Co-funded by the
Erasmus+ Programme
of the European Union

USER GUIDE FOR TEACHERS ON THE CODING4GIRLS SERIOUS GAME APPROACH

Erasmus+ no. 2018-1-SI01-KA201-047013

```
when ▾ key presse
set pen color to
clear
pen up
point in direction 90▾
go to x: -56 y: -138
pen down
move 60 steps
wait 1 secs
turn ↺ 90 degrees
wait 1 secs
move 185 steps
wait 1 secs
```

How do you call the operation that allows you to repeat the code?

2020

O3 – Instructional Support Content

O3/A2 - USER GUIDE ON THE CODING4GIRLS SERIOUS GAME APPROACH

Cite this work

Daniela Tuparova, Boyana Garkova, Joze Rugelj, Mateja Bevcic, Spela Cerar, Matej Zapushek, Michela Tramonti, Alden Meirzhanovich Dochshanov, Carlos V. Carvalho, Rita Durão, Ivanichka Nestorova, Rositsa Georgieva, Hariklia Tsalapata, Olivier Heidmann, Kostas Katsimentes, Christina, Taka Roxani, Sotiri Evangelou, Nadia Vlachoutsou, Nataša Hoić-Božić, Martina Holenko Dlab, Ivona Franković, Marina Ivašić Kos, Luigi Tramonti, , Ahu Şimşek, Kadir Fatih Mutlu, Abdurrahman Saygın, USER GUIDE ON THE CODING4GIRLS SERIOUS GAME APPROACH, Erasmus+ no. 2018-1-SI01-KA201-047013, 2020, https://www.coding4girls.eu/results_03.php

Document Data

Deliverable: O3/A2 - User guide on the CODING4GIRLS serious game approach

Intellectual Output No - Title: O3 – Instructional Support Content

Intellectual Output Leader: South-West University "Neofit Rilski" (Bulgaria)

Contributors

South-West University "Neofit Rilski", Bulgaria

Daniela Tuparova, Boyana Garkova, Ivanichka Nestorova, Rositsa Georgieva

UTH – Greece

Hariklia Tsalapata, Olivier Heidmann, Kostas Katsimentes, Christina, Taka Roxani, Sotiri Evangelou, Nadia Vlachoutsou

University of Rijeka – Croatia

Nataša Hoić-Božić, Martina Holenko Dlab, Ivona Franković, Marina Ivašić Kos

EU-Track – Italy

Michela Tramonti, Alden Meirzhanovich Dochshanov and Luigi Tramonti

Virtual Campus - Portugal

Carlos V. Carvalho, Rita Durão.

University of Ljubljana - Slovenia

Jože Rugelj, Mateja Bevčič, Špela Cerar, Tadeja Nemanič, Matej Zapušek

GOI - Turkey

Ahu Şimşek, Kadir Fatih Mutlu, Abdurrahman Saygın

Disclaimer

This project has been funded by the Erasmus+ Programme of the European Union.

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

TABLE OF CONTENTS

1. INTRODUCTION

The project Coding4Girls addresses open and innovative education and training embedded in the digital era by targeting programming skills that are in high demand in a technology driven society. Furthermore, the ability to design algorithms, to code and to program have become relevant personal skills as they are connected with logic reasoning and problem solving abilities. As such, computer science skills are highly desirable as they are part of the abilities required by employers in innovation-driven sectors that will drive a sustainable economic growth in the coming years leading to higher employment and, as a result, social cohesion.

However, there is currently a shortage of computer science skilled personnel and companies face difficulties in finding employees with the required ICT skill set. CODING4GIRLS aims to close this gap by effectively addressing computer science skills in the late basic school years and on the first years of secondary school, which is the time when many students lose interest in computer science. On top, there is a widely recognized gender gap in this area with girls not being interested in pursuing a computer science career.

CODING4GIRLS addresses inclusive education, training, and youth by promoting equal opportunities between girls and boys in the highly rewarding computer science careers. The project addresses this objective by addressing misperceptions and attitudes of learners, teachers, and parents on computer science careers and their benefits to both girls and boys by demonstrating its links to real-life and highlighting the fact that such careers are rewarding independently of gender or background; but also by addressing the under achievement in computer science skills, which are part of the wider science and technology (STEM) group of competencies.

The project helps build the necessary background among school learners that will enable them to succeed in their future endeavours either academic (in tertiary or other continuing education in computer science) or professional (in vocational or professional activities related to computer science). But CODING4GIRLS also address the development of teacher competencies and the profile of the teaching profession by empowering educators to effectively build desirable computer science skills among their learners. It further enables educators to lead initiatives that promote perspectives of gender equality in the pursuit of academic or professional paths in science.

This user guide for teacher is part of intellectual outputs developed in the frame of the project. The teachers will find information about: main concepts grounded in the Coding4Girls methodology – game-based learning, serious games, design thinking; position of girls in ICT and their perception of serious game; features of Coding4Girls game environment with it two components – teacher part and student part; review of game-based learning approaches for teaching programming and examples of useful programming environments. The guide deals also with lessons learning sheets and their adaptation according Coding4Girls methodology and developed game based environment.

.

2. BASIC CONCEPTS

2.1. Serious games

Game is a broad concept, which makes it hard to point out the most important characteristics that make a certain activity a game. For Thorton the most important feature of the game is interactivity. Johnston sees game as an activity that has rules which all of the participants have to obey, one or several goals, interaction and dynamic visuals (Johnston & de Felix, 1993). Malone (Malone T. , 1981) in his theory exposes fantasy, curiosity, challenge, and control as the most important components of every game. The most comprehensive allocation is made by Garris et al. (Garris , Ahlers, & Driskell, 2002) claiming that those components are competition, challenge, social interactions, conversation, and fantasy. Prensky's theory of game based learning is one of the most influential. He argues that game consists those elements: rules, goals and assignments, results and feedback, conflict situations (competition, challenge, and opposition), interaction and fantasy framework (Prensky , 2001). Authors of the book "Serious games" define game as voluntary activity (a form of freedom) separated from real life (imaginary world that may have or not have relation to real life), that absorbs the player's full attention and is played according to established rules that all players have to follow (Michael & Chen, 2006).

One of the most exposed features of the game is interactivity and it refers to interaction with computer, opponent (computer or human controlled) or with other team players. Computer games are divided by the number of players simultaneously engaged in a game to single player, multiplayer and massively multiplayer games. According to the genre we know arcade, action, war, role playing, strategic, board, sports, simulation, and adventure games. Computer game popularity charts shows that the most played games are massively multiplayer role-playing games, strategy and action games.

To really understand the power of game let us first consider children's play. Children's play is known as one of the most important activities that help develop important skills for entire life. Through play child is improving his intellectual capabilities by discovering the first basic concepts from real world and making viable connections between them. Jean Piaget sees play as incorporation of new materials into existing cognitive structure and consolidation of newly learned behaviour. Freud emphasizes the emotional benefits of play arguing that it reduces the objective and instinctual anxiety – helping child to resolve emotional issues. Social constructivist believe that play is important because of development of social skills.

Learning with play is one of the most common activities in early age but positive effects of playing and games on learning are often overlooked in a formal education. A lot of research has been done in a field of game-based learning showing that if learning material is presented in a computer game format it has positive effects on motivation. Game can help students to draw attention on learning material and at the same time it is a source of fun that gives them enjoyment and pleasure. Having those two components in a learning process, gives us relaxed and highly motivated student who is therefore more willing to learn. Other pedagogic benefits are collaborative learning (multiplayer capabilities), experimental and active learning, problem-based learning, and authentic activities. Use of ICT is nowadays common in a formal education, which gives us an opportunity to develop quality materials in order to foster learning.

Serious games usually refer to games used for training, advertising, simulation, or education. They allow learners to experience situations that are impossible in the real world for different reasons, such as safety, cost or time. They are supposed to have defined learning outcomes and are claimed to have positive impacts on the players' development of new knowledge or different skills. To design a good serious game, we need to be able to design and produce good software. But serious games are much more than software. We also need to be able to design and produce good instruction. The potential value of serious games for learning seems high, but there remains resistance to the use of games in the classroom. A reasonable way to convince more teachers to try games is through pedagogy, connecting elements of existing games designs with accepted learning and instructional theories.

Rieber, Smith and Noah (Rieber , Smith , & Noah, 1998) stated already in 1998 that there are two distinct applications of games in education: game playing and game designing. Game playing is the traditional approach where one provides ready-made games to students. Game designing assumes that the act of building a game is itself a path to learning, regardless of whether or not the game turns out to be interesting to other people. The idea of "learning by designing" is based on the assumption that active participation in the design and development process is the best way to learn something. This approach has gained increased prominence due to the proliferation of computer-based design and authoring tools.

2.2. Game-based learning

There are several reasons that draw educator attention to games (Gee, 2007). In formal education we experience a shift from traditional didactic model, which is focused on instruction, to learner-centred model that emphasizes the active learner's role. We also changed the view of learning goals from lower taxonomic levels, like just recalling information, to higher levels, such as finding and using of information in new settings (Rugelj, Serious computer games design for active learning in teacher education, 2016).

Prensky (Prensky , 2001), Gee (Gee, 2003), and Whitton (Whitton, 2009) defined game based learning as a process of learning with the use of digital games. Games can provide motivation for learning, thus increasing the chance that the desired learning outcomes will be achieved. Learning is defined as the acquisition of knowledge or skills through experience or practice, and what better way to learn than through a game (Pivec & Kearney, 2007), (Pivec, Koubek, & Dondi, 2004) Almost all studies about game based learning show that students are highly motivated when learning materials are presented in a computer game format and that this has positive effects on the acceleration of a learning process (Zapušek & Rugelj, 2013). Students need motivation to focus on what needs to be learned but for any quality learning to occur this is not enough. Comparing learning outcomes of students who learned with computer games and those learning with another type of learning materials shows that there is no significant difference between them. This is usually because of inappropriate game design. Games can be very appealing to students but if they entertain and not teach, the use of games in education does not make much sense. So, we have to find out what are the elements that make computer game an educational computer game.

Gross (Gross, 2003) claims that digital games for educational purposes must have well defined learning goals and have to promote development of important strategies and skills to increase cognitive and intellectual abilities of learners.

According to Malone (Malone T. W., 1981b) and Garris et al. (Garris , Ahlers, & Driskell, 2002), the elements contributing to educational values of digital games are sensual stimuli (visual and audio representations of learning material), fantasy (context presented in imaginary setting), challenge (demanding or stimulating situation) and curiosity (desire to know or learn). These elements must be incorporated on an integrated platform, to structure objectives and rules, a context of meaningful learning, an appealing story, immediate feedback, a high level of interactivity, challenge and competition, random elements of surprise, and rich environments for learning.

A game can be instantiated for learning as it involves mental (and sometimes physical) stimulation and develops practical skills – it forces the player to decide, to choose, to define priorities, to solve problems, etc. Immediate reward (and feedback) is a major motivational factor, whether it is translated as game entities (more life power, access to new levels, etc.) or as neurological impulses (happiness, feeling of achievement, etc.). Games can be social environments, sometimes involving large distributed communities. They imply (Baptista & Vaz de Carvalho, 2010) self-learning abilities (players are often required to seek out information to master the game itself), allow transfer of learning from other realities, and are inherently experiential with the engagement of multiple senses.

Van Eck (van Eck, 2006) argues that games and play can be effective learning environments not because they are fun but because they are immersive, require player to make frequent important decisions, have clever goals, adapt to each player individually and involve social network.

If we consider a model of game based learning by Garris, Ahlers, and Driskell (Garris , Ahlers, & Driskell, 2002), the main characteristic of educational game is that instructional content is blurred within game characteristics. Students are playing the game and having fun, forgetting about the “learning” part of the experience even though they are constantly presented with new concepts which they have to adapt in order to be successful in game. We should foster motivation with game design promoting repeating the cycles within game context. Player is expected to elicit desirable behaviours based on emotional and cognitive reactions that result from interaction with and feedback from gameplay.

Gee (Gee, 2003) argues that features of video games with high learning potential fall into two categories: ‘non-game’ features, which may also appear in non-game contexts, and ‘game’ features, which relate more to the ‘gameness’ of games. Despite this distinction, it should not be assumed that the ‘non-game’ features would work as well for learning if they were detached from the ‘game’.

‘Non-game’ features of games with high learning potential are:

- Empathy for complex systems – to look at complex system from the inside in order to understand how its variables are interacting.

- Simulations of experience and preparations for action.
  In video games, players see the virtual world in terms of how it affords the sorts of actions they need to take to accomplish the goal of winning.
- Distributed intelligence via the creation of smart tool.
  Good video games distributes intelligence between a real-world person and artificially intelligent virtual characters in order to represent macro and micro view of the situation.
- Cross-functional teamwork.

  Good video games may be able to teach collaboration and cross-functional teamwork for institutions like schools and workplaces. Players interact with each other not in terms of their real-world characteristics but through their functional gaming identities. They may also choose to use their real-world race, class, culture, and gender as strategic resources but they are not forced into pre-set racial, gender, cultural or class categories.
- Situated meaning

  Dialogue and experience are essential for people to be able to relate words to actual actions, functions, and problem solving. Since video games are simulations of experience, they can situate language in specific contexts.
- Open-endedness: melding the personal and the social

  In good open-ended games, players construct their own goals, which are based on their own desires, styles and backgrounds. Combination of personal and in game goals produces a state of high motivation.

Features of a ‘good game’ that relates to effective learning are the following:

- Motivation

  Video games are profoundly motivating for players, and it is important to understand the sources of this motivation if it is to provide a foundation for learning.
- The role of failure

  The price of failure is lowered and is often seen as a way to learn the underlying pattern. These features of failure in games allow players to take risks that might be too costly.
- Competition and collaboration

> Many gamers, including young ones, enjoy competition with other players in games but may not see competition as pleasurable and motivating in school. Competition in video games is seen by gamers as social, as much about gaming as winning and losing.

The design of games that relates to:

- Interactivity – player doesn't just passively consume knowledge but has control over content;
- Customization – based on learning styles and providing multiple routes to success;
- Strong identities - Good games offer players identities that trigger a deep investment on the part of the player and which are clearly associated with the functions, skills and goals one has to carry out in the virtual world;
- Well-sequenced problems - In connectionist approaches to learning, it is argued that sequencing is crucial for effective learning in complex domains;
- A pleasant level of frustration - adjusting challenges in such a way that a range of players can experience the game as challenging but do-able;
- A cycle of expertise - repeated cycles of extended practice and tests of mastery;
- ‘Deep’ and ‘fair’ – game must be challenging but set up in a way that leads to success.

Gameplay elements should be initially simple and easy to learn and become more complex the more the player comes to master them.

Computer games used in educational setting are proven to increase motivation. But highly motivated learners are not enough for any quality learning to occur. We also need good learning materials so learners will actually gain new knowledge from materials presented in a computer game form.

Despite the game’s positive influence on motivation, teachers still hesitate to use serious games in teaching. As they pointed out in our investigation, games can be too time-consuming for use in a classroom. Therefore, they often use serious learning games as a home-based learning activity or as a motivation in the introductory lessons.

2.3. Design thinking methodology

Design thinking ideas emerged in the late 1960s, but as a concept, "design thinking" appeared in Stanford University in the late 1980s and 1990s. In the business world, the method became widespread after it was promoted by IDEO's Tim Brown, who said: "Design thinking is a human-centred approach to innovation that draws from the designer's tools to integrate people's needs, technology capabilities and business success requirements. " (IDEO Design thinking, 2008). At the beginning of the 21st century, design thinking became extremely popular and IT companies began to apply the approach to developing their products. Since 2005, for example, SAP has applied design thinking "as a problem-solving philosophy and as an innovative end-user-focused approach" (Innovation Starter, 2015), and as a result, the development of new products leads to the desired result. Companies such as P&G, IBM Design, GOOGLE, AMAZON and Cisco integrate design thinking into their entire business and organization. (Naiman, 2019)

The concept developed by Stanford University identifies design thinking as "a leading methodology for creative problem solving and innovation creation and is used daily by thousands of companies and organizations around the world. The goal is to develop in adolescents the skills of the 21st century - teamwork, problem solving, creativity, empathy, confidence, patience, concentration, experimentation." (Karakehayova, 2019).

The Stanford model consists of 5 stages (Fig. 2.1.)

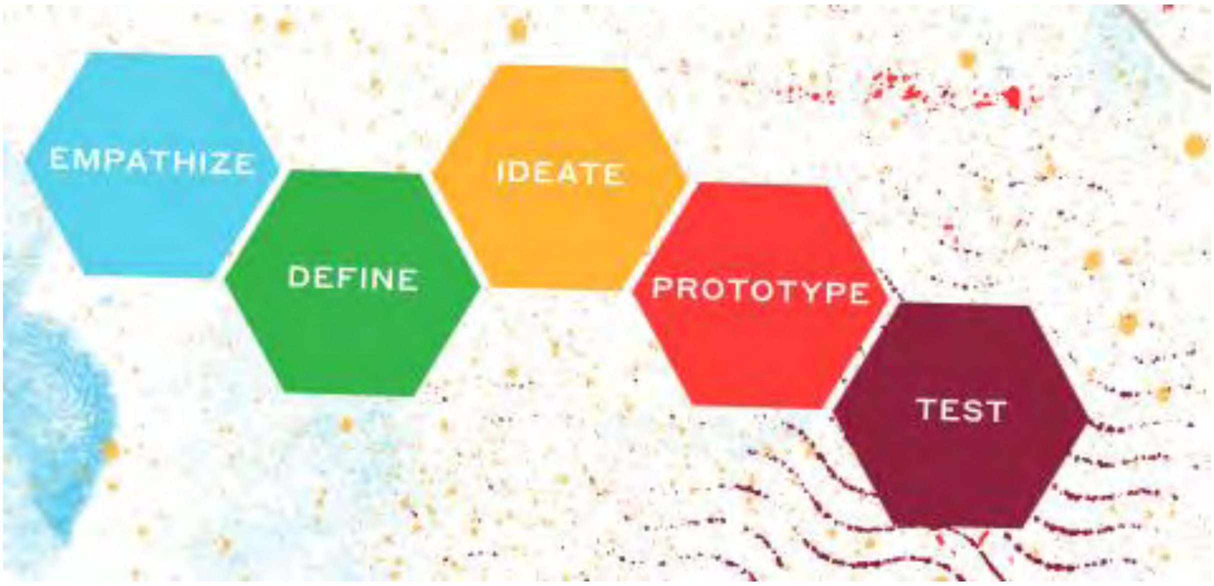


Fig. 2.1. Stages in Stanford design thinking model (Doorley, Holcomb, Klebahn, Segovia, & Utley, 2018)

The main stages and characteristics of design thinking could be summarised as follows (Table 2.1.) *(Georgieva & Tuparova, 2020)*

Table 2.1. Stages and characteristics of design thinking (based on (Innovation Starter, 2015), (Naiman, 2019), (A project of the K-12 Initiative at Stanford University, 2009)

Practices	Principles	Stages		
		Short process	**Extended process**	**dSchool Stanford**
Developing a deeply empathetic understanding of user needs	Discovery - Meeting a new challenge. How to solve it?	Observing and studying the behavior and experience of users, researching and defining a problem and point of view through empathy.		Empathize
Formation of heterogeneous teams	Interpretation - What did I learn and how to interpret it?	Generate many ideas in a short period of time.		Define
Dialogue-based conversations	Ideation - I see an opportunity, how to shape it into an idea?	Selection and classification of ideas		Ideate
Generating desicions through experimentation;	Experimentation - I have an idea how to build it?	Prototyping - a product that receives quick feedback from users.		Prototype
Using a structured and facilitated process.	Evolution - I tried something new, how to develop it?	Testing/Feadback - how to improve the prototype, if necessary		Test
			Testing if the idea works	
			Learning and improvement based on feedback	

Design thinking benefits not only the business but also the non-governmental, educational and health sectors. More and more educational institutions are turning to design thinking. Gradually, good practices for the application of design thinking in education are increasing. (Georgieva & Tuparova, 2020) In the implementation of design thinking approach the teacher need to make a connection between the individual stages in the design thinking model and the appropriate teaching methods that can be applied. The design thinking requires the application of a wide range of teaching methods.

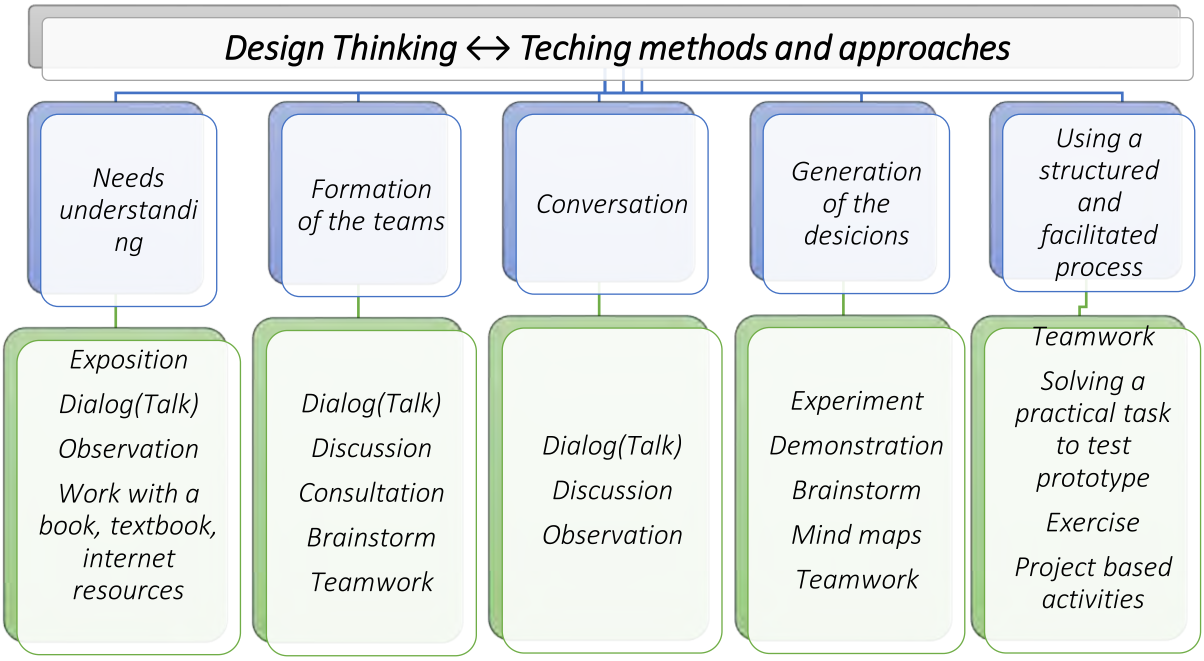


Fig. 2.2. Teaching methods and design thinking (Georgieva & Tuparova, 2020)

2.4. Learning theories and game-based learning

A lot of educational computer games were designed according to behaviouristic theory of learning in the past (Rugelj & Lapina, 2019). They were implemented in the form of programmed instruction. Learners were offered a stimulus in the form of a question or any other type of task or problem to be solved. Learners response immediately by selecting one of the offered answers. If the answer is correct, the game provides some kind of positive response in a form of a positive character reaction or happy tune that stimulates positive emotions. This instance of action-reaction pair enforce connection between a question and the correct answer. In the case of wrong answer, a reaction is provided in a form of negative stimuli and the connection is weakened. Point-and-click games and quizzes have drill and practice concept build in and are typical representatives of games, based on behaviourism. They are suitable to learn basic arithmetic operations or to support memorisation of fractographic data, i.e. learning goals on the lowest levels of Bloom's taxonomy.

Cognitive learning theory emphasizes learner's cognitive activity and formation of appropriate mental models. Learners learn fundamental concepts from her teacher or form learning resources and then use logical deduction to gain new knowledge. Puzzles and strategy games as an environment for decision-making examples of the cognitivist approach. The most advanced forms of cognitive theory based games are based on intelligent tutoring systems, where machine learning algorithms are employed to model student and expert knowledge in order to provide personalized learning material.

Constructivism is an alternative view suggesting that learners construct their own knowledge; a number of individually constructed knowledge representation, all equally valid. Learning is an active process of constructing (rather the acquiring knowledge), built recursively on knowledge that user already has. In a process of construction, sensory data is combined with existing knowledge to create new viable mental models, which are in turn the basis for further construction.

Constructivist learning emphasizes discovery and inquiry learning arguing that students should be placed in an environment (which can be modeled with computer game) where they construct their own knowledge. Three fundamental principles define the constructivist view of learning:

- each person forms her own representation of knowledge,
- learning occurs when the learners' exploration uncovers an inconsistency between their current knowledge representation and their experience,
- learning take place within a social context and interaction between learners and their peers is a necessary part of the learning process.

Learning materials provide instruction that consists of supporting knowledge construction rather than declaring the knowledge in behaviouristic fashion. Computer game simulations replicate various real-life scenarios in computer game format. They present model of abstracted reality in which learner inhabit a certain role. The task of teacher is to provide guidance and feedback when student is learning, i.e. constructing viable mental models.

Games can lead to changes in attitudes, behaviour, and skills of the player. Shute and Ke (Shute & Ke, Games, Learning, and Assessment, 2012) found out that there is convergence between the core elements of a good game and the characteristics of productive learning. Game design has a lot to teach us about learning, and contemporary learning theory can teach us about designing better games (Shute, Rieber, & Van Eck, 2012). Marshall McLuhan, Canadian philosopher of communication theory, who foresaw World Wide Web in the sixties of previous century, when he talked about global village, stated: "Anyone who makes a distinction between games and learning doesn't know the first thing about either." (Shute, Rieber, & Van Eck, 2012)

Whitton and Moseley (Whitton & Moseley, 2012) proposed a framework for good practice in serious games design from an active learning perspective. According to his guidelines, the game environment should support active learning by encouraging exploration, problem-solving and enquiry, engender engagement with explicit and achievable goals, be appropriate for the learning context, support and provide opportunities for reflection, provide equal opportunities for all students, provide ongoing support with a gradual introduction of increasing complexity, and be supported with some help or hints.

Several studies have shown that serious games can support learning with motivation, engagement and fun (Hijon-Neira, Velazquez-Iturbide, Pizarro-Romero, & Carrico, 2015). Learning programming requires many competences such as logical thinking, problem solving, and the ability to understand abstract concepts. For this reason, many students find computer programing difficult to learn. This fact can lead to low motivation to study introductory programming courses. In order to improve motivation and to enhance students' learning attitude towards programming, teachers are looking for simulative approaches to learning.

Programming is best learned by practice and, if students are to learn effectively, at least some of this practice will have to be self-directed or in collaboration with peers. Teacher's key role is to persuade students to do this and thus to motivate them (Feldgen & Clua, 2004).

Students in the introductory computer programming course design and develop programs (Rugelj & Lapina, 2019). This is typical an active learning approach. If we introduce project work in groups, which has proven to be a very effective way of learning programming (Nančovska Šerbec, Kaučič, & Rugelj, 2008), we actually can talk about the trialogical learning principle. The trialogical learning plan consists of forms of learning in which students collaboratively develop, change or create common artefact (i.e. computer program in our case) in a systematic process (Kafai, 1995). It focuses on the interaction that occurs with the creation of concrete artefacts, not just between people ("dialogical approach"), or within one's mind ("monological" approach) (Paavola & Hakkarainen, 2005).

3. GIRLS, ICT AND SERIOUS GAMES

3.1. Position of women in ICT sector – state of the art

The digital society places increasing demands on the competencies of modern peoples. There is a growing need for people who are not only digitally literate, but also have the skills to create and maintain software applications in various fields of human activity. The media constantly comments on the shortage of qualified specialists in the sector of information and communication technologies (ICT). There is a continuous growth of this sector.

In 2019, about 7.8 million people worked as ICT specialists across the European Union (EU). (Eurostat, 2020). But tis is only 3,9% of the all employed in EU.

According to (Eurostat, 2020) in the EU-27 in 2011 women in the IT sector were 17% and in 2019 - about 17.9%. According to Eurostat data for the period 2007-2019, on average for the 27 EU member states, women in the IT sector decreased from 22.5% to 17.9%. In 2019, women are 17.3% of the total number of persons employed in the EU with an ICT education, down from 20.2% in 2009. (Eurostat, 2020).

The current state in employment of women in ICT sector in Coding4girls project partners countries is presented in Fig. 3.1.

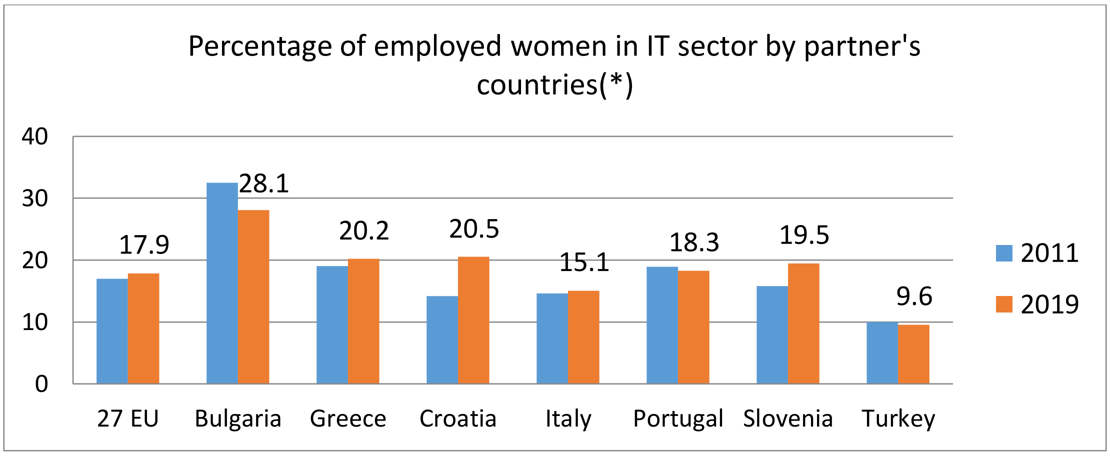


Fig. 3.1. Employed women in ICT sector (in percentage) in 2011 and 2019

3.2. Attitudes of the girls to serious games

The results of study carried out by Vermeulen et al. (Vermeulen, Van Looy, Courtois, & De Grove, 2011) showed that gender differences were consistently present, but previous experience substantially affect the findings (Rugelj & Lapina, 2019). Women are playing games more frequently but shorter periods of time than men. They prefer abstract, short and easy to master games such as casual (e.g. Tetris) and social network games, while men are more likely to play ‘core’ genres. This category refers to skill-based games which are time-consuming and generally feature high-quality three-dimensional graphics such as shooters, fighting, action-adventure, sports, racing, strategy, role-playing and MMO games. Non-core genres include platform, adventure, simulation, party, serious, classic and casual games. Core genres are played more by males than by females. The study has found that women are fonder of puzzle games, while racing, rhythm, simulation and virtual games are played by both women and men. Another study discovered that women favour party games (such as music and dance games) and classic ‘retro’ games.

Alserri et al (Alserri, Zin, & Wook, 2018) made an extensive survey of related papers on effective Serious Game elements, such as motivational elements of digital games players, effective educational game elements, and female preference elements. They used the results to create conceptual model for gender-based engagement in Serious Games. We use this model in the Coding4Girls project to increase girls' motivation for programming through the appropriate selection of games that will be designed and implemented by students in game-design-based learning.

The authors stated that the study has revealed that the motivation to play a specific type of digital game depends on gender. The stereotype that females do not play computer games is no longer valid. The differences in gender preferences for digital games, found out in this research, are very similar to the already presented differences, identified by Vermeulen et al. (Vermeulen, Van Looy, Courtois, & De Grove, 2011). Females prefer explorative and creative gameplay more than males. They play more frequently but for less time than men, and their preferences for game genres are also different. Males have been found to spend more time playing computer games than females. Females also prefer puzzle games, social games with rewards offered in the games, educational games, simulation game genres, as well as collaborative, and exploration gaming, virtual life, virtual world, and party games. Moreover, females like to play adventure games, but they prefer to observe others first before playing themselves. The motivation preference elements in gaming for females are challenge, escapism, fun, social interaction, motivation, fantasy, competition, and arousal. Both males and females like racing, simulation, and virtual games.

Phan et al. (Phan, Jardina, Hoyle, & Chaparro, 2012) came to very similar conclusions in their study about gender differences between male and female gamers in terms of computer game usage, preference, and behaviour.

According to the study of Tuparova at all. (Tuparova, Tuparov, & Veleva, 2019 b), there are differences in boys’ and girls' preferences to features and elements of games such as: Graphical design; Sound; Possibility to play the game on different devices: computer, tablet, smart phone, etc.; Possibility to involve many players; Possibility to play on-line; Possibility to switch to more complex game levels. The most preferred feature of the games for girls is *Possibility to switch to more complex game levels*. For the boys most preferred future is *Logic and plot/scenario of the game.* The lowest-interest feature for girls is the sound in the games, while for the boys - awards in the game. Regarding to the end used device the smartphone is the most used gaming device by both boys and girls. For girls, the second-in-use device is the laptop, followed by a tablet. For boys second-in-use are a desktop computer and a laptop. The authors observed that boys show greater preference than girls to story games and girls have greater preference than boys to memory games. Gender-independent are preferences for the following game types: games with social elements; attention/concentration, games for speed of reactions, role games, puzzles, adventure games. The lowest-interest type of game for both girls and boys is the puzzle game.

We have to take into account specific situation in the Coding4Girls project, which is actually based on game design-based learning and prepare tasks which are interesting and motivating for girls. A programming environment that presents programming as a means for creating animated movies (i.e. storytelling) or simple games can be suitable for girls as most of them can come up with an idea for a story or simple game they would like to create (Wolz, Barnes, & Bayliss, 2009). Both are naturally sequential and are unlikely to require advanced programming concepts immediately, they are a form of self-expression and provide girls an opportunity to experiment with different roles, and non-programming friends can readily understand and appreciate an animated story or game, which provide an opportunity for positive feedback. Kelleher at all (Kelleher , Pausch, & Kiesler, 2007) used Storytelling Alice and Generic Alice for both cases respectively. They claim that several studies of children programmers have found that when girls and boys have similar experience with computer programming, they are equally interested in and effective at learning to program. But programming performance is correlated with the amount of time users spent programming and their prior programming experience.

4. TEACHING PROGRAMMING THROUGH GAMES

This section will be focused on presenting and comparing different approaches and platforms/game environments that can be used for teaching programming skills to children.

4.1. Approaches for teaching programming through games

There is currently a wide variety of approaches for teaching programming to novice coders. The approaches could be classified as:

- Learning programming by game-based design - also known as game-design based learning corresponds to the use of programming languages to learn how to code by designing and creating games.;
- Learning programming in game-based environment.
- Learning programming by game design in game-based environment. The Coding4Girls methodology belongs to this category of programming learning environments.

One example is the use of block based visual programming languages that are especially user friendly, easy to operate (on a drag-and-drop basis), prevent errors regarding syntax and logic (Ouahbi, Kaddari, Darhmaoui, Elachqar, & Lahmine, 2015) and present content in a very intuitive fashion. Scratch (Resnick, et al., 2009), https://scratch.mit.edu/ , Snap! (Weintrop & Wilensky, 2015) https://snap.berkeley.edu/ , or Alice 2/3 https://www.alice.org/about/ are well-known and successful examples of visual programming languages. Next, we have a table comparing the characteristics of the most well-known visual programming languages (and environments).

Table 4.1. Characteristics of some visual block-based programming environments.

	PLATFORM	TARGET GROUP	LANGUAGE(S)	FREE/PAID
Scratch	Web-based (but with offline version)	8-16	Visual computer programming language	Free
Snap!	Web-based (but with offline version)	12-20	Visual computer programming language	Free
Alice	Windows, Mac OS X, Linux	12-20	Visual computer programming language	Free
Tynker	Web-based, iOS	7+	Visual computer-programming language, JavaScript, Python	Paid

Computer games can also be used to learn how to code, either by encouraging students to develop and create their very own video games (learning by game-design) or by allowing them to play serious games whose learning outcomes encompass learning outcomes related to programming (game-based learning). The computer games purposed for educational activities have the potential to create a motivating and fun learning environment, as they contain activities that meet educational standards, learning objectives, provide feedback and can achieve high educational outcomes (Georgieva & Tuparova, 2019).

The idea behind using games to teach programming comes from the fact that these, by being more engaging and motivational, allow for students to learn computational thinking and programming skills in entertaining and familiar environments, before transferring those skills to learning a programming language (Kazimoglu, Kiernan, Bacon, & Mackinnon, 2012), 2012). Still according to Kazimoglu et al. (Kazimoglu, Kiernan, Bacon, & Mackinnon, 2012), serious games for learning programming and computational thinking should not only be created for fun, and should also consider a curriculum and skills, making a difference between different programming constructs and encouraging good programming practices (by ways of gamification, like achieving a high score).

Serious games can also be useful for programming by turning unpleasant operations into interesting experiences —Shabanah et al (Shabanah, Chen, Wechsler, Carr, & Wegman, 2010) created an Algorithm Game Designer aimed at facilitating the creation of algorithm games, in order to improve engagement in algorithm visualization systems. Grivokostopoulou et al (Grivokostopoulou, Perikos, & Hatzilygeroudis, 2016) developed a game for teaching AI algorithms, based on the Pacman game, so that through algorithm visualizations and animations students could be helped in their learning process, by demonstrating the way an algorithm operates, its functions and how to make proper decisions based on specific parameters.

With the developing widespread adoption of faster internet connection speeds and rising availability of computers in everyone's homes, online game-based platforms like CodeCombat (https://codecombat.com/) and LightBot (https://lightbot.com/) have begun to appear. Combéfis et al. (Combéfis, Beresnevičius, & Dagiene, 2016) reviewed the main types of online platforms used for teaching programming skills, and have concluded that the following factors can make a serious game more motivating and successful: 1) a feedback and assessment process; 2) aesthetics; 3) collaboration and multiplayer aspects; 4) guidance through the game; 5) no negative consequences; 6) music; 7) a medium-level of challenge to keep the interest of the players.

Regarding the learning outcomes of Serious Games for Learning Programming, according to an extensive review of 49 serious programming games by Miljanovic et al (Miljanovic & Bradbury, A Review of Serious Games for Programming, 2018), most serious games for programming focus on problem solving and fundamental programming concepts, with few games focusing on data structures, development methods and software design.

There are also games that, while they may not teach programming skills, are capable of teaching in an indirect but effective way key computational thinking skills, like abstraction (eg. Tetris), decomposition (eg. Sudoku), algorithm thinking (eg. Puzzle games), evaluation (eg. Memory Games), and generalization (eg. Pattern Games) (Frankovic, Hoic-Bozic , & Nacinovic-Prskalo, 2018).

In game-design based learning, we often have the combination of the use of visual programming languages and their block-based environments with the process of design and coding of games. In a study that compared using the Scratch and Pascal (procedural language through command line or text editor) coding environments with novice programmers, it was found that learners were much more motivated to continue their studies in computer sciences when using Scratch (Ouahbi, Kaddari, Darhmaoui, Elachqar, & Lahmine, 2015). The study also found that the creation of games and stories made learners more creative and autonomous while learning programming. Another study, conducted by Weintrop et al (Weintrop & Wilensky, 2015) , comparing the use of Snap! and Java by high school students found that the blocks-based approach had an easier learning curve than Java – thus, it is in accordance with the view that blocks-based programming (like Scratch and Snap!) is more accessible to novice programmers. According to the study findings, this was due to the fact that blocks are easier to read, due to the visual nature of the blocks that provide cues on how they can be used, they are easier to compose, and serve as memory aids.

Integrated approach for teaching programming and it influences to achievements of 14 years old students is discussed in (Tuparova, Nikolova, & Tuparova, 2020). This approach combines two stages:

- “Using existing educational computer games: for knowledge and skills acquiring, motivation of learning activities, development of algorithmic thinking; for experimenting with existing educational computer games to inquire of game rules, interface elements and their properties and events;
- Designing and developing educational computer games including mathematical model of the game, design of graphical user interface (GUI); coding, testing and verifying.”

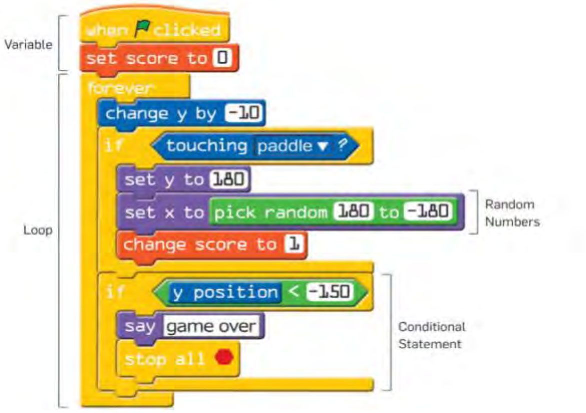
For implementation of the approach is used C# programming environment. The model is tested with 126 students with specialisation in Informatics, divided in two groups – Experimental and Control group. Achievements of the students are measured through practical task and paper-based test. The results show that there is no difference between achievements of boys and girls in experimental group. But authors found that girls in experimental group achieved higher results than girls in control group.

4.2. Game based environments for teaching programming

Next, we present some examples of the use of different environments that use games to teach coding and programming.

4.2.1. Learning by game-design

Scratch



```
when ⚑ clicked
set score to 0
forever
    change y by -10
    if touching paddle ▾ ?
        set y to 180
        set x to pick random 180 to -180
        change score to 1
    if y position < -150
        say game over
        stop all ●
```

Fig.4.1 Sample Scratch Script

Scratch (https://scratch.mit.edu/) is a block-based programming environment, that is, it allows for learners to assemble programs by snapping together code blocks and receiving visual feedback (Weintrop & Wilensky, 2015). This way, learners familiarize themselves with the fundamentals of programming without having to worry about syntax (Ouahbi, Kaddari, Darhmaoui, Elachqar, & Lahmine, 2015).

Scratch can be used as an introduction to programming, especially among younger students, but also as a way to facilitate the transition to other programming environments, therefore introducing and fostering interest in computer science (Wolz, Maloney, & Pulimood, 2008). According to a research study conducted by Meerbaum-Salant et al. (2013), most students can achieve a reasonable level of Computer Science concepts through Scratch. However, difficulties arise when teaching specific topics that require a greater level of abstraction. This can, however, be solved if students are accompanied in the process by the teacher.

The Scratch website supports a worldwide network of users, hosting a community of programmers which allows users to share projects (Meerbaum-Salant , Armoni, & Ben-Ari, 2013). Scratch is available in more than 50 languages, including Bulgarian, Croatian, English, Greek, Italian, Portuguese, Slovenian and Turkish. It also has an offline editor that allows for the program to be downloaded to a computer and run without an internet connection.

Snap!

Snap! (https://snap.berkeley.edu/) is also a visual block-based programming language whose design is based on Scratch but adding some additional features. Snap!, similarly to Scratch, is web-based, but also has the possibility to be run offline through a browser. On top of the features of Scratch, Snap! adds class lists, class procedures, class sprites, class costumes, class sounds and class continuations, thus making it more suitable for more

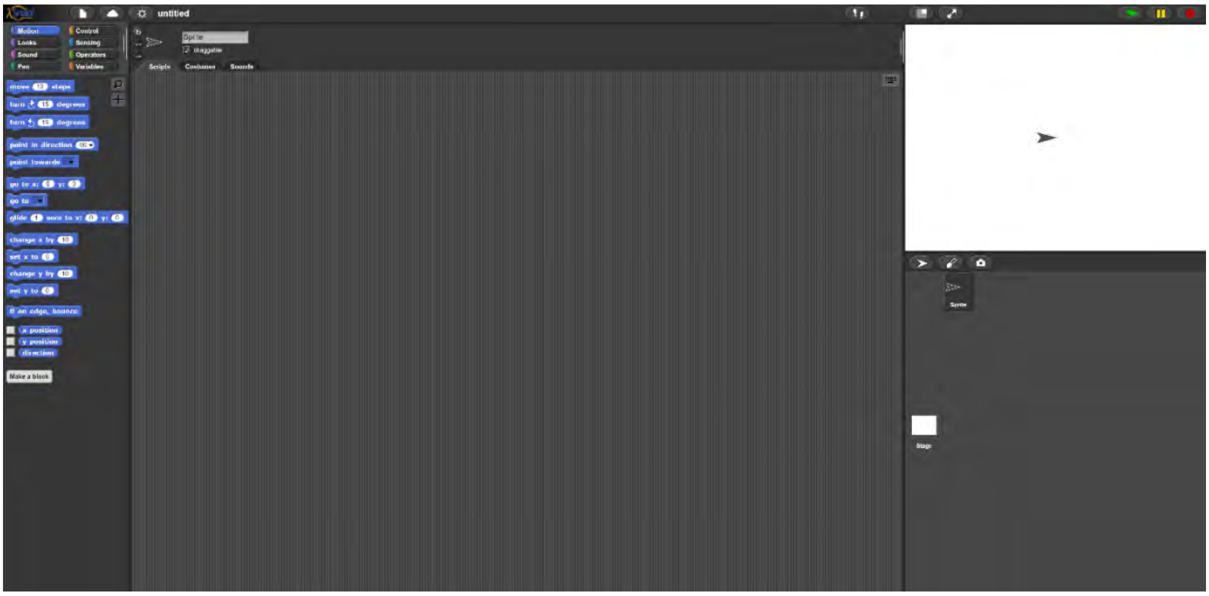
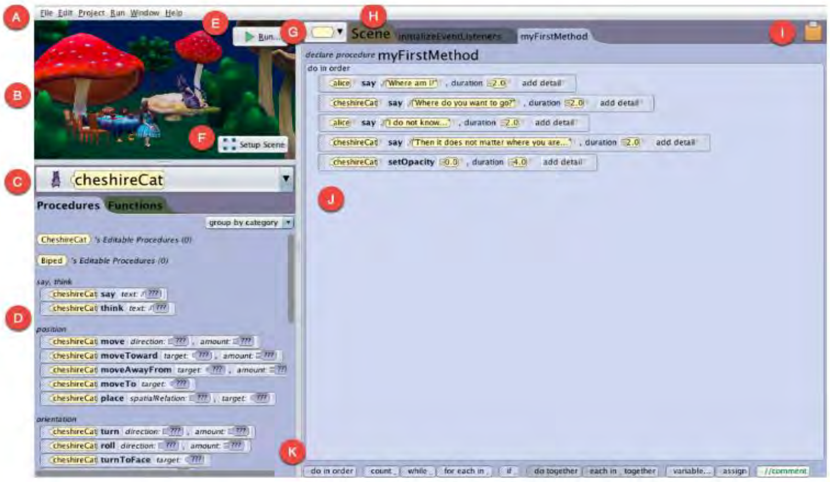


Fig.4.2 Snap! User Interface

advanced audiences than Scratch. Snap! is available in over 40 languages, including Bulgarian, Croatian, English, Greek, Italian, Portuguese, Slovenian and Turkish.

Alice



```
declare procedure myFirstMethod
do in order
    alice say "Where am I?" , duration 2.0  add detail
    cheshireCat say "Where do you want to go?" , duration 2.0  add detail
    alice say "I do not know..." , duration 2.0  add detail
    cheshireCat say "Then it does not matter where you are..." , duration 2.0  add detail
    cheshireCat setOpacity 0.0 , duration 4.0  add detail
```

```
CheshireCat 's Editable Procedures (0)
Biped 's Editable Procedures (0)
say, think
    cheshireCat say text: ???
    cheshireCat think text: ???
position
    cheshireCat move direction: ??? , amount: ???
    cheshireCat moveToward target: ??? , amount: ???
    cheshireCat moveAwayFrom target: ??? , amount: ???
    cheshireCat moveTo target: ???
    cheshireCat place spatialRelation: ??? , target: ???
orientation
    cheshireCat turn direction: ??? , amount: ???
    cheshireCat roll direction: ??? , amount: ???
    cheshireCat turnToFace target: ???
```

Fig. 4.3 Alice 3 Code Editor

Alice https://www.alice.org/ is a block-based programming environment that aims to motivate learners to program by fostering their creativity through the creation of animations, interactive narratives or simple 3D games (Kelleher , Pausch, & Kiesler, 2007). It allows for the creation of virtual worlds with a Virtual World Editor where learners can add 3D objects and add functions and methods to existing 3D objects. Once the Virtual World is created the learner can write code to develop the logic of the game/narrative/animation (Sykes, 2007). Alice is a good programming language for learners with no previous experience, as it allows for them to see the animated programs run as they write their code (Cooper, Dann, & Pausch, 2000).

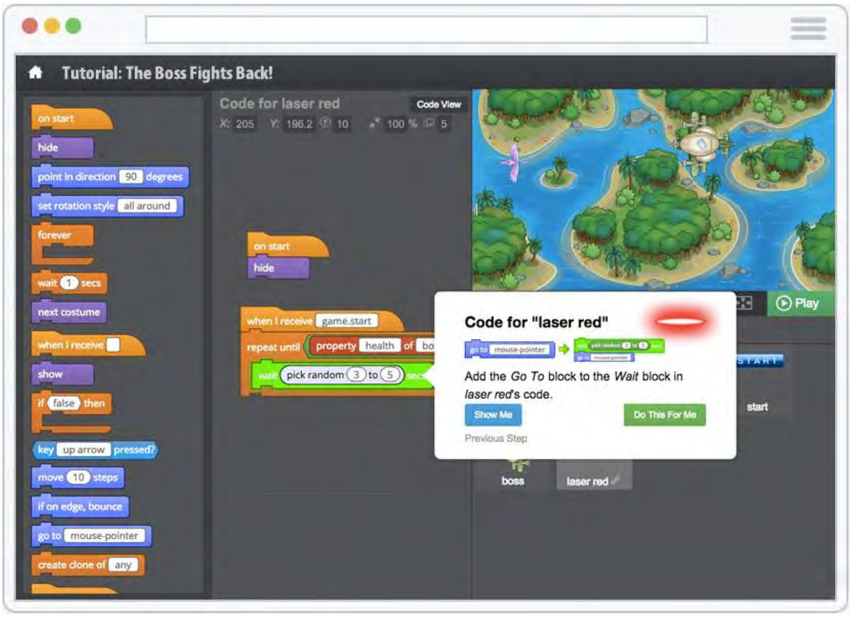
Moreover, another study that focused on Storytelling Alice (a programming environment based on Alice 2 with additional story writing creative tools) found that although both Generic Alice and Storytelling Alice were equally successful at teaching programming concepts to girls, with Storytelling Alice girls were more motivated and increased their time programming which had a positive impact in programming.

Alice 3, the most recent instalment of the series, is available in 13 languages, including English, Portuguese, Greek, Slovenian, and Bulgarian, English, Spanish, Portuguese and German.

Tynker

Tynker https://www.tynker.com/ is an educational programming platform based on a drag-and-drop visual programming language. Unlike the other programming environments presented, this one is a commercial product. One aspect it adds to Scratch consists in how it approaches coding as a game, in which the users are required to create a program to make the characters move, interact, and perform different tasks. It presents problems the players need to solve so that children begin to recognize patterns (Geist, 2016). Tynker also provides a built-in tutor to give step-by-step instructions, so that the learner can learn how to apply coding concepts. The learner is also able to visualize the blocks in JavaScript code, enabling them to understand the block in a text-based programming language.

Tynker provides over 23 Programming Courses, 11 iPad courses and 2000+ coding activities and it provides individual plans and family plans (for up to 4 members). It is only available in English.



Tutorial: The Boss Fights Back!

Code for "laser red"

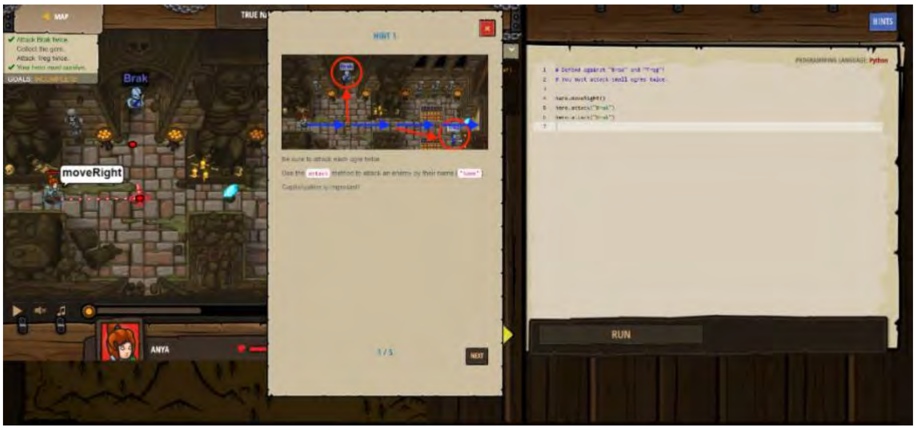
Add the *Go To* block to the *Wait* block in *laser red*'s code.

Fig.4. 4 Tynker User Interface

4.2.1. Learning programming in Game-based environment

Next, we present some platforms based on games meant to develop coding abilities.

Code Combat



✓ Attack Brak twice.
Collect the gem.
Attack Treg twice.
✓ Your hero must survive.
GOALS: INCOMPLETE

Be sure to attack each ogre twice.
Use the attack method to attack an enemy by their name "Name".
Capitalization is important!

```
# Defend against "Brak" and "Treg"!
# You must attack small ogres twice.

hero.moveRight()
hero.attack("Brak")
hero.attack("Brak")

```

Fig. 4. 5 Code Combat Gameplay

Code Combat https://codecombat.com/ is a web-based adventure game. It has plans for both individual students and classes, providing also resources for teachers to use during classes. It has more than 100 free-to-play and subscriber-only levels. It is available in over 50 languages, including Bulgarian, Croatian, English, Greek, Italian, Portuguese, Slovenian and Turkish.

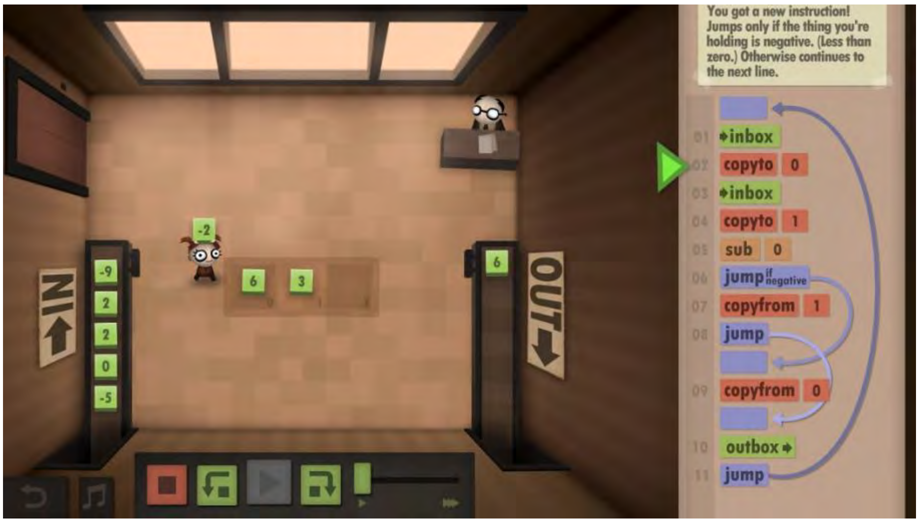
Learners are required to write their own code (either in JavaScript or Python) to make the characters move, interact, and achieve different tasks. It is not block-based, requiring learners to write their own code. It provides hints along the way and introduces concepts gradually. The game is divided in 5 different worlds, introducing different concepts as the player progresses through the game: 1) Kithgard Dungeon – syntax, methods, parameters, strings, loops and variables; 2) Backwoods Forest – If/else, Boolean logic, relational operators, functions, object properties, event handling, input handling; 3) Sarven Desert – Arithmetic, counters, while-loops, break, continue, arrays, string comparison, finding min/max; 4) Cloudrip Mountain – Object literals, remote method, invocation, for-loops, complex functions, drawing, modulo; 5) Kelvintaph Glacier – Advanced Techniques.

According to Miljanovic et al. (Miljanovic & Bradbury, A Review of Serious Games for Programming, 2018) the game's educational content comprises different conceptual aspects of algorithms design and problem solving, fundamental programming concepts (such as Syntax & Semantics, Variables & Primitive Data Types; Expressions & Assignments, Input &

Output, Conditionals & Iteratives, Functions & Parameters, and Recursion) and fundamental data structures (such as Arrays, Heterogeneous Aggregates, and Abstract Data Types).

Human Resource Machine

Human Resource Machine http://tomorrowcorporation.com/humanresourcemachine is a puzzle game based on a visual programming language. In this game, the player has to automate a task by programming an office worker, progressing through increasingly difficult puzzles (Johnson & all, 2016). The player must create a sequence of moves, by dragging and dropping instructions, with new commands being gradually introduced to accomplish more complex operations. Through this visual programming language it teaches fundamental programming concepts through algorithm comparison.



You got a new instruction! Jumps only if the thing you're holding is negative. (Less than zero.) Otherwise continues to the next line.

```
inbox
copyto 0
inbox
copyto 1
sub 0
jump if negative
copyfrom 1
jump
copyfrom 0
outbox
jump
```

Fig. 4. 6 Human Resource Machine Gameplay

LightBot

LightBot https://lightbot.com/ is a puzzle game with similar mechanics to Human Resource Machine, requiring logic thinking to progress through the levels. The player must guide a robot to light up tiles and solve 40 levels. The commands used in Lightbot appear as icons.

Although there is no explicit learning of a programming language, players "… develop an understanding of sequencing and implementation of algorithms" (Miljanovic & Bradbury, A Review of Serious Games for Programming, 2018)(p.6) through a focus on problem solving skills. Nevertheless, through the game students learn different fundamental programming concepts, such as conditionals, iteratives, recursion and debugging strategies. As the space for the tiles is limited, the learner must also consider code efficiency.

A study conducted by Mathrani et al (Mathrani, Christian, & Ponder-Sutton, 2016) used Lightbot with students of the secondary education level and found that students enjoyed playing the games and that this was an effective way of learning programming constructs like functions, procedures, conditionals and recursions. Most of the students involved also agreed that the approach was an effective way for them to understand concepts that would otherwise be more difficult to grasp.

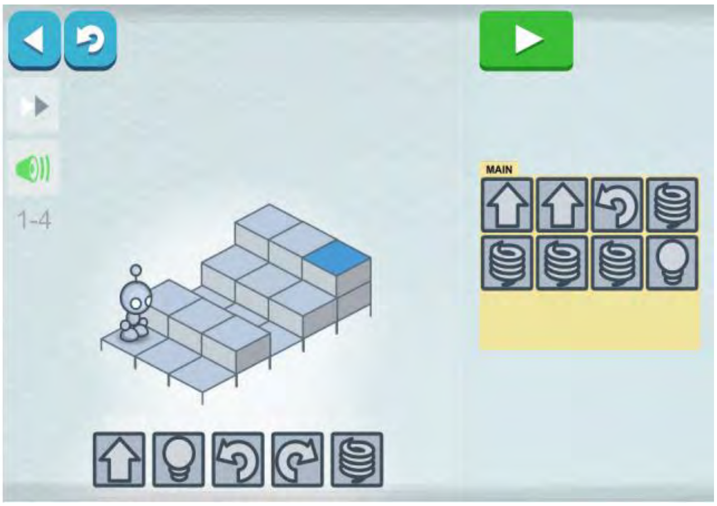
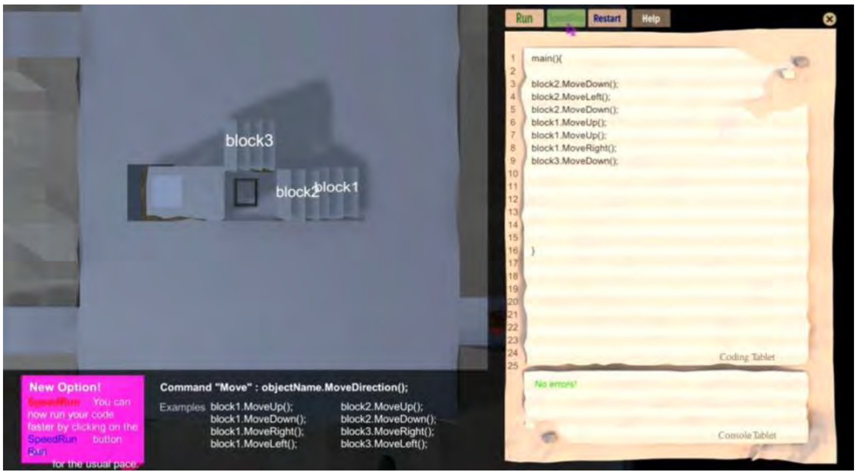


Fig. 4. 7 Lightbot Interface

May's Journey

May's Journey is the only game from this list that is directly targeted towards middle school girls. It is a 3D puzzle game in which the players solve a maze through the game's custom programming language, which is inspired by Java. The game was designed to attract girls to Computer Science by teaching the basics of programming. The developer employs pseudo-code to facilitate the transition between visual programming and real programming



```
main(){

block2.MoveDown();
block2.MoveLeft();
block2.MoveDown();
block1.MoveUp();
block1.MoveUp();
block1.MoveRight();
block3.MoveDown();






}
```

No errors!

New Option! SpeedRun You can now run your code faster by clicking on the SpeedRun button Run for the usual pace.

Command "Move" : objectName.MoveDirection();

```
Examples block1.MoveUp();       block2.MoveUp();
         block1.MoveDown();     block2.MoveDown();
         block1.MoveRight();    block3.MoveRight();
         block1.MoveLeft();     block3.MoveLeft();
```

Fig. 4. 8 May's Journey Gameplay

languages. The teaching content includes basic instructions and sequence logic, loops, variables, if statements, comparators and Boolean logic, operations on integers and operations on strings (Jemmali , 2016).

There are two phases in the game, an exploration phase with the mechanics of a typical game and a coding phase. In the coding phase, the interface is split so that the player can type the code while getting visual feedback of the program run. Hints for the code are also provided during the exploration phase. In the story, the hero is a girl who lives in a world that is falling apart and is separated from her friend, needing to fix the game world and solve mysteries. Each part of the mystery is revealed gradually, motivating the players to play more. As pre-teens are the main target group, they made the main character look like a middle school girl, so that the player can project herself unto the girl. The design also had in mind girls preferences towards design (high lightness, warm colour scheme and less aggressive illustration. The test group seemed also to appreciate the storyline.

No Bug's Snack Bar

No Bug's Snack Bar is a research serious game with a drag and drop block-based approach, focusing on problem solving. The learning outcomes relate to variable manipulation, sequence of actions, conditionals and iteratives and debugging strategies (Vahldick, 2017).

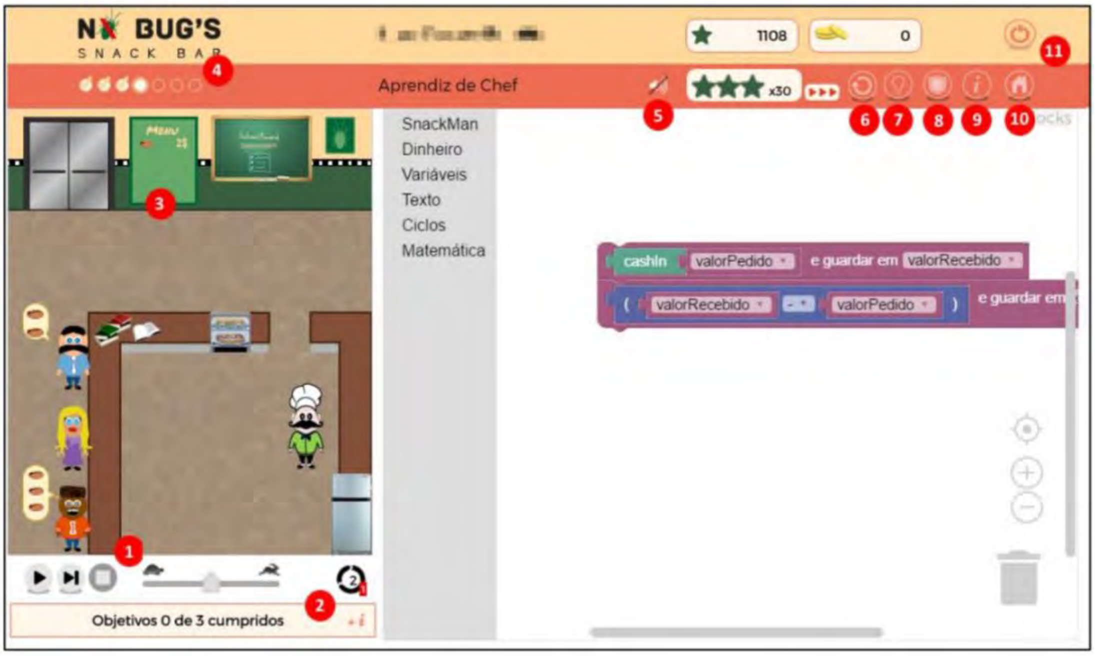
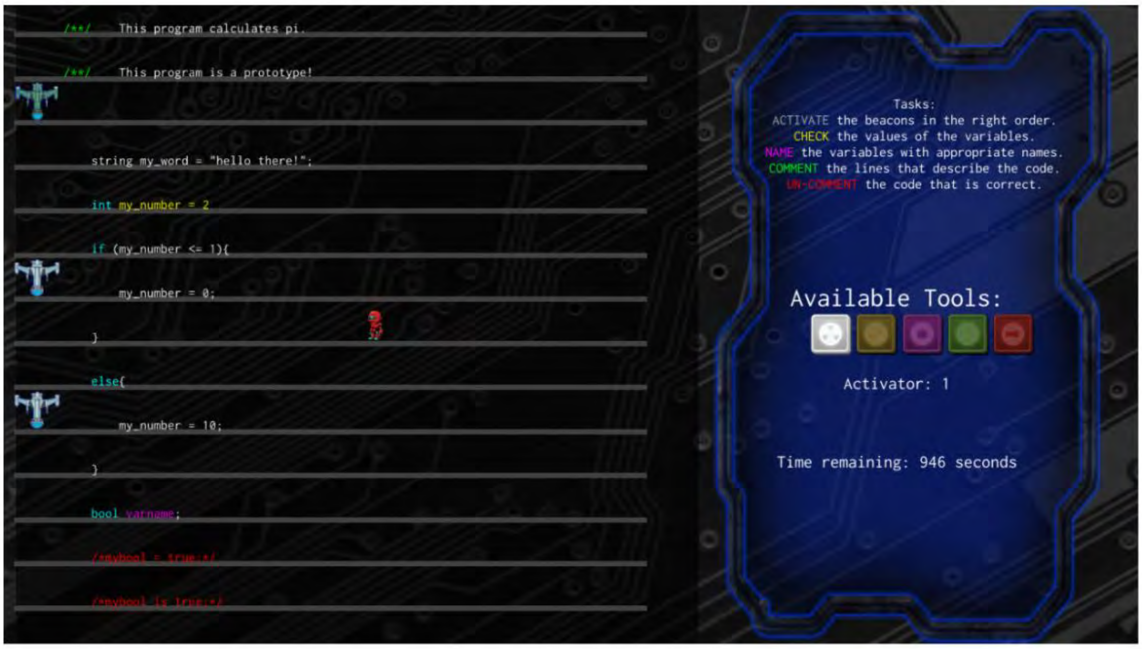

NO BUG'S
SNACK BAR
1108
0
Aprendiz de Chef
x30
SnackMan
Dinheiro
Variáveis
Texto
Ciclos
Matemática
cashIn valorPedido e guardar em valorRecebido
(valorRecebido - valorPedido) e guardar em
Objetivos 0 de 3 cumpridos
1
2
3
4
5
6
7
8
9
10
11

Fig. 4. 9 No Bug's Snack Bar Gameplay

In the game, the main character works at a Snack Bar and has to clear different missions using code. One innovation integrated in this game is a tool for teachers to monitor how students are progressing. There is also a gamification approach, with the student gaining points as they progress through the games and find ways to better their solutions to the problems presented. The game incorporates an assistant with hints and an administrative system for the teacher to see who are the learners that need help so that he/she can send them personalized hints. The presence of a teacher was, therefore, seen as fundamental.

Robot ON!

Robot ON! is another research puzzle-type game aimed at undergraduate students who are learning C++. In this game, the player is a scientist that must activate a 'Mech Suit' through a series of tasks. Once the player finishes a level, he activates a new robot system. Teachers can create their own levels using other programming languages. The player has different tools, which are colour coded, with the colour on the code corresponding to the different tools. The player is introduced to different tools one at a time, accompanied by tutorials (Miljanovic & Bradbury, 2016).



```
/**/    This program calculates pi.
/**/    This program is a prototype!

string my_word = "hello there!";
int my_number = 2
if (my_number <= 1){
    my_number = 0;
}
else{
    my_number = 10;
}
bool varname;
/*mybool = true;*/
/*mybool is true;*/
```

Tasks:
ACTIVATE the beacons in the right order.
CHECK the values of the variables.
NAME the variables with appropriate names.
COMMENT the lines that describe the code.
UN-COMMENT the code that is correct.

Activator: 1

Time remaining: 946 seconds

Fig. 4. 10 Robot ON! Gameplay

Instead of focusing on writing code, this game focus on programming comprehension, in order to teach debugging skills and understand code written by others. As the game progresses, the player is given the following tools: 1) activator tool (to learn control flow); 2) commenter and un-commenter tools (to learn code behaviour); 3) namer tool (to learn variable purpose); and 4) checker tool (to learn data flow).

Educational Pacman Game

Educational Pacman Game is a research serious game designed to teach search algorithms, allowing students to see how different search algorithms behave through an annotated graphical depiction of them (Grivokostopoulou, Perikos, & Hatzilygeroudis, 2016).

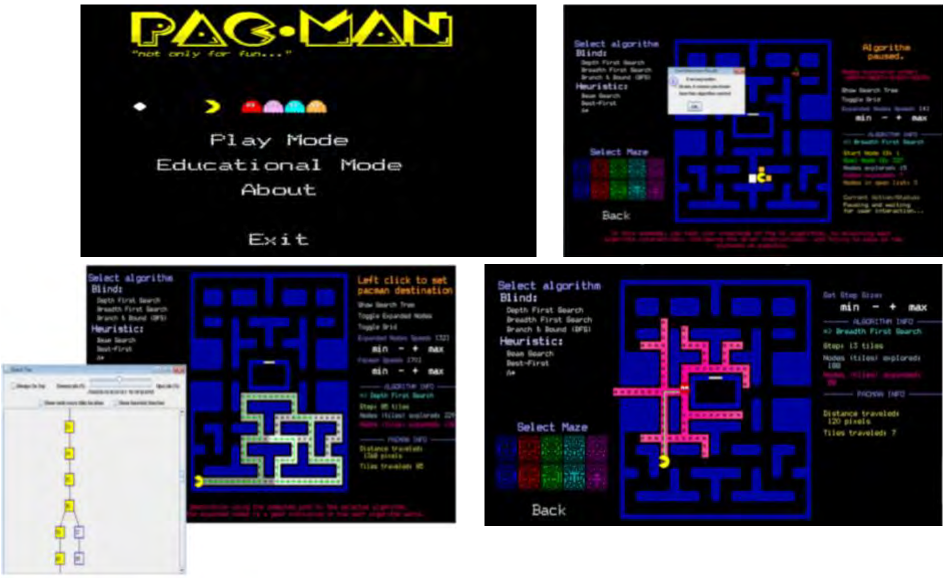


Fig. 4. 11 Educational Pacman Game

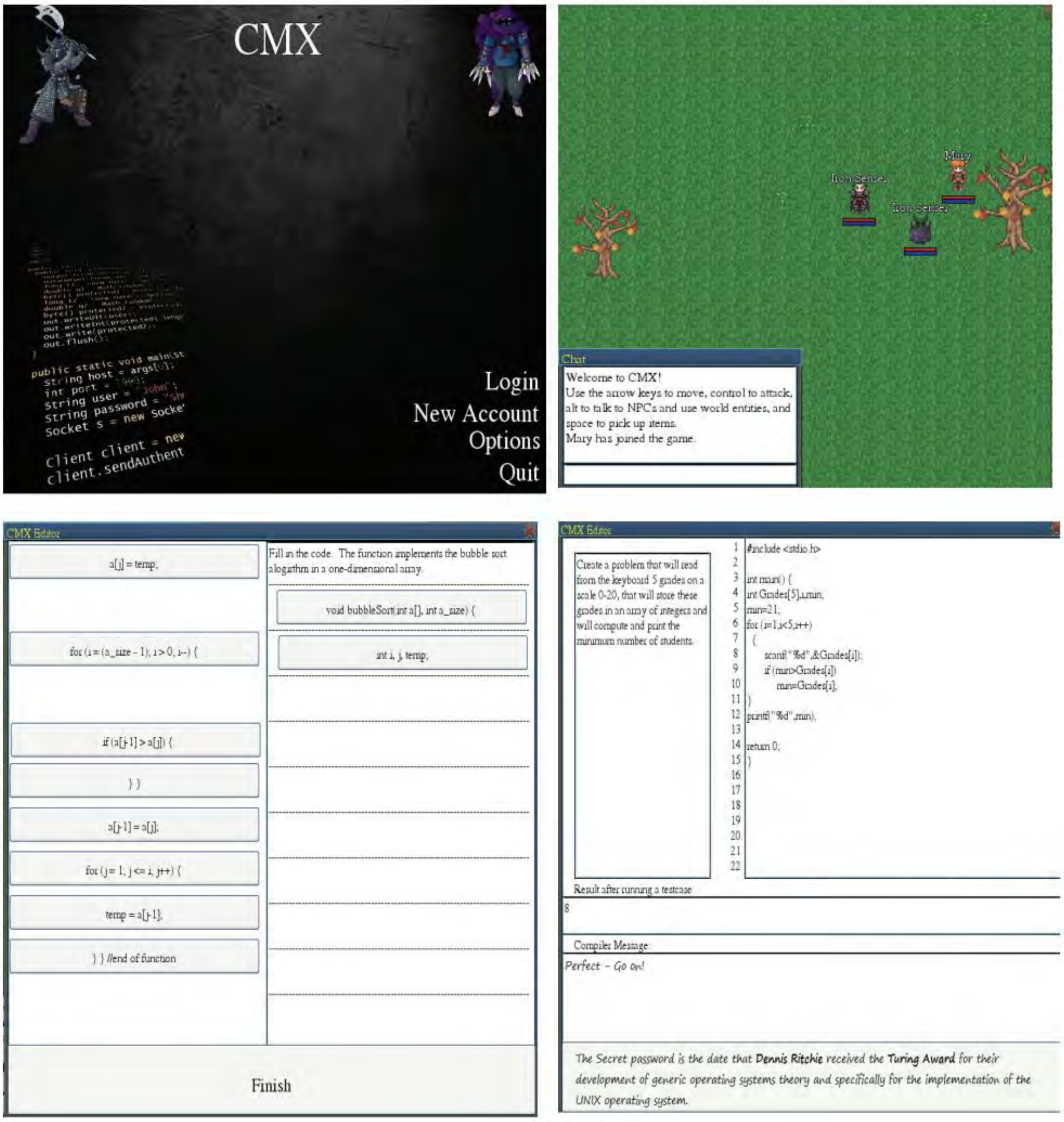
The game has two modes:

1) <u>Educational Mode: </u>the student can read a textual description and the graphical flowchart and pseudocode of an algorithm of his choice. The game also allows for the student to learn the algorithm via visualizations on the different Pacman Maze.

2) <u>Playing Mode</u>: the student has to solve maze levels within different conditions and with a time constraint. These levels are made so that the student has to apply a specific search algorithm to move the Pacman in the maze — the student is asked to, from a random position, reach to a cheery or power-up by moving the character on the specific algorithm.

CMX

CMX is a Massive Multiplayer Online Role Playing Game (MMORPG) for learning programming. In the game there are two teams — crackers and hackers — who compete against each other to find the passwords on a global toxic waste factory. They are assisted by Senseis, that assist them in learning the programming language C. The game includes three levels of Senseis reachable by unlocking a password in the previous level. The students seemed to increase their comprehension level of C — they could not only answer theory questions, but also lay executable programs by dragging and dropping tools, as well as writing programs in C (Malliarakis, Satratzemi, & Xinogalos, 2014).



Welcome to CMX!
Use the arrow keys to move, control to attack, alt to talk to NPCs and use world entities, and space to pick up items.
Mary has joined the game.

Fill in the code. The function implements the bubble sort alogirthm in a one-dimensional array.

```
a[j] = temp;
for (i = (a_size - 1); i > 0; i--) {
if (a[j-1] > a[j]) {
} }
a[j-1] = a[j];
for (j = 1; j <= i; j++) {
temp = a[j-1];
} } //end of function
```

```
void bubbleSort(int a[], int a_size) {
int i, j, temp;
```

Create a problem that will read from the keyboard 5 grades on a scale 0-20, that will store these grades in an array of integers and will compute and print the minumum number of students.

```
#include <stdio.h>

int main() {
int Grades[5],i,min;
min=21;
for (i=1;i<5;i++)
  {
     scanf("%d",&Grades[i]);
     if (min>Grades[i])
        min=Grades[i];
}
printf("%d",min);

return 0;
}
```

8

Perfect - Go on!

The Secret password is the date that **Dennis Ritchie** received the **Turing Award** for their development of generic operating systems theory and specifically for the implementation of the UNIX operating system.

Fig. 4. 12 CMX Game Environment

Table 4.2. Comparison of game environment for teaching programming

GAME	PLATFORM	TARGET GROUP	LANGUAGE(S)	LEARNING UNITS	TYPE OF GAME
Code Combat	Web-based	9+	JavaScript, Python	syntax, methods, parameters, strings, loops and variables, If/else, Boolean logic, relational operators, functions, object properties, event handling, input handling, Arithmetic, counters, while-loops, break, continue, arrays, string comparison, finding min/max, Object literals, remote method, invocation, for-loops, complex functions, drawing, modulo	Commercial Game (*Freemium*)
Human Resource Machine	Windows, Mac OS X, Linux, iOS, Android	9+	Visual Programming Language-based	Input & output and conditionals & iteratives algorithm comparison	Commercial Game (*Paid*) Puzzle Game
Lightbot	iOS, Android, Windows, Mac OS X	9+	Visual Programming Language-based	Conditionals and Iteratives and Recursion Debugging Strategies	Commercial Game *Paid* Puzzle Game
May's Journey	Windows	12-18	Custom Programming Language (inspired from an Object-Oriented Language like Java)	Basic instructions and sequence logic, loops, variables, if statements, comparators and Boolean logic, operations on integers, operations on strings	Research Game Puzzle Game
No Bug's Snack Bar	Web-based	18+	Visual Programming Language	Variable Manipulation, Sequence of Actions, Conditional & Iteratives, Debugging Strategies	Research Game
Robot ON!	Windows	18+	C++	Syntax & Semantics, Variable & Primitive Data Types, Expressions & Assignments, Conditionals & Iteratives, Program Comprehension	Free/Research Game
Educational 'Pacman' Game	Windows	18+	Game for learning AI search Algorithm	Algorithms (Basic textual description of Algorithms and corresponding graphical flowchart along with Pseudocode)	Research Game
CMX	Windows	18+	C	Theory on arrays, outputs of variables after an array's program's execution, syntax of array programs, entry of variable values in an array, calculation of the array's sum, calculation of the maximum value of the array,	Research Game MMORPG

5. CODING4GIRLS GAME PLATFORM FOR TEACHERS AND STUDENTS

5.1. Coding4Girls platform and design thinking approach

A perfect combination of ICT and the serious game is shown in the project result of *O2-Promoting the Development of Programming Skills among Girls through Serious Games* aiming at building programming skills among young people, mainly girls, in basic and secondary education by using a software developed in the framework of the project.

This software was designed on overcoming the existing gap between male and female participation in computer science education and careers by introducing methodological learning interventions that make computer science attractive to everyone. This software and the learning activities to be carried out were developed with the aim to incentivize the girls' involvement in computer science. They were designed following the design thinking approach, meant as a creative process that helps students to design meaningful solutions collaboratively together with their peers.

This design process is very effective thanks to its structured framework for identifying challenges, gathering information, generating potential solutions, refining ideas, and testing solutions.

The process is recursive by nature and demands interaction. Each stage in the process requires is revisiting and invoking throughout a learning experience to encourage experimentation, solution feasibility, and reflection. (Luka, 2014)

Designed thinking activities enable highly collaborative experience in and outside the classroom. Students are directly engaged in information gathering, knowledge generation, communication and presentation.

In the Coding4Girls project, the designed thinking approach exploits the main advantages of game-based learning to encourage and motivate students in their learning process. It uses some important game elements (e.g. points and challenges) and scenarios which make a certain activity more fun by increasing the degree of involvement, motivation and results achieved.

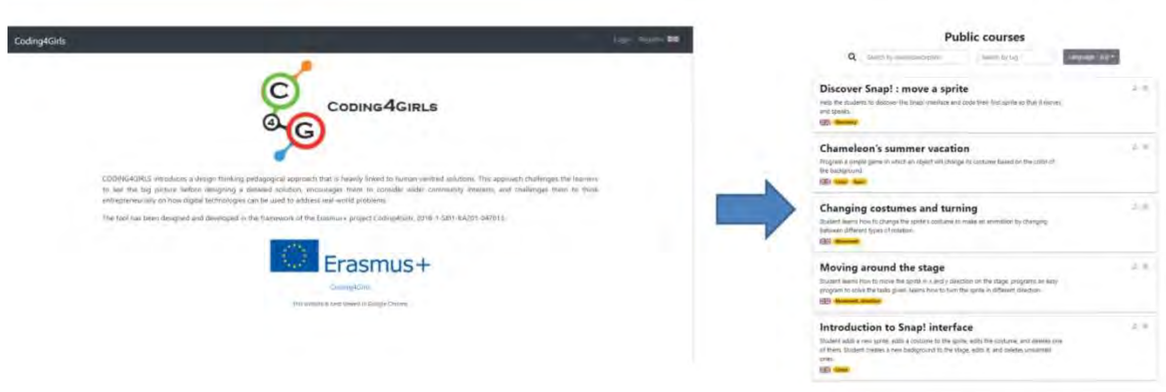
In particular, the use of the challenges while helping students to see the big picture before designing a detailed solution encourages and challenges them to think entrepreneurially about digital technologies and how they can be used to address real-world problems.

The software, developed in the framework of the project, consists of two different interconnected parts: the Teachers' Platform and the Student Game Environment.

5.2 The Teacher's Platform

The Teachers' Platform is a web-based platform where teachers can design and prepare specific courses to develop coding skills of their students by using Snap! Canvas.

Inside the platform, every teacher has at their disposal both a private and public area. In the first one, they can prepare their courses based on their students' needs. The second one is a repository of all created courses by other fellow teachers. The aim is to share, use or get inspired by the learning activities already prepared by other colleagues (Fig. 5. 1). Of course, every teacher can personalize all the public courses according to the specificity of their classes.



CODING4GIRLS introduces a design thinking pedagogical approach that is heavily linked to human centred solutions. This approach challenges the learners to see the big picture before designing a detailed solution, encourages them to consider wider community interests, and challenges them to think entrepreneurially on how digital technologies can be used to address real-world problems.

The tool has been designed and developed in the framework of the Erasmus+ project Coding4Girls, 2018-1-SI01-KA201-047013.

Coding4Girls

This website is best viewed in Google Chrome.

Public courses

Discover Snap! : move a sprite
Help the students to discover the Snap! interface and code their first sprite so that it moves and speaks.

Chameleon's summer vacation
Program a simple game in which an object will change its costume based on the color of the background.

Changing costumes and turning
Student learns how to change the sprite's costume to make an animation by changing between different types of rotation.

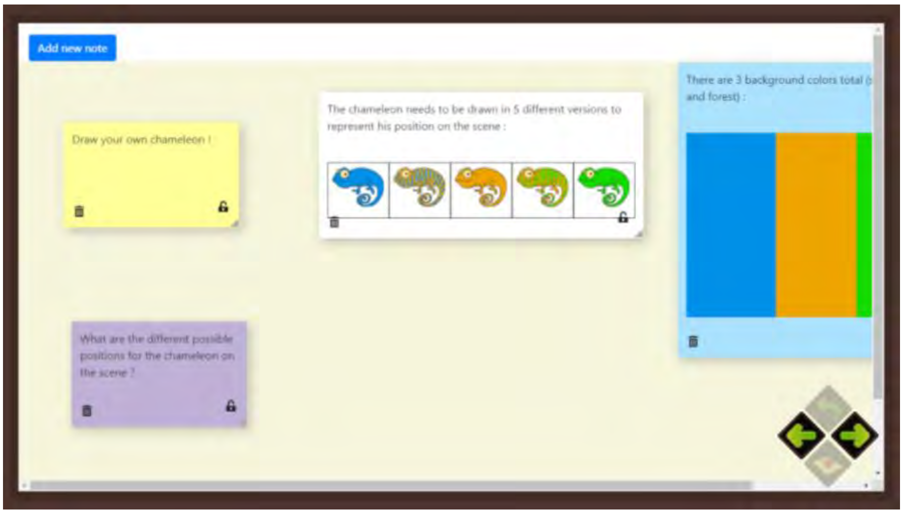
Moving around the stage
Student learns how to move the sprite in x and y direction on the stage, programs an easy program to solve the tasks given, learns how to turn the sprite in different direction.

Introduction to Snap! interface
Student adds a new sprite, adds a costume to the sprite, edits the costume, and deletes one of them. Student creates a new background to the stage, edits it, and deletes unwanted ones.

Fig. 5.1 Teacher Platform and some coding courses available in it.

These courses are constituted as a grouping space for thematically related activities, all connected to an overarching issue. The problem is presented at the very beginning of the course to the students who can brainstorm all together to collaboratively elaborate tentative solutions. This is a very important phase of the learning process where students can generate new idea and be creatively immersed in a problem-solving process.

The brainstorming canvas can be prepared by teachers in the Teachers' platform by using some key questions or some remarks and reflections to be used during the students' discussion. The software offers the possibility to prepare post-it on a virtual board by using text, images or video as in the following Figure 5.2:



Draw your own chameleon !

The chameleon needs to be drawn in 5 different versions to represent his position on the scene :

There are 3 background colors total (
and forest) :

What are the different possible positions for the chameleon on the scene ?

Fig. 5.2. Brainstorm zone in the Student Game Environment for students' contribution and discussion.

All the post-its have a padlock icon which can be open when the post-it is editable or locked when the post-it cannot be edited. Only teachers have the possibility of locking and unlocking post-its, students can only know if a post is locked or not without being able to act on it.

After the brainstorming phase, the students are given in a step-by-step fashion; specific activities (presented in consecutive order) designed to present the tools necessary to solve the overarching problem. These coding activities will be presented by using Snap! canvas, through both the half-baked tasks to challenge the students in the problem solution and the final solution presentation.

In Figure 5.3 the structure of each C4G course published in the Teachers' platform is shown.

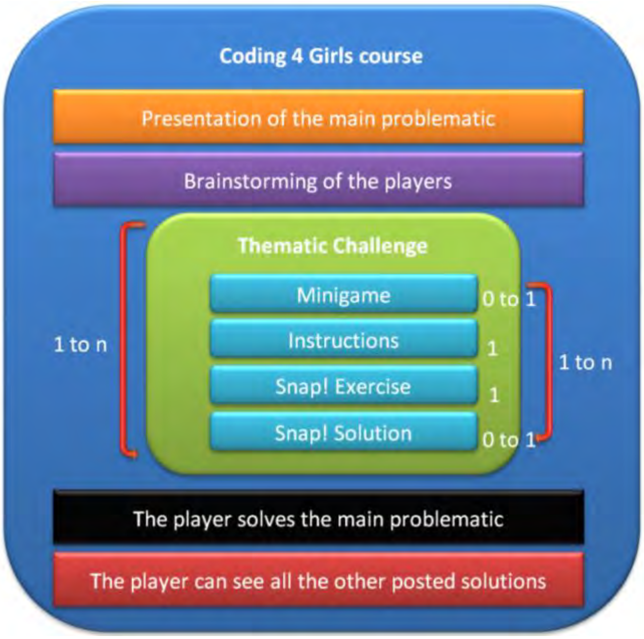
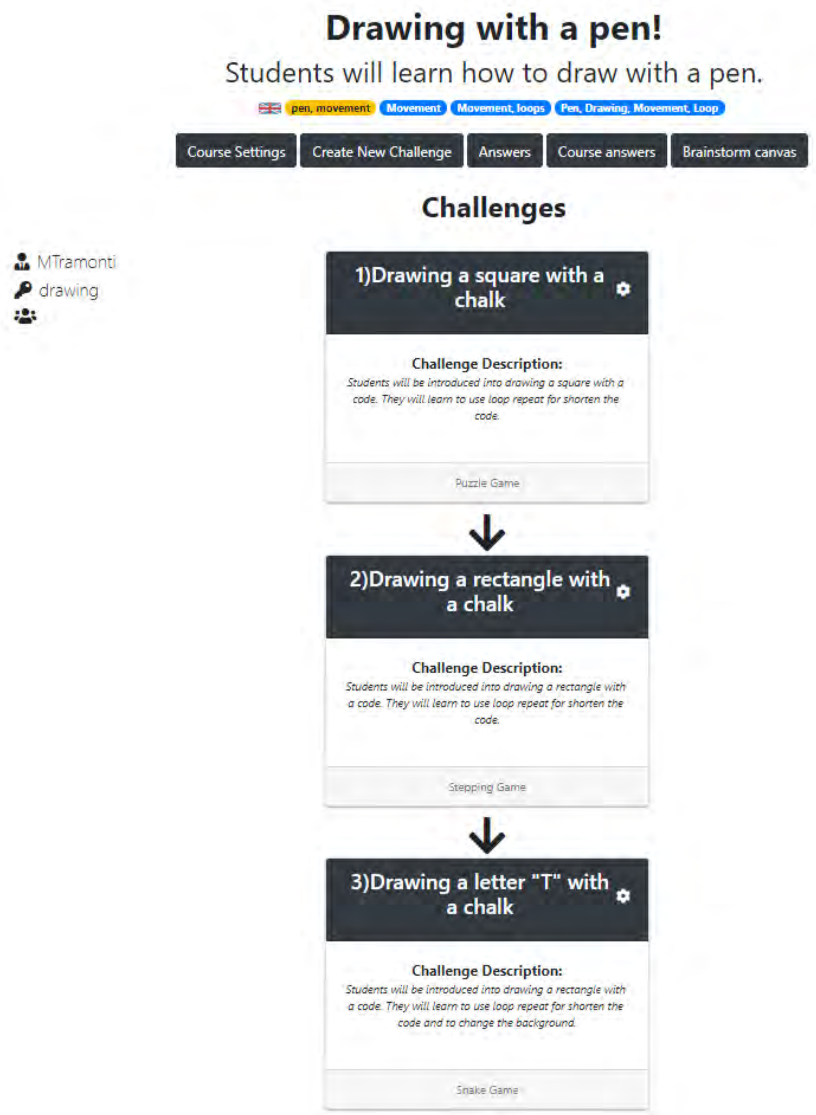

Coding 4 Girls course
Presentation of the main problematic
Brainstorming of the players
Thematic Challenge
Minigame
0 to 1
Instructions
1
Snap! Exercise
1
Snap! Solution
0 to 1
1 to n
1 to n
The player solves the main problematic
The player can see all the other posted solutions

Fig. 5.3 Structure of a C4G course

Each grouped activities in the course are called challenges, which are presented to students in a specific order set, previously, by the teacher (Figure 5.4). For example, if a teacher wants to create a course on basic programming knowledge, the first activity could concern the concept of booleans, the second conditional structures and the last one - the loops. This will allow teachers to prepare customized learning steps for their students in the Teachers' Platform and the students to be lead gradually to the final learning objective in the Students' Game Environment.

Of course, students need to play and solve all the challenges in the order established by the teachers before achieving the final solution.



Drawing with a pen!

Students will learn how to draw with a pen.

Challenges

1)Drawing a square with a chalk

Challenge Description:
Students will be introduced into drawing a square with a code. They will learn to use loop repeat for shorten the code.

Puzzle Game

2)Drawing a rectangle with a chalk

Challenge Description:
Students will be introduced into drawing a rectangle with a code. They will learn to use loop repeat for shorten the code.

Stepping Game

3)Drawing a letter "T" with a chalk

Challenge Description:
Students will be introduced into drawing a rectangle with a code. They will learn to use loop repeat for shorten the code and to change the background.

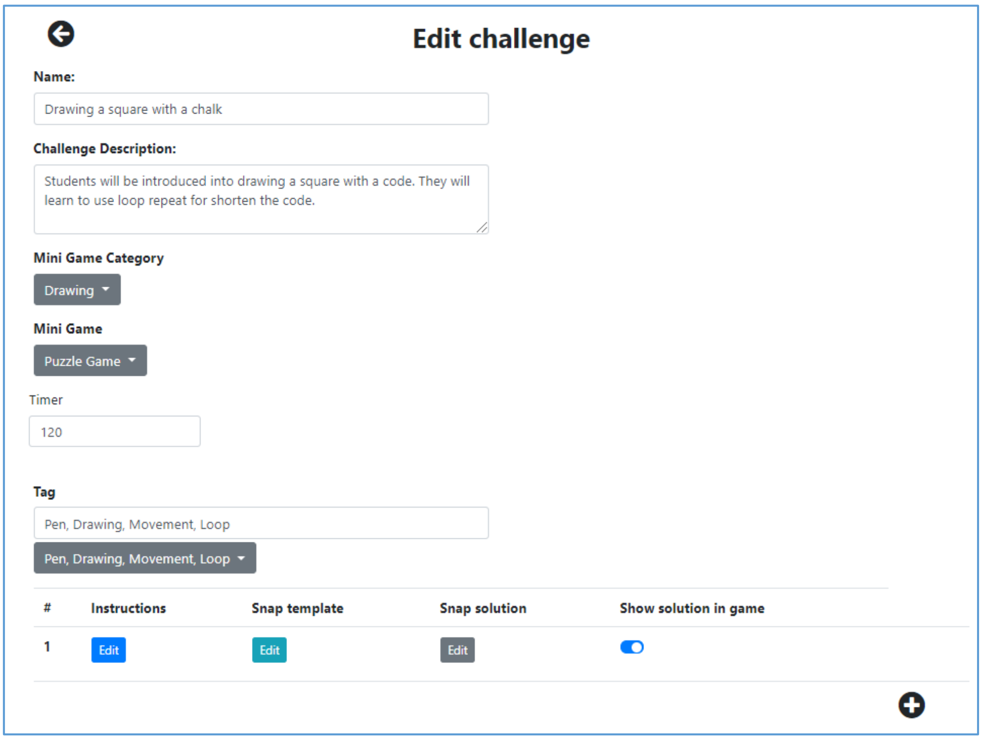
Snake Game

Fig. 5.4 An example of challenges in a coding course in the Teacher Platform.

Each challenge might be associated with a mini-game developed by the project team. The aim is to help the students understand better what could be the final result of a specific coding property studied.

For example, if a challenge is concerned with the study of "loop" property, Match-3 is one of the mini-games developed that could be associated with it.

Therefore, each challenge might have, besides the Snap! canvas to solve the task, a mini-game that the student will need to play following a page with instructions defined by the teacher during their preparation in the Teachers' Platform (Figure 5.5).



Edit challenge

Name: Drawing a square with a chalk

Challenge Description: Students will be introduced into drawing a square with a code. They will learn to use loop repeat for shorten the code.

Mini Game Category: Drawing

Mini Game: Puzzle Game

Timer: 120

Tag: Pen, Drawing, Movement, Loop

Pen, Drawing, Movement, Loop

#	Instructions	Snap template	Snap solution	Show solution in game
1	Edit	Edit	Edit	●

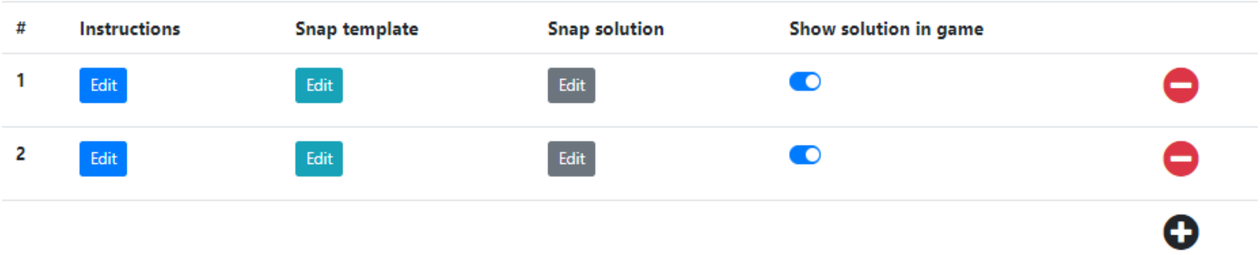
Fig. 5.5 Inside view of a challenge in the Teachers' Platform

All the games offered (currently 11) to the students are related to and simplify the actual programming concepts upon which the C4G courses and scenarios have been designed. However, it is not mandatory to include a mini-game in the challenge. This is only a teacher's choice.

Every challenge has constituted of three parts:

1. Instructions, where the teacher can give some clues or explanations about the task to be carried out by their students.
2. Snap template, where students will try to solve the assigned task by using block coding of the open-source Snap! in the Students' Game Environment
3. Snap solution where the teachers can decide to show or not the final solution of the task.

If the challenge is complex, in terms of programming knowledge, it can be structured into more than one task (Figure 5.6). In this manner, teachers will be able to lead, step by step, their students, through different phases, to solve a complex activity breaking down into smaller parts.

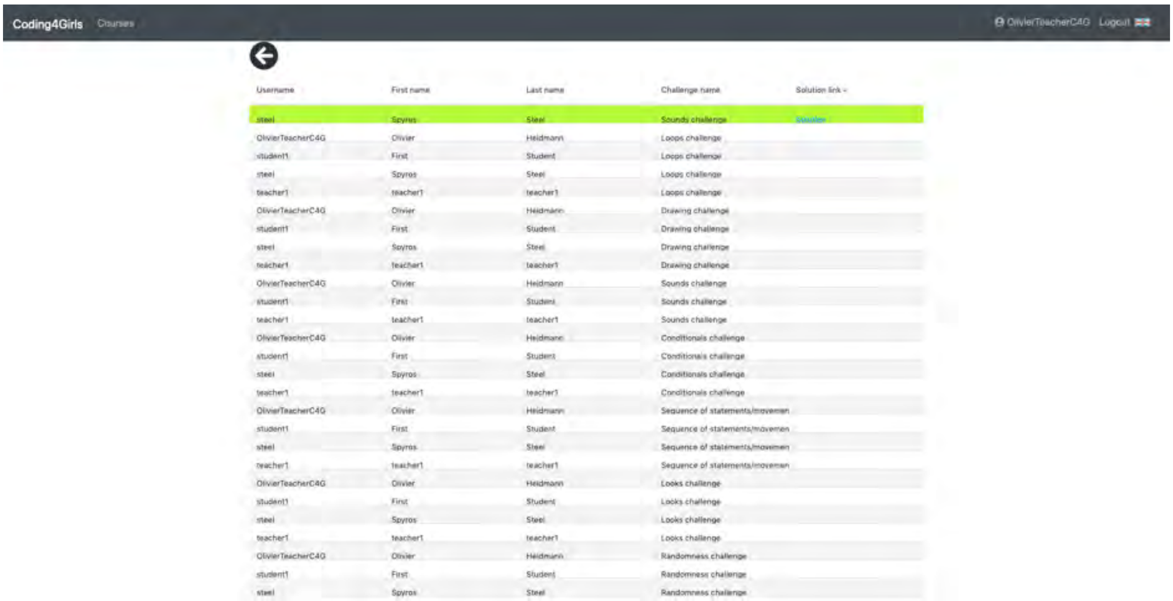


#	Instructions	Snap template	Snap solution	Show solution in game	
1	Edit	Edit	Edit	●	⊖
2	Edit	Edit	Edit	●	⊖
					⊕

Fig. 5.6 Breaking down into smaller tasks of a complex challenge

Moreover, the teachers can collect all the Snap! answers of their students after their submission in each challenge, displayed in a table in the Teachers' Platform. The table contains the list of the users and its details for every challenge that exists in the course. Every row is reported the username, first name, last name, challenge name or solved challenges.

If there isn't a submitted solution the "Solution link" column will be empty. Otherwise, the row will be highlighted and the word "Solution" will appear on the last column and, clicking on it, the submission of the user will be shown (Figure 5.7).



Username	First name	Last name	Challenge name	Solution link ▾
steel	Spyros	Steel	Sounds challenge	Solution
OlivierTeacherC4G	Olivier	Heidmann	Loops challenge	
student1	First	Student	Loops challenge	
steel	Spyros	Steel	Loops challenge	
teacher1	teacher1	teacher1	Loops challenge	
OlivierTeacherC4G	Olivier	Heidmann	Drawing challenge	
student1	First	Student	Drawing challenge	
steel	Spyros	Steel	Drawing challenge	
teacher1	teacher1	teacher1	Drawing challenge	
OlivierTeacherC4G	Olivier	Heidmann	Sounds challenge	
student1	First	Student	Sounds challenge	
teacher1	teacher1	teacher1	Sounds challenge	
OlivierTeacherC4G	Olivier	Heidmann	Conditionals challenge	
student1	First	Student	Conditionals challenge	
steel	Spyros	Steel	Conditionals challenge	
teacher1	teacher1	teacher1	Conditionals challenge	
OlivierTeacherC4G	Olivier	Heidmann	Sequence of statements/movemen	
student1	First	Student	Sequence of statements/movemen	
steel	Spyros	Steel	Sequence of statements/movemen	
teacher1	teacher1	teacher1	Sequence of statements/movemen	
OlivierTeacherC4G	Olivier	Heidmann	Looks challenge	
student1	First	Student	Looks challenge	
steel	Spyros	Steel	Looks challenge	
teacher1	teacher1	teacher1	Looks challenge	
OlivierTeacherC4G	Olivier	Heidmann	Randomness challenge	
student1	First	Student	Randomness challenge	
steel	Spyros	Steel	Randomness challenge	

Fig. 5.7 The answers to the challenge by users with a solution

In this way, the teachers can always monitor and track the progress and results of their students.

5.3. The Students' Game Environment

The students will access these challenges through the Student Game Environment that is a Unity 3D videogame downloadable software where students can discover and complete the courses prepared by their teachers in a fun, engaging and playful manner.

The courses, using elements of the design thinking approach, present to the students an overarching issue to solve and present the tools to solve it in a step-by-step approach. They, as above mentioned, are prepared by teachers and published in the Teachers' Platform, but students can access them through the Students' Game Environment.

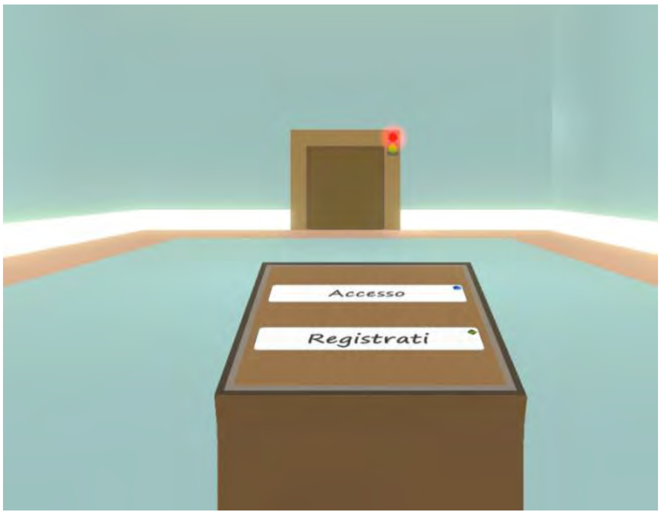


Fig. 5.8 The first room to access the Students' Game Environment

The students can join a course by entering the correct code in the terminal of the portal. The code is unique and should be provided by teachers who have already prepared the course on the Teachers' Platform. Once, the course was submitted, its name is visible on the top of the portal, which indicates also the currently selected course (Figure 5.9).

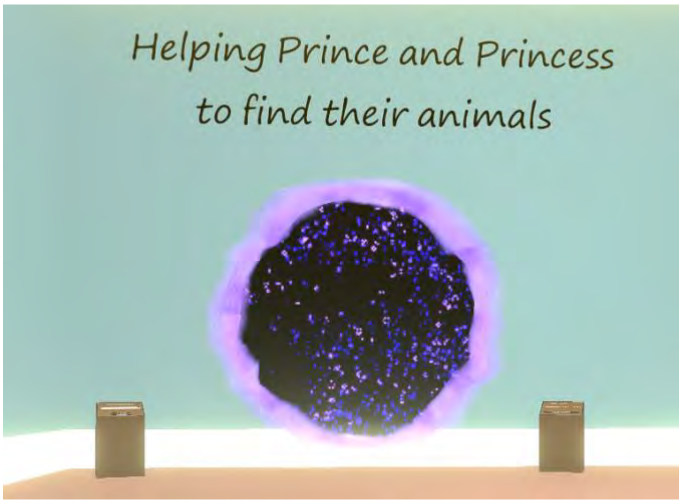


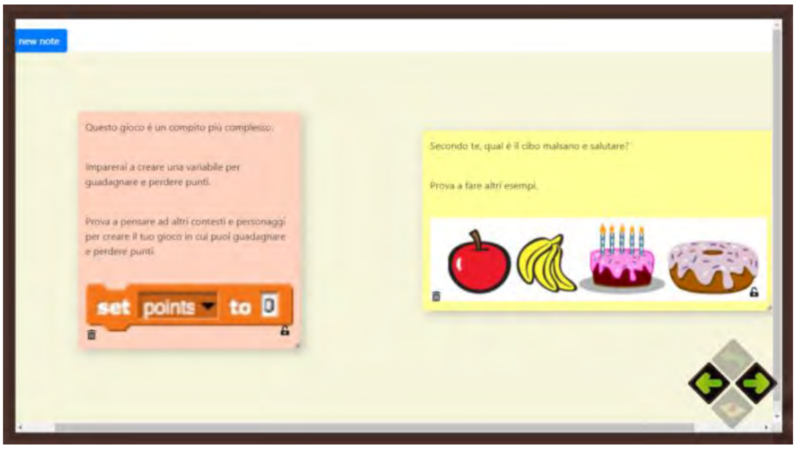
Fig. 5.9 Second room with two terminals: one to join, for the first time, a course and one to select a course among already registered

Students are encouraged to design and code games that address specific needs or issues (depending on teachers' choice). It is a "low entry high ceiling approach" allowing pupils to start with easy problems until engaging more challenging tasks. Teachers design and present to their students "half-baked" scenarios in which a solution is partially ready by challenging them to complete the task. Therefore, teachers can prepare small and manageable modules by using Coding4Girls (C4G) software suitable for their students' needs and knowledge.

Following the Design Thinking approach, the project team has prepared some courses and learning scenarios designed to challenge students on solving a specific coding issue. Currently, the 21 learning scenarios, collected in the report "Collections game design-based learning sheets targeting teachers", have been re-adapted to the design thinking approach following the structure of the C4G software.

The scenarios have different levels from basic to advanced for more capable students and they are available in English, Slovenian, Italian, Croatian, Bulgarian and Greek.

The tasks can be solved individually and collaboratively by arousing group discussion in the classroom or, virtually, on the Students Game Environment. The software offers a brainstorming space where students can discuss and share their ideas by placing and managing multimedia post-it as shown in Figure 5.11. All the students involved in the course can write their contribution which is visible to everyone enrolled in the specific course.



Questo gioco è un compito più complesso.

Imparerai a creare una variabile per guadagnare e perdere punti.

Prova a pensare ad altri contesti e personaggi per creare il tuo gioco in cui puoi guadagnare e perdere punti.

Secondo te, qual è il cibo malsano e salutare?

Prova a fare altri esempi.

Fig. 5.11: Brainstorming area in the Learning Scenario *Buying food for a picnic* in the Students' Game Environment

The following figure shows the panel of challenges in the Student Game Environment of the course created by the teacher in the Teacher Platform. The number from 0 to 13 represents the challenges. In this panel, challenge number 0 represents the general instructions given at the beginning of the game and opening it will take students to the global problem instructions, the related Snap! template followed by the brainstorm canvas. While the other challenges, from the number 1 to 13, represent the tasks prepared by the teachers in their platform.

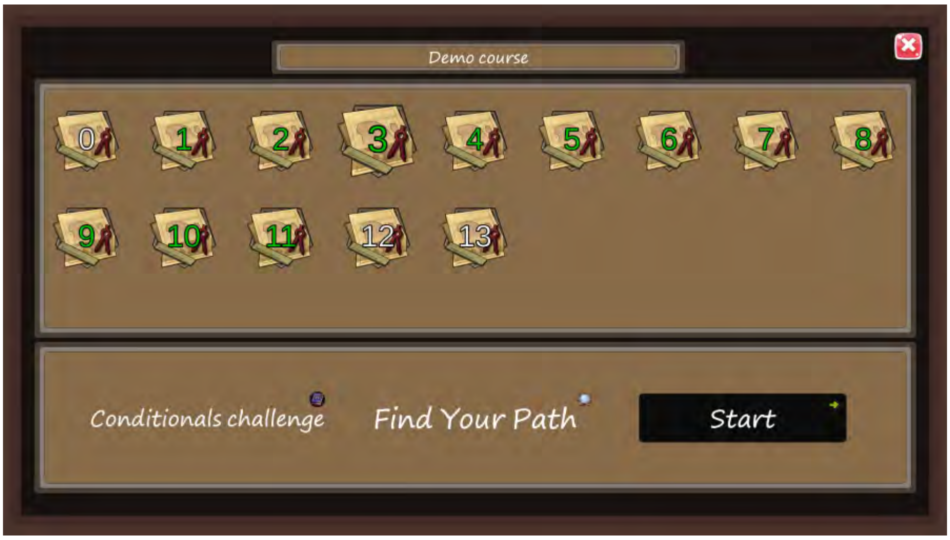


Fig. 5.12. The panel of challenges in the Student Game Environment

Through selecting one of them, the challenge details will appear on the bottom of the screen, with the challenge's name on the left, the challenge's associated mini-game in the middle and a button to start the challenge. Figure 5.12 shows the name of the challenge, corresponding to the topic to be studied, in this case, "Conditional challenge". Besides the challenge topic, the system shows the 3D mini-game associated with it, e.g. "Find your path".

The teachers can decide which mini-game (optional) their students will play by selecting one of the existing mini-games, such as Match3 game, Dice game, Inventory game, Find your path game, Stepping game, Sound game, Snake game, Puzzle game, Pattern matching game and a multiple-choice question quiz.

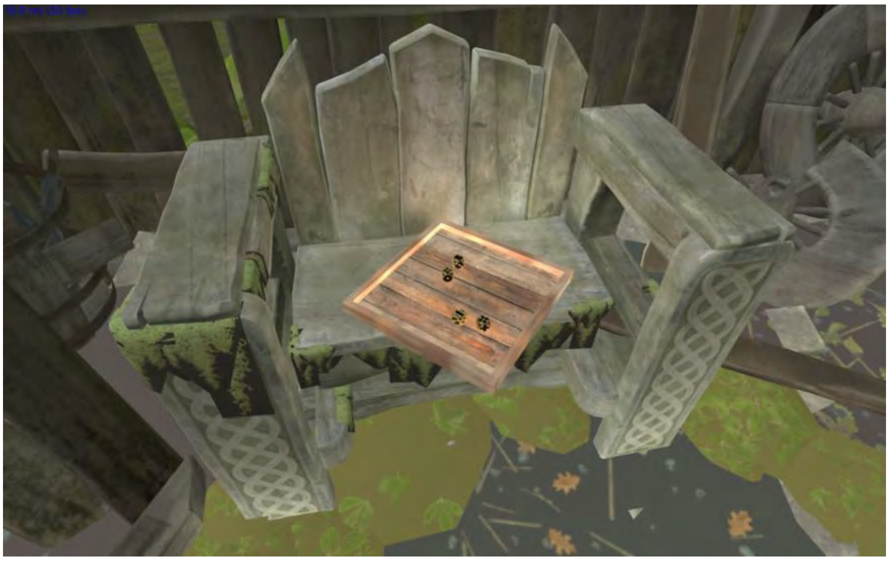
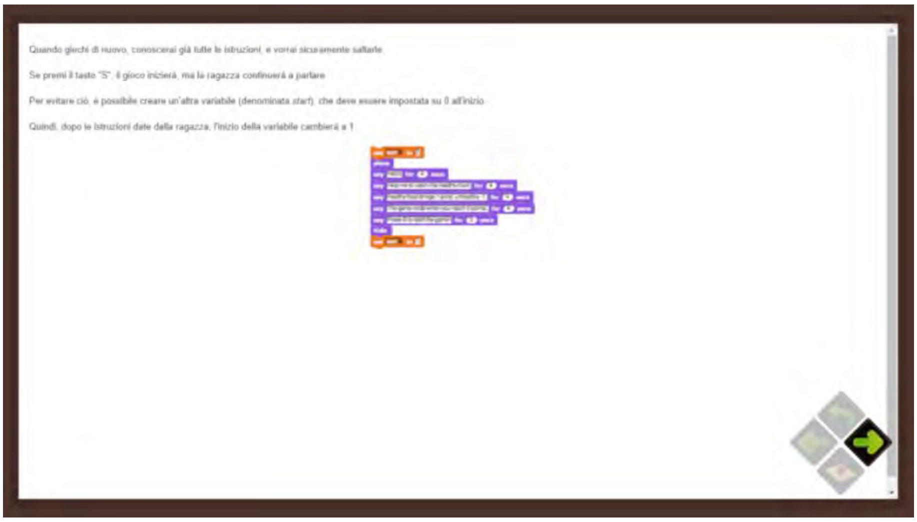


Fig. 5.13 An example of an available mini-game - Dice Game

Each challenge, in the Students' Game Environment, is structured as follows: one introductory mini-game illustrating the coding concept; one instructional page for the task to be fulfilled in Snap; two Snap! canvas - one to be used to solve the task and the other to display the activity solution.

The instructional page is in HTML (Figure 5.14), usually enriched by images or videos, to present the context and specific aim of the learning task to be fulfilled in Snap!



Quando giochi di nuovo, conoscerai già tutte le istruzioni, e vorrai sicuramente saltarle.

Se premi il tasto "S", il gioco inizierà, ma la ragazza continuerà a parlare.

Per evitare ciò, è possibile creare un'altra variabile (denominata *start*), che deve essere impostata su 0 all'inizio.

Quindi, dopo le istruzioni date dalla ragazza, l'inizio della variabile cambierà a 1.

Fig. 5.14 An example of the challenge's instructional page in the Students' Game Environment

This page will be followed by a Snap! canvas (Figure 14), based on a template provided by the teacher containing the coding activity or problem to be executed or solved by students.

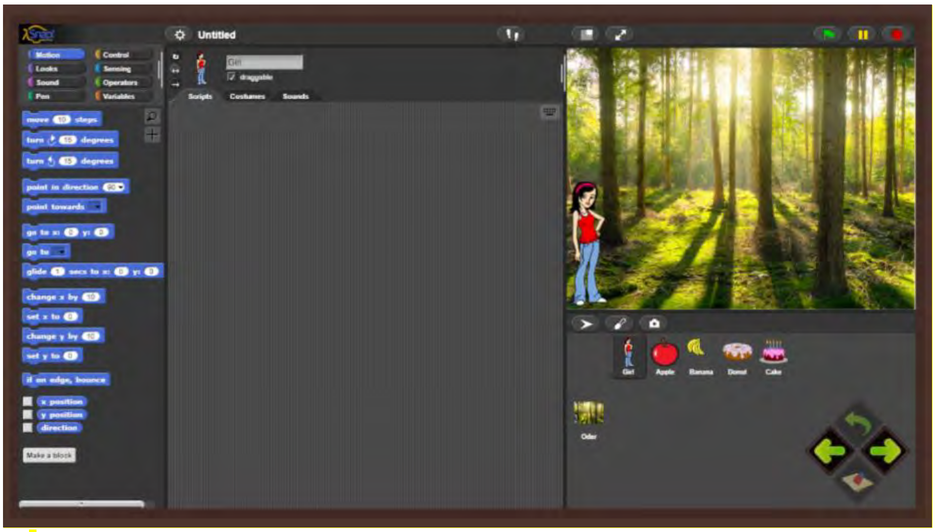
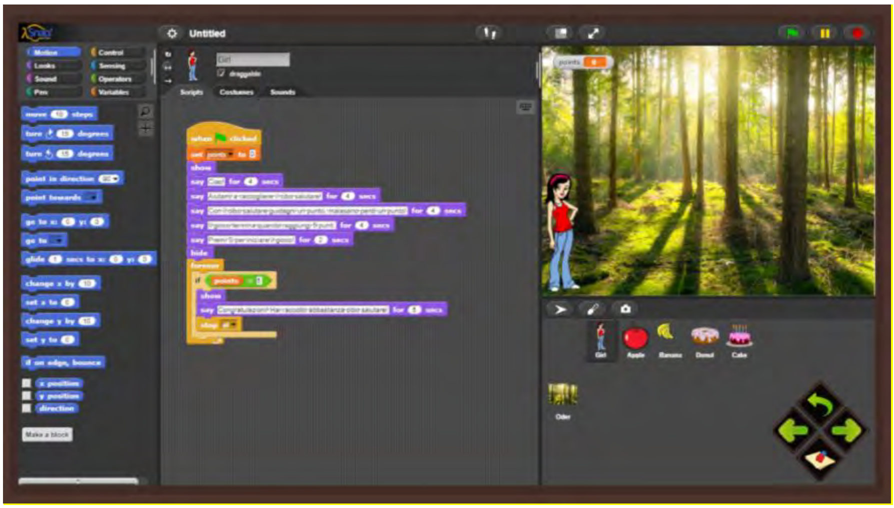


Figure 3.15: An example of the Snap!'s template provided by the teacher to solve the coding task in the Students' Game Environment

Another Snap! canvas will be provided to display the solution of the challenge proposed. However, only the teacher can decide if solution canvas is shown or not. It will depend on the teaching process management selected by the instructor.



```
when 🏁 clicked
set points to 0
show
say Ciao! for 4 secs
say Aiutami a raccogliere il cibo salutare! for 4 secs
say [illegible] for 4 secs
say [illegible] for 4 secs
say [illegible] for 2 secs
hide
forever
    if points = [illegible]
        show
        say Congratulazioni! Hai raccolto abbastanza cibo salutare! for [illegible] secs
        stop all
```

Fig. 5.16 An example of the Snap!’s canvas providing with the challenge’s solution by the teacher for their students in the Students’ Game Environment

Since every course assembles more challenges, these steps will be repeated as many time as the number of the total challenges identified by the teachers. This allows breaking one complex coding activity into simple elementary steps where the answer and/or the solution to the preceding activity becomes the template in which the next activity is to be executed.

In the end, the course is constituted of a certain number of challenges which present the programming teaching through an incremental scaffolding process.

Once the students have completed all the challenges of the course (Figure 7), they are lead back to the initial coding problem and will be asked to synthesize the new knowledge they just acquired. At the very end of the course, the players can see also all the solutions to the problem proposed by the other students by arousing confrontation moments.

This C4G approach and tools allow students to develop, improve and reinforce coding skills through personalized training activities combining fun and learning tasks.

6. EXAMPLES OF LESSONS (LEARNING SHEETS)

6.1. Learning scenarios sheets;

In the frame of the project Coding4Girls have been developed 22 learning scenarios sheets. They describe end-to-end blended learning activities that deploy the CODING4GIRLS serious game and design thinking approaches. The learning sheets are available in English as well as the national languages of project partners – Bulgarian, Croatian, Greek, Italian, Portuguese, Slovenian and Turkish. All of them are available at the web site of the project. (https://www.coding4girls.eu/results_03.php)

The prepared learning sheets present in concise manner information that will help instructors integrate the proposed serious games and design thinking learning methodologies into their teaching practices. They follow the CODING4GIRLS active, game-based learning design and include information for each learning activity to be developed for building programming skills for girls and boys. The following information are available:

- Overall educational objective of the corresponding learning activity
- Concepts covered by the learning activity
- Specific learning objectives
- Expected learning outcomes
- Step-by-step use of the CODING4GIRLS game design-based learning approach
- Assessment methods for evaluating the knowledge developed
- Questions for initiating discussion among learners in the context of class collaboration.

Teachers can use the scenarios and games in the proposed sequence or can select them freely according to their preferences and needs. Also teacher have to take care about adaptation some of scenarios according to knowledge of the students about some concepts learned in mathematics – e.g. coordinate system, coordinates, negative numbers, fractions etc..

Learning sheets cover both the generic functionality of the proposed serious game, including user interaction processes and feedback generation as well as descriptions of all learning activities that will be implemented in the proposed serious game.

Prepared learning sheets follow from basic with one programming concept to more advanced with multiple programming concepts. Following table represents the proposed order of activities.

Table 6.1. List of learning sheets developed in the project Coding4Girsl

BASIC LEARNING SCENARIOS		
1	**Introduction to Snap! interface** Getting familiar with Snap! visual programming environment	UL
2	**Time to bring your sprite to life** Finding programming blocks, connect them, move a sprite, make sprite say something	UL
3	**Moving around the stage** Making a meaningful sequence of blocks	UL
4	**Changing costumes and turning**	UL
5	**Sounds of the farm** Adding, importing, recording and playing sound	UL
6	**Chameleon’s summer vacation, simple version** Getting familiar with events, color sensing, Boolean values, checking and responding to two different game states	UL
7	**Helping Prince and Princess to find their animals** Using conditionals, drawing	UL
8	**Drawing with a chalk** Using loops, turning, changing background	UL
9	**Picking up trash and cleaning the park** Getting familiar with variables, duplicating sprites, blocks of code	UL
10	**Feeding the cats** Using variables (inside/outside the loop), loops, random numbers, string concatenation, operators, input	UL
11	**Guessing the number of cats in a shelter**	UL

	Using random values, variables input, conditionals, comparison operators, counter	
ADVANCED LEARNING SCENARIOS		
12	**Catching healthy food** Using variables, conditionals, loop, point in direction, random	UL
13	**Storytelling**	SWU
14	**Drawing**	UNIRI
15	**Catch the mouse** Using loops, conditionals, variables	UL
16	**Buying food for a picnic** Using variables, conditionals, operators	UL
17	**Operations**	SWU
18	**Recycling**	SWU
19.1	**Play a piano 1**	SWU
19.2	**Play a piano 2**	UNIRI
20	**Test**	SWU
21	**Simplified PACMAN game** Using event-based object movement, colour sensing, Boolean values, checking and responding to two different game states	UL

All 22 learning sheets are presented in Appendix 1. In Appendix 2 are given codes for all games presented in Public section of games in the Coding4Girls environment.

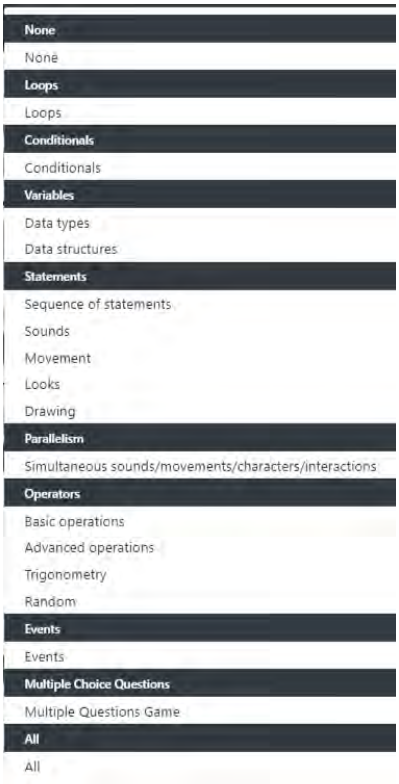
6.2. Examples with use of game environment developed in the frame of Coding4Girls project

The Students' Game Environment is a Unity-based serious videogame available on Windows and Mac OS X. When playing the game, the user first discovers an introductory scene presenting the required European disclaimers with the project and Erasmus+ logo being displayed for 4 seconds.

The player is arriving after that in a big empty room named the lobby, controlling an invisible avatar and viewing the game world through a first-person type of perspective. A console is placed in the middle of the room, allowing for the user to log on the C4G server by using their credentials or by creating a new account. Once logged in, a door slides open and the player can proceed to another room, where they can either subscribe to a new course or access a course already activated. Once the desired course is selected, the user will step through a floating portal to proceed with the game and discover the exact content of the selected course.

A course is formed by a collection of linked coding challenges using the Snap! Platform, each illustrated by a mini game. Ideally, each challenge will illustrate a facet or a step of the lesson to be learned in the course.

By default, C4G has defined some basic coding concepts which are each illustrated by a mini-game: Loops, Conditionals, Variables, Statements, Parallelism, Operators, Events. Besides, teachers can also decide to replace the mini-game by a multiple-choice question quiz or by nothing at all, leaving only Snap! exercises in the challenges.



None
None
Loops
Loops
Conditionals
Conditionals
Variables
Data types
Data structures
Statements
Sequence of statements
Sounds
Movement
Looks
Drawing
Parallelism
Simultaneous sounds/movements/characters/interactions
Operators
Basic operations
Advanced operations
Trigonometry
Random
Events
Events
Multiple Choice Questions
Multiple Questions Game
All
All

Fig. 6.1 The Mini-Game Category available in the Teachers' Platform

Each challenge includes coding tasks to be solved in Snap! canvas and can be associated with mini-game displayed in the challenge' panel (Figure 6.2).

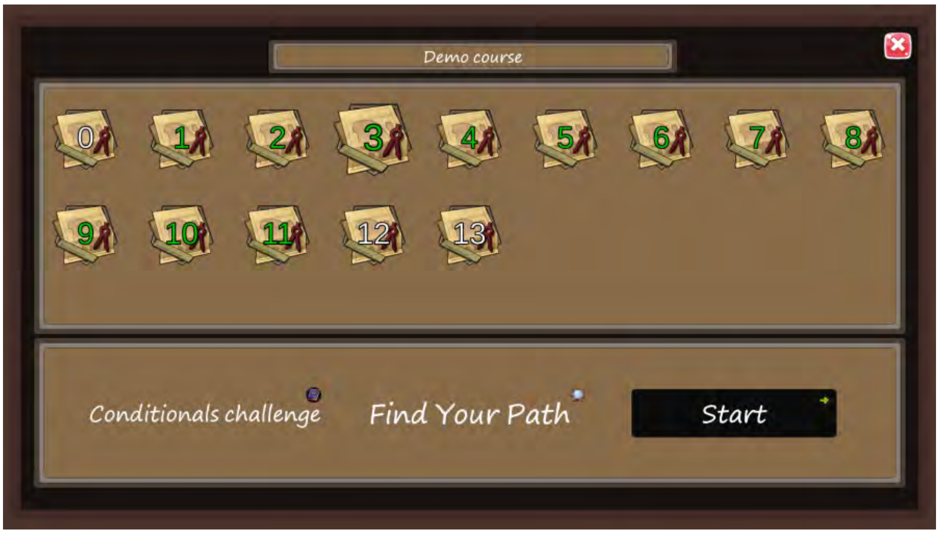
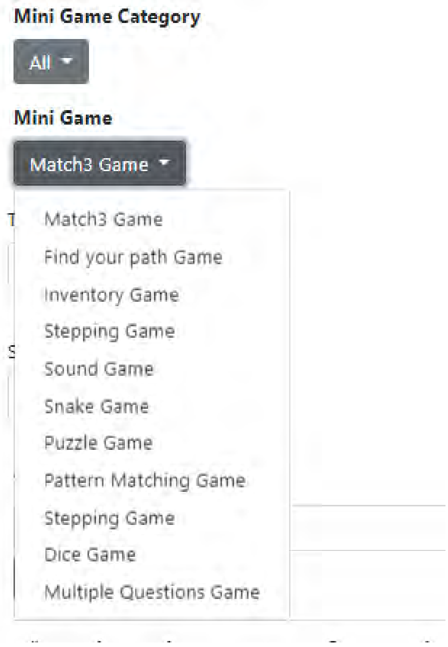


Fig. 6.2 Challenge's panel in the Students' Game Environment

The integration of the mini-game into a specific challenge is to illustrate the programming concept behind the task designed by the teachers. It's not mandatory. This means that the teacher can decide to have or not a mini-game for their students in the challenge and which mini-game to play by selecting them from a list of existing mini-games.

The necessity to associate a mini-game to the challenge should be considered if it leads an added value in terms of understanding, concept-proof, mechanics visualization and, of course, engaging and motivating better the students in their learning path.

For the moment 11 different mini-games exist, ranging from a Match3 game to a multiple-choice question quiz.



Mini Game Category
All

Mini Game
Match3 Game

Fig. 6.3 The mini-games available in the Teachers' platform

When the user enters a challenge attached to a mini-game, they will be teleported to the location where the mini-game takes place. There are many locations for the different mini-games varying from sandy beaches to snowy mountains.

Fig. 6.4 An example of a location associated with one mini-game

Among the available mini-games, there are Snake game, Match3 game, Dice game, Inventory game, Find your path game, Stepping game, Sound game, Puzzle game, Pattern matching game and a multiple-choice question quiz.

Each mini-game has presents its difficulties and the range of available mini-games covers the different possible age groups of the students and/or the programming topics taught in the courses.

One of the simplest mini-games is the snake game, set on a riverbank and where the player needs to follow the wooden arrow sign to a red circle to start the activity. (Figure 6.5.)

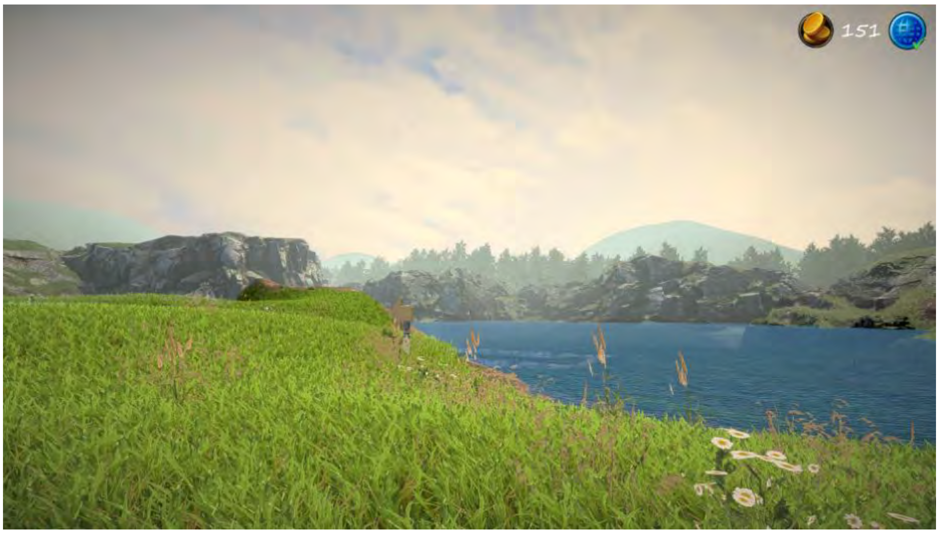


Fig. 6.5 An example of the location associated with a snake game

The snake game (Figure 6.6) takes place in the water. You control the snake using the 4 keyboard arrow and try to survive as long as possible. The snake increases in size by eating the bright green dots but can die if they circle on their tail, hit the border of the screen or eat a red dot. Each eaten green dots adds one point to the score.

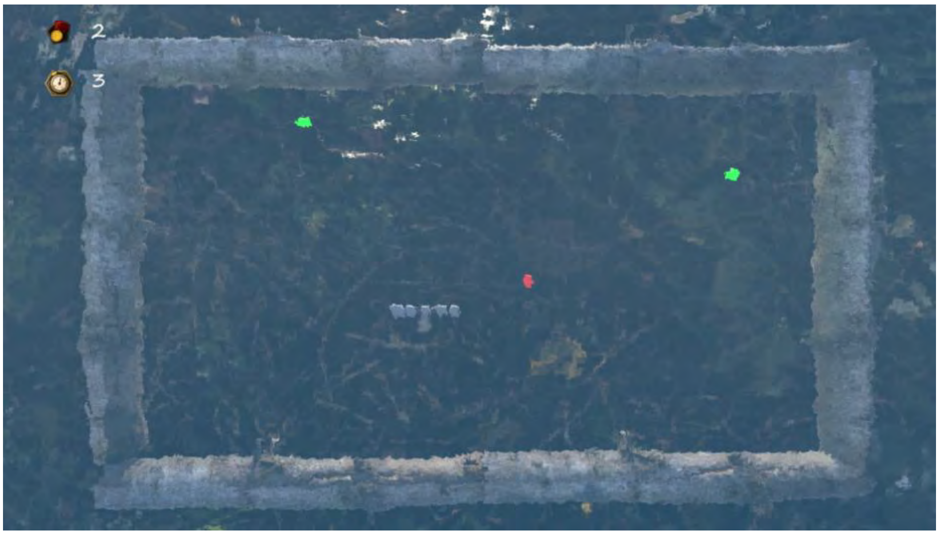


Fig. 6.6 Snake game

This simple minigame offers a good illustration of several programming concepts such as loops, movements, changing graphics and so forth.

Another example is the Match3 game. It takes place in a snowy, elevated region that the players are able to roam freely.

Fig. 6.7 The location associated with a Match3 game

The Match3 mini-game is based on a typical Match3 activity where you have to drag and drop adjacent squares in order to make them disappear if they form a group of three tiles or more of the same type.

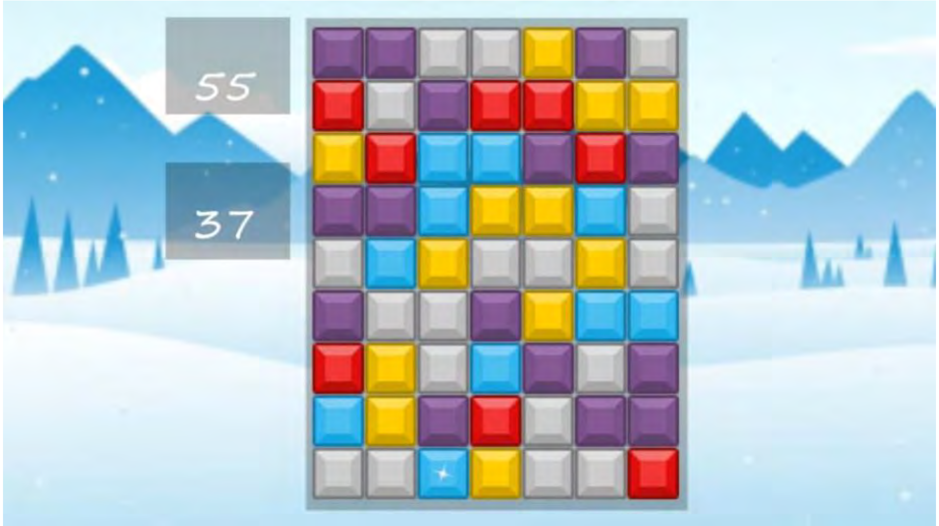


Fig. 6.8 Match3 mini-game

The dice game takes place in dark woods. The interactive part of the mini-game is inside an abandoned cabin, on the chair across the door. It is a wooden board with 2 dice on it.

Both the player and the Artificial Intelligence should roll the dices in turn and the one with the biggest sum of the two dices wins the round.

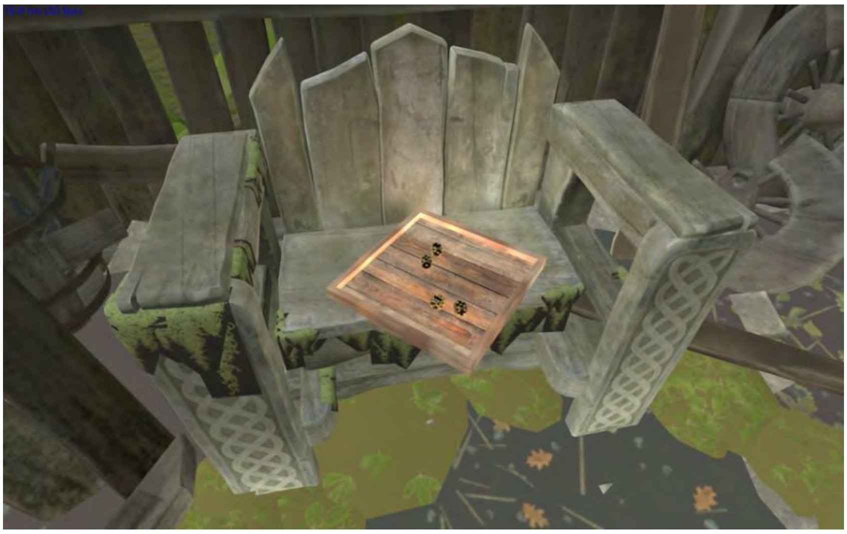


Fig. 6.9 Dice game

The inventory game takes place in the fields, next to a small forest. There are colourful spheres floating around the area and the player needs to collect them, according to the instructions delivered by the teacher on the wooden sign.

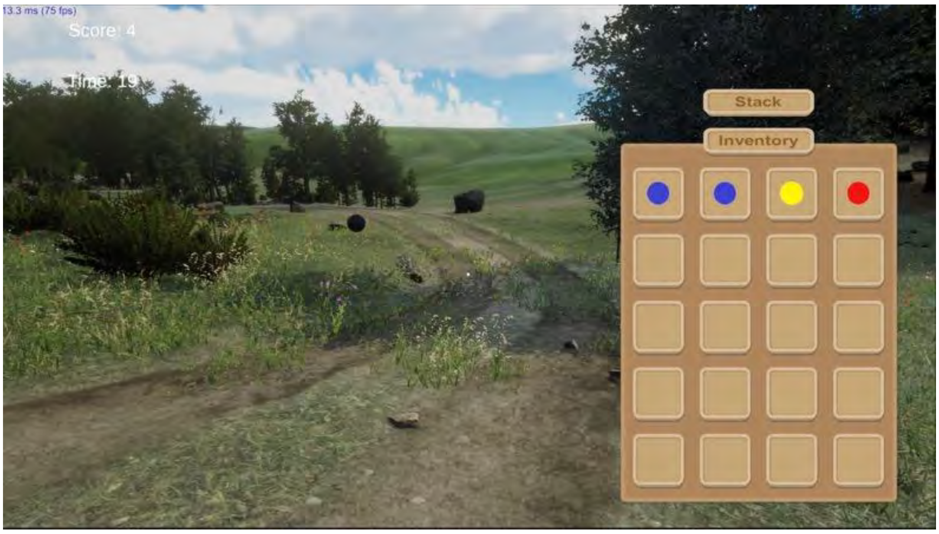


Fig. 6.10 Inventory game

The Stepping mini-game takes place on one beach of the island at sunset. The player needs to answer the question written on the big rock in the centre of the beach by stepping on the letter spelling the correct answer.

When the player steps on one of the stone with a letter forming the answer, the letter will be highlighted in green and appear on the second big rock next to the one where the question is written. If the user steps on a wrong stone it will be highlighted red and they will need to start again the spelling of the answer.

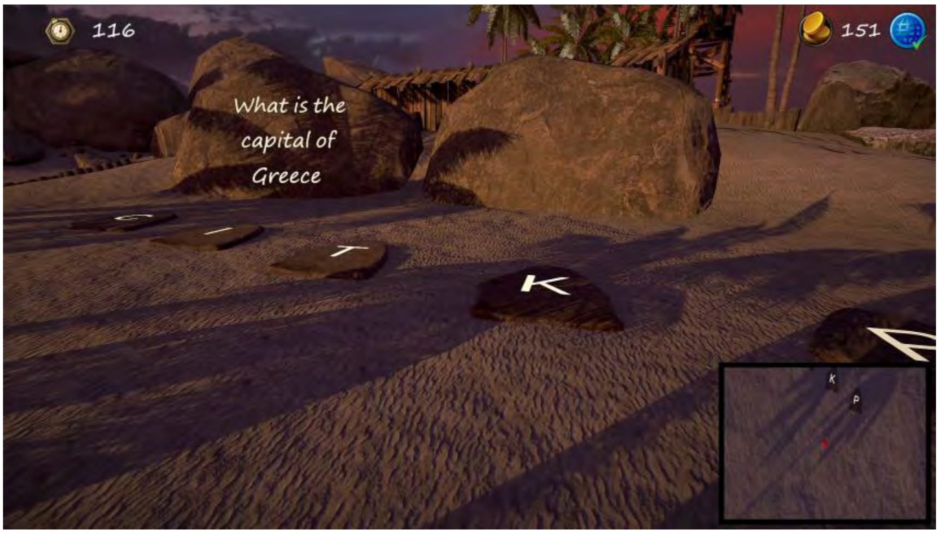


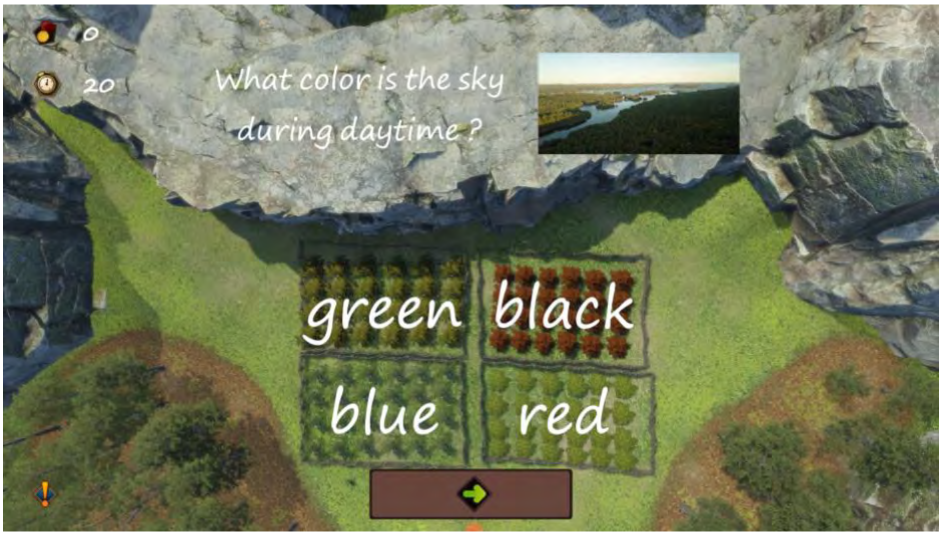
Fig. 6.11 Stepping game

The multiple-choice quiz is set in a field on the side of a cliff and the players are free to roam at the beginning and in order to start the mini-game.

Questions appear on the top of the cliff and each field will contain the four possible answers, alongside with the images, if any.

The medal icon represents the current score of the student, based on the correct answers and the clock countdowns the seconds to end this challenge.

The arrow button at the bottom allows you to go to the next question.



What color is the sky during daytime ?

Fig. 6.12 Multiple choice quiz

The puzzle game takes place in a rocky area of the map. The aim is to find the scattered panels in the area and solve the puzzles by creating a line starting from the white circle of the panel going to the red square.

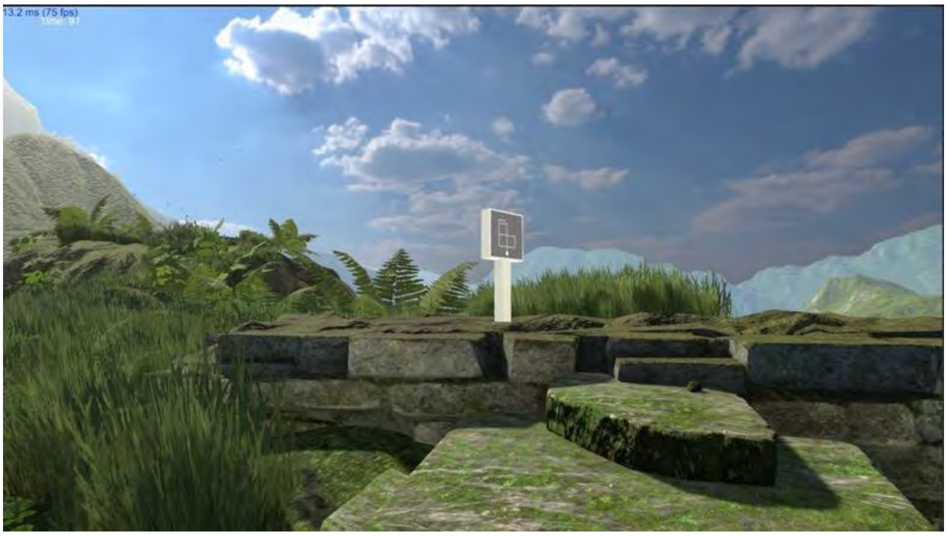


Fig. 6.13 A panel in the puzzle game

The pattern matching game takes place on the banks of a river. There are some boxes with numbers written on their side will start floating on the river towards the sea. The goal of the game is for the player to pick them up in order to complete a certain math operation.

In other terms, the players need to use the floating numbers and available operations to reach the defined number written on a blue table.

The target numbers and the floating numbers are generated randomly but in a way that makes sure, there is always a possible combination to reach the target number.

The available operations depend on the teacher and can be one of the following:

- Basic operations (+, -, *, /)
- Advanced operation (power, square root, modulo)
- Trigonometry (Cos, Sin, Tan)
- Booleans (AND, OR, XOR).

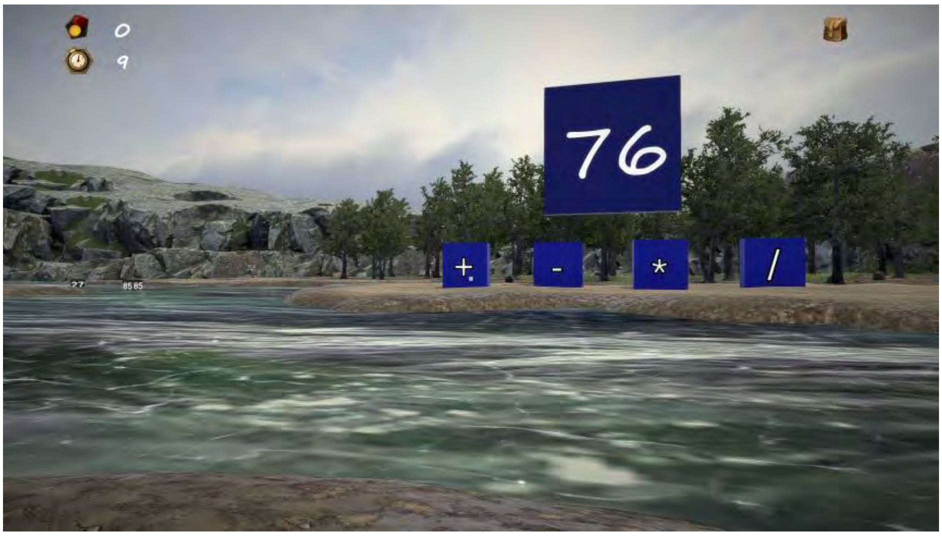


Fig.6.14 Pattern matching game

Another example of mini-game is the sound game which requires a lot of attention from the players to succeed in their task. It takes place in a tropical forest and the player needs to find the five scattered panels and solve for each of them a puzzle based on the sounds they can hear in the jungle.

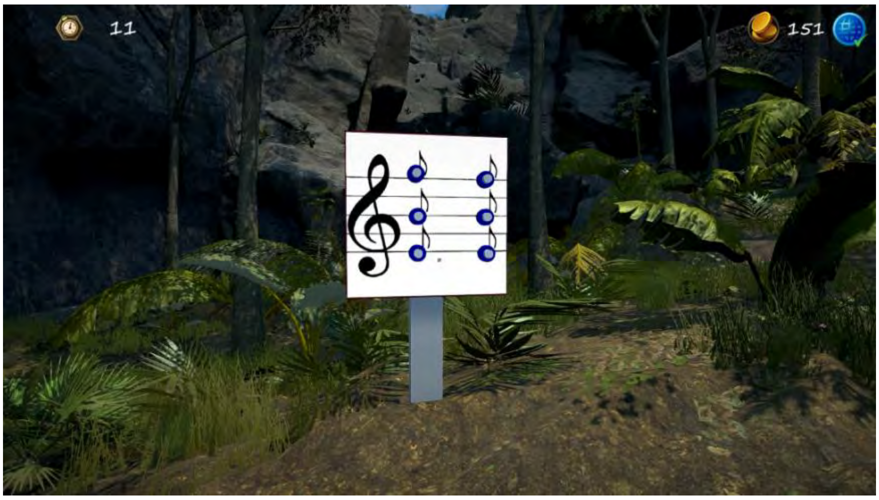


Fig. 6.15 One of the panels in the sound game

Each column of the panel represents one sound and each row represents the tone of the sound. The buttons on top are for the high tones and the ones on the bottom for the low. When the user is near a panel, a combination of 3 possible sounds will play (one low pitched bird call, one middle pitched bird call and one high pitched bird call). The combination of bird calls is repeated in a loop, each repetition separated by the others by a 3 seconds silence.

The player will need to pay great attention to the order in which the bird calls are played and reproduce it on the panel accordingly.

For example on the first panel (Figure 6.16) the user can hear first a low pitched call and then a high pitched one. The correct answer is hence to press the low button on the left of the panel and the high button on the right.

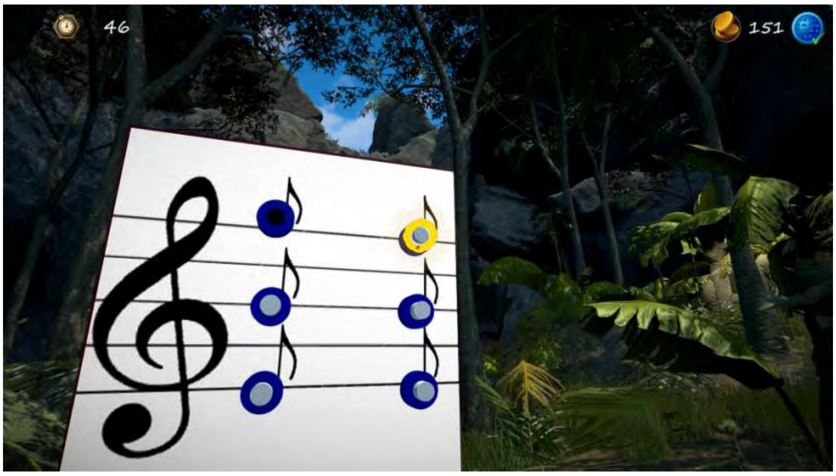


Fig. 6.16 Students' answer displayed on the panel in the Sound game.

To select a certain button, the user simply has to point the mouse cursor at it and the button will be highlighted in yellow. If the pressed button is the correct one then it will appear green, otherwise, it will appear as black.

Sound is often important to any game, but it is crucial in this one and at the center of the gameplay loop. It’s because the players should recognize between high, middle and low tones and, therefore, they need to concentrate on them.

The Students’ Game Environment may be used by all students, mainly aged 10 to 15, and aims to encourage equitable participation in coding activities inside and outside the classroom. It shows the programming concepts through a 3D virtual environment that the students can explore, mini-games demonstrating concepts, a collaborative environment for sharing ideas among class members towards a potential solution, reviews of the solutions of others for building further perspective on tackling a problem, and comparison of solutions to the one that is introduced by the teacher.

Future development and improvement can be foreseen to enrich the C4G software with many more mini-games to help illustrate further programming concepts. The modular nature of the C4G serious game and the way it was designed in Unity allows this sort of future extensions.

APPENDIX 1. LEARNING SHEETS

BASIC LEARNING SCENARIOS

Learning Scenario 1 - Introduction to Snap! interface

Learning Scenario Title	Introduction to Snap! interface
Previous programming experience	/
Learning Outcomes	General learning outcomes: ● get familiar with Snap! visual programming environment Specific learning outcomes: ● Student is able to add a new sprite ● Student is able to add a costume to a sprite and edit it ● Student is able to centre the sprite, so that rotation works appropriately ● Student is able to add a new background to stage and edit it
Aim, Tasks and Short Description of Activities	Student adds a new sprite, adds a costume to the sprite, edits the costume, and deletes one of them. Student creates a new background to the stage, edits it, and deletes unwanted ones. **Aim: By the end of the hour students will draw their favourite character and its living environment, real or imaginary, in order to use it in a game. To make the activity more motivating for all students, the drawing of sprites has been identified in scientific studies to be suitable for this target group.**
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Teacher demonstration Individual work

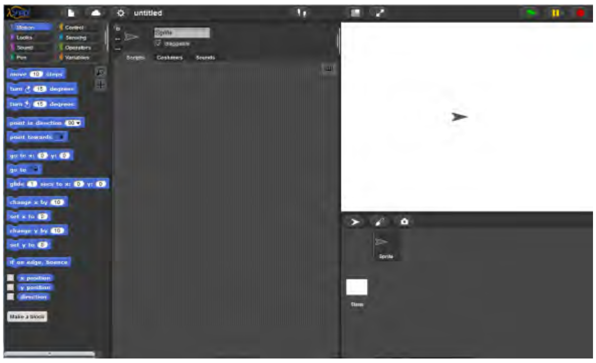
<table>
<tr><td><b>Teaching Forms</b></td><td>Frontal work
Individual work</td></tr>
<tr><td><b>Teaching summary</b></td><td>

(Motivation-Introduction, Implementation, Reflection and evaluation)

By the end of the hour students will draw their favourite character and its living environment, real or imaginary, in order to use it in a game.

[Step 1]

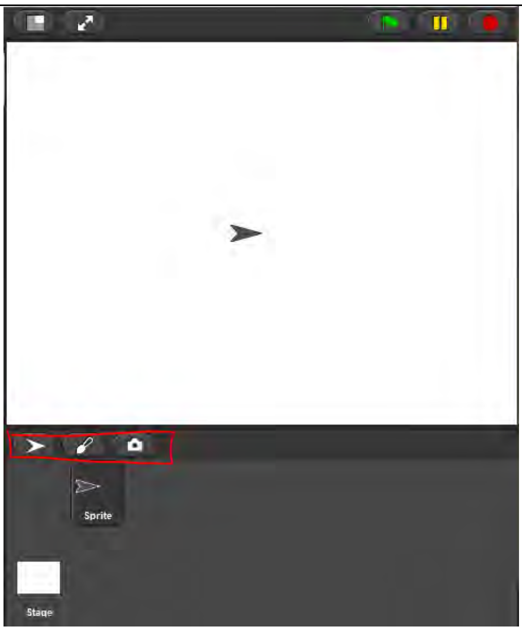
Show students the webpage where they can find Snap! (https://snap.berkeley.edu/). Show them different parts of the interface: section with blocks, section where they can assemble scripts/change costumes/add sounds, stage with sprite on it, list of sprites.

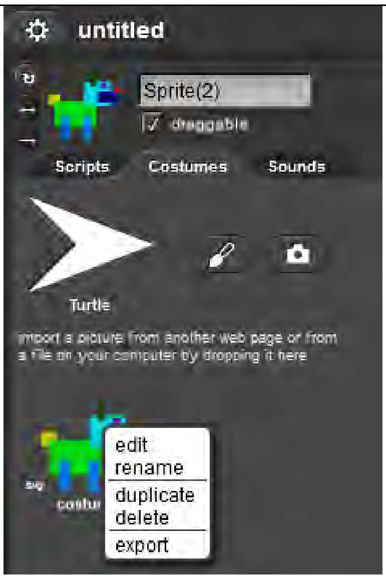


[Step 2]

You can create a new sprite by clicking one of the three buttons:

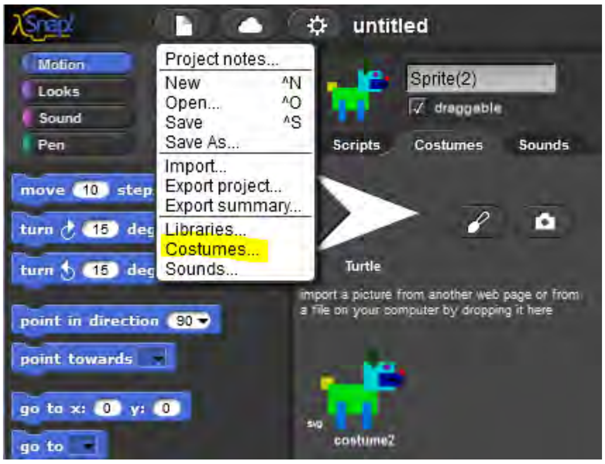
</td></tr>
</table>

	 You will try to draw a new Sprite, therefore click on the paintbrush, and a pop-up window opens where you can draw your sprite in a similar way as in Paint. Task for students: Draw your first sprite. You have 10 minutes. After the sprite is drawn, you should make sure that the rotation centre of the sprite is where you want it to be. To do this use ⌖. Task for students: centre your sprite [Step 3] To edit your sprite, choose Costumes tab, that is only visible, when your sprite is clicked. Right click on a costume you want to edit and choose edit. You can also duplicate your costume or delete it in the same menu.


untitled
Sprite(2)
draggable
Scripts
Costumes
Sounds
Turtle
import a picture from another web page or from a file on your computer by dropping it here
costu
edit
rename
duplicate
delete
export

[Step 4]

To import an already existing costume, click on the icon with a piece of paper drawn on it, and choose Costumes…

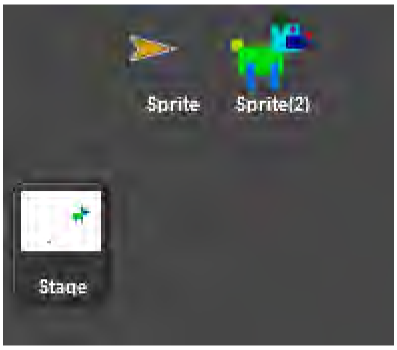
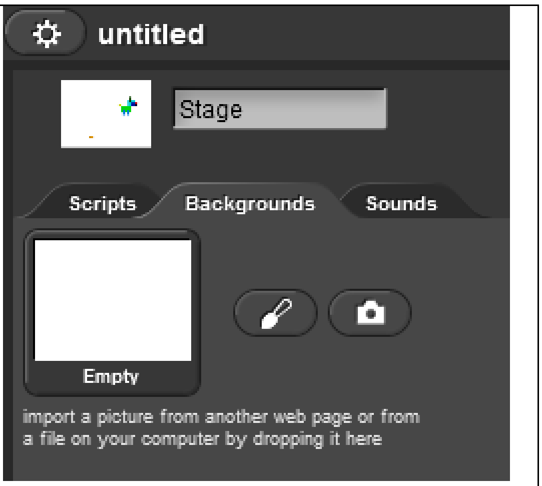


Again, this option will only be shown, when your sprite is clicked on under the stage.

Task for students: select a costume and add it to the sprite

[Step 5]

Now you have your character, and you should add some background to the stage. To do so, first click on the Stage instead on the character under the stage. To add a new background, choose Backgrounds tab:

	  untitled Stage Scripts Backgrounds Sounds Empty import a picture from another web page or from a file on your computer by dropping it here Task for students: draw your own background. Task for students: search through the existing backgrounds and add one of them to the import one of them, so that you have two. Task for students: Find a way to edit your background. Find a way to delete one of your backgrounds, so that only one is left. Reflection and evaluation: Did the students manage to draw their character and environment where (s)he lives? Did they have any problems? How did they solve them?
Tools and Resources for the Teacher	https://snap.berkeley.edu/
Resources/materials for the Students	Instructions for student (C4G1_InstructionsForStudent.docx)

Learning Scenario 2 - Time to bring your sprite to life

Learning Scenario Title	Time to bring your sprite to life
Previous programming experience	/
Learning Outcomes	General learning outcomes: ● Student knows where to find programming blocks and how to connect them into a sequence ● Students knows how to move a sprite ● Student knows how to make spirit say something Specific learning outcomes oriented on algorithmic thinking: ● Making a meaningful sequence of blocks
Aim, Tasks and Short Description of Activities	Student finds out where the programming blocks are stored and how to find the appropriate ones, what categories of blocks are there, and how to connect blocks into a sequence
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Teacher demonstration Individual work
Teaching Forms	Frontal work Individual work
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) You will make your character move and say something during this hour. You can show them an example of a program they will program in this hour. [Step 1]

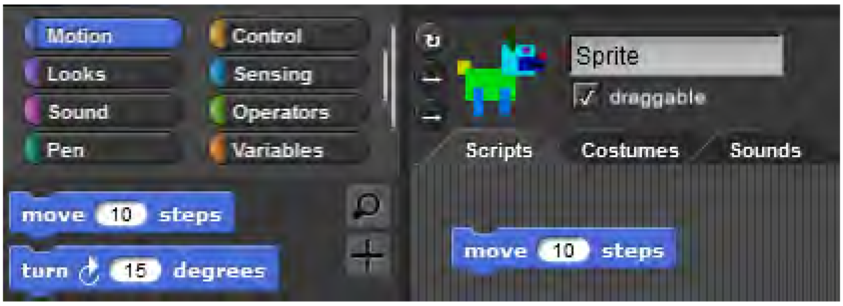
First let's look at where the programming blocks that are available for you to use. Where are they?

On the left hand side, you can find different categories of the blocks: Motion, Looks, Sounds, Pen, Control, Sensing, Operations, and Variables. We will first use blocks.

Task for students: First find the block and then double-click on it. What did it do?

[Step 2]

To start connecting block into a program, you have to drag-and-drop your blocks to the Scripts tab.



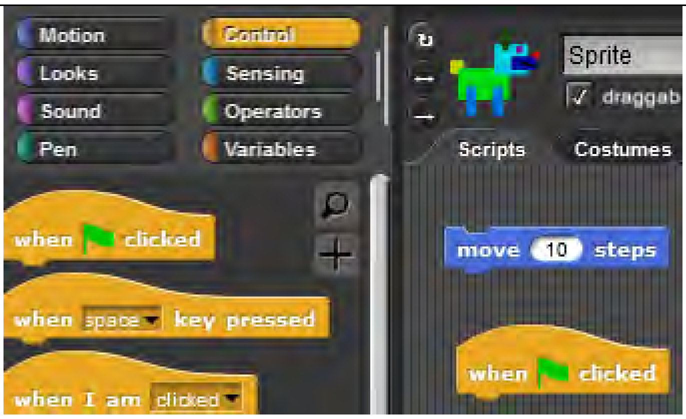
You can double-click on the block inside Scripts tab to execute the code.

[Step 3]

The programs in Snap! are usually started by clicking on the green flag.

Task for students: click through different categories types and try to find a block that starts the program if the green flag is clicked on.

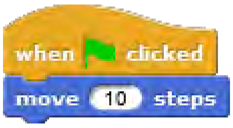
Solution:



```
when [green flag] clicked
when [space] key pressed
when I am [clicked]
```

```
move (10) steps
when [green flag] clicked
```

If you want the program to work in a correct sequence of steps, the blocks have to be connected as with the puzzles. Like this;



```
when [green flag] clicked
move (10) steps
```

Now every time you click on the green flag, the sprite will move for 10 steps, but from different position on the picture.

[Step 4]

If a block has some white space on it, this means that you can change the numbers or letters written there.

Task for students: Make sure your character moves for 30 steps at a time instead of just 10.

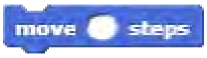
[Step 5]

Make your character say something. Where are you going to find the block say? Try out what is the difference between say Hello! for 2 secs and say Hello!, and explain it to your neighbour.

[Step 6]

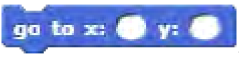
You found both say commands in Looks category. The main difference is that with say Hello! you do not tell the program to wait for __ seconds before the code continues or that it should stop saying it at any time.

[Step 7]

Take your character form the previous hour. By dragging in on the stage move it to the left side of the stage and write a program, that makes the character  from its position on the left to the right side of the stage. After each move, the character should say something. Make more than just one move.

Try it out. Did the character end on exactly same position every time your program is ran? Can you find a block that would make sure your character always starts from the same position and doesn't run off stage?

Tip for teacher: if the character runs off stage, you can call it back on stage by clicking on it with your right mouse button and choosing show.

The block you are looking for is . To determine which x and y are ok, you can move your character to the spot you want it to be on and clicking on x position and y position (on the bottom of Motion category of blocks) and the current x and y will show. You just have to write them into the white spaces in go to block.

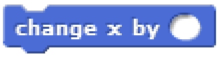
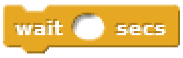
Reflection and evaluation:

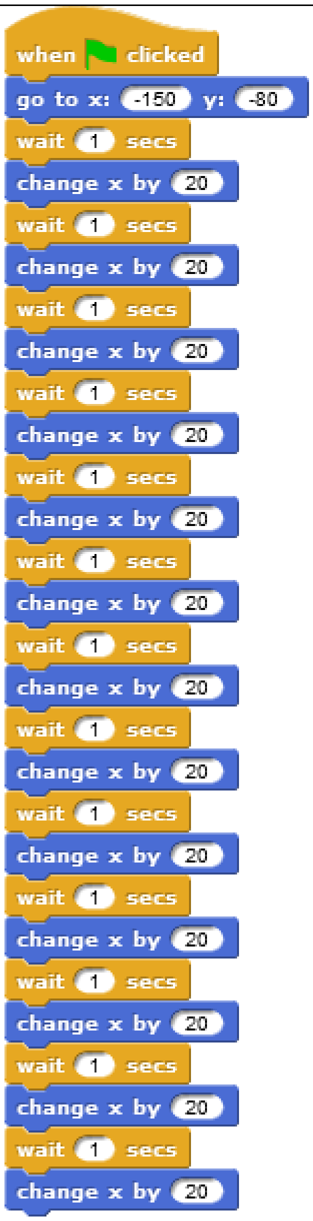
How many times did your character have to repeat the move and say sequence to complete the task? Is the number the same for everyone in the class? Why is that?

Tools and Resources for the Teacher	**Example program:** https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_dog_goes_home
Resources/materials for the Students	● Instructions for student (C4G2_InstructionsForStudent.docx) ● If student didn't draw her own sprite and background, she can use: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_dog_goes_home_tmp

Learning Scenario 3 - Moving around the stage

Learning Scenario Title	Moving around the stage
Previous programming experience	- Student knows where to find programming blocks and how to connect them into a sequence
Learning Outcomes	General learning outcomes: - Making a meaningful sequence of blocks Specific learning outcomes oriented on algorithmic thinking: - Student positions the sprite on the stage - Student changes x and y position of the sprite - Student uses repeat x loop - Student learns that direction of the sprite's movement in move __ steps is relative to the direction the sprite is turned to
Aim, Tasks and Short Description of Activities	**Short description:** Student learns how to move her sprite in x and y direction on the stage, programs an easy program to solve the tasks given, she learns how to turn her sprite in a different direction and how this affects move __ steps block **Tasks:** create a program that moves a sprite in the x direction, create a program that moves a sprite in the y direction, create a program that combines movement in the x and y directions. **Aims:** differentiate between movement in x and y direction on the stage and uses repeat loop
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Teacher demonstration Individual work
Teaching Forms	Frontal work Individual work

Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) You will help different animals to accomplish their goals. To do so, you will need to give them instructions how to move around the stage. [Task 1] Open Catch the ball and add code to the dog so that it catches the ball. Use  change x by ◯ and  wait ◯ secs blocks to make an animation of a dog moving towards the ball. A possible solution to the task:



```
when ⚑ clicked
go to x: -150 y: -80
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
wait 1 secs
change x by 20
```

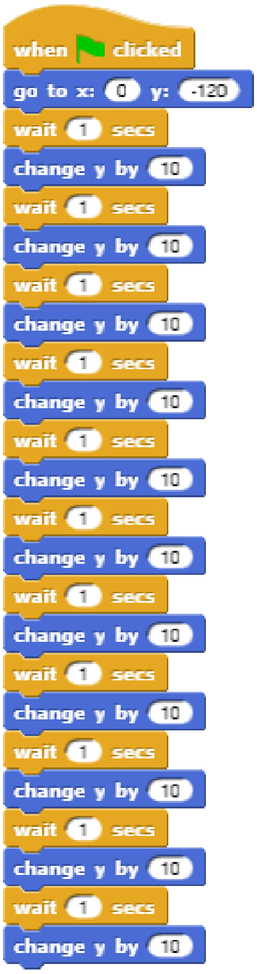
As you can see, the x changes, when you move to the left or to the right. If the x is 0, your sprite is in the middle of the stage. All that is left of the middle, needs - in front of the number and the more away it is, the greater the number. Right of the middle, x values are numbers greater than 0.

Tip: If done with older students, who know decimals, waiting time can be shorter, e.g. 0.1. If they know what coordinate system is, some explanation can be omitted.

[Task 2]

Open Help monkey climb the tree, and add code to the monkey to fetch the bananas. Use `change y by ( )` and `wait ( ) secs` blocks, to make an animation of a monkey climbing on the palm tree.

A possible solution of the task:



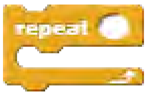
```
when green flag clicked
go to x: 0 y: -120
wait 1 secs
change y by 10
wait 1 secs
change y by 10
wait 1 secs
change y by 10
wait 1 secs
change y by 10
wait 1 secs
change y by 10
wait 1 secs
change y by 10
wait 1 secs
change y by 10
wait 1 secs
change y by 10
wait 1 secs
change y by 10
wait 1 secs
change y by 10
wait 1 secs
change y by 10
```

As you can see, the y changes, when you move up or down. If the y is 0, your sprite is in the middle of the stage. All that is higher than the middle has y greater than 0. If you want your sprite to be below the middle line on the stage, it is just as if you go diving: you say that you are below the water by putting - in front of the number and say, how many “meters” below the water you are and on the stage you say - how many steps below the middle line you are. If you want to climb back down from the tree, use change y by -10.

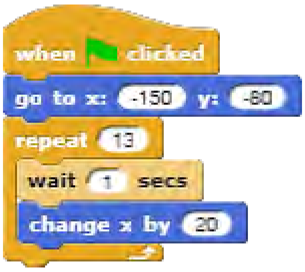
Tip: If done with older students, who know decimals, waiting time can be shorter, e.g. 0.1. If they know what coordinate system is, some explanation can be omitted.

[Step 3]

In both tasks you had to interchangeably use two blocks. How many times did you have to **repeat the code?**

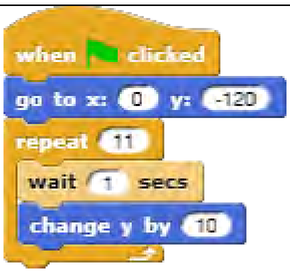
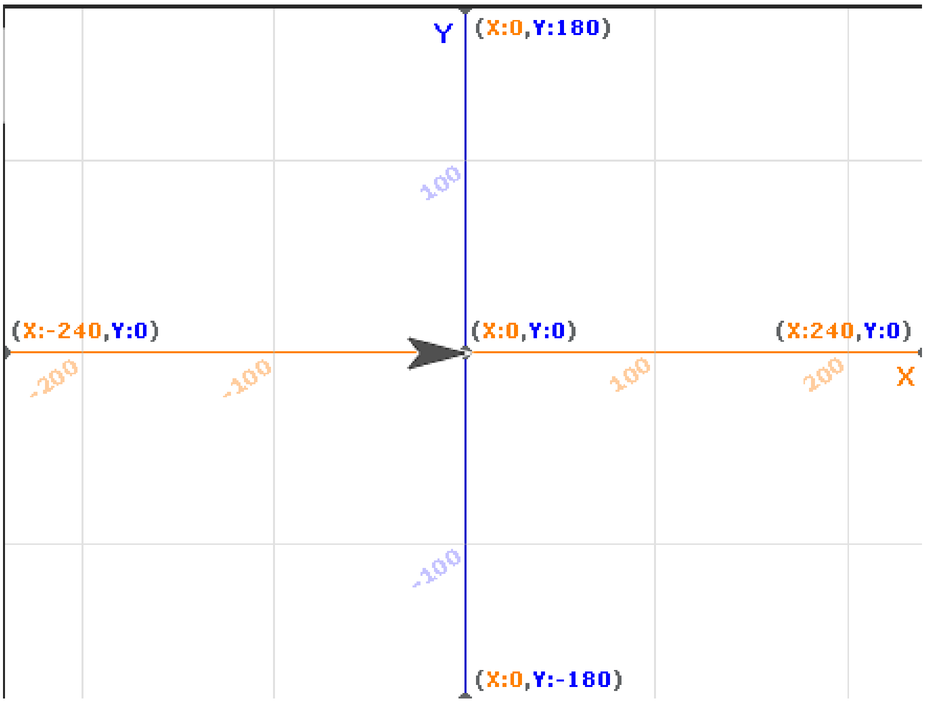
There is a shorter way of writing this code by telling the computer to repeat your code a given number of times. This is repeat __ loop. You can use it when the same action or a sequence of actions repeats itself more then once. Try to change your code for both tasks so, that you use  repeat loop. The code you want to repeat has to be put inside this block, and you have to write how many times it should be repeated in the blank space.

Code for the dog:



```
when [flag] clicked
go to x: -150 y: -80
repeat 13
    wait 1 secs
    change x by 20
```

Code for the monkey:

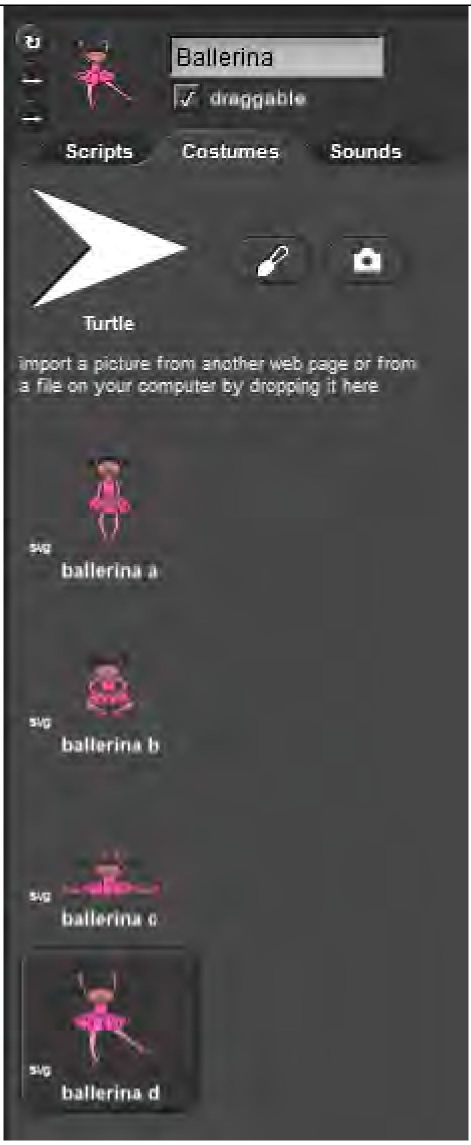
	 when ⚑ clicked go to x: 0 y: -120 repeat 11 wait 1 secs change y by 10 Task: Try to make the dog run to the ball and back. Task: Try to make monkey climb the tree and back down. What did you like the most? You can help yourself with x and y position of the sprite by using XY Grid background in Snap:  Y (X:0,Y:180) 100 (X:-240,Y:0) (X:0,Y:0) (X:240,Y:0) -200 -100 100 200 X -100 (X:0,Y:-180)
Tools and Resources for the Teacher	- A possible solution to Catch the ball: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_moving_x - A possible solution to Help monkey climb a tree: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_moving_y

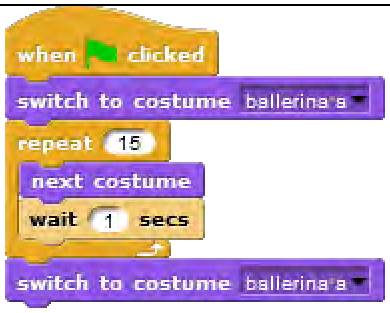
Resources/materials for the Students	● Catch the ball: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Catch_the_ball ● Help monkey climb the tree: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Help_monkey_climb_the_tree ● Instructions for student (C4G3_InstructionsForStudent.docx)

Learning Scenario 4 - Changing costumes and turning

Learning Scenario Title	Changing costumes and turning
Previous programming experience	Movement
Learning Outcomes	General learning outcomes: ● Making a meaningful sequence of blocks Specific learning outcomes oriented on algorithmic thinking: ● Student changes sprite’s costume to make an animation ● Students changes rotation of characters
Aim, Tasks and Short Description of Activities	**Short description:** Student learns how to change the sprite's costume to make an animation. She also learns how to change between different types of rotation of the sprite. **Tasks:** create a program that changes the sprite’s costume. in each program set appropriate type of rotation for each sprite **Aims:** know hot to change sprite’s costume and how to set appropriate type of rotation of the sprite
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Demonstration Individual work
Teaching Forms	Frontal Individual work

Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) You will learn how to make an animation of a sprite so that it looks like it is walking, dancing,... [step 1] Open a new empty project, click on icon that looks like a white piece of paper, and select Costumes… Click on ballerina a, and click on Import. Do the same with ballerina b, ballerina c, and ballerina d. In Costumes tab of your sprite, you now have 4 ballerina costumes. You can rename Sprite to Ballerina, by changing the text above the Costumes tab:

	 Ballerina ☑ draggable Scripts Costumes Sounds Turtle svg ballerina a svg ballerina b svg ballerina c svg ballerina d import a picture from another web page or from a file on your computer by dropping it here Now go back to Scripts tab and try to create a code, that will start when the green flag is clicked, and 15 times change every second change the appearance of the Ballerina. You will need to use next costume block. Make sure our Ballerina starts and finishes her dance with both legs on the floor. Start and end position are not part of her dance. Solution:

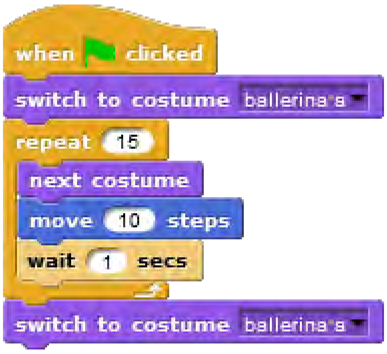


```
when ⚑ clicked
switch to costume ballerina'a
repeat 15
    next costume
    wait 1 secs
switch to costume ballerina'a
```

[Step 2]

Our ballerina doesn't want to be on the same position all the time, so she makes a small movements every time she changes a costume. Add this movement to her dance.

Possible solution:

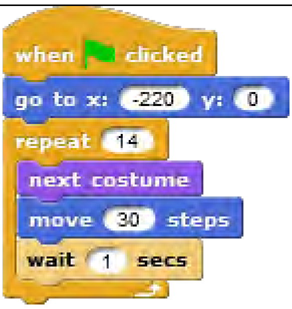


```
when ⚑ clicked
switch to costume ballerina'a
repeat 15
    next costume
    move 10 steps
    wait 1 secs
switch to costume ballerina'a
```

[Step 3]

Open a new empty project and import avery walking costumes. Add a suitable background for Avery to walk on. Create an animation of Avery walking from left side of the stage to the right side of the stage. Try to figure out, how to animate Avery in a way, that her steps will look connected as in real life.

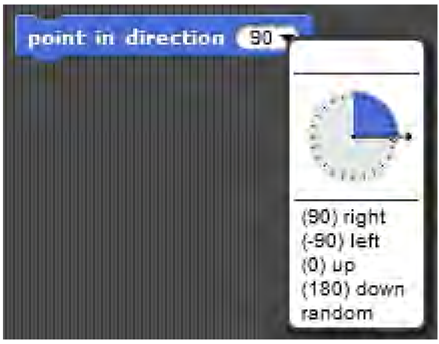
Possible solution:



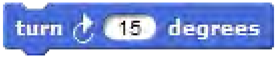
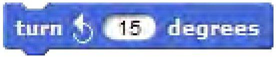
```
when [flag] clicked
go to x: -220 y: 0
repeat 14
    next costume
    move 30 steps
    wait 1 secs
```

[Step 4]

Until now, you always wrote a program where a sprite only moved in one direction. In this task, you will have to turn the mouse, in order to reach the cheese. To make her turn, you can either choose:

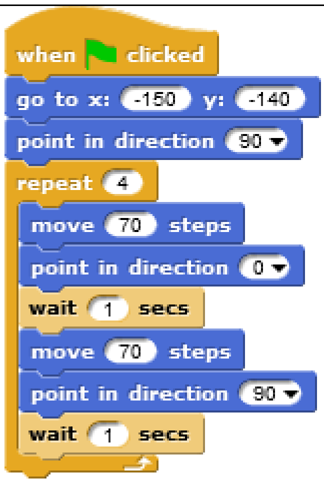
a) 

where you tell her in which direction she has to look or

b) you can tell her to turn for a certain angle clockwise  or counterclockwise . A full circle has 360 degrees, so if you want to turn in the opposite direction from where you are now, you turn for 180 degrees. If you want to turn to your left you turn 90 degrees counterclockwise. If want to turn to your right you turn 90 degrees clockwise.

Open https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Find_cheese. Write a program that mouse has to follow to reach the cheese if she has to walk only on the green area. Make mouse point in the direction she is heading and move __ steps block. To see how the mouse moves, use wait 1 second in between the lines.

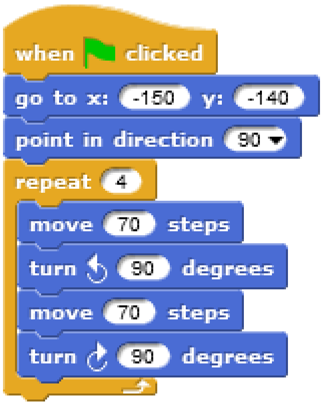
Solution:



```
when [flag] clicked
go to x: -150 y: -140
point in direction 90
repeat 4
  move 70 steps
  point in direction 0
  wait 1 secs
  move 70 steps
  point in direction 90
  wait 1 secs
```

Now try to write a program with turn 90 degrees.

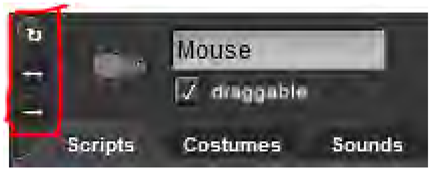
Solution:



```
when [flag] clicked
go to x: -150 y: -140
point in direction 90
repeat 4
  move 70 steps
  turn ↺ 90 degrees
  move 70 steps
  turn ↻ 90 degrees
```

[Step5]

As you have seen, the mouse has turned in different directions to reach the cheese. Sometimes you don't want your sprite to turn upside down, but to just turn to the left or to the right so it doesn't walk on its head. To make sure your sprite turns like you want it to, you have to click on appropriate icon left of your sprite:



The circular arrow means, that your sprite can turn in any direction (like your mouse)

	The <-> arrow means that your spirit will only turn to the left or to the right (this is what you would use for the dog not to walk on its head The last -> arrow means that the sprite will always look as it is (you could use this for the monkey) Try to rewrite your programs for the dog and the monkey so that they first go the the object and back by turning. Make sure you change their rotation style properly.
Tools and Resources for the Teacher	● Ballerina program solutions: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_dancing ● Avery walking: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Avery_walking ● Find cheese solution: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Find_cheese_solution
Resources/materials for the Students	● Find cheese: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Find_cheese ● Instructions for student (C4G4_InstructionsForStudent.docx)

Learning Scenario 5 - Sounds of the farm

Learning Scenario Title	Sounds of the farm
Previous programming experience	● Student is able to add a background. ● Student is able to add a new sprite. ● Student knows how to make sprite say something.
Learning Outcomes	General learning outcomes: ● add sound from Snap's media library, ● import sound from other media, ● record a new sound, ● play sound when a key is pressed. Specific learning outcomes oriented on algorithmic thinking: ● student adds sound from Snap's media library and plays it when a certain key is pressed, ● student imports sound from computer and plays it when a certain key is pressed, ● student records a new sound and plays it when a certain key is pressed.
Aim, Tasks and Short Description of Activities	**Short description**: Program simple game in which player learns the sounds of animals by pressing certain keys. **Tasks**: In the first step student has to choose scene background. Than, student has to program the woman farmer to tell the instructions: 1) If you want to hear the dog, click on the key "D"!; 2) If you want to hear the cow, click on the key "C"!; 3) If you want to hear the sheep, click on the key "S"!; 4) If you want to hear the pig, click on the key "P"!; 5) If you want to hear the horse, click on the key "H"!. After that, student has to program the task as directed by the woman farmer. **Aim**: Students will be introduced how to add a new sound and how to use it. They will also learn how to use the sound block ("*play sound*

	[name_of_sound]”) and the control block (“*when [the_key] key pressed”*).
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Active learning, game-design based learning
Teaching Forms	Frontal teaching Individual work
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) **Motivation-Introduction** We motivate students by playing the game (they don’t see the code). The goal of the lesson is to make the game like this. If you want to hear the cow, click on the key "C"! [Step 1] The first step is to determine the background of the game. The background has to contain different animals. We have three options: 1. the students draw the background themselves; 2. the students search for free image online; 3. we provide background for students (if we want to save time). Students already know how to add background, so they do it individually.

[Step 2]

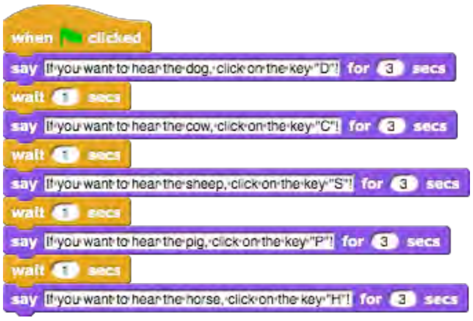
The second step is to add the woman farmer. We have the same options like in the first step:

1. the students draw the woman farmer themselves;
2. the students search for free image of the woman farmer online;
3. we provide image of the woman farmer for students (if we want to save time).

Students already know how to add a new sprite, so they do it individually.

[Step 3]

Next, students have to program the instructions for the player. The instructions are given by the woman farmer. Students do that by using *Looks/say[string]* and *wait[n]* block. Students already know how to do this, so they do it individually.



```
when ⚑ clicked
say [If you want to hear the dog, click on the key "D"!] for (3) secs
wait (1) secs
say [If you want to hear the cow, click on the key "C"!] for (3) secs
wait (1) secs
say [If you want to hear the sheep, click on the key "S"!] for (3) secs
wait (1) secs
say [If you want to hear the pig, click on the key "P"!] for (3) secs
wait (1) secs
say [If you want to hear the horse, click on the key "H"!] for (3) secs
```

Implementation

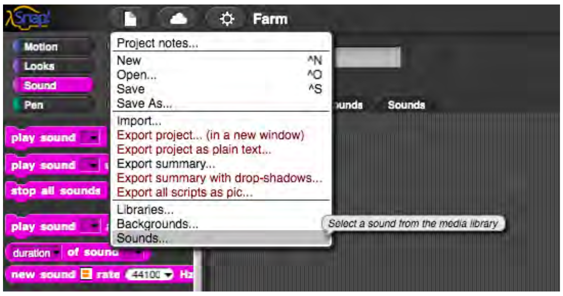
Next we show students how to add sound in the game. We have three options:

1. importing a sound from the Snap's media library;
2. importing a sound from our computer by dragging it into Snap!;
3. recording a new sound in Snap!

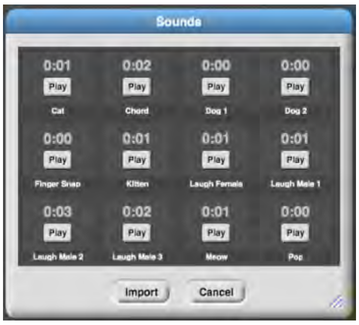
We show students all three options in the form of frontal teaching. When we introduce them all, they start to program the following tasks individually (with the support of the teacher).

[Step 4]

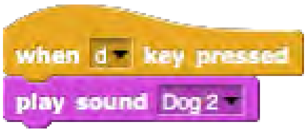
Students have to program the dog's sound. When the player presses the "D" key, the dog has to bark. First, students import the sound from the Snap's media library to background's sound tab.



Next, they choose the sound of the dog (Dog 1 or Dog 2).



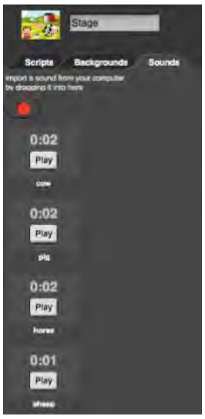
Students have to program the sound of the dog which will be played when the key “D” is pressed. They do that by using *Control/when [the_key] key pressed* block and *Sound/play sound [name_of_sound]* block.



```
when d key pressed
play sound Dog 2
```

[Step 5]

Students have to program sounds of animals. First, they have to add sounds from their computer. They do that by dragging the sounds in the background’s sounds tab.



Stage

import a sound from your computer by dragging it into here

0:02 Play cow

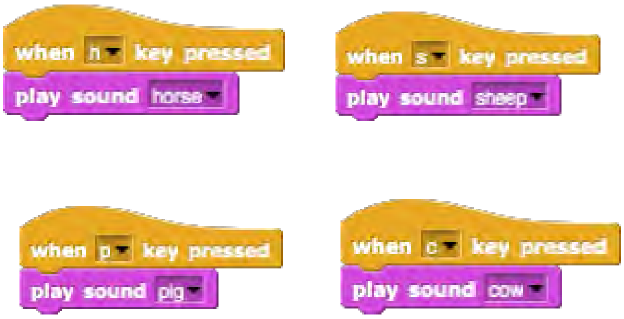
0:02 Play pig

0:02 Play horse

0:01 Play sheep

Once we have the sounds imported, we can right-click the sounds to rename them. In our case they are called a cow, a pig, a horse and a sheep.

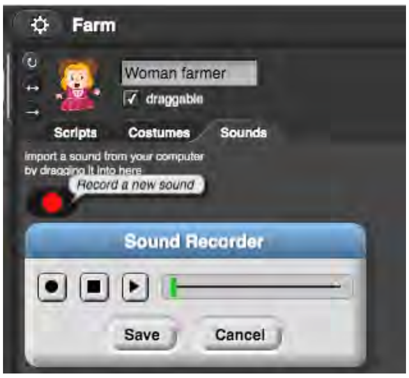
Next, students have to add the sound in background's scripts. They do that by using *Control/when[the_key] key pressed* and *Sound/play sound[name_of_sound]* block.



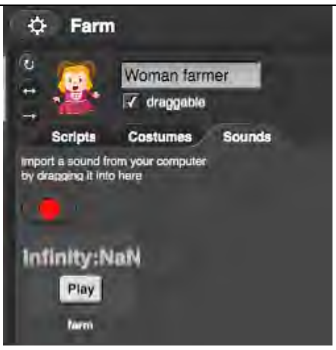
```
when h key pressed
play sound horse

when s key pressed
play sound sheep

when p key pressed
play sound pig

when c key pressed
play sound cow
```

[Step 6]

Next step is to program the woman farmer's welcome greeting. When player start the game the woman farmer has to say: "Welcome to my farm". First, students have to record the woman farmer's welcome greeting. They do that with sound recorder (red button) located in the (woman farmer's) Sounds tab. When they record the sound, they have to save it (Save button).



Once we have the sound saved, we can right-click it to rename it. In our case it is called a farm.



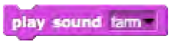
Woman farmer

☑ draggable

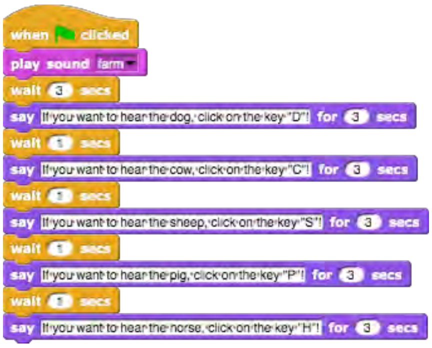
Import a sound from your computer by dragging it into here

Infinity:NaN

Now students have to add the sound in woman farmer’s scripts. They do that by using *Sound/play sound[name_of_sound]* block.



```
play sound farm
```



```
when (flag) clicked
play sound farm
wait 3 secs
say If you want to hear the dog, click on the key "D"! for 3 secs
wait 1 secs
say If you want to hear the cow, click on the key "C"! for 3 secs
wait 1 secs
say If you want to hear the sheep, click on the key "S"! for 3 secs
wait 1 secs
say If you want to hear the pig, click on the key "P"! for 3 secs
wait 1 secs
say If you want to hear the horse, click on the key "H"! for 3 secs
```

[Additional task]

Student can upgrade the farm as he or she like by adding new sprites (farmer, hen, tractor, ...) and sounds.

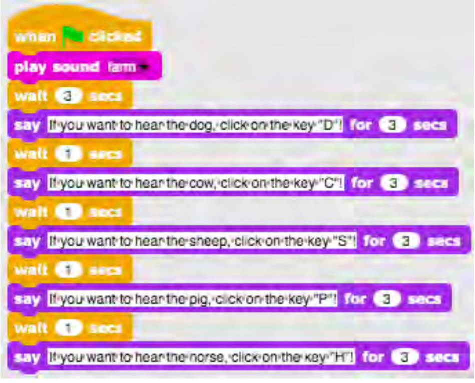
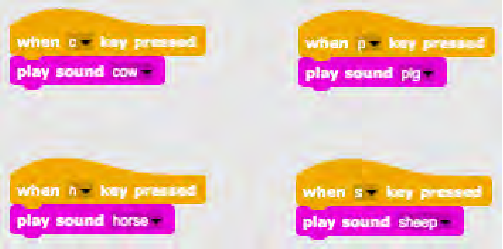
Reflection and evaluation

Students summarize:

- how they added sounds in their code;
- which blocks they used to insert sound into the code;
- which control blocks they used in their code;
- why and how they used sound blocks and control blocks.

[Final Code]

The woman farmer

	 when clicked play sound farm wait 3 secs say If you want to hear the dog, click on the key "D"! for 3 secs wait 1 secs say If you want to hear the cow, click on the key "C"! for 3 secs wait 1 secs say If you want to hear the sheep, click on the key "S"! for 3 secs wait 1 secs say If you want to hear the pig, click on the key "P"! for 3 secs wait 1 secs say If you want to hear the horse, click on the key "H"! for 3 secs *The background*  when c key pressed play sound cow when p key pressed play sound pig when h key pressed play sound horse when s key pressed play sound sheep
Tools and Resources for the Teacher	● Whole activity in Snap!: https://snap.berkeley.edu/project?user=tadeja&project=Farm ● Website of free images: https://pixabay.com/ ● Website of free sounds: https://www.zapsplat.com/ ● Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena. ● Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.
Resources/materials for the Students	● Template in Snap!: https://snap.berkeley.edu/project?user=tadeja&project=Sounds%20of%20the%20farm_0 ● Website of free images: https://pixabay.com/ ● Website of free sounds: https://www.zapsplat.com/ ● Instructions for student (C4G5_InstructionsForStudent.docx)

Learning Scenario 6 - Chameleon's summer vacation

Learning Scenario Title	Chameleon's summer vacation
Previous programming experience	no prior programming knowledge is required
Learning Outcomes	General learning outcomes: ● event based object movement, ● single or multiple color sensing, ● Boolean value readings in logical expressions, ● defining, differentiating, dynamically checking and responding to different game states, Specific learning outcomes oriented on algorithmic thinking: ● student implements object movement with arrow keys using events and takes into account restrictions, ● student uses a sensing color block to get the boolean value for single or multiple color sensing reading, ● student realizes object state can be expressed with the colors the object is touching, ● student differentiates between two (basic) five (full) different states and knows how to express them with logical expressions, ● student realizes that position of the object is dynamically changing and uses forever loop to repeatedly check the current state, ● student uses if sentence to give different responses based on the current position of the object.
Aim, Tasks and Short Description of Activities	Short description: Program simple game in which the object will change its costume based on the color of the background. Tasks: Students have to program chameleon to change his looks (costume) and also tell where he is in five different situations: 1) when swimming in the sea, he has to change his color to blue and say "I am swimming in the sea", 2) when he is between the sea and the beach his

	skin turns half blue-half sandy color and he says “I am between the sea and the beach”, 3) on the beach, he takes on a sandy color and says “I am relaxing at the beach”, 4) between the beach and the forest, he turns half green-half sandy color and says “I am between the beach and the forest”, 5) in the forest, his skin turns green and he says “I am cooling off in the tree shade”. **Students will be introduced to sensing color block and how to use it in logical expressions in order to differentiate between dynamically changing game states and give the right responses.**
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	active learning, collaborative learning, problem solving
Teaching Forms	frontal teaching individual work/working in pairs/group work
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) Chameleon went on a summer vacation. He likes to bathe in the sea, enjoy relaxing at the beach and when it’s too hot he likes to go to the shelter of nearby trees to cool itself. Because he is a chameleon he changes his color according to its current background. **[Basic version]** In the basic version we have to differentiate between two states. [Step 1] We ask students to edit the scene background so it is divided into two parts of the same color, blue and sandy, each representing a different place. Color blue is for the sea and sandy for the beach. We can instruct students to include other items to make the background more realistic,

such as: waves, shells, sand castles, sun umbrellas, etc… They have to be careful not to choose items that are bigger and entirely colored with different colors than the background. In that case color sensing block won't be able to recognize which part of the scene the character is on.

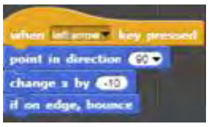
[Step 2]

They have to draw a chameleon and paint his skin in two different colors:

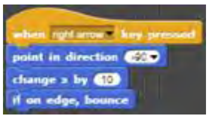
[Step 3]

First they have to make their chameleon move in four directions using keys. They can choose their own key combination (e.g. arrow keys or WASD). At this point we assume that they know how to do it from previous activities. We have to remind students that character can move out of the scene if we don't use appropriate block when programming movement (bounce if on edge block).

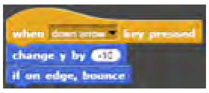
To make chameleon movement a little more realistic, we want him to turn left or right to face the horizontal direction we are facing (using a *point in direction* block).



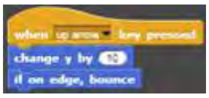
```
when left arrow key pressed
point in direction 90
change x by -10
if on edge, bounce
```



```
when right arrow key pressed
point in direction -90
change x by 10
if on edge, bounce
```



```
when down arrow key pressed
change y by -10
if on edge, bounce
```



```
when up arrow key pressed
change y by 10
if on edge, bounce
```

[Step 4]

Co-funded by the Erasmus+ Programme of the European Union

We introduce students to the concept of character sensing the color (colors) that he is touching. With the block "touching color?" we can get information in a form of Boolean values – True or False if he is touching a certain color. Because we get Boolean value from this block we can use it in the head of If sentence where it is decided if we are going to execute commands listed in its body or not.

Next we discuss with the students what are the different positions of chameleon on the scene and how can we express them using touching color? block.

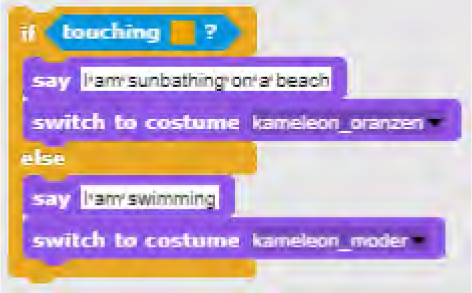
There are two:

1. He is touching the color blue -> Touching color [blue]?
2. He is touching the sandy color -> Touching color [sandy]?

When he is touching certain color we have to change its appearance and we also have to make him say where he is. We can change the appearance of a Sprite by switching between its costumes. This is done with *Looks/switch to costume[option]* block where we select which one of the possible costumes we want to display. In order to make chameleon speak we use *Looks/say[text]* block.

Because there are only two possibilities we can use "if - else" conditional block.

We can choose which color are we going to check and implicitly other color will fall into "else" case. In the sample code we chose sandy color:

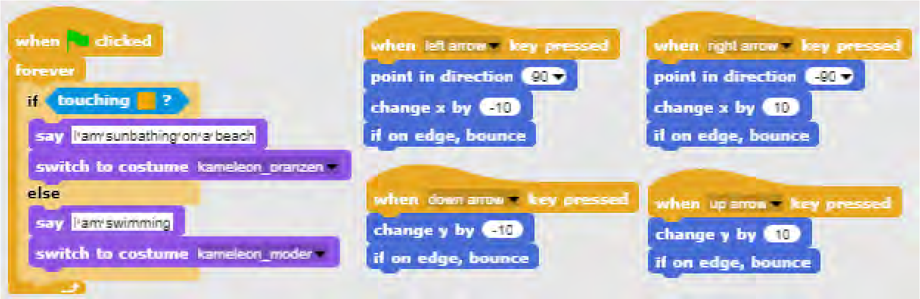


```
if <touching [ ] ?>
    say [I'am sunbathing on a beach]
    switch to costume [kameleon_oranzen ▾]
else
    say [I'am swimming]
    switch to costume [kameleon_moder ▾]
```

[Step 5]

For situations when we have to execute certain commands for the entire duration of the program we use – forever loop. Everything written under the body of forever loop is going to execute over and over again. We discuss with the students that in our case this is exactly what we want/need in order to create this game.

[Final Code]



```
when 🟩 clicked
forever
    if <touching [ ] ?>
        say [I'am sunbathing on a beach]
        switch to costume [kameleon_oranzen ▾]
    else
        say [I'am swimming]
        switch to costume [kameleon_moder ▾]

when [left arrow ▾] key pressed
point in direction (90 ▾)
change x by (-10)
if on edge, bounce

when [right arrow ▾] key pressed
point in direction (-90 ▾)
change x by (10)
if on edge, bounce

when [down arrow ▾] key pressed
change y by (-10)
if on edge, bounce

when [up arrow ▾] key pressed
change y by (10)
if on edge, bounce
```

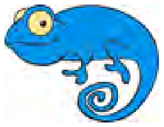
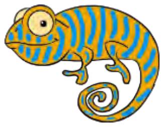
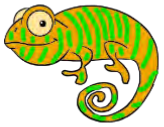
[Full version]

[Step 1]

We ask students to edit the scene background so it is divided into three parts of the same color, each representing a different place: blue color is for the sea, sandy color for the beach and green for the forest. They can add other items to make a background more realistic such as: waves, shells, sand castles, sun umbrellas, trees, etc... but they have to be careful that added items are not bigger than the main character itself, because in this case character won't touch any of three colors and Snap’s sensing feature won’t be able to recognize on which part of the scene the character is.

[Step 2]

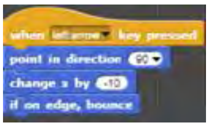
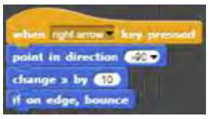
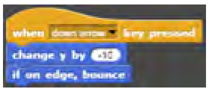
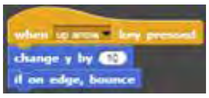
They have to draw a chameleon and paint his skin in five different combinations representing his position on the scene:

[Step 3]

First they have to make their chameleon move in four directions using keys. They can choose their own key combination (e.g. arrow keys or WASD). At this point we assume that they know how to do it from some other activity. We have to warn the students not to forget that the character can move out of the scene if we don’t use appropriate block when programming movement (bounce if on edge block).

To make chameleon movement a little more realistic, we want him to turn left or right to face the horizontal direction we are facing (using a *point in direction* block).

 when [left arrow] key pressed point in direction (90) change x by (-10) if on edge, bounce	 when [right arrow] key pressed point in direction (-90) change x by (10) if on edge, bounce	 when [down arrow] key pressed change y by (-10) if on edge, bounce	 when [up arrow] key pressed change y by (10) if on edge, bounce

[Step 4]

We introduce students to the concept of character sensing the color (colors) that he is touching. With the block “touching color?” we can get information in a form of Boolean values – True or False if he is touching single or even multiple colors at the time. Because we get Boolean value from this block we can use it in the head of If sentence where it is decided if we are going to execute commands listed in its body or not.

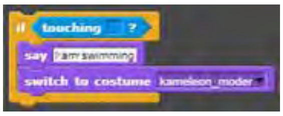
Next we discuss with the students what are the different positions of the chameleon on the scene and how can we express them using touching color? block.

We quickly find out there are five:

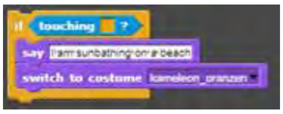
1. He is entirely on the blue part -> Touching color [blue]?
2. He is between the blue and sandy part -> Touching color[blue]? AND Touching color [sand]?
3. He is entirely on the sandy part -> Touching color [sand]?
4. He is between the sandy and green part -> Touching color[sand]? AND Touching color [green]?
5. He is entirely on the green part -> Touching color [green]?

When he is touching a certain color(s) we have to change its appearance and we also have to make him say where he is. We can change the appearance of a Sprite by switching between its costumes. This is done with *Looks/switch to costume[option]* block where we select which one of the possible costumes we want to display. In order to make a chameleon speak we use *Looks/say[text]* block.

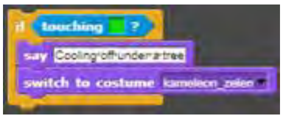
First we take care of the simpler situations where chameleon is entirely on the same color part of the scene:



```
if touching 🟦 ?
  say I am swimming
  switch to costume kameleon_moder
```

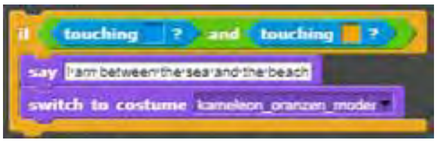


```
if touching 🟧 ?
  say I am sunbathing on a beach
  switch to costume kameleon_oranzen
```

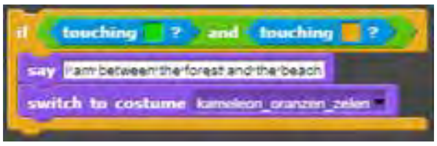


```
if touching 🟩 ?
  say Cooling off under a tree
  switch to costume kameleon_zelen
```

Next we form a logical expression with the use of logical operator AND, because we want to verify if chameleon is touching two colors at the same time:



```
if touching 🟦 ? and touching 🟧 ?
  say I am between the sea and the beach
  switch to costume kameleon_oranzen_moder
```



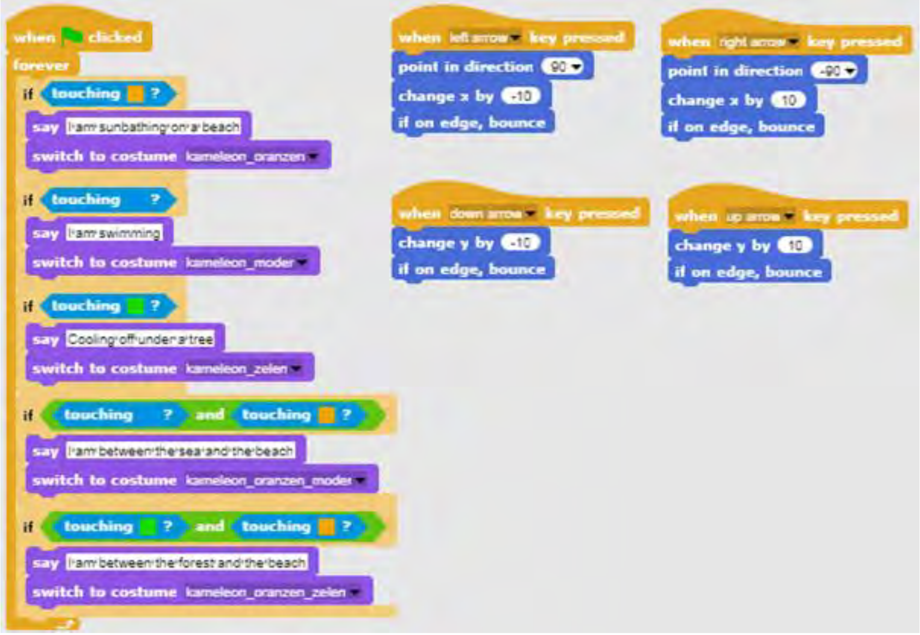
```
if touching 🟩 ? and touching 🟧 ?
  say I am between the forest and the beach
  switch to costume kameleon_oranzen_zelen
```

If we combine the conditional sentences above and put them under *When Green Flag clicked* block event, we notice that these conditions will be checked exactly once. We help them notice that because we control the movement of the main character, chameleon position will be changing all the time during the game. This is why we have to constantly check those conditions not only once, but literally all the time!

[Step 5]

For situations when we have to execute certain commands for the entire execution of the program we use – forever loop. Everything written under the body of the forever loop is going to execute over and over again. We discuss with the students that in our case this is exactly what we want/need in order to create this game.

[Final Code]

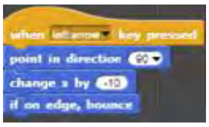
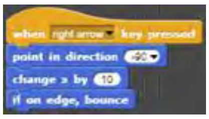
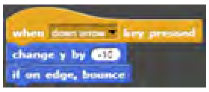
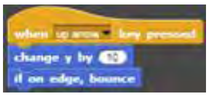


```
when 🏳 clicked
forever
  if <touching 🟧 ?>
    say I am sunbathing on a beach
    switch to costume kameleon_oranzen
  if <touching ?>
    say I am swimming
    switch to costume kameleon_moder
  if <touching 🟩 ?>
    say Cooling off under a tree
    switch to costume kameleon_zelen
  if <<touching ?> and <touching 🟧 ?>>
    say I am between the sea and the beach
    switch to costume kameleon_oranzen_moder
  if <<touching 🟩 ?> and <touching 🟧 ?>>
    say I am between the forest and the beach
    switch to costume kameleon_oranzen_zelen

when left arrow key pressed
point in direction 90
change x by -10
if on edge, bounce

when right arrow key pressed
point in direction -90
change x by 10
if on edge, bounce

when down arrow key pressed
change y by -10
if on edge, bounce

when up arrow key pressed
change y by 10
if on edge, bounce
```

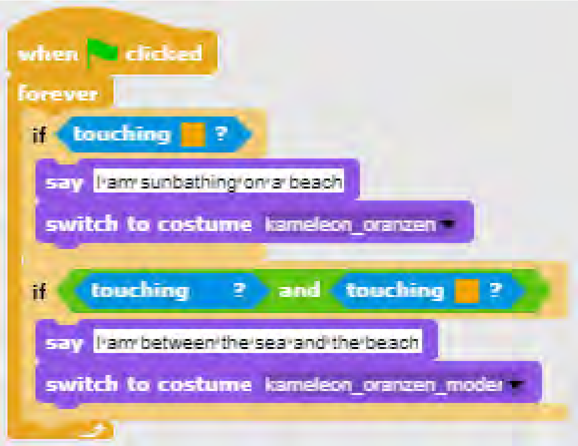
[Students adjust the code]

In order to simplify this activity we can prepare some of the code beforehand in a template file and instruct students to complete it.

Students who followed suggested learning path already learned about moving the object with keys. So we can include the movement code in a template file. They can change the keys settings from arrow keys to custom arrangement (e.g. WASD).

 when left arrow key pressed point in direction 90 change x by -10 if on edge, bounce	 when right arrow key pressed point in direction -90 change x by 10 if on edge, bounce	 when down arrow key pressed change y by -10 if on edge, bounce	 when up arrow key pressed change y by 10 if on edge, bounce

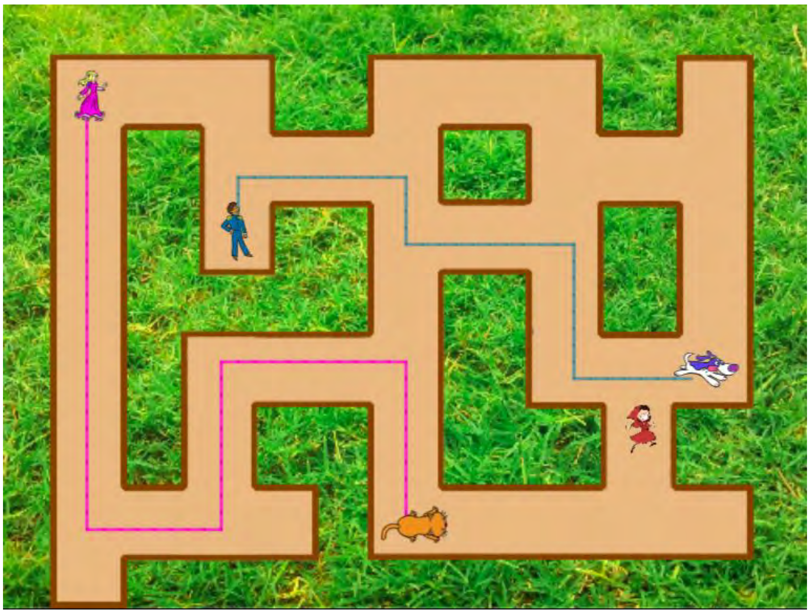
To help them understand the concept of forever loop and how to use it for detecting background color we can include code for detecting two situations: 1) the object is entirely on one color, 2) the object is

	touching two colors at the same time. We instructed them to complete the code for every case.
	Suggested code template:
	
	`when ⚑ clicked`
	`forever`
	`  if touching 🟧 ?`
	`    say I am sunbathing on a beach`
	`    switch to costume kameleon_oranzen`
	`  if touching ? and touching 🟧 ?`
	`    say I am between the sea and the beach`
	`    switch to costume kameleon_oranzen_moder`

Tools and Resources for the Teacher	- Whole activity in Snap!: Basic: https://snap.berkeley.edu/project?user=zapusek&project=chameleon_simple Full: https://snap.berkeley.edu/project?user=zapusek&project=chameleon - Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena. - Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.
Resources/materials for the Students	- Template in Snap!: https://snap.berkeley.edu/project?user=zapusek&project=chameleon_template - Half-baked activity in Snap!: https://snap.berkeley.edu/project?user=zapusek&project=chameleon_half_baked - Instructions for student (C4G6_InstructionsForStudent.docx)

Learning Scenario 7 - Helping Prince and Princess to find their animals

Learning Scenario Title	Helping Prince and Princess to find their animals
Previous programming experience	Adding text for the sprite Object movement with arrow keys using events Using conditional for *object is touching* for object state Using events
Learning Outcomes	General learning outcomes: ● Conditionals for *object is touching* certain color ● Moving to coordinates ● Pen up, pen down ● Pen color Specific learning outcomes oriented on algorithmic thinking: ● Student uses if sentence for object state and puts the object back, if touching certain color ● Student sets starting x and y coordinates for sprite ● Student uses pen up and pen down for drawing a line / path ● Student changes pen color depending on the pair he is connecting ● Student realizes that at the beginning he has to clear all previous paths
Aim, Tasks and Short Description of Activities	Short description: Girls has to help the Princess to find her cat and the Prince to find his dog. She does that by going to the Princess and showing her, with drawing a line, the way to her cat; similar the Girl shows the Prince the way to his dog. On this way the Girl has to avoid the meeting between animals so their paths may not cross. Tasks: In the first step students have to choose the appropriate background (a maze). They add five sprites in the maze – their sprite (a girl), a princess, a prince, a cat and a dog. Next they program moving with keys (using events) for the girl, where they have to pay attention that the sprite does not walk on the grass. Later they program drawing with a pen and changing pen color with events. They also have to program the starting event, which clears the path and the girl gives the instructions.

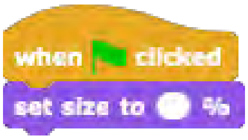
	Aim: Students will be introduced into drawing with key movement. Beside that they will learn how to use conditionals to prevent the sprite moving all across the screen.
Duration of Activities	30 min
Learning and Teaching Strategy and Methods	Active learning, game-design based learning, problem solving
Teaching Forms	Frontal teaching Individual work
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) It is initially given to the students: ● Background ● Girl sprite ● Movement code for one direction The girl decides to help the Princess to find her cat and the Prince to find his dog by showing (drawing) the path to their animals. To avoid confusion, the paths should be different colors and may not crossed. 

[Step 1]

We ask students to edit the scene background – a maze. For implementing "if touching color" either the background (grass) has to be monochrome or the path has to have a monochrome frame, like in our case. To avoid those "problems" with finding appropriate background we give them this background.

[Step 2]

Students already have the girl sprite at the beginning. They need to find another four sprites and put them in the maze. For all sprites they have to set the appropriate size (which is smaller then the width of the paths in the maze. For each sprite they use the code:

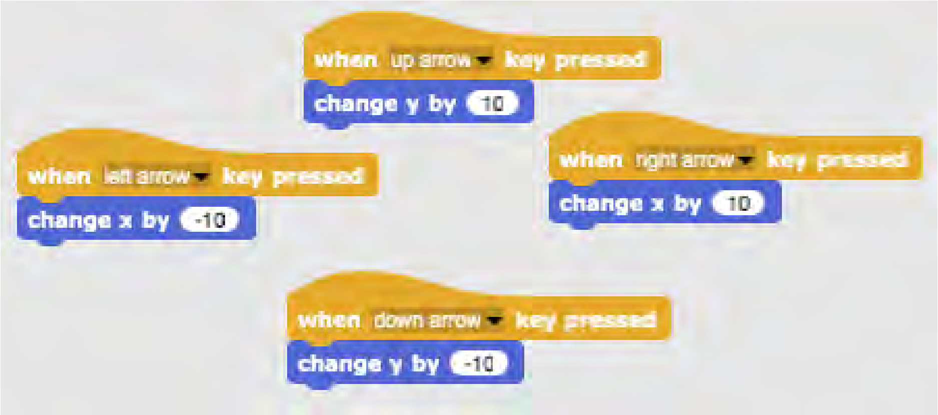


```
when 🟩 clicked
set size to ( ) %
```

Recommended size for the girl is 8%, other sprites can be bigger.

[Step 3]

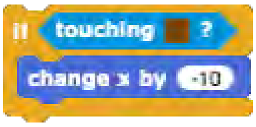
After that they have to make the girl's movement in four directions using keys. We assume that they already know how to do this from previous activities. Anyway, we give them the code for one direction, which helps them to make another three.



```
when up arrow key pressed
change y by 10

when left arrow key pressed
change x by -10

when right arrow key pressed
change x by 10

when down arrow key pressed
change y by -10
```

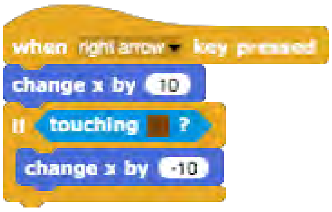
[Step 4]

In the next step they have to prevent the girl's movement across the meadow. They do this by adding a conditional block if touching brown color. If the girl is touching the brown color (end of path), she moves for 10 steps back. We don't see those two steps and it's like the girl stays at the same position. This is a code for moving to the right, so 10 steps back means changing x by -10.



```
if touching [■] ?
  change x by -10
```

They add this code under the previous code, e.g. for the right arrow:

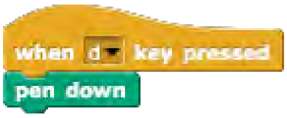


```
when right arrow key pressed
change x by 10
if touching [■] ?
  change x by -10
```

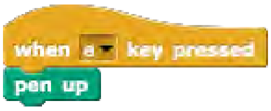
Similar needs to be done for other three directions.

[Step 5]

Next they program drawing. They do this by *pen up* and *pen down* blocks using events *when key pressed*.



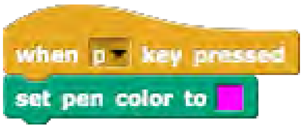
```
when d key pressed
pen down
```



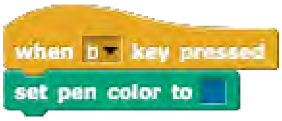
```
when e key pressed
pen up
```

When the key "D" is pressed and the girl moves, she draws a line. When the key "E" is pressed, the drawing stops.

Similar they set pen color by pressing the key.



```
when p key pressed
set pen color to [■]
```



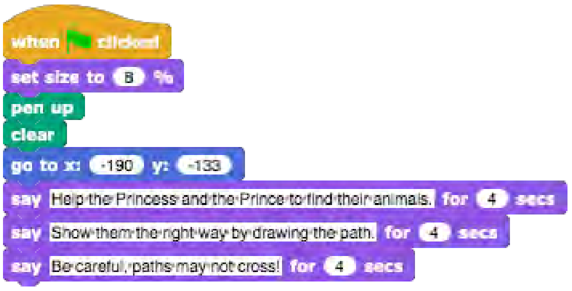
```
when b key pressed
set pen color to [■]
```

[Step 6]

Finally they program *when clicked green flag* event, where students add some instructions which the girl tells at the beginning.

When playing the game, stop it and play it again, students will see that is good to add following blocks: *pen up* (in case it stayed down from the previous playing), *clear* (clears the path from previous playing) and *go to x, y* (the girl always starts at these coordinates, which are inside the path and not on the grass).

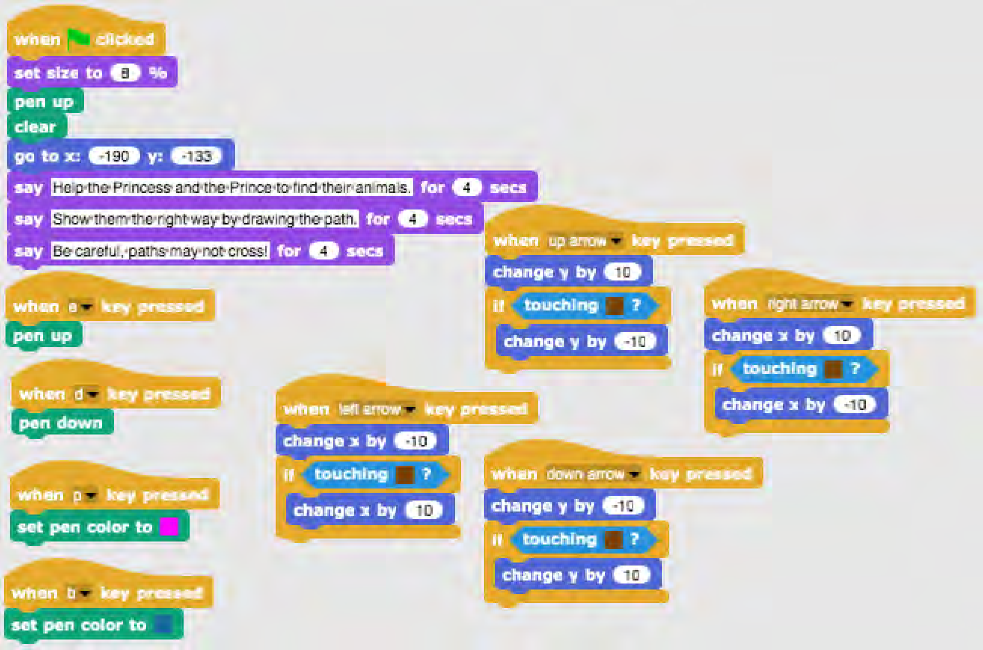
To determine the starting coordinates for the girl, we grab a girl with the mouse and drop her where we want her to start. Then we click on *Motion* blocks, where we can find *x position* and *y position*. By clicking on *x position* we find out the x position of the girl, similar with y.



```
when ⚑ clicked
set size to 8 %
pen up
clear
go to x: -190 y: -133
say Help the Princess and the Prince to find their animals. for 4 secs
say Show them the right way by drawing the path. for 4 secs
say Be careful, paths may not cross! for 4 secs
```

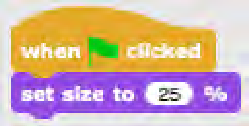
[Final Code]

Girl



```
when 🏴 clicked
set size to (8) %
pen up
clear
go to x: (-190) y: (-133)
say [Help the Princess and the Prince to find their animals.] for (4) secs
say [Show them the right way by drawing the path.] for (4) secs
say [Be careful, paths may not cross!] for (4) secs

when [a ▾] key pressed
pen up

when [d ▾] key pressed
pen down

when [p ▾] key pressed
set pen color to [■]

when [b ▾] key pressed
set pen color to [■]

when [up arrow ▾] key pressed
change y by (10)
if <touching [■] ?>
    change y by (-10)

when [right arrow ▾] key pressed
change x by (10)
if <touching [■] ?>
    change x by (-10)

when [left arrow ▾] key pressed
change x by (-10)
if <touching [■] ?>
    change x by (10)

when [down arrow ▾] key pressed
change y by (-10)
if <touching [■] ?>
    change y by (10)
```

E.g. Princess



```
when 🏴 clicked
set size to (25) %
```

[Additional tasks]

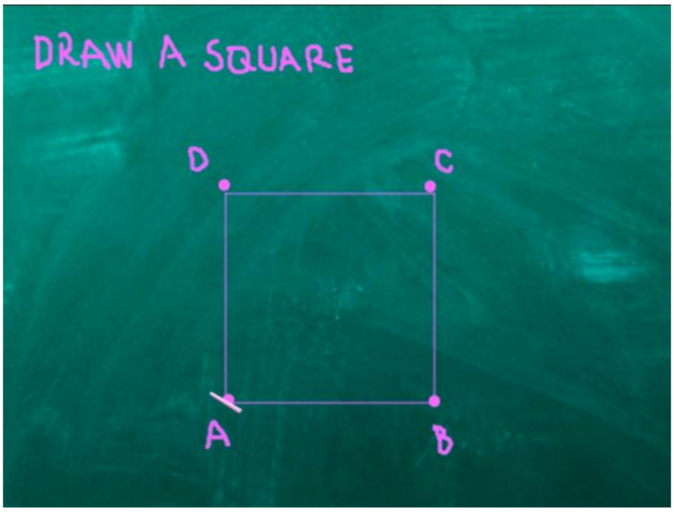
Students can add additional tasks according to their wishes or they can follow the tasks below:

- Set starting coordinates for the Prince and the Princess and write a code for their movement. Set the appropriate size for them. They should draw a path to their animals.
- Add another sprite (animal) for the girl.
- Each sprite should draw with a different color.
- Adjust the initial instructions.
- Add instructions for moving a sprite and drawing by clicking a sprite. E.g. the Princess says: “You move me with pressing the keys W, S, A and D. I draw the path by pressing the key 3. I stop drawing by pressing the key 4. Help me to find my cat!”

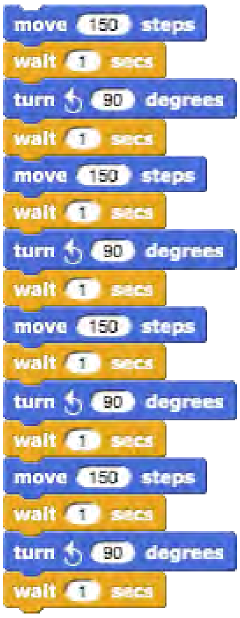
Tools and Resources for the Teacher	● Whole activity in Snap!: https://snap.berkeley.edu/project?user=mateja&project=Helping%20Prince%20and%20Princess%20to%20find%20their%20animals ● Activity in Snap! with additional tasks (possible solution): https://snap.berkeley.edu/project?user=mateja&project=Helping%20Prince%20and%20Princess%20to%20find%20their%20animals%20%2B%20Add.%20Task ● Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena. ● Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.
Resources/materials for the Students	● Half-baked activity in Snap!: https://snap.berkeley.edu/project?user=mateja&project=Helping%20Prince%20and%20Princess%20to%20find%20their%20animals%20-%20Part ● Instructions for student (C4G7_InstructionsForStudent.docx)

Learning Scenario 8 - Drawing with a chalk

Learning Scenario Title	Drawing with a chalk
Previous programming experience	Adding text for the sprite Drawing with pen (pen up, pen down, set color) Moving with steps Using loops Using events
Learning Outcomes	General learning outcomes: ● Loop repeat ● Turning for 90 degrees ● Point in direction ● Changing background Specific learning outcomes oriented on algorithmic thinking: ● Student uses loop repeat when the same blocks repeat 2/4 times ● Student uses turning for 90 degrees when drawing different shapes (square, rectangle, “T” letter) ● Student understands the meaning point in direction 90 ● Student knows how to change background with combination of an event when a key is pressed
Aim, Tasks and Short Description of Activities	Short description: The player gets three different backgrounds and has to connect dots into three different shapes – a square, a rectangle and a “T” letter. Tasks: Students choose the “boardS” background and start with drawing a square. Their starting position is the dot “A”. When drawing a square, they repeat certain steps 4 times, so instead of writing the same code 4 times, they can use a loop *repeat 4* times. Then they draw a rectangle, also with using a loop repeat, this time *repeat 2* times. In their last task they have to connect dots in a shape of letter “T”, where they have to find out the number of steps. They can use loop *repeat* where possible.

	Aim: Students will be introduced into drawing different shapes with a code. They will learn to use loop repeat for shorten the code and to change a background.
Duration of Activities	60 min
Learning and Teaching Strategy and Methods	Active learning, game-design based learning, problem solving
Teaching Forms	Frontal work Individual work / Work in pairs
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) It is initially given to the students: ● Three backgrounds with all the dots they have to connect ● Chalk sprite The chalk wants to draw a square, a rectangle and to connect dots in a shape of letter "T" but it doesn't know how to move and how to turn. Write a code and show the chalk how to do it! [Step 1]  Students starts with this background. They write a code for drawing a square. Starting from the dot "A", they move X steps to the dot "B", turn

90 degrees on the left, move X steps to the dot “C”, turn 90 degrees on the left, move X steps to the dot “D”, turn 90 degrees on the left, move X steps to the dot “A” (and turn 90 degrees on the left).

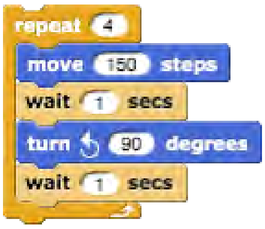


```
move 150 steps
wait 1 secs
turn ↺ 90 degrees
wait 1 secs
move 150 steps
wait 1 secs
turn ↺ 90 degrees
wait 1 secs
move 150 steps
wait 1 secs
turn ↺ 90 degrees
wait 1 secs
move 150 steps
wait 1 secs
turn ↺ 90 degrees
wait 1 secs
```

Using *turn 90 degrees* is the easiest way, since we can always use turning for 90 degrees (it only depends if we want to turn left or right). Using *point in direction 0, 90, 180, -90* is another option, but it’s a bit more complicated because we have to separate 4 possibilities and we can not use a loop *repeat*.

Wait 1 secs block is added just to see the drawing / all steps. Without this block the whole code happens in a second. Students should try it without this block to understand its meaning.

We ask student how would they shorten the code, if possible. Is there some part than repeats? The answer is yes. Instead of writing the same code 4 times, in programming we use loop *repeat*.

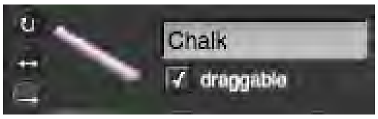


```
repeat 4
    move 150 steps
    wait 1 secs
    turn ↺ 90 degrees
    wait 1 secs
```

If we want to actually see what we draw, we have to put a block *pen down* before the *repeat* loop.

pen down

If we want the chalk is not rotating when turning, we click on *don't rotate* in direction block.

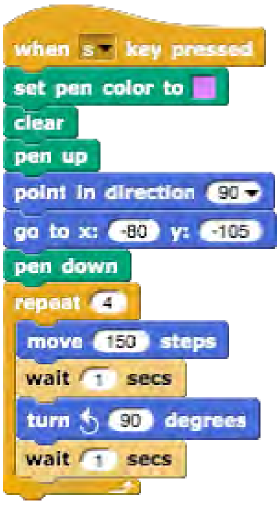


Chalk

☑ draggable

[Step 2]

For activating the code, students use the event block, e. g. *when S key is pressed*. They can also *set pen color*, and, like they already know from the previous activities, following blocks: *pen up* (in case it stayed down from the previous playing), *clear* (clears the drawing from previous playing) and *go to x, y* (that the chalk always starts at these coordinates). Sometimes happens that we stop the program during the play and a sprite is then rotated in "a strange direction". This is a problem when starting a game again, if a sprite is rotated wrong, it will go for example down and not on the right on the first step. To avoid this problem, we add a block *point in direction 90*.

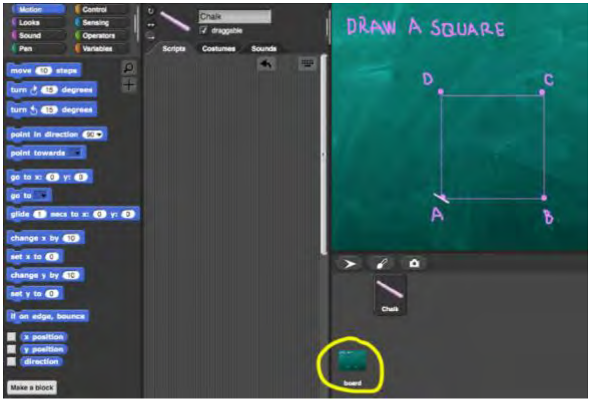


```
when s key pressed
set pen color to
clear
pen up
point in direction 90
go to x: -80 y: -105
pen down
repeat 4
    move 150 steps
    wait 1 secs
    turn ↻ 90 degrees
    wait 1 secs
```

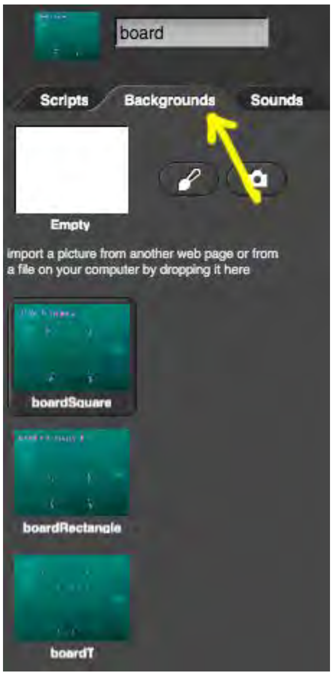
[Step 3]

After drawing a square, we want to draw a rectangle. This means we have to change the background. We will do this with two steps:

a) We click on the background (named *board*, on the right side of the screen).

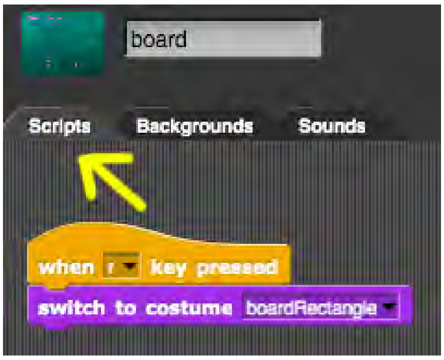


Clicking on *Backgrounds* we can see all three needed backgrounds (*boardSquare, boardRectangle, boardT*), already prepared for this activity.



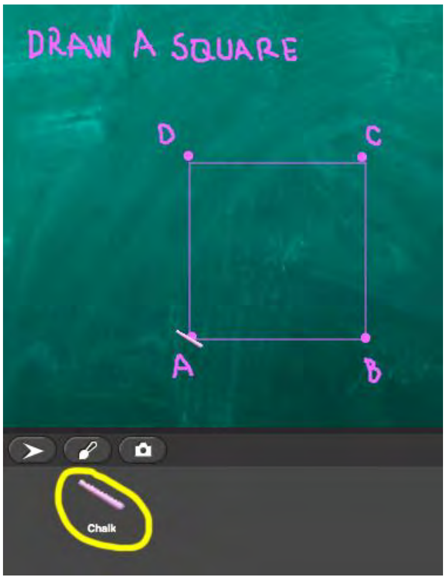
import a picture from another web page or from a file on your computer by dropping it here

To write a code students have to click on *Scripts*. To program changing background they choose an event block *when R key pressed* and then *switch to costume boardRectangle*.

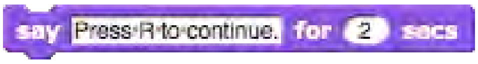


```
when r key pressed
switch to costume boardRectangle
```

b) We click back on the chalk.

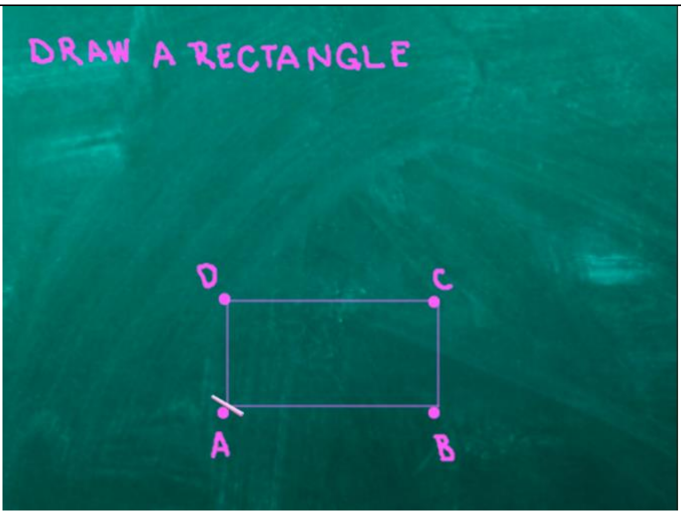


Under the code from [Step 2] students add a block, where they'll tell a player what to do to change the background, which is, press the key "R".



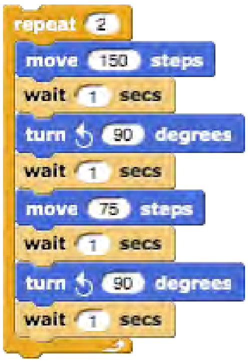
```
say Press R to continue. for 2 secs
```

[Step 4]



After pressing the “R” key, background changes to this one. Similar to before, they need to connect dots and draw a rectangle. Students can copy the previous blocks of code and correct them so the program will draw a rectangle.

They change the loop *repeat*. Now, this loop will repeat 2 times.

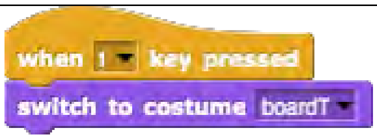


```
repeat 2
    move 150 steps
    wait 1 secs
    turn ↺ 90 degrees
    wait 1 secs
    move 75 steps
    wait 1 secs
    turn ↺ 90 degrees
    wait 1 secs
```

[Step 5]

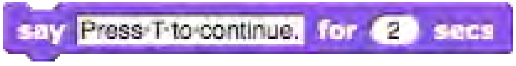
After drawing a rectangle, students will connect dots in a shape of letter “T”. This means they have to change the background, so in this step they actually repeat the [Step 3], they just change the letter (“T”) and costume (boardT):

a) They click on the background (named *board*, on the right side of the screen), where they write a code for changing background. They will do this with *when T key pressed* and then *switch to costume boardT*.



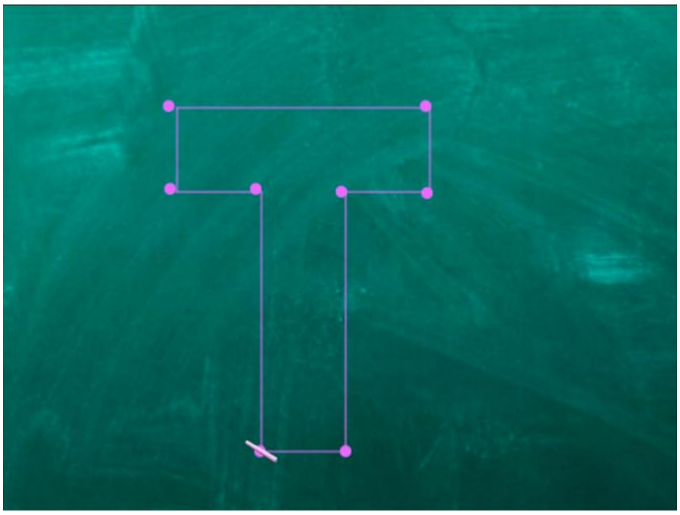
```
when t key pressed
switch to costume boardT
```

b) They click back on the chalk and under the code from [Step 4] add a block, where they'll tell a player what to do to change the background, which is, press the key “T”.



```
say Press T to continue. for 2 secs
```

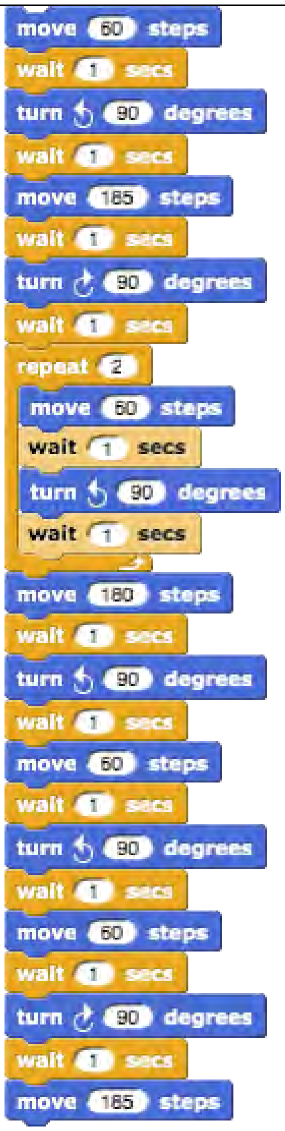
[Step 6]



After pressing the “T” key, background changes to this one. Similar to before, they need to connect dots and draw a letter “T”. Students can copy the previous blocks of code and correct them.

Students will have to change the starting coordinates, which are not the same as before. They already know how to determine the right coordinates from previous activity.

Then they write a code for drawing a letter “T”. They have to find out the number of steps. One possible solution is:

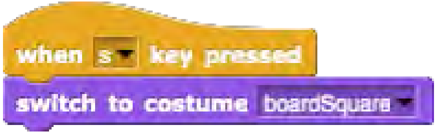


```
move 60 steps
wait 1 secs
turn ↺ 90 degrees
wait 1 secs
move 185 steps
wait 1 secs
turn ↻ 90 degrees
wait 1 secs
repeat 2
    move 60 steps
    wait 1 secs
    turn ↺ 90 degrees
    wait 1 secs
move 180 steps
wait 1 secs
turn ↺ 90 degrees
wait 1 secs
move 60 steps
wait 1 secs
turn ↺ 90 degrees
wait 1 secs
move 60 steps
wait 1 secs
turn ↻ 90 degrees
wait 1 secs
move 185 steps
```

[Step 7]

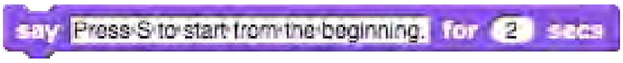
Since we changed the background, we can not return to the first background to draw a square. So students will have to add one last code. They repeat [Step 3/5].

a) They click on the background (named *board*, on the right side of the screen), where they write a code for changing background. They will do this with *when S key pressed* and then *switch to costume boardSquare*.



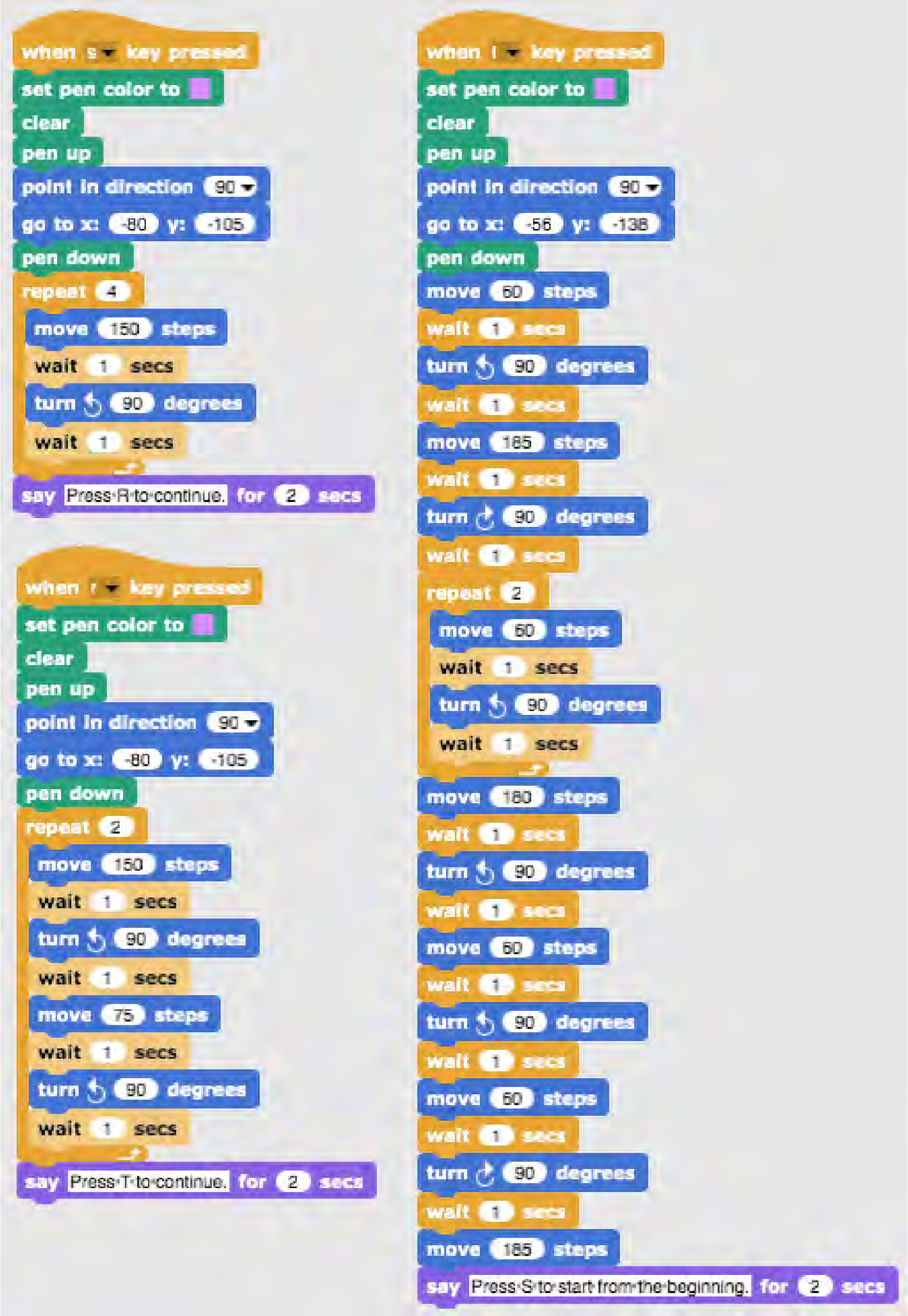
```
when s key pressed
switch to costume boardSquare
```

	b) They click back on the chalk and under the code from [Step 6] add a block, where they'll tell a player what to do to change the background, which is, press the key "S".



```
say [Press S to start from the beginning.] for (2) secs
```

[Final Code]



```
when [s] key pressed
set pen color to [ ]
clear
pen up
point in direction (90)
go to x: (-80) y: (-105)
pen down
repeat (4)
    move (150) steps
    wait (1) secs
    turn ↺ (90) degrees
    wait (1) secs
say [Press R to continue.] for (2) secs
```

```
when [r] key pressed
set pen color to [ ]
clear
pen up
point in direction (90)
go to x: (-80) y: (-105)
pen down
repeat (2)
    move (150) steps
    wait (1) secs
    turn ↺ (90) degrees
    wait (1) secs
    move (75) steps
    wait (1) secs
    turn ↺ (90) degrees
    wait (1) secs
say [Press T to continue.] for (2) secs
```

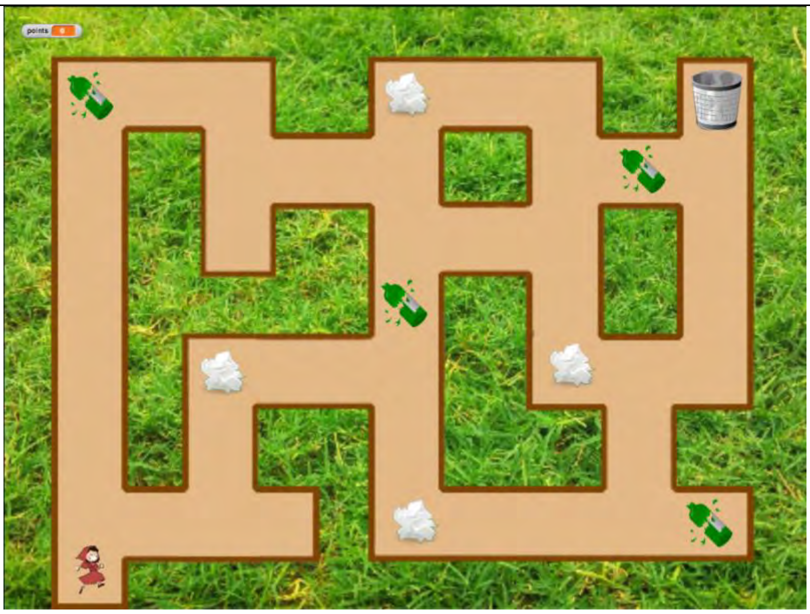
```
when [t] key pressed
set pen color to [ ]
clear
pen up
point in direction (90)
go to x: (-56) y: (-138)
pen down
move (60) steps
wait (1) secs
turn ↺ (90) degrees
wait (1) secs
move (185) steps
wait (1) secs
turn ↻ (90) degrees
wait (1) secs
repeat (2)
    move (60) steps
    wait (1) secs
    turn ↺ (90) degrees
    wait (1) secs
move (180) steps
wait (1) secs
turn ↺ (90) degrees
wait (1) secs
move (60) steps
wait (1) secs
turn ↺ (90) degrees
wait (1) secs
move (60) steps
wait (1) secs
turn ↻ (90) degrees
wait (1) secs
move (185) steps
say [Press S to start from the beginning.] for (2) secs
```

	[Additional tasks] Students can add additional tasks according to their wishes or they can follow the tasks below: ● Add a new background and draw some dots. ● Write a code that connects the dots. You can draw a background or you can use a given one.
Tools and Resources for the Teacher	● Whole activity in Snap!: https://snap.berkeley.edu/project?user=mateja&project=Drawing%20with%20a%20chalk ● Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena. ● Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.
Resources/materials for the Students	● Half-baked activity in Snap!: https://snap.berkeley.edu/project?user=mateja&project=Drawing%20with%20a%20chalk%20-%20Part ● Instructions for student (C4G8_InstructionsForStudent.docx)

Learning Scenario 9 - Picking up trash and cleaning the park

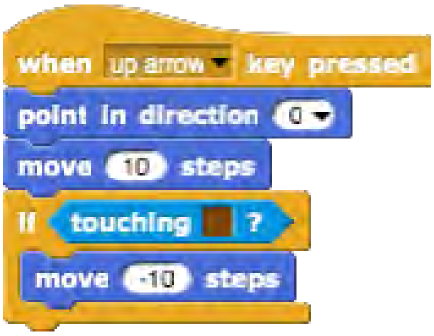
Learning Scenario Title	Picking up trash and cleaning the park
Previous programming experience	Setting starting coordinates Setting size for sprite Adding text for sprite Object movement with arrow keys using events Using conditional *object is touching* for object state
Learning Outcomes	General learning outcomes: ● Variables ● Show and hide sprites ● Duplicate sprites ● Duplicate block of code ● Conditionals Specific learning outcomes oriented on algorithmic thinking: ● Student uses variable for counting collected waste ● Student uses hide sprite when a sprite is touched and show sprite at the beginning ● Student knows how to duplicate a sprite (from one bottle to e.g. 4 bottles) ● Student knows how to duplicate a block of code (from a bottle sprite to a paper sprite) ● Student knows how to use conditionals for checking if a sprite is shown and if all trash is picked up
Aim, Tasks and Short Description of Activities	Short description: The park is full of trash and the girl decides to clean it up. When she collects all the trash she throws it in the trash can. Tasks: Students start with setting starting coordinates for the girl. They game ends when the girl collects all the trash and puts it in the bin. To do this, students will have to use variables for counting points (1 collected trash = 1 point). When the girl touches the trash, she picks it up, the trash hides and number of points increases for 1. When she picks up all the trash, she goes to the trash can. If she does not pick up

	all trash and goes to the trash can earlier, the trash can says to come back when she picks up all the trash. **Aim: Students will learn how to use variables and how to duplicate a block of code or even a whole sprite.**
Duration of Activities	45 min
Learning and Teaching Strategy and Methods	Active learning, game-design based learning, problem solving
Teaching Forms	Frontal teaching Individual work
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) It is initially given to the students: ● Background ● Girl sprite (with the movement code), bottle sprite, paper sprite and trash can sprite The girl wants to take a walk and enjoy her day in the park. When she comes there, she sees the park is full of trash. She decides to pick up all the trash. When she does that, she can finally lay down and enjoy the sunny day in a clean park.

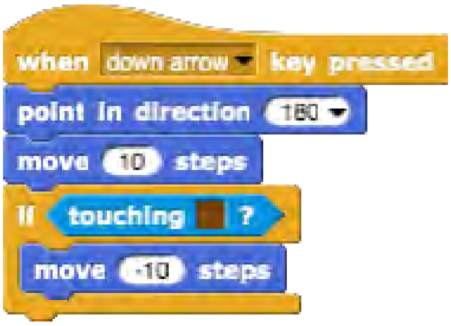


[Step 1]

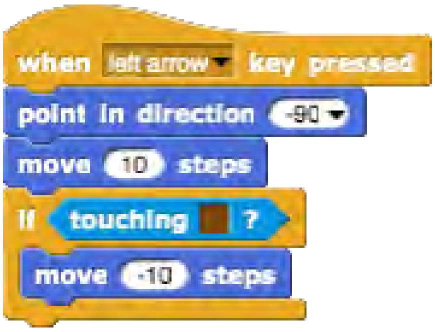
The background is given and also the girl sprite with a code for movement with keys and conditional for touching the brown line.



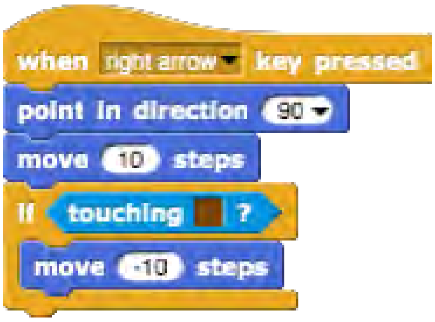
```
when up arrow key pressed
point in direction 0
move 10 steps
if touching ? 
  move -10 steps
```



```
when down arrow key pressed
point in direction 180
move 10 steps
if touching ? 
  move -10 steps
```



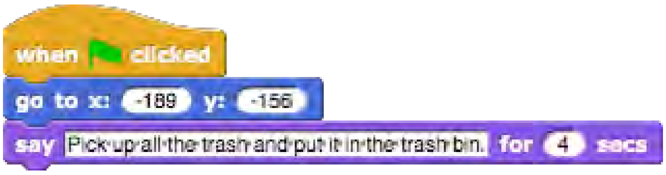
```
when left arrow key pressed
point in direction -90
move 10 steps
if touching ? 
  move -10 steps
```



```
when right arrow key pressed
point in direction 90
move 10 steps
if touching ? 
  move -10 steps
```

Students have to set the starting coordinates for the girl with *go to x, y* block. The coordinates are chosen on their own, it’s only important they are on the path. Students already know how to set the coordinates from previous activities. They also add some instructions.

E.g.:



```
when [flag] clicked
go to x: -189 y: -156
say [Pick up all the trash and put it in the trash bin.] for 4 secs
```

[Step 2]

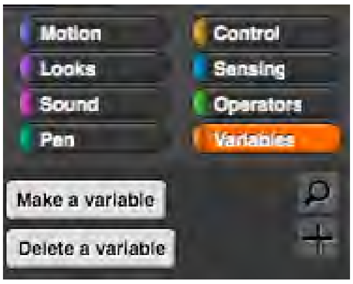
For counting number of trash the girl picked up, we will use variables.

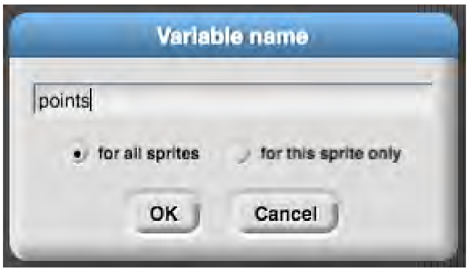
What is a variable?

A variable is like a box where we store some information.

In our case, we can see our variable as a box, named points. When the girl picks up a trash, a trash is stored in a variable *points*. This variable counts how many trash did the girl pick.

How do we make a variable?





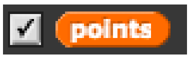
Variable name

points

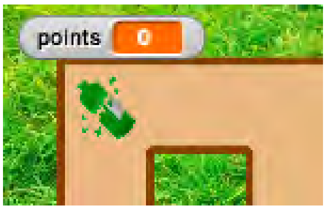
● for all sprites ○ for this sprite only

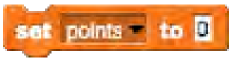
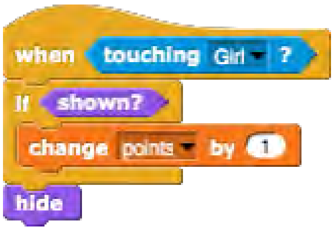
OK Cancel

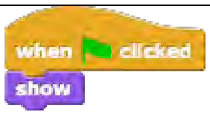
We select an orange block *Variables*, then click on button *Make a variable*, write a *Variable name* and click OK. Then a block *points* appears.



If the box is checked, the variable with its value will be visible on the screen:



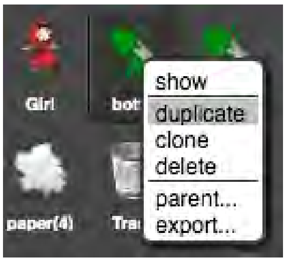
	At the beginning of the game, the value of the variable has to be 0, since there is no trash picked up. Under the code from [Step 1] student adds a block *set* __ *to 0.* By clicking on drop down menu they choose appropriate variable, which is *points*.  set points to 0 [Step 3] Students write a code for a bottle. The idea is that the sprite disappears (which means hide) when it touches the girl. So the code will start when the sprite is touching the girl. Then we have to think in which case she picks up the trash. If we said the trash hides when it's picked up, we can only pick it up if it's still there = is shown. If the sprite (bottle) is still there, we pick it up "and put it in the variable box". Before we had 0 elements in the variable *points*, now we have 1. We can see that by picking up trash we change number of variable (*points*) by 1, which is, increase by 1. When the trash is picked up, we hide it.  when touching Girl ? if shown? change points by 1 hide Now we can test if our code is correct. We click on green flag and pick up the bottle. The bottle has to disappear and number of points has to be 1. Then we want to play the game again and we click again on the green flag. What happens? Where is the bottle now? The bottle is hidden, we hid it before. So on the beginning of the game, we have to program that the bottle is shown. We do this by selecting block *show*.



```
when clicked
show
```

[Step 4]

Now students want to have more bottles in their game so they can easily duplicate their sprite. They right click on the sprite and choose duplicate.

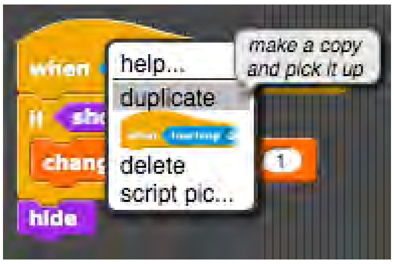


Now they just click with a mouse on the new bottle and drag it somewhere inside the maze.

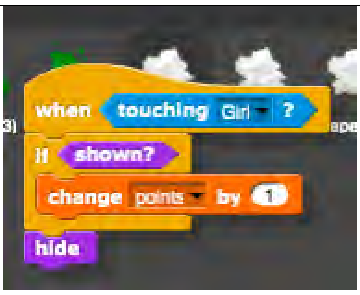
They can repeat this step and duplicate the bottle again.

[Step 5]

Now students want to have the same code for the paper sprite. They can duplicate the code of the bottle by right clicking on the block of code:



And drop it in the paper sprite by clicking with the mouse on the paper sprite.



```
when touching Girl ?
if shown?
    change points by 1
hide
```

They repeat this step to duplicate the block of code *when green flag clicked – show*.

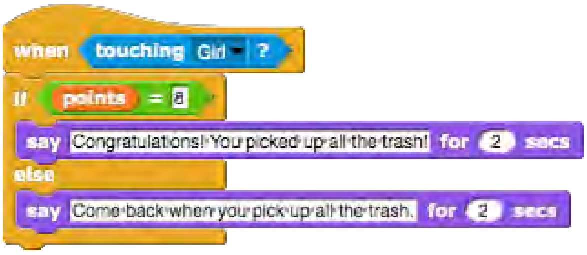
They can also repeat [Step 4] and duplicate the whole paper sprite to have more paper trash in the maze.

[Step 6]

The last thing students have to do is write a code for the trash can. The sprite trash can is already given, they can move It wherever inside the maze.

Also this code will activate when the girl touches it.

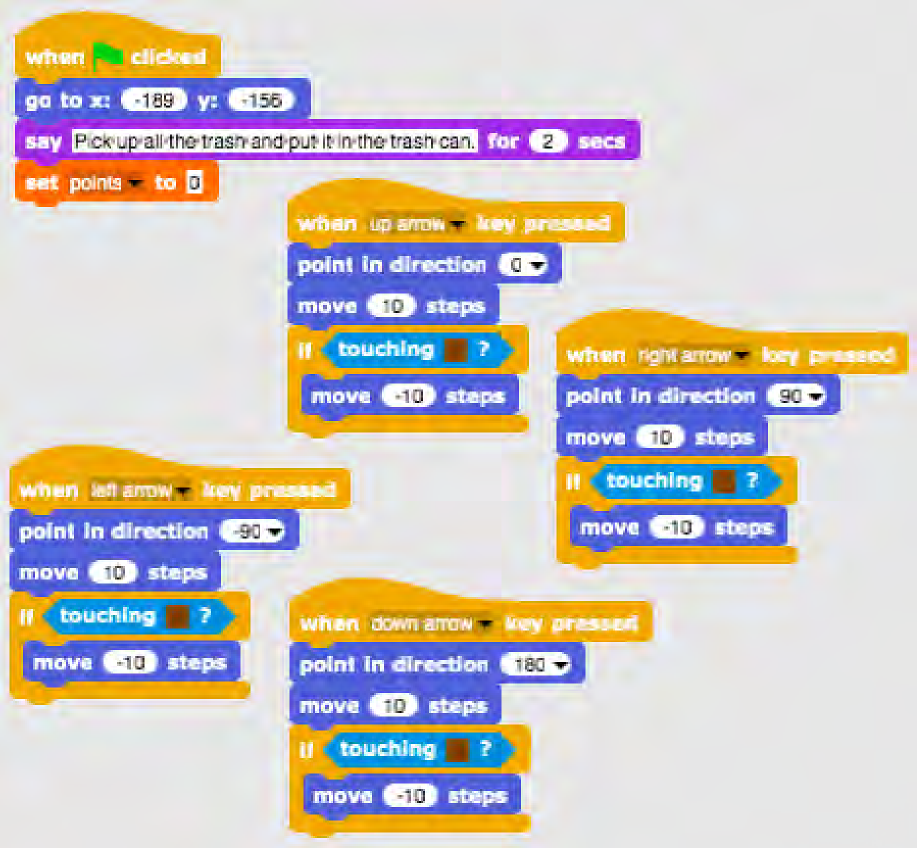
The trash can will have to check if all trash is picked up. Thanks to variable *points*, this will be easy to do. Let's say we have 8 trash sprites in the game, so students have to check if the number of points is equal to 8. If it is, that means all trash is picked up, otherwise is not. They will use if statement to program this and they will add some text to tell the player if he picked up all the trash or not.



```
when touching Girl ?
if points = 8
    say Congratulations! You picked up all the trash! for 2 secs
else
    say Come back when you pick up all the trash. for 2 secs
```

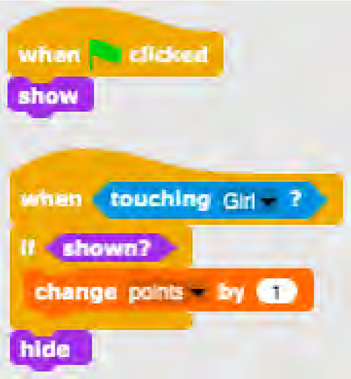
[Final Code]

Girl



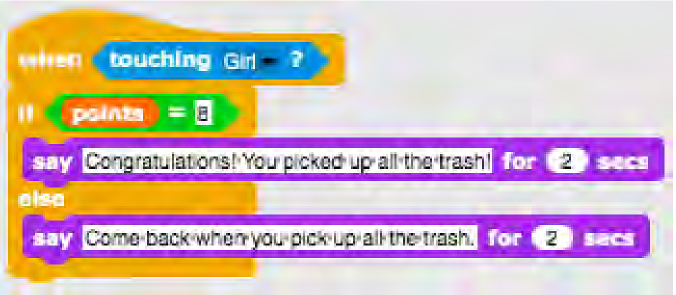
```
when ⚑ clicked
go to x: -189 y: -156
say Pick up all the trash and put it in the trash can. for 2 secs
set points to 0

when up arrow key pressed
point in direction 0
move 10 steps
if touching ■ ?
    move -10 steps

when right arrow key pressed
point in direction 90
move 10 steps
if touching ■ ?
    move -10 steps

when left arrow key pressed
point in direction -90
move 10 steps
if touching ■ ?
    move -10 steps

when down arrow key pressed
point in direction 180
move 10 steps
if touching ■ ?
    move -10 steps
```

Bottles / Papers



```
when ⚑ clicked
show

when touching Girl ?
if shown?
    change points by 1
hide
```

Trash can



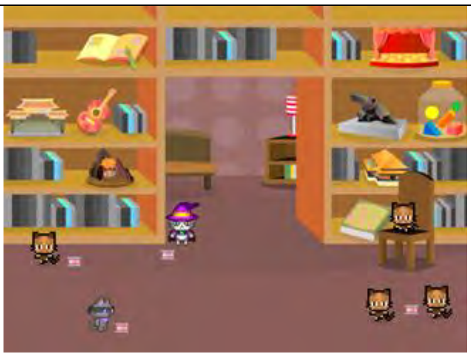
```
when touching Girl ?
if points = 8
    say Congratulations! You picked up all the trash! for 2 secs
else
    say Come back when you pick up all the trash. for 2 secs
```

	[Additional tasks] Students can add additional tasks according to their wishes or they can follow the tasks below: ● Add another type of waste (e.g. bio-waste). ● The trash can says e.g. “You picked up X bottles, Y papers and Z watermelons”. ● If a player picks up all the trash, the trash can says: “Congratulations! You picked up all the trash!” ● If a player does not pick up all the trash, the trash can tells him which trash has not been picked up, e.g. “You did not pick up all the bottles. You did not pick up all the watermelons.” and “Come back when you pick up all the trash”.
Tools and Resources for the Teacher	● Whole activity in Snap!: https://snap.berkeley.edu/project?user=mateja&project=Picking%20up%20trash%20and%20cleaning%20the%20park ● Activity in Snap! with additional tasks (possible solution): https://snap.berkeley.edu/project?user=mateja&project=Picking%20up%20trash%20and%20cleaning%20the%20park%20%2B%20Add.%20Task ● Lajovic, S. (2011). *Scratch. Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena. ● Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.
Resources/materials for the Students	● Half-baked activity in Snap!: https://snap.berkeley.edu/project?user=mateja&project=Picking%20up%20trash%20and%20cleaning%20the%20park%20-%20Part ● Instructions for student (C4G9_InstructionsForStudent.docx)

Learning Scenario 10 - Feeding the cats

Learning Scenario Title	Feeding the cats
Previous programming experience	● conditionals (if, if-else blocks) ● printing the text (block say)
Learning Outcomes	General learning outcomes: ● setting and increasing the variable value, ● assigning variable value inside/outside the loop, ● for loop (repeat n times), ● random numbers, ● string concatenation, ● operators: logical, arithmetic, ● input Specific learning outcomes oriented on algorithmic thinking: ● Student recognizes the situation for using repeat n times loop, ● student differentiates between assigning the value in every iteration of the loop and once before the loop. ● student uses input block to get the number from a player, ● student knows how to use arithmetic operators to generate the right answer, ● student uses if - else sentence to check the correctness of player input and gives an appropriate response, ● student know how to use a variable to count correct answers.
Aim, Tasks and Short Description of Activities	Short description: Program a game in which the player will have to perform ten multiplication calculations and count the correct answers. Task: Program the activity in which shelter keeper Martha will repeatedly ask the player for the number of cats she can feed in a certain room. The number depends on the number and size of the bowls. For each room those two numbers have to be assigned randomly. We also have to have a counter that will count the right

	answers. First shelter keeper has to explain the assignment for the player and then the game begins. Game is over when she asks for the number of cats 10 times. Each time she has to give a response if the input number is correct or not. After activity she has to summarize how successful the player was, she tells how many times the player answered correctly and how many times she was wrong. **Students will be introduced to the concept of multiple variable random value assignment inside a loop and how it is different from when we do it outside a loop. They will also learn about how to get, test and count correct player inputs.**
Duration of Activities	45 min
Learning and Teaching Strategy and Methods	active learning, collaborative learning, problem solving
Teaching Forms	frontal teaching individual work / working in pairs / group work
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) Shelter keeper is trying to feed her cats in ten different rooms. In every room there are a random number of bowls (2 to 10), which have different sizes (1 to 5) but inside each room all of the bowls are the same size. The size of the bowl tells how many cats can eat from it, for example if bowl size is 3 that means 3 cats can eat from it. Help find the number of cats she can feed in each room. [Step 1] First we instruct students to design an interesting background for the game. If we want to save time, we can provide it for them.

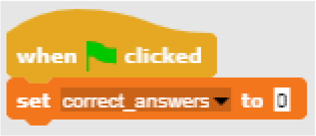


[Step 2]

We have to select a new costume for the default turtle sprite that will represent cat shelter keeper.

[Step 3]

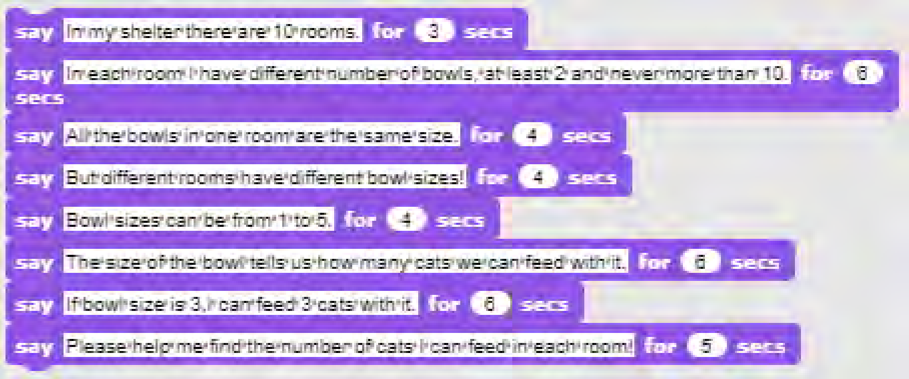
In order to store needed values we need three variables: 1) for storing the number of correct answers, 2) for assigning the random value for number of bowls inside each room (2-10) and 3) for assigning the random value for bowl capacity (1-5). The correct answer counter will have to be set to 0 and the other two do not have to be set before the loop because we will assign them new random values in each iteration of the loop. We also want to count the rooms, but we don’t need a special variable to count it. We are going to use the same variable as in for loop. Its number will be initialized to a value 1 and then increased by 1 for each iteration until value 10 is reached. This replicates the room counting.



```
when ⚑ clicked
set correct_answers ▾ to 0
```

[Step 4]

Next we have to program the instructions for the player. We do that by using Looks/say[string] and wait [n] seconds block.



```
say In my shelter there are 10 rooms. for 3 secs
say In each room I have different number of bowls, at least 2 and never more than 10. for 6 secs
say All the bowls in one room are the same size. for 4 secs
say But different rooms have different bowl sizes! for 4 secs
say Bowl sizes can be from 1 to 5. for 4 secs
say The size of the bowl tells us how many cats we can feed with it. for 6 secs
say If bowl size is 3, I can feed 3 cats with it. for 6 secs
say Please help me find the number of cats I can feed in each room! for 5 secs
```

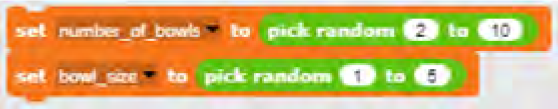
[Step 5]

We discuss with the students what are the actions that will happen in each room and thus be the same. These are commands that will have to be placed inside the loop block to be executed in each iteration of the loop.

First we will have to randomly assign a value (1-10) for the number of bowls and bowl size in that room (1-5). Then we will have to ask a player how many cats we can feed in that room. Her answer will have to be tested for correctness and we will have to give an appropriate response and remember if it was correct (correct answer counter). At the end of each iteration we will also have to increase the room number by 1.

[Step 6]

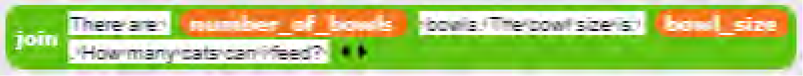
To randomly assign the values for the number of bowls and their size we will use Variables/set [options] value with Operators/pick random [n] to [m].



```
set number_of_bowls to pick random 2 to 10
set bowl_size to pick random 1 to 5
```

[Step 7]

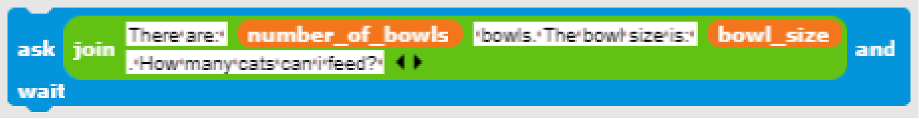
We want to ask the player for the number of cats we can feed inside Sensing/ask [string] and wait block, because otherwise it will be displayed for a certain seconds and then updated with a new line of text. In that way players can easily forget how many bowls/sizes are in the current room. In order to make a string that will be constructed from a combination of text and references to variables we use Operators/join [string1][string2] block. We will have to expand this block so it fits the entire sentence.



```
join [There are: ] (number_of_bowls) [ bowls. The bowl size is: ] (bowl_size) [. How many cats can i feed?]
```

[Step 8]

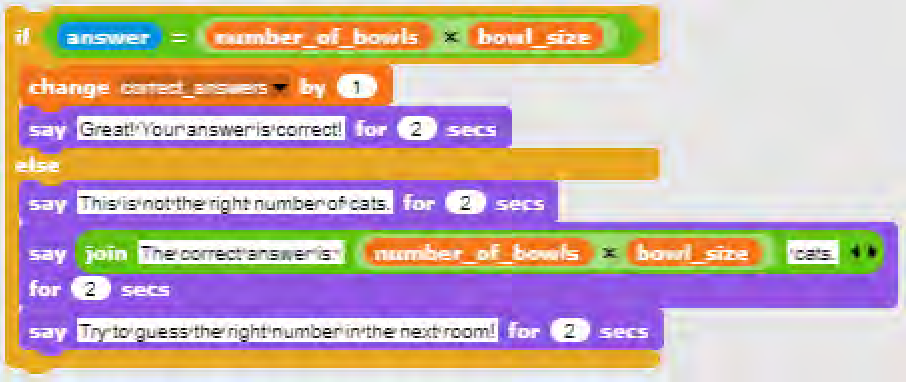
We have to put this long string inside Sense/Ask [string] and wait block in order to get the answer from the player.



```
ask (join [There are: ] (number_of_bowls) [ bowls. The bowl size is: ] (bowl_size) [. How many cats can i feed? ]) and wait
```

[Step 9]

When the player answers we have to check the correctness. There are only two possible situations, the player can be correct or wrong, so we will use If-Else block. The right answer is the value of multiplying the number of bowls with the bowl size. We have to check if the player's answer is equal to that number. If the answer is correct we increase the correct answer counter by 1 and give response. If not, we only give response. We don’t have to count wrong answers because we can calculate it from the correct answer counter.



```
if <(answer) = ((number_of_bowls) × (bowl_size))>
    change [correct_answers ▾] by (1)
    say [Great! Your answer is correct!] for (2) secs
else
    say [This is not the right number of cats.] for (2) secs
    say (join [The correct answer is: ] ((number_of_bowls) × (bowl_size)) [ cats.]) for (2) secs
    say [Try to guess the right number in the next room!] for (2) secs
```

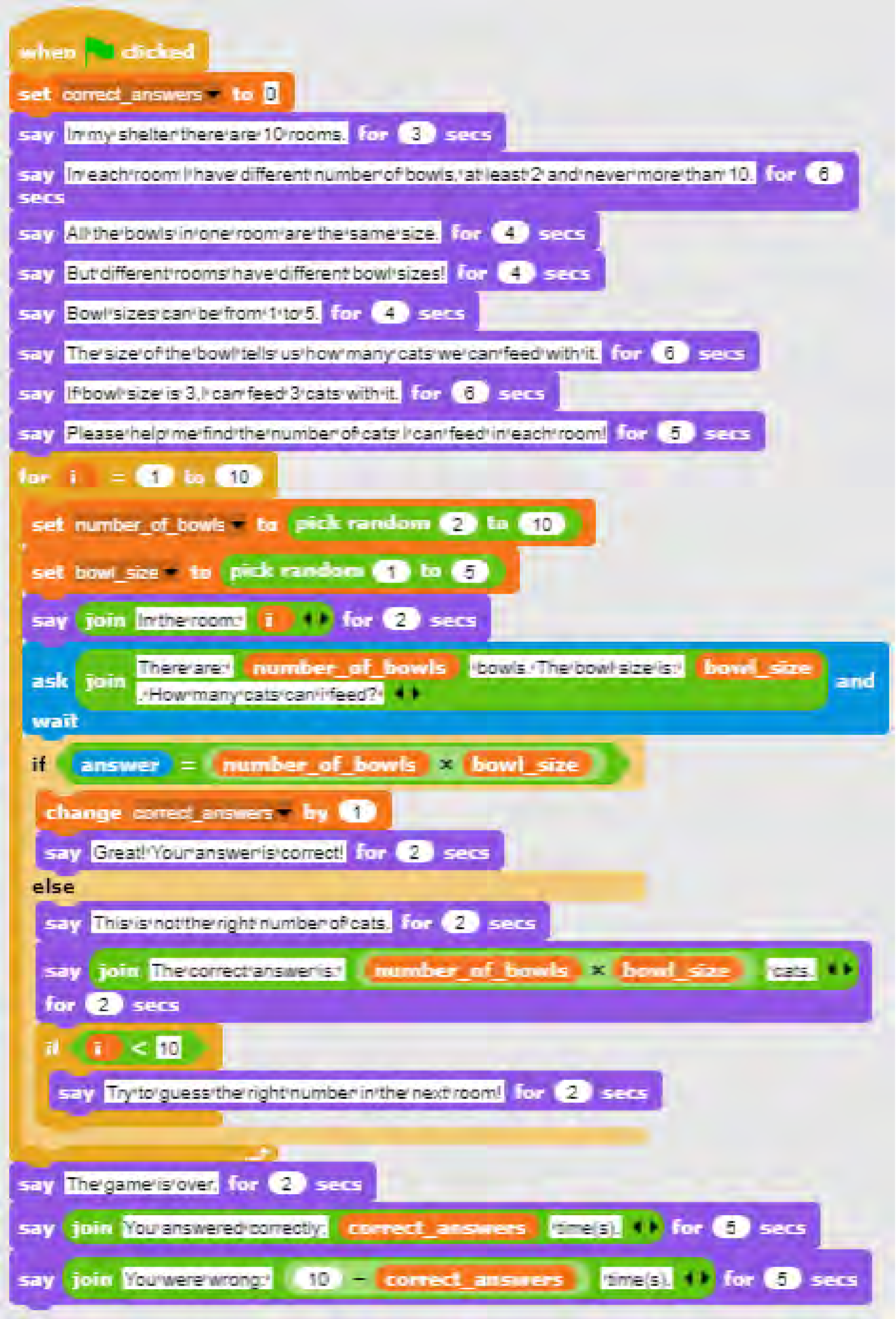
[Step 11]

Now we have to select a loop. As mentioned earlier, it is best to choose for loop because the variable that is used for iterating replicates the counting of rooms.

[Step 12]

When the loop stops, the game is over. We provide the information about player achievement. Number of correct answers is stored in the correct answer counter; the number of wrong answers can be calculated.

[Final code]

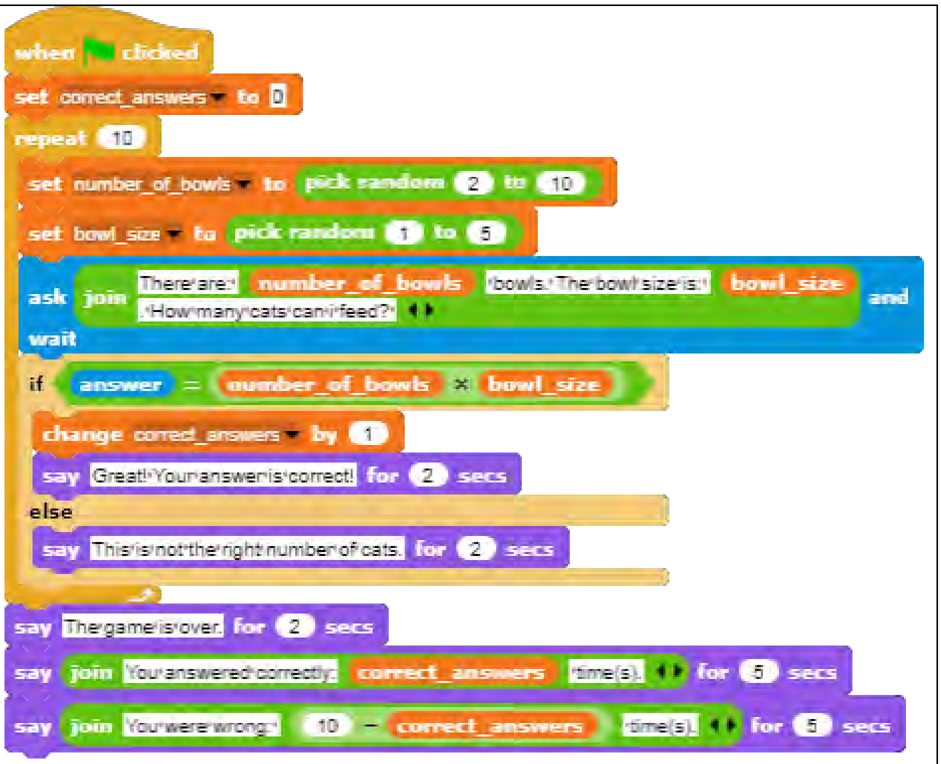


```
when [flag] clicked
set correct_answers to 0
say In my shelter there are 10 rooms. for 3 secs
say In each room I have different number of bowls, at least 2 and never more than 10. for 6 secs
say All the bowls in one room are the same size. for 4 secs
say But different rooms have different bowl sizes! for 4 secs
say Bowl sizes can be from 1 to 5. for 4 secs
say The size of the bowl tells us how many cats we can feed with it. for 6 secs
say If bowl size is 3, I can feed 3 cats with it. for 6 secs
say Please help me find the number of cats I can feed in each room! for 5 secs
for i = 1 to 10
    set number_of_bowls to pick random 2 to 10
    set bowl_size to pick random 1 to 5
    say join In the room: i for 2 secs
    ask join There are: number_of_bowls bowls. The bowl size is: bowl_size . How many cats can I feed? and wait
    if answer = number_of_bowls × bowl_size
        change correct_answers by 1
        say Great! Your answer is correct! for 2 secs
    else
        say This is not the right number of cats. for 2 secs
        say join The correct answer is: number_of_bowls × bowl_size cats. for 2 secs
        if i < 10
            say Try to guess the right number in the next room! for 2 secs
say The game is over. for 2 secs
say join You answered correctly: correct_answers time(s). for 5 secs
say join You were wrong: 10 − correct_answers time(s). for 5 secs
```

[Basic version of the activity]

To save time we can use the basic version of the scenario. In a basic version all the essential concepts are included, other functionalities described above can be used as later upgrades.

<table>
<tr>
<td></td>
<td>



```
when 🏁 clicked
set correct_answers to 0
repeat 10
    set number_of_bowls to pick random 2 to 10
    set bowl_size to pick random 1 to 5
    ask join There are: number_of_bowls bowls. The bowl size is: bowl_size . How many cats can I feed? and wait
    if answer = number_of_bowls × bowl_size
        change correct_answers by 1
        say Great! Your answer is correct! for 2 secs
    else
        say This is not the right number of cats. for 2 secs
say The game is over. for 2 secs
say join You answered correctly: correct_answers time(s). for 5 secs
say join You were wrong: 10 − correct_answers time(s). for 5 secs
```

</td>
</tr>
<tr>
<td><b>Tools and Resources
for the Teacher</b></td>
<td>

- Whole activity in Snap!:
https://snap.berkeley.edu/project?user=zapusek&project=cat_feeding_2
- Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena.
- Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.

</td>
</tr>
<tr>
<td><b>Resources/materials
for the Students</b></td>
<td>

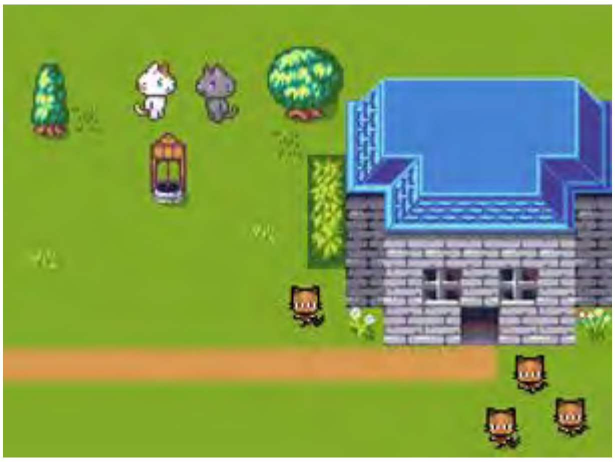
- Template in Snap!:
https://snap.berkeley.edu/project?user=zapusek&project=cat_feeding_template
- Instructions for student (C4G10_InstructionsForStudent.docx)

</td>
</tr>
</table>

Learning Scenario 11 - Guessing the number of cats in a shelter

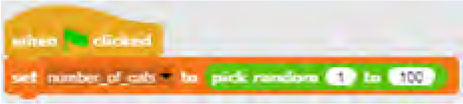
Learning Scenario Title	Guessing the number of cats in a shelter
Previous programming experience	● conditionals (if block) ● printing the text (block say)
Learning Outcomes	General learning outcomes: ● random values, ● variable assignment, ● user input, ● repeat until loop, ● comparison operators, ● counter Specific learning outcomes oriented on algorithmic thinking: ● student assigns random value to the variable, ● student uses input block to get the number from a player, ● student uses repeat until loop to repeatedly ask player to input the number and perform a value testing, ● student performs value testing with if sentence and comparison operators and gives appropriate response, ● student sets the condition of the repeat loop to test if game is over, ● student realizes that she doesn't have to test if the game is over, because it is implicatelly covered in condition, ● student implements a counter for counting player guesses and uses the final value to differentiate between both possible outcomes.
Aim, Tasks and Short Description of Activities	Short description: Program simple game in which at the beginning a random number from 1 to 100 will be randomly assigned to a variable. Player will try to guess it by typing in numbers. She will get a response if the input number will be: more, less or equal the random value.

	Task: Program cat shelter Martha to randomly set the number of cats, ask the player for her or his name and then explain the task for her/him. Next Martha has to greet the player with her/his name and then repeatedly ask for a number. When player inputs her/his guess, she must respond: 1) if the input number is lower than actual number, she says: "the number of cats is higher", 2) if the input number is higher than actual number, she says: "the number of cats is lower", 3) if the input number is correct, she says: "Excellent, you guessed the right number". Program a counter that will count every player try. When the player guesses the right number you have to check if the number of tries is less than 5. In that case the player gets the cat, otherwise not. **Aim: Students will be introduced to repeat until loop and how to set the condition to implicitly track the condition that stops the game. They will also learn how to use variables in different situations: to set a random value, as a counter or to get the players input.**
Duration of Activities	45 min
Learning and Teaching Strategy and Methods	active learning, collaborative learning, problem solving
Teaching Forms	frontal teaching individual work/ working in pairs / group work

Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) Cat shelter keeper Martha wants you to guess the exact number of cats that she has in her shelter. The number can be anywhere between 1 and 100. When the player types the number she answers if current input number is less, more or equal to the right number of cats. If a player guesses the number of cats in less than five tries, she gets the cat, otherwise she is prompted to play again. [Step 1] First task is to make an interesting background for the game. Students can draw it themselves or use free license images from the internet. To save time, we can prepare the background beforehand.  [Step 2] We have to select a new costume for the default turtle sprite that will represent cat shelter keeper.

[Step 3]

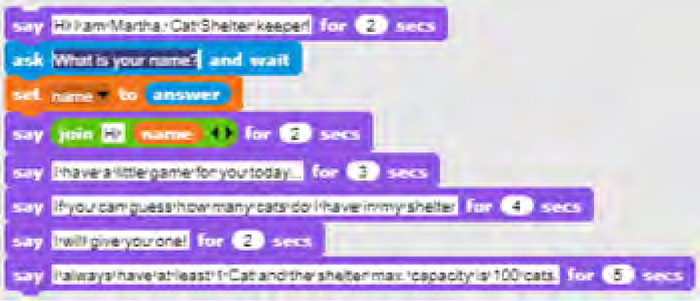
We discuss with students that this game can be interesting for playing it more than once, if the number of cats is randomly set. In order to have that random number available for guess numbers comparisons, we also have to store it in a variable. Variables are now (we assume they do not yet know the concept of lists) the only way to remember a certain value in Snap. This has to happen when the program starts (Event/When green flag is clicked).



```
when ⚑ clicked
set number_of_cats ▾ to pick random 1 to 100
```

[Step 4]

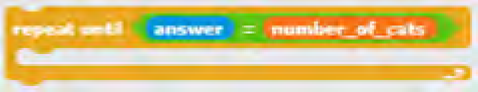
Shelter keeper asks the player for her name in order to greet her. This is done with Sense/ask[string] and wait block. Player answer is stored in a built-in variable named *answer*. In order to greet her, we have to join the string stored in the variable *answer* with some greeting. This is done with Operators/join[string1][string2] block. To display the text, we use Looks/say [string] for n seconds block. We also use those blocks to write instructions for playing the game. We can also emphasize that it is important to be attentive to the duration of displaying the text.



```
say Hi I am Martha, Cat Shelter keeper! for 2 secs
ask What is your name? and wait
set name ▾ to answer
say join Hi name ◂▸ for 2 secs
say I have a little game for you today... for 3 secs
say If you can guess how many cats do I have in my shelter for 4 secs
say I will give you one! for 2 secs
say I always have at least 1 Cat and the shelter max capacity is 100 cats. for 5 secs
```

[Step 5]

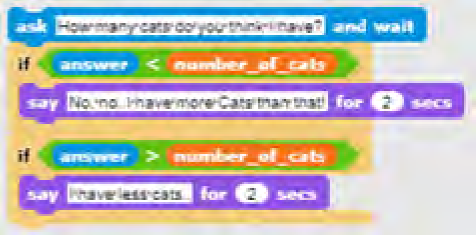
We discuss with students that it is not possible to predict how many times players will have to guess in order to find the right number. She can get very lucky and guess it in her first try, maybe it will take her 5 guesses, or even more, we cannot tell! This is the reason we have to choose the right loop for the given task. Shelter keeper has to repeatedly ask for a number and give an appropriate response until the player guesses the right number. The only loop we can implement the desired execution is repeated until[condition] loop. Condition is relatively easy to see, we have to loop it till the player answer, that is stored in built-in variable answer equals the value stored in *cat_number* variable.



```
repeat until answer = number_of_cats
```

[Step 6]

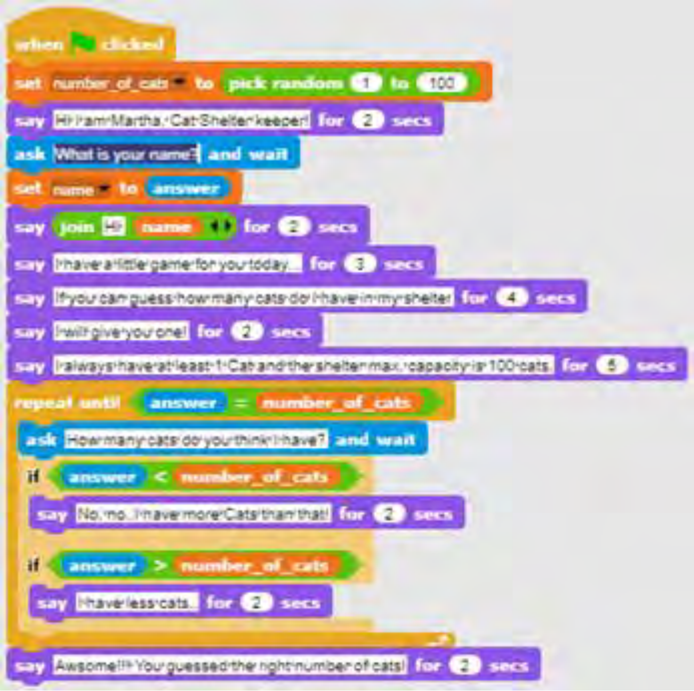
Next, we have to ask students what are the commands that will go into the loop body. What is the activity or commands that will be repeated until the player guesses the right number? First, we have to ask the player to input a number, then we have to make a response based on the value of that number.



```
ask How many cats do you think I have? and wait
if answer < number_of_cats
  say No, no... I have more Cats than that! for 2 secs
if answer > number_of_cats
  say I have less cats. for 2 secs
```

[Step 7]

Last thing to explain or discuss with the students is when this loop will end and what that implies. When player answers will be equal to the number of cats both conditions in the loop body would be false, so the loop will go in the next iteration, checking the loop condition. Condition will be true this time, so the loop will terminate and commands that follow the loop will be executed. To paraphrase, when the loop terminates we know that the player guessed the number right. So now we can respond accordingly.



```
when 🏴 clicked
set number_of_cats ▾ to pick random 1 to 100
say Hi I am Martha, Cat Shelter keeper! for 2 secs
ask What is your name? and wait
set name ▾ to answer
say join Hi name for 2 secs
say I have a little game for you today... for 3 secs
say If you can guess how many cats do I have in my shelter for 4 secs
say I will give you one! for 2 secs
say I always have at least 1 Cat and the shelter max. capacity is 100 cats. for 5 secs
repeat until answer = number_of_cats
    ask How many cats do you think I have? and wait
    if answer < number_of_cats
        say No, no... I have more Cats than that! for 2 secs
    if answer > number_of_cats
        say I have less cats. for 2 secs
say Awsome!!! You guessed the right number of cats! for 2 secs
```

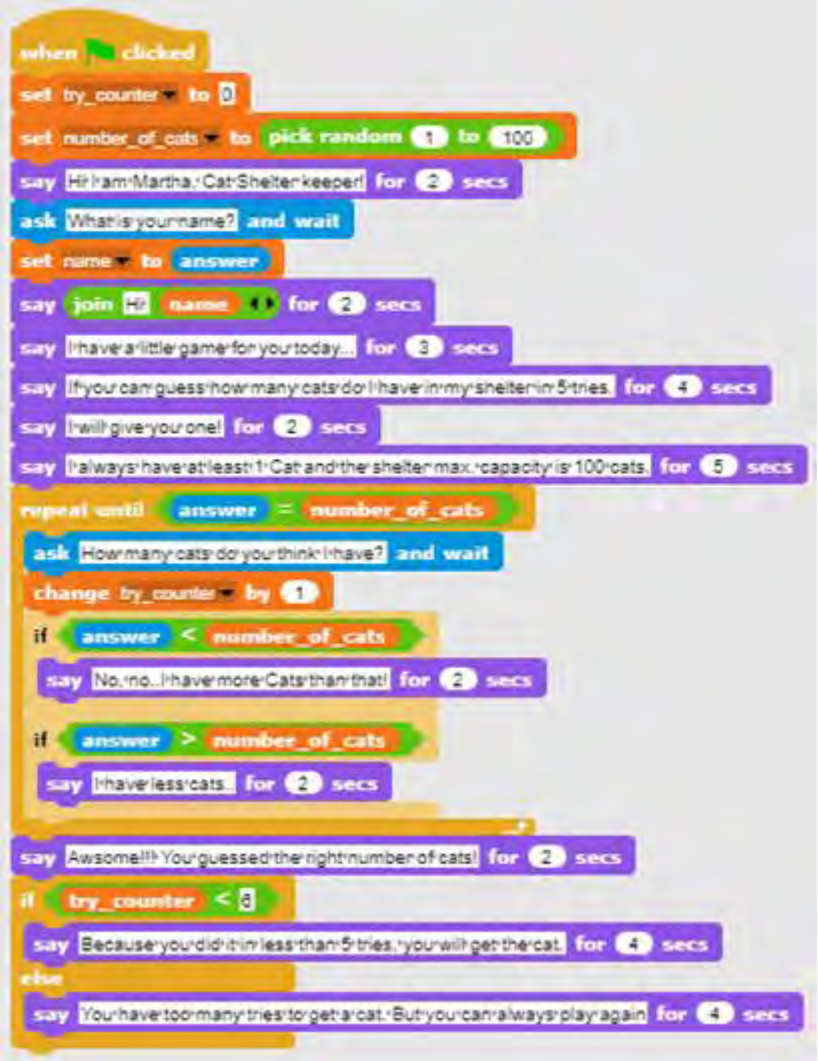
[Step 9]

We have to create a new variable that will have the role of a counter and initialize it to the value 0. We discuss with students the importance of variable initialization and difference between setting the value, and increasing it. When we set the value of a variable, the previous value is lost. This is not ok for a counter. If we increase variable value by some number, we add that number to whatever value variable had earlier. This is exactly what we want in this situation. Every time a player inputs a new number we want to increase it by 1.

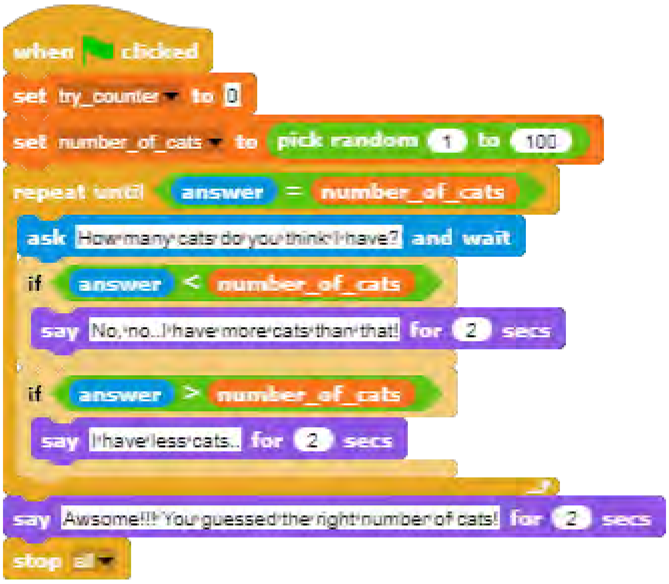
[Step 10]

After the right answer, we have to check the value of the counter variable in order to decide if the player will get the cat or not. Because Snap only has logical operators less (<) and doesn't have operators less or equal, the condition for deciding if a player gets the cat is *cat_counter* < 6. This is also a good example for using If-Else condition block because we differentiate between the two cases.

[Final code]



```
when clicked
set try_counter to 0
set number_of_cats to pick random 1 to 100
say Hi! I am Martha, Cat Shelter keeper! for 2 secs
ask What is your name? and wait
set name to answer
say join Hi! name for 2 secs
say I have a little game for you today... for 3 secs
say If you can guess how many cats do I have in my shelter in 5 tries. for 4 secs
say I will give you one! for 2 secs
say I always have at least 1 Cat and the shelter max. capacity is 100 cats. for 5 secs
repeat until answer = number_of_cats
  ask How many cats do you think I have? and wait
  change try_counter by 1
  if answer < number_of_cats
    say No, no...I have more Cats than that! for 2 secs
  if answer > number_of_cats
    say I have less cats. for 2 secs
say Awsome!!! You guessed the right number of cats! for 2 secs
if try_counter < 6
  say Because you did it in less than 5 tries, you will get the cat. for 4 secs
else
  say You have too many tries to get a cat. But you can always play again for 4 secs
```

	[Basic version of the activity] To save time we can use the basic version of the scenario. In a basic version all the essential concepts are included, other functionalities described above can be used as later upgrades.  when 🏳 clicked set try_counter ▾ to 0 set number_of_cats ▾ to pick random 1 to 100 repeat until answer = number_of_cats ask How many cats do you think I have? and wait if answer < number_of_cats say No, no...I have more cats than that! for 2 secs if answer > number_of_cats say I have less cats.. for 2 secs say Awsome!!! You guessed the right number of cats! for 2 secs stop all ▾
Tools and Resources for the Teacher	- Whole activity in Snap!: https://snap.berkeley.edu/project?user=zapusek&project=cats_in_a_shelter - Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena. - Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.
Resources/materials for the Students	- Template in Snap!: https://snap.berkeley.edu/project?user=zapusek&project=cats_in_a_shelter_template - Instructions for student (C4G11_InstructionsForStudent.docx)

ADVANCED LEARNING SCENARIOS

Learning Scenario 12 - Catching healthy food

Learning Scenario Title	Catching healthy food
Previous programming experience	Adding test for sprite Showing and hiding sprite Using point in direction Using random Using variables for counting points Using loop repeat Using forever loop Using conditionals
Learning Outcomes	General learning outcomes: ● Variables ● Conditionals ● Loop ● Point in direction ● Random Specific learning outcomes oriented on algorithmic thinking: ● Student uses variable for preventing the game to start before the girl ends talking (optional) ● Student uses if statement for checking (with help of a variable) if the food can start moving ● Student uses loop repeat for food's movement until points are less than 5 ● Student uses point in direction 180 (down) for sprites moving down ● Student uses random for picking number of steps ● Student uses random for moving to random position ● Student uses random for moving to x (random), y (fixed) position (optional)
Aim, Tasks and Short Description of Activities	Short description: The girl is catching food. She has to be careful, only healthy elements bring points!

	Tasks: Students have to program two different sprites, a girl who gives instructions, tells what to do to start the game and counts points; and food which is randomly falling from the top of the screen. Additionally, students can add a variable and if statement for preventing the food movement before a girl stops talking. **Aim: Students will learn how to randomly move for X steps and choose a position and also how to use variables and conditionals for preventing other events.**
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Active learning, game-design based learning, problem solving
Teaching Forms	Individual work / Work in pairs
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) The girl is catching food. Each healthy food brings 1 point, while each unhealthy takes 1 point. The game starts with some instructions, given by a girl. Then she disappears and the food appears. When the player collects 5 point, food disappears and a girl reappears.

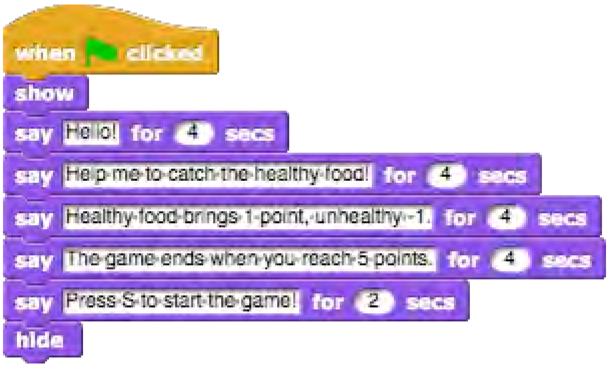
[Step 1]

This activity is meant as an individual work or work in pairs. A teacher gives some clues, explains some harder parts and helps when needed.

It is initially given to the students:

- Background
- Girl sprite

Students choose background and add a main sprite, e.g. a girl. The girl gives some instructions at the beginning and then she hides. Like we saw from previous activities, it's good to write a block *show* when the flag is clicked (when playing again, if the sprite remains hidden). The code is, for example:



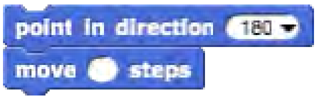
```
when flag clicked
show
say [Hello!] for (4) secs
say [Help me to catch the healthy food!] for (4) secs
say [Healthy food brings 1 point, unhealthy -1.] for (4) secs
say [The game ends when you reach 5 points.] for (4) secs
say [Press S to start the game!] for (2) secs
hide
```

We will return to this sprite later. Let's write a code for a fruit now.

[Step 2]

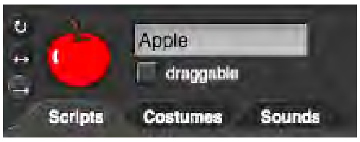
Students add a new sprite, a healthy food, e.g. an apple.

First, they program a sprite movement, which is from top to bottom, so they select following blocks:

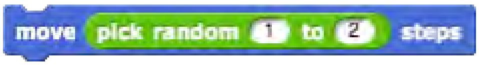


```
point in direction 180
move ( ) steps
```

If they don't want their apple to be upside down, they can choose the third option *don't rotate* in direction block.



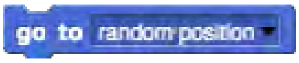
To make a game more interesting, number of steps can be randomly chosen, so the speed will not be always the same. E.g.:



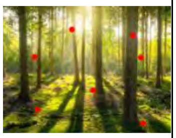
```
move (pick random 1 to 2) steps
```

Next step is to think about what happens when the apple comes to the bottom of the screen?

In this case students can use a block *touching edge* in combination of *if statement*. If the apple touches the edge, it will be moved on some random position. Blocks for movement offers us next block:

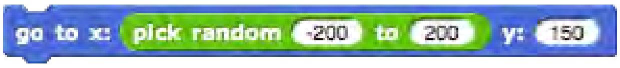


```
go to random position
```



This command will randomly choose x any y coordinates and the apple could appear anywhere on the screen (look at red dots on the picture).

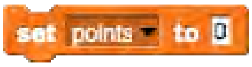
If we want the apple to appear always on the top of the screen, the y value can be fixed, and only the x value will be picked randomly. With the following code the apple will always appear on the top of the screen (look at red dots on the picture).



```
go to x: pick random -200 to 200 y: 150
```

[Step 3]

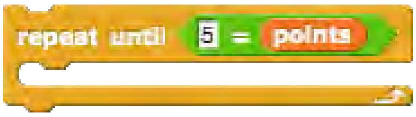
Students can now make a variable, *points*, which they will use for counting. Points have to be set to 0 at the beginning (on girl's sprite).



```
set points to 0
```

[Step 4]

If we want the apple to move repeatedly, we need a loop. Students can use a loop *repeat until* and set a condition. For example, they want the game to finish when they reach 5 points. So the condition will be *points* = 5 and the loop will repeat until the condition is false. When the condition is true, this is the player reaches 5 points, the loop will stop.

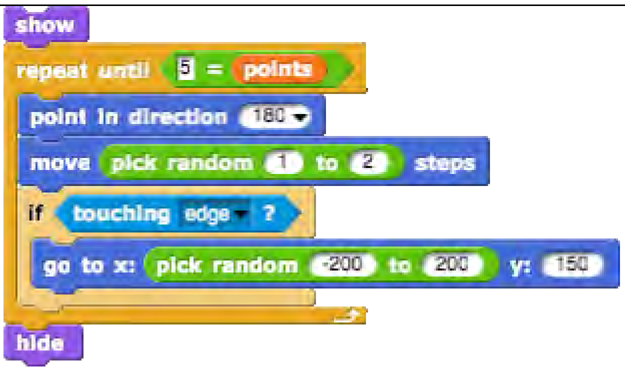


```
repeat until 5 = points
```

[Step 5]

We don't want the apple is shown at the beginning, but after the girl gives her instructions. Students can program the apple to show *when key is pressed.* Of course, they had to add a block *show* before the loop repeat and *hide* after that. The whole code for now looks like:

Co-funded by the
Erasmus+ Programme
of the European Union

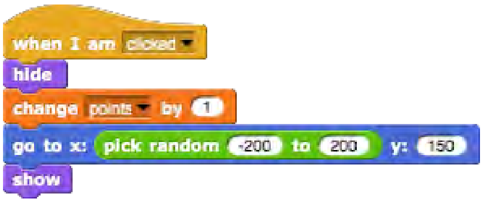


```
show
repeat until 5 = points
    point in direction 180
    move pick random 1 to 2 steps
    if touching edge ?
        go to x: pick random -200 to 200 y: 150
hide
```

[Step 6]

What happens when the apple *is clicked* (or *mouse-entered*)?

The apple has to hide, count points, change position and show again. Points will be changed by 1 and for position students can use the same code as before.



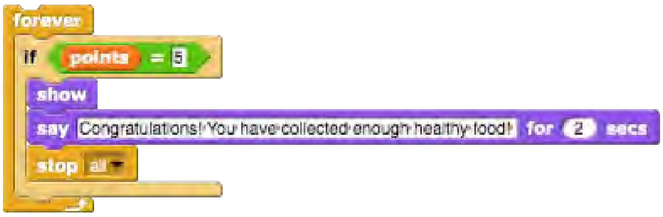
```
when I am clicked
hide
change points by 1
go to x: pick random -200 to 200 y: 150
show
```

[Step 7]

Let's move back to the girl.

The girl has now to reappear and say, e.g. *Congratulations!*

We'll need a loop forever, which will check if we reached 5 points. If we did, the girl will show and say something. After that we'll add a block *stop all*. Let students figure it out what does this stop mean (without stop, girl will be saying "Congratulations…" forever).

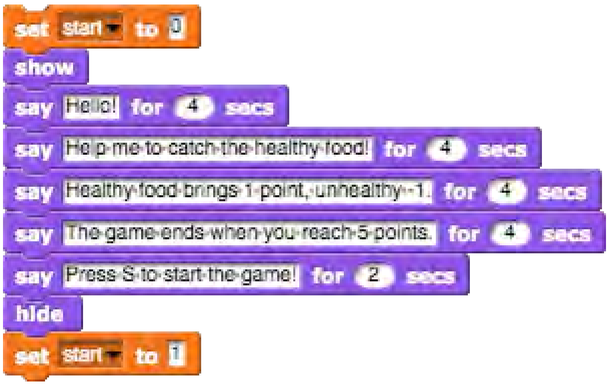


```
forever
    if points = 5
        show
        say Congratulations! You have collected enough healthy food! for 2 secs
        stop all
```

[Step 8]

When playing the game again, when students will already know all the instructions (from [Step 1]) and they will surely want to skip them. They can press the "S" before so the game will begin, but the girl will be still talking.

To prevent that, we can create another variable (named *start*), which has to be set to 0 at the beginning. Then, after the girl's instructions, the variable start will change to 1.

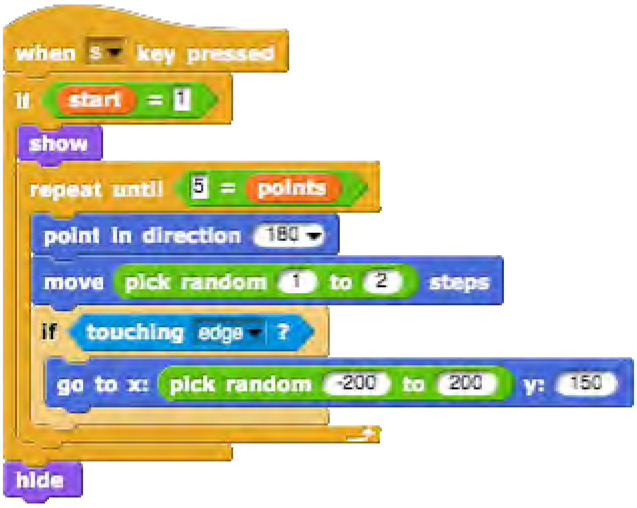


```
set start to 0
show
say Hello! for 4 secs
say Help me to catch the healthy food! for 4 secs
say Healthy food brings 1 point, unhealthy -1. for 4 secs
say The game ends when you reach 5 points. for 4 secs
say Press S to start the game! for 2 secs
hide
set start to 1
```

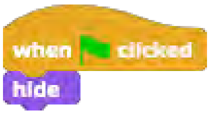
Now we have to program the apple to start only if the variable *start* is equal to 1, which students will do with *if statement*. With this, students won't be able to run a game before the girl stops talking.

Another thing can happen when we play the game again. If we stop the game when we have for example 3 points, the apple will not disappear. In this case when starting the game again, the apple will be visible before the girl ends with giving instructions. Since we do not want this, we add a code that apple hides at the beginning of the game.

The apple's code is now:



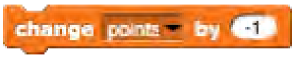
```
when s key pressed
if start = 1
  show
  repeat until 5 = points
    point in direction 180
    move pick random 1 to 2 steps
    if touching edge ?
      go to x: pick random -200 to 200 y: 150
hide
```



```
when [flag] clicked
hide
```

[Step 9]

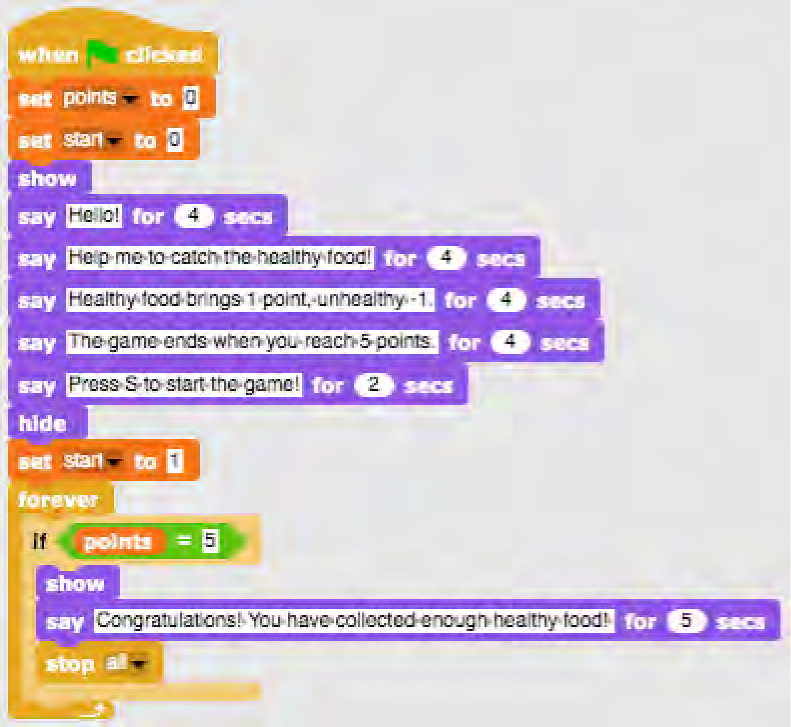
Students can now duplicate the apple sprite many times and change them costume (if they want). The code will be the same. The only change is with unhealthy food, where they will lose 1 point by clicking it.



```
change points by -1
```

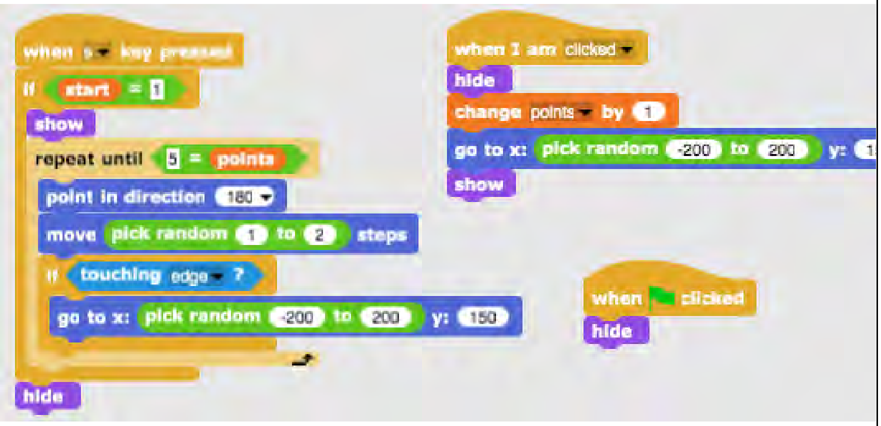
[Final Code]

Girl



```
when ⚑ clicked
set points ▾ to 0
set start ▾ to 0
show
say Hello! for 4 secs
say Help me to catch the healthy food! for 4 secs
say Healthy food brings 1 point, unhealthy -1. for 4 secs
say The game ends when you reach 5 points. for 4 secs
say Press S to start the game! for 2 secs
hide
set start ▾ to 1
forever
    if points = 5
        show
        say Congratulations! You have collected enough healthy food! for 5 secs
        stop all ▾
```

Apple



```
when s ▾ key pressed
if start = 1
    show
    repeat until 5 = points
        point in direction 180 ▾
        move pick random 1 to 2 steps
        if touching edge ▾ ?
            go to x: pick random -200 to 200 y: 150
hide
```

```
when I am clicked ▾
hide
change points ▾ by 1
go to x: pick random -200 to 200 y: 1
show
```

```
when ⚑ clicked
hide
```

[Additional tasks]

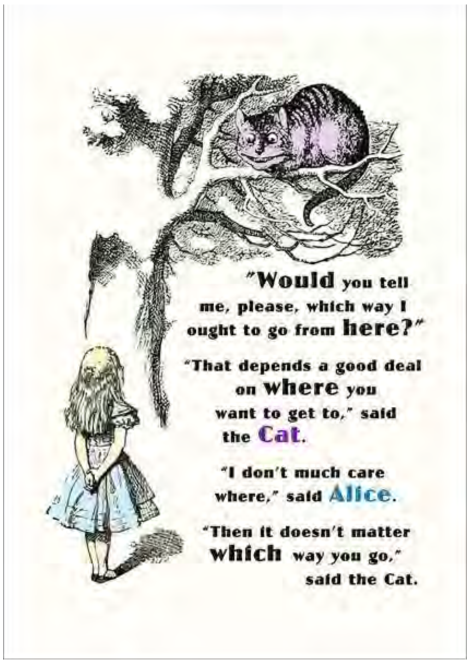
Students can add additional tasks according to their wishes or they can follow the tasks below:

- Change the game so that a bowl sprite is catching food.

	● Add a new sprite (a bowl). Draw it, find it online or use attached picture/s of the bowl. ● Set the starting position of the bowl (e.g. at the bottom of the screen) and write a code for the bowl’s movement (left and right, if you want also up and down). Food sprites have to disappear and reappear at a random location by touching the bowl (and not on mouse-clicking the food as before). ● Change the rules – let the game end when a player scores 20 points (he wins) or when he picks up 3 unhealthy foods (he loses). ● Add more food sprites to make the game more interesting. ● Change the bowl costume when a player scores e.g. 5, 10, 15 points.
Tools and Resources for the Teacher	● Whole activity in Snap!: https://snap.berkeley.edu/project?user=mateja&project=Catching%20healthy%20food ● Activity in Snap! with additional tasks (possible solution): https://snap.berkeley.edu/project?user=mateja&project=Catching%20healthy%20food%20%2B%20Add.%20Task ● Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena. ● Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.
Resources/materials for the Students	● Half-baked activity in Snap!: https://snap.berkeley.edu/project?user=mateja&project=C4G12_Catching%20healthy%20food%20-%20Part

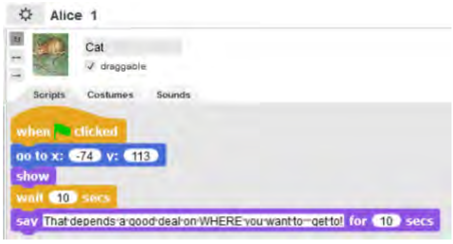
	- Instructions for student (C4G12_InstructionsForStudent.docx) - Images: bowl1.png, bowl2.png, bowl3.png, bowl4.png

Learning Scenario 13 - Storytelling

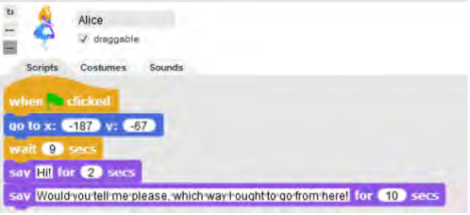
Learning Scenario Title	Storytelling
Previous programming experience	Showing and hiding sprite Using conditionals Using say (from looks group) Using wait for…seconds
Learning Outcomes	General learning outcomes: ● Moving and size-changing ● Broadcasts ● Compose the structure of storytelling ● Changing the background of the scenes Specific learning outcomes oriented on algorithmic thinking: ● Students plan dialogues and activities of the sprites in the story ● Students use sending of broadcasts for dialogue between sprites ● Students use moving and size changing for sprites ● Students use show and hide of sprites ● Students refactor and extend the code of sprites
Aim, Tasks and Short Description of Activities	Short description: The rabbit tells the story about Alice in Wonderland. He starts the storytelling with several sentences against the backdrop labelled *Alice in Wonderland*. The story of Alice begins in the forest. Alice walks and wonders "Where am I?" /To realise Alice's moving away, gradually with the movement her size is reduced/. Alice arrives at a crossroads and sees the Cheshire Cat on a tree. A conversation begins between Alice and the Cheshire Cat.  "Would you tell me, please, which way I ought to go from here?" "That depends a good deal on where you want to get to," said the Cat. "I don't much care where," said Alice. "Then it doesn't matter which way you go," said the Cat.

	The conversation is presented in the picture. Tasks: Students have to experiment with a brief example of the story of the meeting between Alice and the Cat based on synchronizing the dialogue through a waiting block. They then review a second version of the story using broadcast messages. Messaging commands are entered. The students complete the code of the characters according to the text from the picture. The task is complicated by the introduction of changing the stage decor through broadcasting and moving Alice through the woods before her meeting the cat. **Aim: Students will learn how to plan storytelling, how to use broadcast messages for synchronisation of the activities of sprites and stage changes.**
Duration of Activities	90 minutes
Learning and Teaching Strategy and Methods	Active learning, game-design-based learning, problem-solving
Teaching Forms	Individual work / Work in pairs/ Frontal Discussion
Teaching summary	(Motivation-Introduction, Implementation, Reflection and Evaluation) 1. The teacher discusses with the students the story of Alice in Wonderland and shows the picture of Alice meeting the Cheshire cat. She explains that Alice's story can be recreated using coding. Students are tasked with starting the project and looking at the sprites' codes. https://snap.berkeley.edu/project?user=ddureva&project=Alice_1 Discussion: Who starts talking first? When does Alice get involved and when - the Cat? Why is there no synchronization in the dialogue of the characters? The answer lies in the inaccurate calculation of the times in which each of the characters *"talks"* and *"the lack of a timeout to wait for a character to finish his or her replies"*.

The codes are commented on and the table is completed:



```
when 🏴 clicked
go to x: -74 y: 113
show
wait 10 secs
say That depends a good deal on WHERE you want to  get to! for 10 secs
```



```
when 🏴 clicked
go to x: -187 y: -67
wait 9 secs
say Hi! for 2 secs
say Would you tell me please, which way I ought to go from here! for 10 secs
```

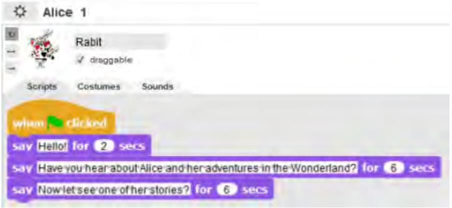
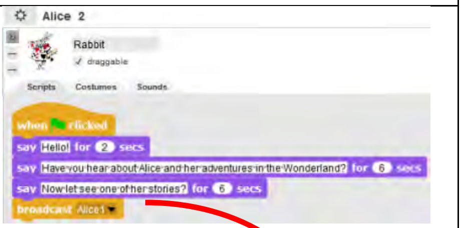
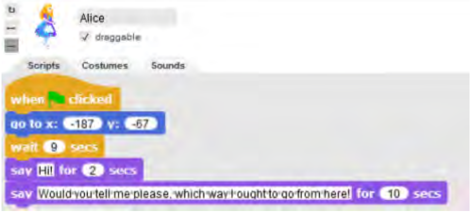
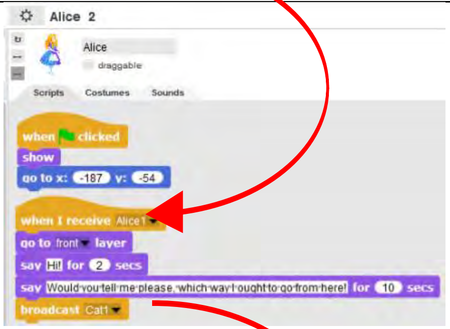
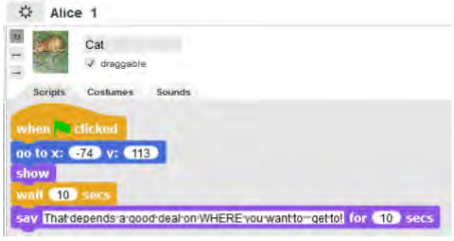
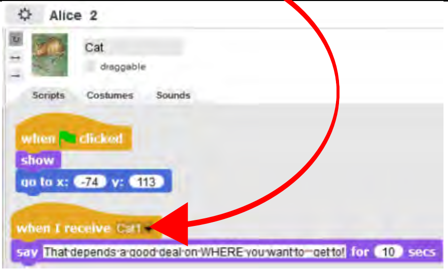
Sprite	Activity	Start from beginning	End time	Duration
Rabbit	Say: Hello! Have you heard about Alice and her adventures in Wonderland? Now let's see one of her stories.	0	14	14
Alice	Say: Would you tell me please, which way I ought to go from here?	9	21	12
Cat	Say: That depends a good deal on WHERE you want to go.	10	20	10

The conclusion is that synchronizing with the *wait for… second* block can lead to errors in the behavior of the characters in storytelling.

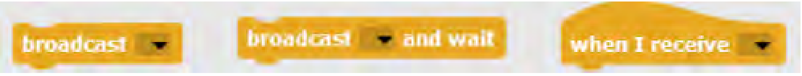
2. The teacher is tasked with starting and reviewing the project code https://snap.berkeley.edu/project?user=ddureva&project=Alice_2

What are the unfamiliar commands so far?

The codes from Alice_1 and Alice_2 are compared.

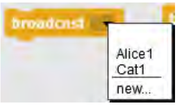
Alice_1	Alice_2
 Alice 1 Rabit when clicked say Hello! for 2 secs say Have you hear about Alice and her adventures in the Wonderland? for 6 secs say Now let see one of her stories? for 6 secs	 Alice 2 Rabbit when clicked say Hello! for 2 secs say Have you hear about Alice and her adventures in the Wonderland? for 6 secs say Now let see one of her stories? for 6 secs broadcast Alice1
 Alice when clicked go to x: -187 y: -67 wait 9 secs say Hi! for 2 secs say Would you tell me please, which way I ought to go from here! for 10 secs	 Alice 2 Alice when clicked show go to x: -187 y: -54 when I receive Alice1 go to front layer say Hi! for 2 secs say Would you tell me please, which way I ought to go from here! for 10 secs broadcast Cat1
 Alice 1 Cat when clicked go to x: -74 y: 113 show wait 10 secs say That depends a good deal on WHERE you want to get to! for 10 secs	 Alice 2 Cat when clicked show go to x: -74 y: 113 when I receive Cat1 say That depends a good deal on WHERE you want to get to! for 10 secs

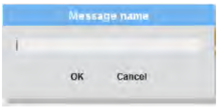
2. Blocks for broadcasting are introduced:



It is discussed that *broadcast* messages target all characters, but can only be received by some of the characters. The *broadcast… and wait* block requires all characters who have received the message to perform their actions and then the actions of the sprite that sent the message continues.

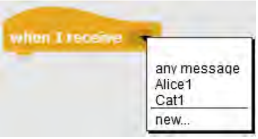
The teacher demonstrates how to name a *broadcast* message and how it is used in the event *When I receive …*

1. 

2. 

3. Introducing a name. OK

Use in an event:



1. , 2. The message that should be received by the sprite is selected from the list.

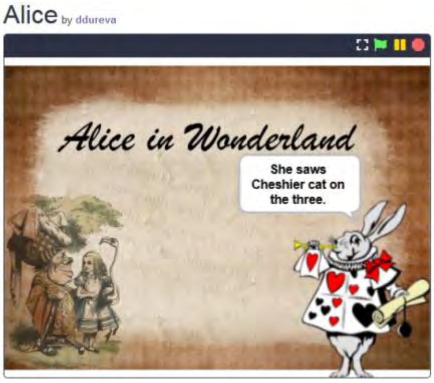
3. The group discusses how to complete the story in the picture. How to name messages: e.g. The message from Cat to Alice to be Alice2 and from Alice to Cat - Cat1.

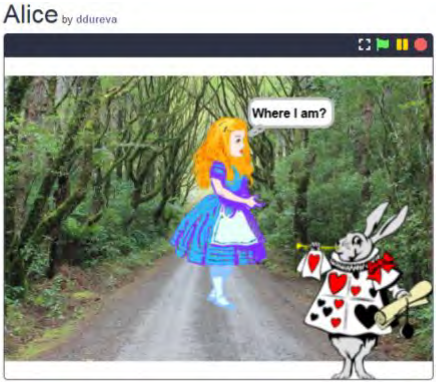
4. Students complete the story in pairs.

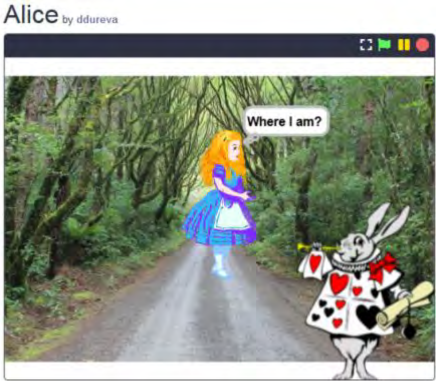
5. The teacher comments that storytelling often requires a change in stage costumes. *“Let's make Alice's story more complete by starting the story of the Rabbit against an introductory backdrop, moving the action into the forest where Alice is walking and wondering "Where am I?" And her size gradually decreases as she moves farther away. Then she finds herself at a crossroads and sees The Cheshire cat. The conversation begins between the two.*

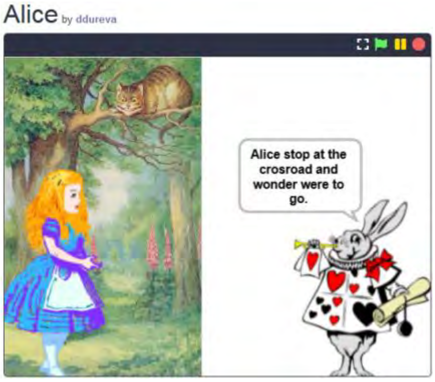
6. The teacher demonstrates the project.

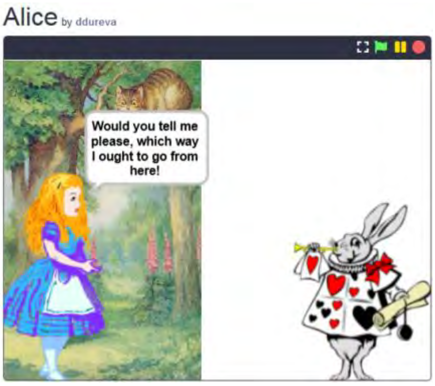
https://snap.berkeley.edu/project?user=ddureva&project=Alice

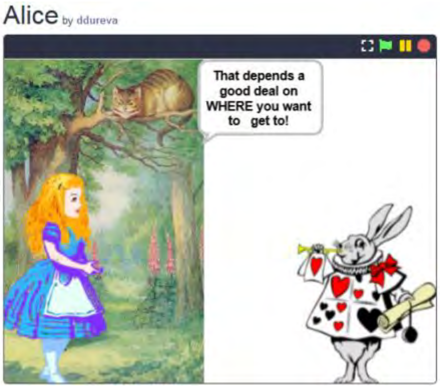









Alice by ddureva
Would you tell me please, which way I ought to go from here!

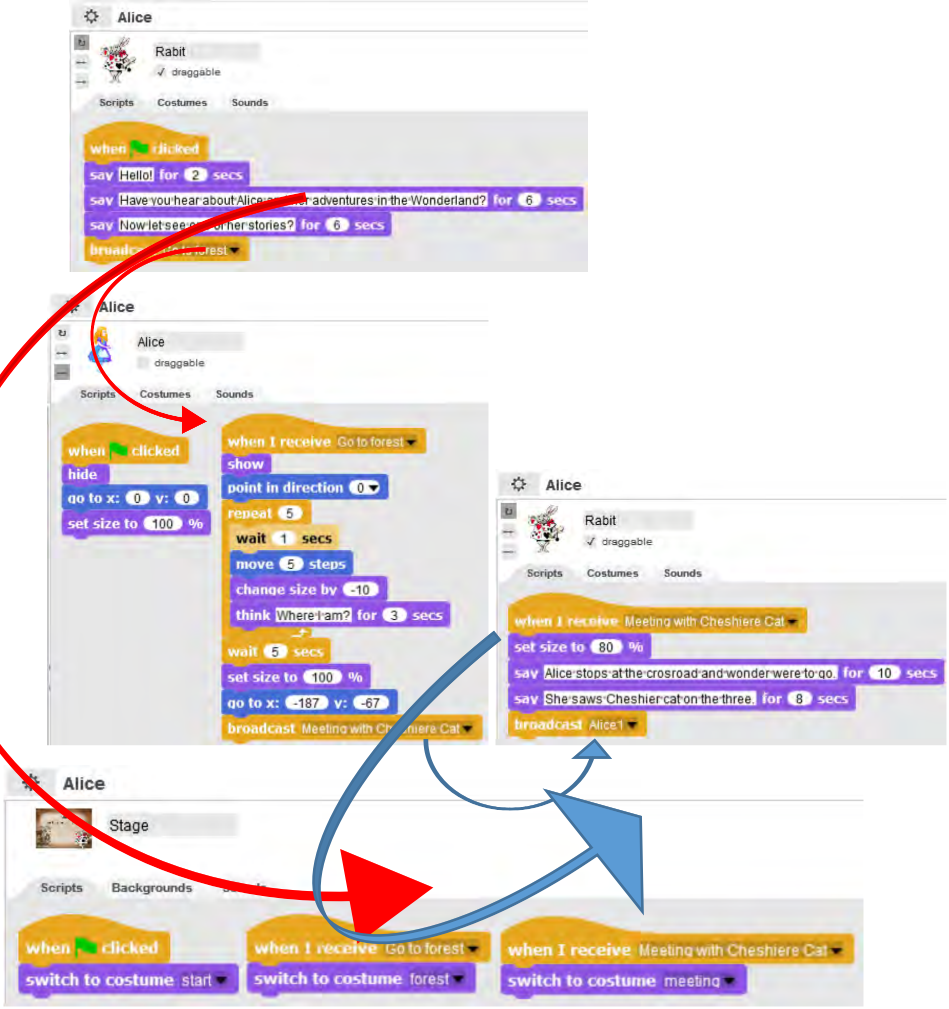


Changes in the scenes and actions of characters are commented on.

"When does a scene change? When does Alice appear and what are her actions? When does the cat appear and what are his actions?"

The scenes in the Alice_2 project are discussed. There are 3 scenes, one already used. Which scene to use to get started? What should be done to prevent the sprites of Alice and the Cat from being displayed at the start?

How to change the stage décor? A *broadcast* can be used to change the stage set by the Rabbit after his introductory words. Alice appears when the scene is changed with the message *Go to forest.*



Alice – Rabit ✓ draggable – Scripts

```
when [flag] clicked
say [Hello!] for (2) secs
say [Have you hear about Alice and her adventures in the Wonderland?] for (6) secs
say [Now let see one of her stories?] for (6) secs
broadcast [Go to forest]
```

Alice – Alice □ draggable – Scripts

```
when [flag] clicked
hide
go to x: (0) y: (0)
set size to (100) %
```

```
when I receive [Go to forest]
show
point in direction (0)
repeat (5)
  wait (1) secs
  move (5) steps
  change size by (-10)
  think [Where I am?] for (3) secs
wait (5) secs
set size to (100) %
go to x: (-187) y: (-67)
broadcast [Meeting with Cheshiere Cat]
```

Alice – Rabit ✓ draggable – Scripts

```
when I receive [Meeting with Cheshiere Cat]
set size to (80) %
say [Alice stops at the crosroad and wonder were to go.] for (10) secs
say [She saws Cheshier cat on the three.] for (8) secs
broadcast [Alice1]
```

Alice – Stage – Scripts

```
when [flag] clicked
switch to costume [start]
```

```
when I receive [Go to forest]
switch to costume [forest]
```

```
when I receive [Meeting with Cheshiere Cat]
switch to costume [meeting]
```

When Alice is on the path in the woods, she walks and "wonders", so for greater realism, her size decreases by -10%. This is repeated 5 times using the *repeat loop.*

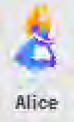
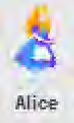
When she reaches the junction, the scene is changed with the message *"Meeting the Cheshire Cat"*. This message is received at the same time by the Rabbit, which reduces its size to 80% and he continues to tell the story with his size reduced. At this stage the cat sprite is not shown because it is present as part of the decor. It appears on the Cat1 message. The teacher

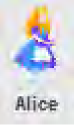
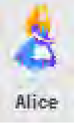
	can explain that the cat was cut from the decor using an external graphics editor. (Unfortunately, Snap! does not provide much graphics editor capabilities, unlike Scratch 3.0). After the release of the Rabbit message, the *Alice 1* story continues as it was done in the *Alice 2* project. 3. The teacher comments that in order to tell a story, one must first invent a plot. An additional table may be used to describe the scenario of the story. (Appendix 1) At the teacher's discretion, the finished table may be given or partially completed and students, guided by the illustration, may complete it. 4. The students are tasked with describing the examined scenarios and completing the story of the Alice_2 project in pairs.
Tools and Resources for the Teacher	Whole activity in Snap!: https://snap.berkeley.edu/project?user=ddureva&project=Alice
Resources/materials for the Students	"Would you tell me, please, which way I ought to go from here?" "That depends a good deal on where you want to get to," said the Cat. "I don't much care where," said Alice. "Then it doesn't matter which way you go," said the Cat. ● https://snap.berkeley.edu/project?user=ddureva&project=Alice_1 ● https://snap.berkeley.edu/project?user=ddureva&project=Alice_2 ● Instructions for student (C4G13_InstructionsForStudent.docx)

Appendix 1. Story plots/Scenarios

Name	Design	Actions	Notes
1. Start	Alice in the Wonderland	The story starts with the scene (When the green flag is clicked)	Against this background, the Rabbit introduces the story.
2. Forest		The scenery appears when the Rabbit rounds up his introduction (A *Go to forest* message has been sent)	Against this background, Alice appears positioned in the center of the stage. She starts moving, wondering *"Where am I?"*. The sprite gradually reduces its size 5 times by 10%. When it reaches the end of the path (at a crossroads), the scene changes to *Meeting*. (Alice sends message -broadcast *Meeting with Cheshire Cat*)
3. Meeting		Appears when *Alice's* message *Meeting with Cheshire Cat* is received.	Here Alice and the cat are part of the background. To use Alice's sprite, prior to the message, she is positioned so that she covers her image from the decor. The Cat sprite appears at a later stage. As the scene changes, the Rabbit continues to tell the story. Later a conversation takes place between Alice and the Cheshire Cat.

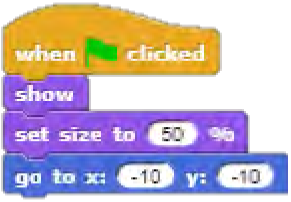
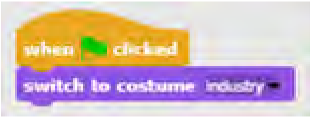
Sprites

Sprite	Actions	Stage background
	At the Start: Says: Hello! (For 2 sec.) Says: Have you heard about Alice and her adventures in Wonderland? (For 6 sec.) Says: Now let's see one of her stories! (for 6 sec.) Sends the *Go to forest* message.	
	At the Start: Hides from stage; at centre stage position and 100% size, ready to be displayed against the new background.	
	At the Start Hides from the stage; positioned at x: -74, y: 113 (Positions are predetermined after the Cat sprite has been set on the *Meeting* stage.)	
	Receives a *Go to forest* message: The sprite shows on stage. Repeated 5 times: waiting for 1 sec .; moving 5 steps; size reduction (change by -10); wondering: *Where am I?* Preparing for next decor: waiting 5 sec; restoring the sprite's size (100% change) and positioning at x: -187, y: -67 Sends Message: *Meeting with Cheshire Cat*.	
	No action. Just becomes visible from previous decor.	
	Receives the message: *Meeting with Cheshire Ca*t. Resizes to 80%	

	He says: *"Alice stops at the crossroads and wonders were to go."* (for 10 seconds). He says, *"She saw the Cheshire Cat on the three."* (for 8 sec.) Sends a message *Alice1*	
	Receives the *Alice1* message. Moves to the front (This is necessary because the Cat appears after her, which prevents Alice's lines from appearing in a dialogue box if she is not in the front layer). She says: *"Hi!"* (for 2 sec.) She says: *"Would you tell me please, which way I ought to go from here!"* (for 10 seconds). Sends a *broadcast* message to the Cat: *Cat1*.	
	Receives the *Cat1* message. The sprite shows on stage. It says: *"That depends a good deal on WHERE you want to get to!"* (for 10 seconds). Sends an *Alice2* message.	
	Receives the *Alice 2* message. Says: …………………………………………………………………… Sends a *Cat2* message.	
	Receives the *Cat2* message. Says: …………………………………………………………………… Sends a *Rabbit1* message.	
	Receives the *Rabbit1* message. Says: "What's the moral of the story?" (for 8 sec.) Says: "To know which way to go, one has to determine his or her goal first."	

Learning Scenario 14 - Drawing

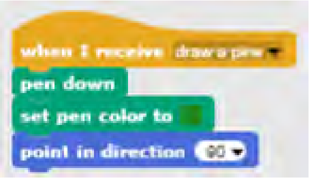
Learning Scenario Title	Drawing
Previous programming experience	Adding sprite Using point in direction Using variables for counting point Using loop repeat Using conditionals
Learning Outcomes	General learning outcomes: ● Variables ● Conditionals ● Loop ● Point in direction ● Operators Specific learning outcomes oriented on algorithmic thinking: ● student uses pen to draw ● student uses loops to draw ● student changes the value of a variable when drawing ● student uses point in direction to draw objects on the stage ● student uses broadcast to control sprite ● student uses conditionals to change stage ● student uses operator > to change stage
Aim, Tasks and Short Description of Activities	Short description: The climate has changed a lot, the air is heavily polluted due to industry. Trees need to be planted to improve air quality! Tasks: To improve air quality, students have to program sprite to draw two types of different trees - pine and oak, and buttons that symbolize those types of trees. When a button is clicked, a specific tree type is drawn. **Aim: Students will learn to draw in Snap!, to change the colour and the pen thickness, and how to use variables and conditionals that cause a new event.**

Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Active learning, game-design based learning, problem solving
Teaching Forms	Individual work / Work in pairs
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) At the beginning of the game, an industry that causes climate change and a variable that displays air quality are shown on stage. Trees need to be planted to improve air quality. Two different types of trees can be drawn, pine and oak. When a pine is drawn, the air is improved by 3, and by drawing an oak the air is improved by 2 units. When the air quality reaches 10 units, the stage background changes to a meadow. [Step 1] Students have to open the Improve *the Climate* program which contains a template of backgrounds (industry and grass) and sprites (a pencil, a pine, an oak and sprite named clear). Also, add new sprite – pencil ("pencil a" from the offered sprites). Because the sprite is too large, it should be reduced to 50%. The starting position of the pencil (coordinates) should also be specified, e.g. X = -10, y = -10.  when 🟩 clicked show set size to 50 % go to x: -10 y: -10  when 🟩 clicked switch to costume industry ▾

[Step 2]

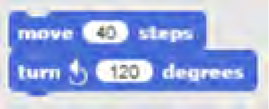
Pencil sprite should receive “oak” and “pine” messages and draw appropriate trees in response to the message. First, mark the pencil sprite and add the code that will enable drawing pine using a pen when the sprite receives the "pine" message.

A point in direction should be set to 90 to draw the canopy in the shape of a triangle, and its color should be set.



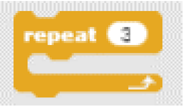
```
when I receive draw a pine ▾
pen down
set pen color to 🟩
point in direction 90 ▾
```

To draw a pine tree canopy, move sprite 40 steps, rotate left by 120 degrees.



```
move 40 steps
turn ↺ 120 degrees
```

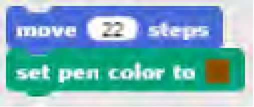
This movement should be repeated three times.



```
repeat 3
```

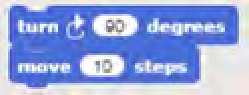
After the canopy, the trunk should also be drawn. For the trunk to be in the proper position, move 22 steps.

After that, set the pen color to brown.



```
move 22 steps
set pen color to 🟫
```

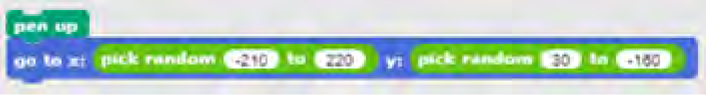
Turn 90 degrees to the right, and then move 10 steps.



```
turn ↻ 90 degrees
move 10 steps
```

This movement should be repeated 3 times.

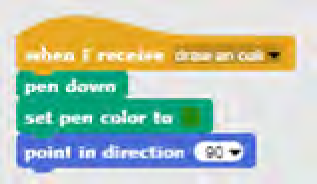
In the end, it is necessary to lift the pen up so the sprite will not left a trace during the next movement. Also, the pen should be moved to the random position.



```
pen up
go to x: pick random -210 to 220 y: pick random 30 to -180
```

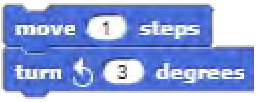
[Step 3]

Similarly, it is necessary to add the code to pencil sprite for drawing oaks. The oak should be drawn when the sprite receives the message "oak". A point in direction should be set to 90 to keep the canopy round, the pen should be down and color should be set.



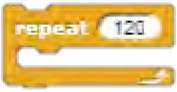
```
when I receive draw an oak ▾
pen down
set pen color to 🟩
point in direction 90 ▾
```

To draw oak tree canopy, move sprite 1 step and turn 3 degrees left after each step.



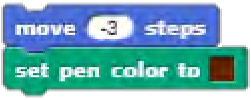
```
move 1 steps
turn ↺ 3 degrees
```

This movement should be repeated 120 times.



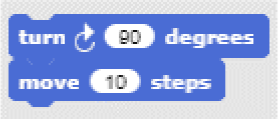
```
repeat 120
```

Once the canopy is finished, the trunk should be drawn. The pencil sprite should be moved to the centre of the drawn circle, by -3 steps, and the colour of the pen changed to brown.



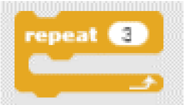
```
move -3 steps
set pen color to
```

To draw the trunk, sprite should be turned right 90 degrees and moved 10 steps.



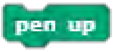
```
turn ↻ 90 degrees
move 10 steps
```

This part is repeated three times.



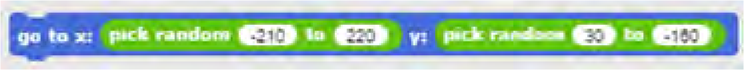
```
repeat 3
```

When the drawing is done, it is necessary to lift the pen up so that it does not draw the line when moving the pencil sprite.



```
pen up
```

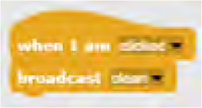
After the oak is drawn, the pen should be moved to the random position.



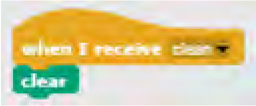
```
go to x: pick random -210 to 220 y: pick random 30 to -160
```

[Step 4]

Next students have to add the code that makes all the drawn trees are deleted when the player clicks on the Clear sprite. When the Clear sprite is clicked with the mouse, it broadcasts a message to clear all trees. When the Pencil sprite receives a message, it deletes all drawn trees.



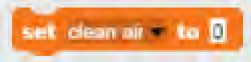
```
when I am clicked
broadcast clear
```



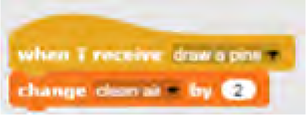
```
when I receive clear
clear
```

[Step 5]

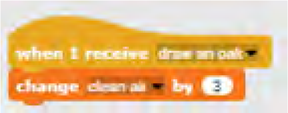
Create a new variable "clean air" to show the current air quality. Set the initial value on 0 and show the variable on the stage.


set clean air to 0

Each time a pine is drawn the air improves by 2 units so add the block to the pine sprite that will change the value of the "clean air" variable by 2 each time the pine is clicked.

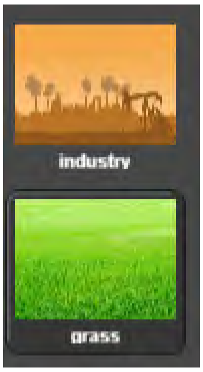

when I receive draw a pine
change clean air by 2

Each time an oak is drawn the air improves by 3 units so add the block to oak sprite that will change the value of the "clean air" variable by 3 each time the pine is clicked.

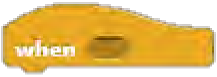

when I receive draw an oak
change clean air by 3

[Step 6]

When the "clean air" variable reaches 10, the stage should change to grass. Therefore, from the downloaded materials add a new background “grass” for the stage (background is from the downloaded materials,).

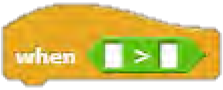


Add hat block „When“ from the „Control“ palette to pencil sprite.



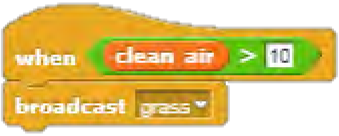
```
when
```

Then, add operator >.



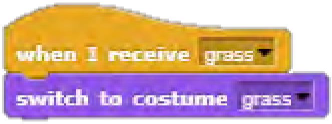
```
when [ ] > [ ]
```

Define that sprite broadcasts the message „grass“ when the „clean air“ variable is greater than 10.



```
when (clean air) > 10
broadcast grass
```

Add the code to the stage to change the costume to “grass” when the “grass” message is received.



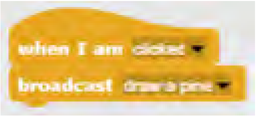
```
when I receive grass
switch to costume grass
```

[ADDITIONAL TASK]

You can upgrade the game by adding animals that they appear when the air is not polluted anymore.

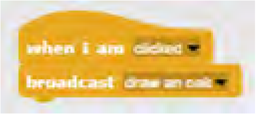
[Final code]

Pine



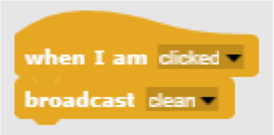
```
when I am clicked
broadcast draw a pine
```

Oak

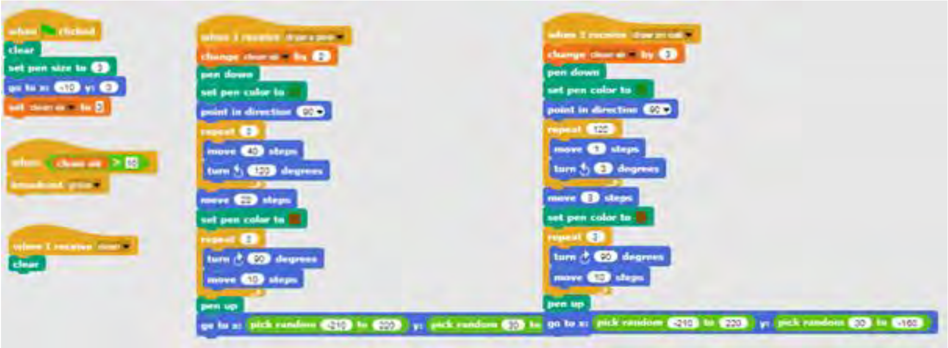
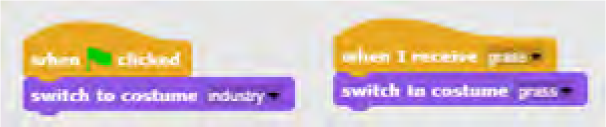


```
when I am clicked
broadcast draw an oak
```

X



```
when I am clicked
broadcast clean
```

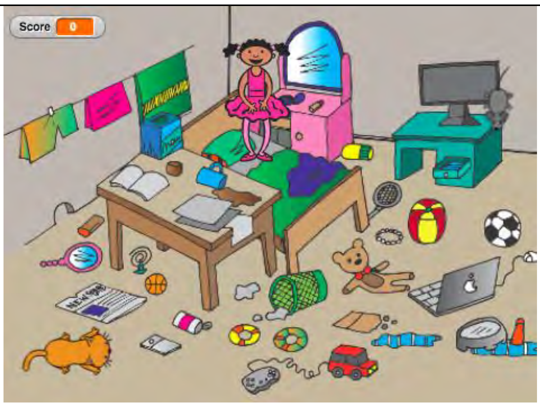
	Pencil  when 🏁 clicked clear set pen size to 3 go to x: -10 y: 0 set clean air to 5 when < clean air > 10 > broadcast grass when I receive clear clear when I receive draw a pen change clean air by 2 pen down set pen color to 🟩 point in direction 90 repeat 3 move 40 steps turn ↺ 120 degrees move 22 steps set pen color to 🟫 repeat 2 turn ↻ 90 degrees move 10 steps pen up go to x: pick random -210 to 220 y: pick random 30 to when I receive draw an oak change clean air by 1 pen down set pen color to 🟩 point in direction 90 repeat 120 move 1 steps turn ↺ 3 degrees move 3 steps set pen color to 🟫 repeat 2 turn ↻ 90 degrees move 10 steps pen up go to x: pick random -210 to 220 y: pick random 30 to -160 Stage  when 🏁 clicked switch to costume industry when I receive grass switch to costume grass
Tools and Resources for the Teacher	Snap! project "Drawing": https://snap.berkeley.edu/project?user=tadeja&project=Improve%20the%20climate (9.1.2020)
Resources/materials for the Students	● Programming language Snap!: https://snap.berkeley.edu/ (9.1.2020) ● Instructions for student (C4G14_InstructionsForStudent.docx)

Learning Scenario 15 - Catch the mouse

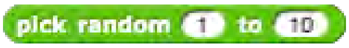
Learning Scenario Title	Catch the mouse
Previous programming experience	● Student is able to add a background. ● Student is able to add a new sprite. ● Student is able to add a new sound. ● Student knows how to make sprite say something. ● Student knows how to change sprite's costume to make an animation. ● Student is able to implement object movement with arrow keys using events and takes into account restrictions. ● Student is able to differentiate between two different states and knows how to express them with logical expressions. ● Student knows how to use conditionals.
Learning Outcomes	General learning outcomes: ● forever loop; ● random numbers; ● counter; ● timer. Specific learning outcomes oriented on algorithmic thinking: ● student uses forever loop to move the sprites; ● student uses random numbers to determine the position of the sprite, move sprite for random steps and turn sprite for random degrees; ● student implements counter for counting mice catch and uses the final value to summarize how successful player is; ● student uses timer to determine the end of the game.
Aim, Tasks and Short Description of Activities	**Short description**: Program a game in which player (the cat) will have to catch the mouse. **Task**: Program the activity in which the cat will catch the mouse. The cat will be moved by a player with arrow keys and the mouse will move randomly. When the cat touches the mouse, the mouse will hide and appear in a random location. We also have to have a counter that will count the number of times the cat caught the mouse. We also have to need a timer to finish the game. After activity the girl has to summarize how successful player was, she tells how many times did player caught the mouse.

Co-funded by the Erasmus+ Programme of the European Union

	Aim: **Student will be introduced to the concept of multiple variable random value assignment. They will learn how to use the *Operators/pick random[x]to[y]* block.**
Duration of Activities	45 min
Learning and Teaching Strategy and Methods	Active learning, collaborative learning, problem solving, game-design based learning
Teaching Forms	Frontal teaching Working in pairs / group work
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) **Motivation-Introduction** We motivate students by showing the game. We discuss with them about how they would start programming this game. Together with students, we determine the sequence of steps, for example: 1. choose background and add sprites; 2. program the cat to move with the arrow keys; 3. program the mouse to move randomly; 4. program the mouse to hide (and appear in a random location) when it touches the cat; 5. program counter; 6. add timer and determine the end of the game; 7. add the girl and program her to summarize how successful player was; 8. program the girl to jump when she touches the mouse; 9. add sound of the cat/mouse; 10. etc. Students can help with the steps or they make their own rules of the game (but they have to follow bold steps).



We introduce operator for random value assignment.



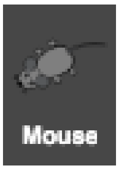
```
pick random 1 to 10
```

Students program the following tasks in pairs/group with the support of the teacher.

Implementation

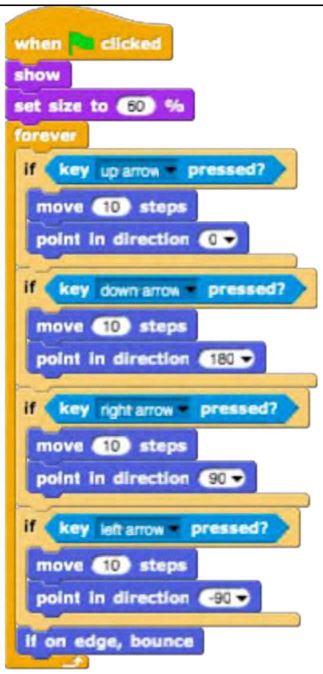
[Step 1]

The first step is to determine the background of the game. Students search for free image online. Next, they add new sprites – the cat and the mouse.



[Step 2]

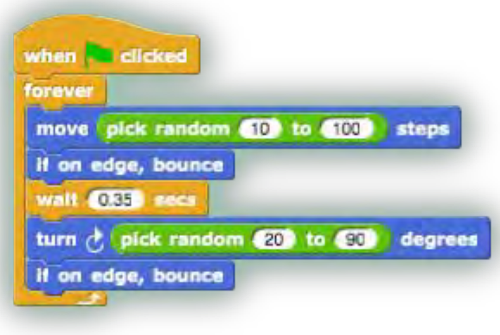
Students program the cat to move with the arrow keys. Here they have to determine what happens if the cat is on edge.



```
when ⚑ clicked
show
set size to 60 %
forever
    if key up arrow pressed?
        move 10 steps
        point in direction 0
    if key down arrow pressed?
        move 10 steps
        point in direction 180
    if key right arrow pressed?
        move 10 steps
        point in direction 90
    if key left arrow pressed?
        move 10 steps
        point in direction -90
    if on edge, bounce
```

[Step 3]

Students have to program the mouse to move randomly. In this case, the idea is that the mouse in an infinite loop takes a random number of steps and turns for a random degree. Students do that with *Motion/move[x]steps* block and *Motion/turn[x]degrees* block into which they insert the *pick random[x]to[y]* operator.

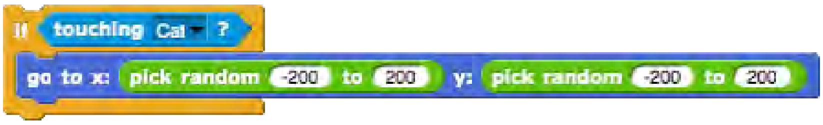


```
when ⚑ clicked
forever
    move pick random 10 to 100 steps
    if on edge, bounce
    wait 0.35 secs
    turn ↻ pick random 20 to 90 degrees
    if on edge, bounce
```

[Step 4]

Next step is to program the mouse to hide when it touches the cat. The idea is that the mouse hides and appears in a random location when it touches the cat. In this case, the game does not end at the first catch of the mouse.

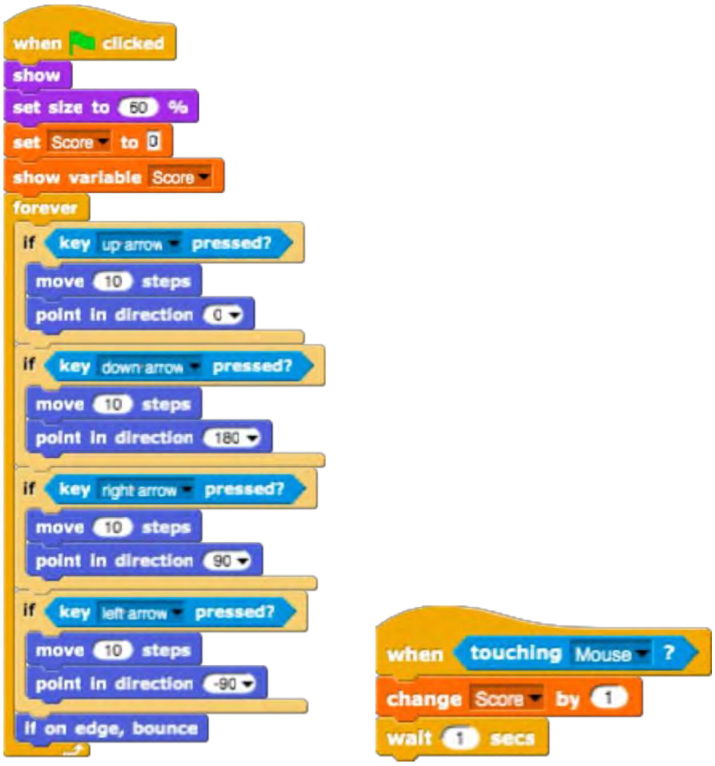
Students can add their own rule here. In any case, they have to use the *pick random[x]to[y]* operator.



```
if touching Cat ?
    go to x: pick random -200 to 200 y: pick random -200 to 200
```

[Step 5]

In the case we want to know the number of times the mouse was caught, we have to add a counter. Students make a new variable – score and add it to the cat's code. The score at the beginning of the game have to always be zero. Students do that with *Variables/set[variable]to[x]* block. If we want the score to be shown to the player of the game, students have to add the *show variable[variable]* block. Then the students add a new control block (*Control/when*) to check if the cat touches the mouse. If the cat touches the mouse, the result is increased by 1 (*Variables/change[score]by[x]*).

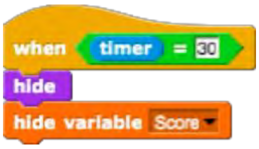


```
when clicked
show
set size to 60 %
set Score to 0
show variable Score
forever
    if key up arrow pressed?
        move 10 steps
        point in direction 0
    if key down arrow pressed?
        move 10 steps
        point in direction 180
    if key right arrow pressed?
        move 10 steps
        point in direction 90
    if key left arrow pressed?
        move 10 steps
        point in direction -90
    if on edge, bounce
```

```
when touching Mouse ?
change Score by 1
wait 1 secs
```

[Step 6]

Students determine when the game ends. They do this with adding the timer. After some time (e.g. 30 seconds) the mouse and the cat disappear, the variable *Score* is hidden and the game is over.

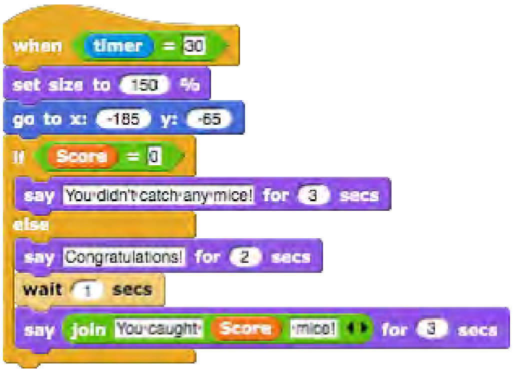


```
when timer = 30
hide
hide variable Score
```

Students have to add these blocks to the cat and mouse script.

[Step 7]

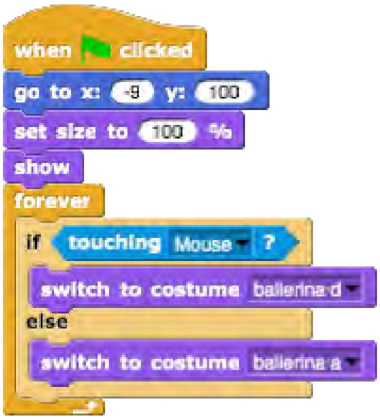
Students have to program the girl to summarize how successful player was. If the player doesn’t catch any mice, the girl says: “You didn't catch any mice!”. Else she says: “Congratulations! You caught *x* mice!”



```
when timer = 30
set size to 150 %
go to x: -185 y: -65
if Score = 0
  say You didn't catch any mice! for 3 secs
else
  say Congratulations! for 2 secs
  wait 1 secs
  say join You caught Score mice! for 3 secs
```

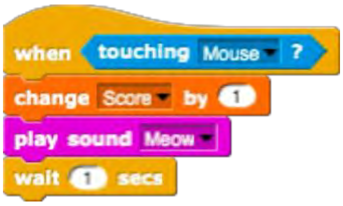
[Additional tasks]

Students can add any elements to their game. For example, the girl who jumps every time she touches a mouse.



```
when clicked
go to x: -9 y: 100
set size to 100 %
show
forever
  if touching Mouse ?
    switch to costume ballerina-d
  else
    switch to costume ballerina-a
```

Students can add sound. For example, they add sound of the cat. The sound plays when the mouse is caught.



```
when <touching Mouse ?>
change Score by 1
play sound Meow
wait 1 secs
```

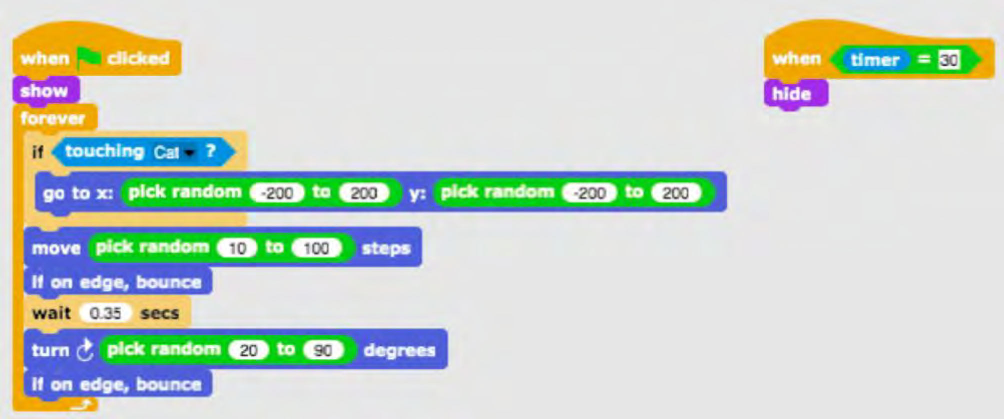
Reflection and evaluation

Students adjust the code:

- the mouse moves 20 to 60 steps forever;
- the mouse goes to location x = 100 when it touches the cat;
- the mouse turns 90 degrees forever;
- etc.

[Final Code]

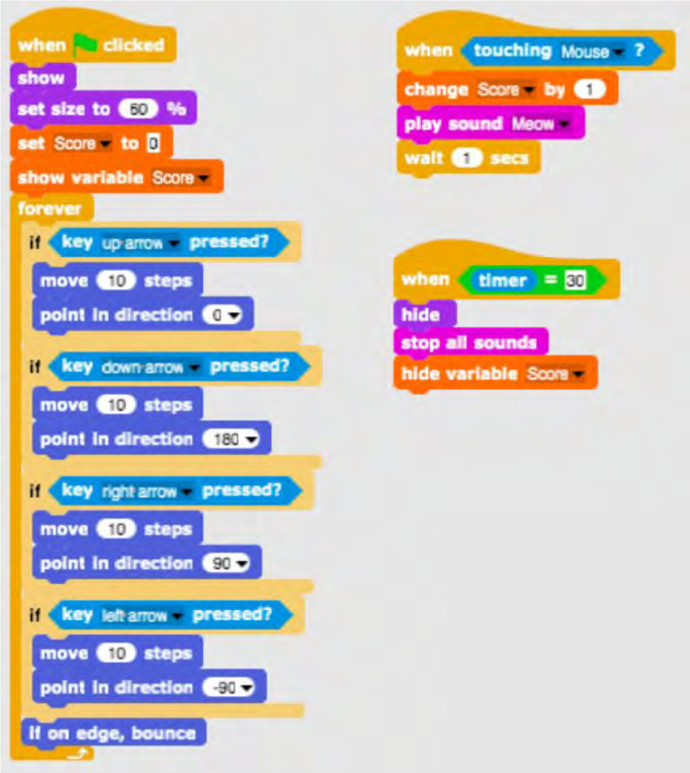
The mouse



```
when 🏴 clicked
show
forever
    if <touching Cat ?>
        go to x: (pick random -200 to 200) y: (pick random -200 to 200)
    move (pick random 10 to 100) steps
    if on edge, bounce
    wait 0.35 secs
    turn ↻ (pick random 20 to 90) degrees
    if on edge, bounce
```

```
when timer = 30
hide
```

The cat

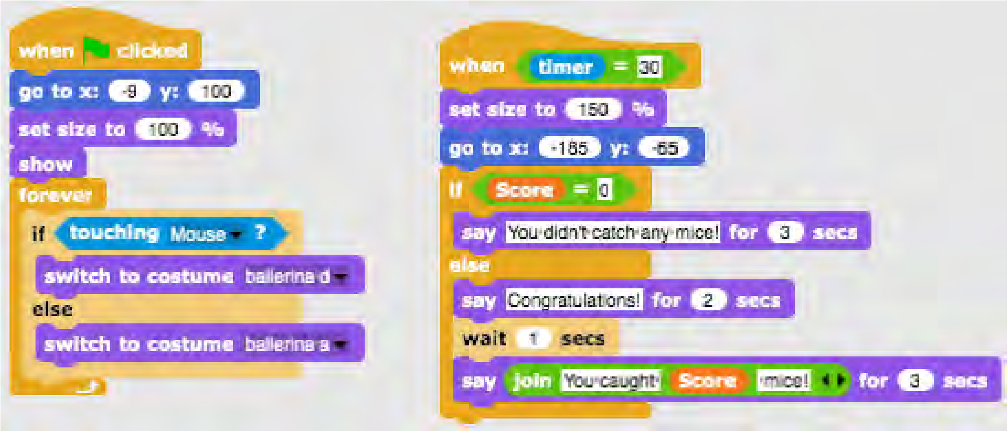


```
when ⚑ clicked
show
set size to 60 %
set Score to 0
show variable Score
forever
  if key up arrow pressed?
    move 10 steps
    point in direction 0
  if key down arrow pressed?
    move 10 steps
    point in direction 180
  if key right arrow pressed?
    move 10 steps
    point in direction 90
  if key left arrow pressed?
    move 10 steps
    point in direction -90
  if on edge, bounce
```

```
when touching Mouse ?
change Score by 1
play sound Meow
wait 1 secs
```

```
when timer = 30
hide
stop all sounds
hide variable Score
```

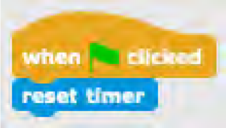
The girl



```
when ⚑ clicked
go to x: -9 y: 100
set size to 100 %
show
forever
  if touching Mouse ?
    switch to costume ballerina d
  else
    switch to costume ballerina a
```

```
when timer = 30
set size to 150 %
go to x: -185 y: -65
if Score = 0
  say You didn't catch any mice! for 3 secs
else
  say Congratulations! for 2 secs
  wait 1 secs
  say join You caught Score mice! for 3 secs
```

The background

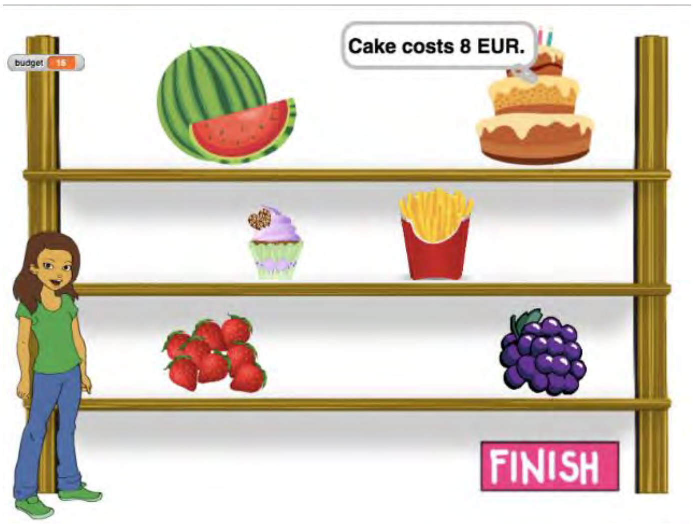


```
when ⚑ clicked
reset timer
```

Tools and Resources for the Teacher	● Whole activity in Snap!: https://snap.berkeley.edu/project?user=tadeja&project=Catch%20the%20mouse ● Website of free images: https://pixabay.com/ ● Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena. ● Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.
Resources/materials for the Students	● Template in Snap!: https://snap.berkeley.edu/project?user=tadeja&project=Catch%20the%20mouse_0 ● Website of free images: https://pixabay.com/ ● Instructions for student (C4G15_InstructionsForStudent.docx)

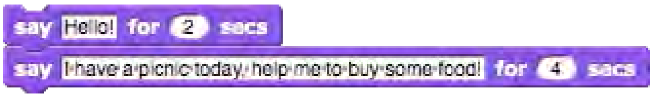
Learning Scenario 16 - Buying food for a picnic

Learning Scenario Title	Buying food for a picnic
Previous programming experience	Adding text for sprite Showing and hiding sprites Using operators Using variables Using string concatenation Using conditionals
Learning Outcomes	General learning outcomes: ● Variables ● Conditionals ● Operators Specific learning outcomes oriented on algorithmic thinking: ● Student uses variables for setting price for different sprites ● Student changes variables’ value, since the budget changes when the player buys food ● Student uses if statement for checking the availability of money ● Student uses operators for joining text - variables’ value - text ● Student uses operators for comparing prices and budget ● Student uses operators (subtraction) for changing variables’ value
Aim, Tasks and Short Description of Activities	Short description: The girl is going to a picnic and needs help with buying some food. She has 15 EUR and can not spend more. When she buys something, the budget’s value changes. It her budget is too low she can not buy the chosen food. Tasks: Students have to program three different sprites: a girl, food (which they can duplicate with slight changes) and finish button. Girl gives instructions, tells how much money the player has and at the end (with clicking on the finish button) she tells how many healthy and unhealthy products the player bought. Food tells its price when the mouse pointer hovers it. If the player has enough money, he can buy a

	product and the budget's value changes. Otherwise the food can not be bought. **Aim: Students will learn how to work with variables: setting different starting values, using conditionals to compare variables' value, changing variables' value, using variables for counting (un)healthy food. In addition, they will repeat adding text, joining texts and if statement.**
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Active learning, game-design based learning, problem solving
Teaching Forms	Individual work / Work in pairs
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) The girl is in a grocery buying food for picnic. She has 15 EUR. She can see the food's price when the mouse pointer hovers it and buy it with clicking on selected food. She can only buy food until she has enough money. Buy clicking on finish button, she tells how many healthy and unhealthy products the player bought. 

[Step 1]

This activity is meant as an individual work or work in pairs. A teacher gives some clues, explains some harder parts and helps when needed. Students choose background and add a main sprite, e.g. a girl. The girl gives some instructions at the beginning, e.g.:



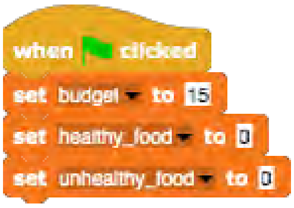
```
say [Hello!] for (2) secs
say [I have a picnic today, help me to buy some food!] for (4) secs
```

[Step 2]

In this game we will need few variables:

- *budget*, for setting the amount of money available,
- *healthy_food*, for counting how many healthy elements the player bought,
- *unhealthy_food*, for counting how many unhealthy elements the player bought,
- a variable for each food, e.g. *watermelon_price*, for setting the price of each food.

At the beginning, the *budget* variable is set to e.g. 15 (EUR). Other two variables are set to 0. This code can be added before the girl's code from [Step 1].



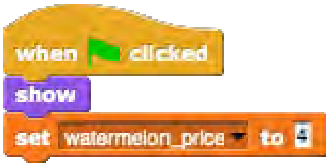
```
when green flag clicked
set budget to 15
set healthy_food to 0
set unhealthy_food to 0
```

[Step 3]

Students add a sprite (food) and choose its costume.

Food's (watermelon's) code needs three control events:

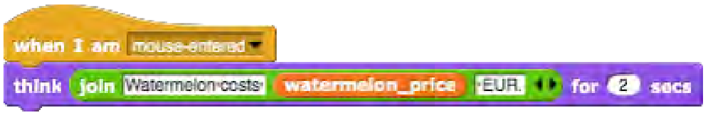
a) *When the green flag clicked*: to show and set the food's price. Let the variable's price be reasonably determined (of course, not 0, but bigger than 1).



```
when green flag clicked
show
set watermelon_price to 4
```

b) *When mouse-entered:* to tell the player how much does the product costs.

Students can use *Looks – thinking* block with use of joining text – variable's value – text, e.g.:

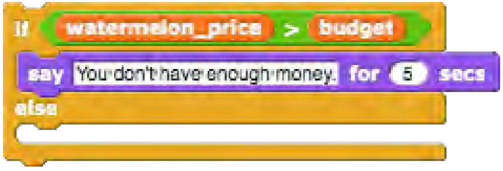


```
when I am [mouse-entered]
think (join [Watermelon costs ] (watermelon_price) [ EUR.]) for (2) secs
```

c) *When clicked*: here they have to make a small reflection.

1) In which case the player can buy the product and in which not?
2) What happens with the budget if he buys the food?
3) How do we count bought products?
4) What happens with the food on the shelf?

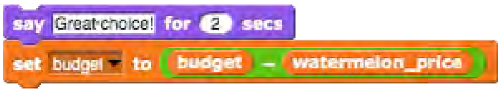
1) The player can buy the product if he has enough money. So students have to compare two variables: *budget* and *watermelon_price*. If the watermelon costs more than he has he can not buy it. Students can add some text to tell the player he can not buy this product.



```
if <(watermelon_price) > (budget)>
  say [You don't have enough money.] for (5) secs
else
```

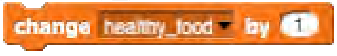
2) If the player has 15 EUR and buys a watermelon for 4 EUR, he now has 15 – 4 = 11 EUR. So the budget value is now: previous *budget value – watermelon_price*.

Students can add some text here as well.



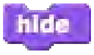
```
say [Great choice!] for (2) secs
set [budget] to ((budget) – (watermelon_price))
```

3) Counting the number of bought products will be realised with changing *healthy_food* variable by 1.



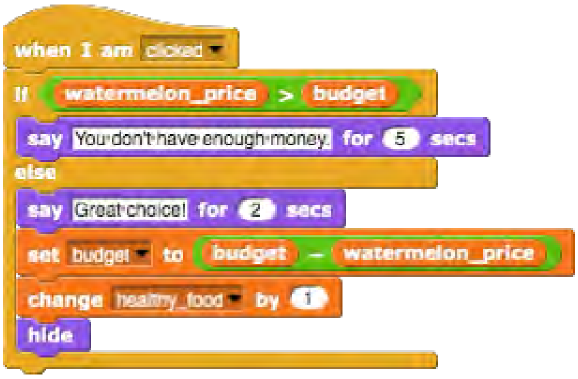
```
change [healthy_food] by (1)
```

4) When the food is clicked, it *hides*.



```
hide
```

One possible solution is:



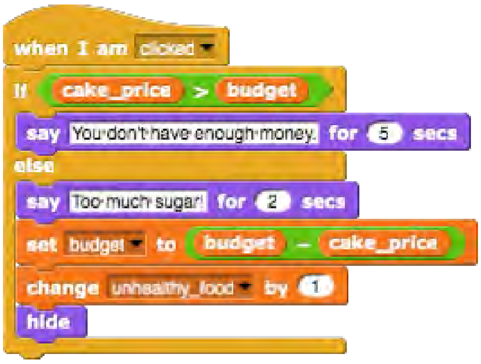
```
when I am clicked
if watermelon_price > budget
    say You don't have enough money. for 5 secs
else
    say Great choice! for 2 secs
    set budget to budget - watermelon_price
    change healthy_food by 1
    hide
```

[Step 4]

To have more food on the shelves, students can duplicate the watermelon sprite. Let's say the second food will be a cake. The code from [Step 3] then needs some changes. Students have to:

- Change the costume
- Make a new variable: *cake_price*
- Set the cake_price to some value
- Change in the code every block of *watermelon_price* with *cake_price*
- Change the response on buying the cake
- Replace *change healthy_food by 1* to *change unhealthy_food by 1*

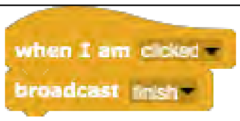
E.g., the *when clicked* code for the cake could be:



```
when I am clicked
if cake_price > budget
    say You don't have enough money. for 5 secs
else
    say Too much sugar! for 2 secs
    set budget to budget - cake_price
    change unhealthy_food by 1
    hide
```

[Step 5]

When the player finishes his buying, he clicks on the *Finish button*. To tell the program that player clicked on the button (finished with buying food), we broadcast a message.



```
when I am clicked
broadcast finish
```

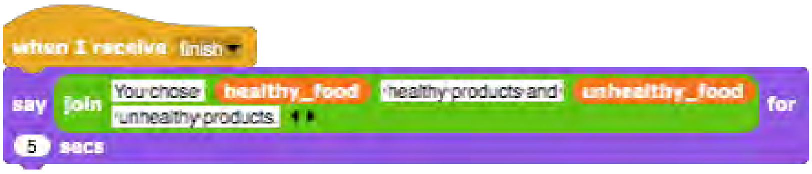
[Step 6]

At the end we return to girl's sprite.

When the player finishes his shopping, we want that the girl tells him how many healthy and unhealthy products he bought.

When the player clicks on the finish button, a message *finish* is sent.

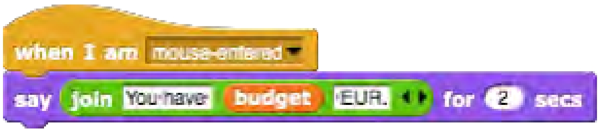
When the girl receives the message *finish*, she tells, e.g. "You chose X healthy products and Y unhealthy products".



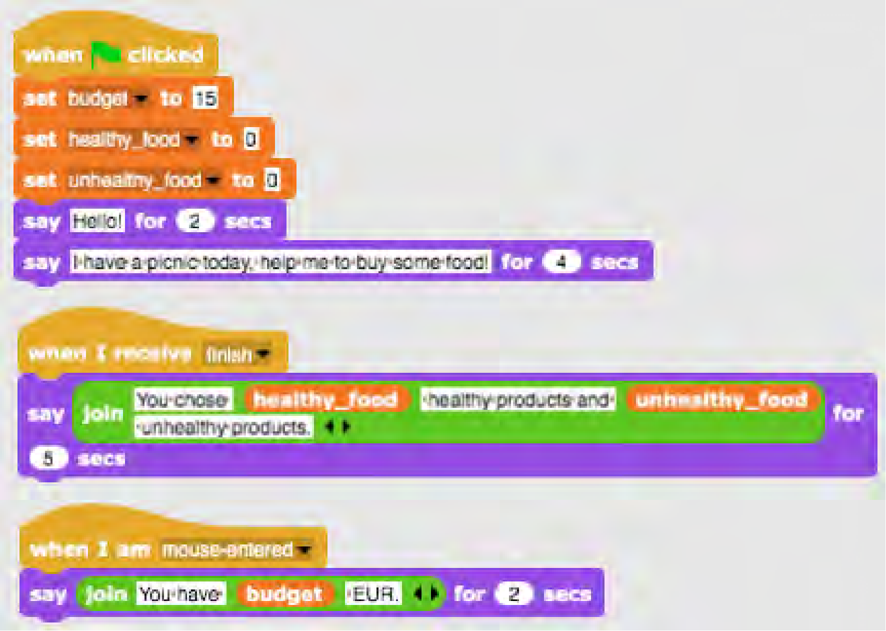
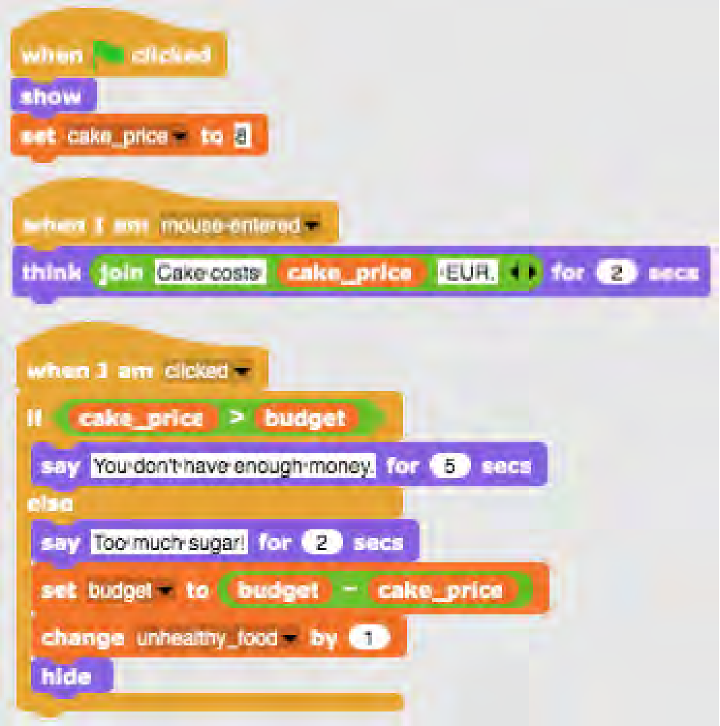
```
when I receive finish
say join You chose healthy_food healthy products and unhealthy_food unhealthy products for 5 secs
```

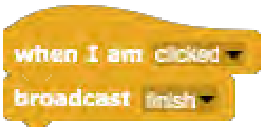
[Step 7]

Anytime during the game, the player can check his budget by mouse-enter the girl. E.g., she can say / think something like:



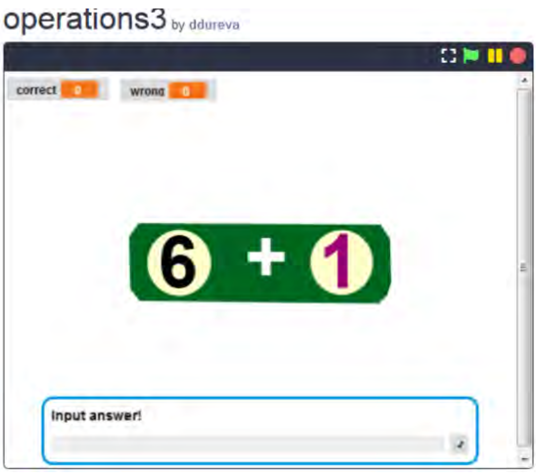
```
when I am mouse-entered
say join You have budget EUR. for 2 secs
```

	[Final Code]
	Girl
	
	`when 🏁 clicked`
	`set budget ▾ to 15`
	`set healthy_food ▾ to 0`
	`set unhealthy_food ▾ to 0`
	`say Hello! for 2 secs`
	`say I have a picnic today, help me to buy some food! for 4 secs`
	
	`when I receive finish ▾`
	`say join You chose  healthy_food  healthy products and  unhealthy_food  unhealthy products. for 5 secs`
	
	`when I am mouse-entered ▾`
	`say join You have  budget  EUR. for 2 secs`
	
	Food
	
	`when 🏁 clicked`
	`show`
	`set cake_price ▾ to 8`
	
	`when I am mouse-entered ▾`
	`think join Cake costs  cake_price  EUR. for 2 secs`
	
	`when I am clicked ▾`
	`if cake_price > budget`
	`    say You don't have enough money. for 5 secs`
	`else`
	`    say Too much sugar! for 2 secs`
	`    set budget ▾ to budget - cake_price`
	`    change unhealthy_food ▾ by 1`
	`    hide`

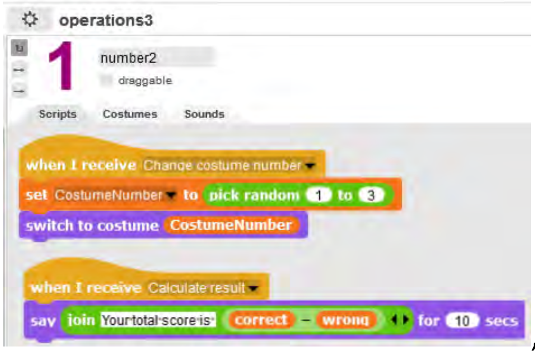
	Finish Button  when I am clicked broadcast finish [Additional tasks] Students can add additional tasks according to their wishes or they can follow the tasks below: ● Change the game so you can buy each food 3 times. ● Give more money to the player at the beginning. ● At the end the girl tells also how many products you bought. E.g. “You bought 2x watermelon, 1x grapes, 2x fries”.
Tools and Resources for the Teacher	● Whole activity in Snap!: https://snap.berkeley.edu/project?user=mateja&project=Buying%20food%20for%20a%20picnic ● Activity in Snap! with additional tasks (possible solution): https://snap.berkeley.edu/project?user=mateja&project=Buying%20food%20for%20a%20picnic%20%2B%20Add.%20Task ● Lajovic, S. (2011). *Scratch. Nauči se programirati in postani računalniški maček.* Ljubljana: Pasadena. ● Vorderman, C. (2017). *Računalniško programiranje za otroke.* Ljubljana: MK.
Resources/materials for the Students	Instructions for student (C4G16_InstructionsForStudent.docx)

Learning Scenario 17 - Operations

Learning Scenario Title	Operations
Previous programming experience	Using variables for counting points and to choose costume of the stage and of the sprite; Using random number to choose stage décor and costume for the sprite. Using repeat loop Using conditionals Using operations for comparison Using sensing for dialogue (ask ….and wait) Using broadcast events
Learning Outcomes	General learning outcomes: ● Variables ● Conditionals ● Loop ● Sensing blocks ● Broadcast events Specific learning outcomes oriented to algorithmic thinking: ● Students use *variables for points counting and for costumes of the stage and of the sprite keeping.* ● Students use variables for points counting ● Students initialise variables for points counting ● Students use conditionals and logical operations ● Students use broadcast event for changing the sprite and calculating the final result.
Aim, Tasks and Short Description of Activities	Short description: Let's check while playing a game whether the player has mastered the arithmetic operations in Snap!. The rules are as follows: Ten times an arithmetic operation with the first operand of 6 is randomly selected, and the second operand is randomly selected to be a number from 1 to 3. The player must enter the correct answer. The right and wrong answers are counted. At the end of the game the correct result is reported.

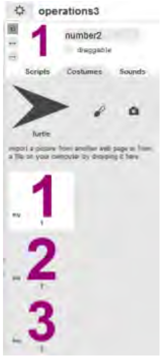
	Task: Students have to define the scenery /stage décor/ and the sprite's costume; to plan the required variables, determine which blocks they need. At the finish they have to create the codes to the stage and the sprite. Additional tasks could be to: ● To assign the sprite, depending on the result, to say: *"Good for you!"* or *"You don't know well the arithmetic operations in Snap! yet!"* **Aim: Students will improve their previously acquired knowledge on variables, random numbers, loops, broadcast.**
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Active learning (discussions, experiment with a previously prepared game), game-design based learning, problem-solving,
Teaching Forms	Individual work / Work in pairs/ Frontal work with the whole class
Teaching summary	(Motivation-Introduction, Implementation, Reflection and Evaluation) 1. The teacher poses the problem regarding the need for a game to determine if the arithmetic operations in Snap! have been mastered and demonstrates the project. https://snap.berkeley.edu/project?user=ddureva&project=operations3.  operations3 by ddureva correct 0 wrong 0 6 + 1 Input answer!

2. The teacher discusses how to formulate the condition of the task. The task is formulated.
   Ten times in a random manner, an arithmetic operation is selected with the first operand 6 and a second operand is also randomly selected, from numbers 1 to 3. The player must enter the correct answer. The right and wrong answers are counted. The result is reported at the end of the game.
3. The variables are commented, as well as the way they are defined, initialized and changed.
4. Random number commands, arithmetic and logic operations, broadcast event commands are revised.
5. It is debated whether the base code is to the stage or to the sprite. In the example, the main code is to the scene, and the sprite' code has scripts for changing the costume and calculating the end result.

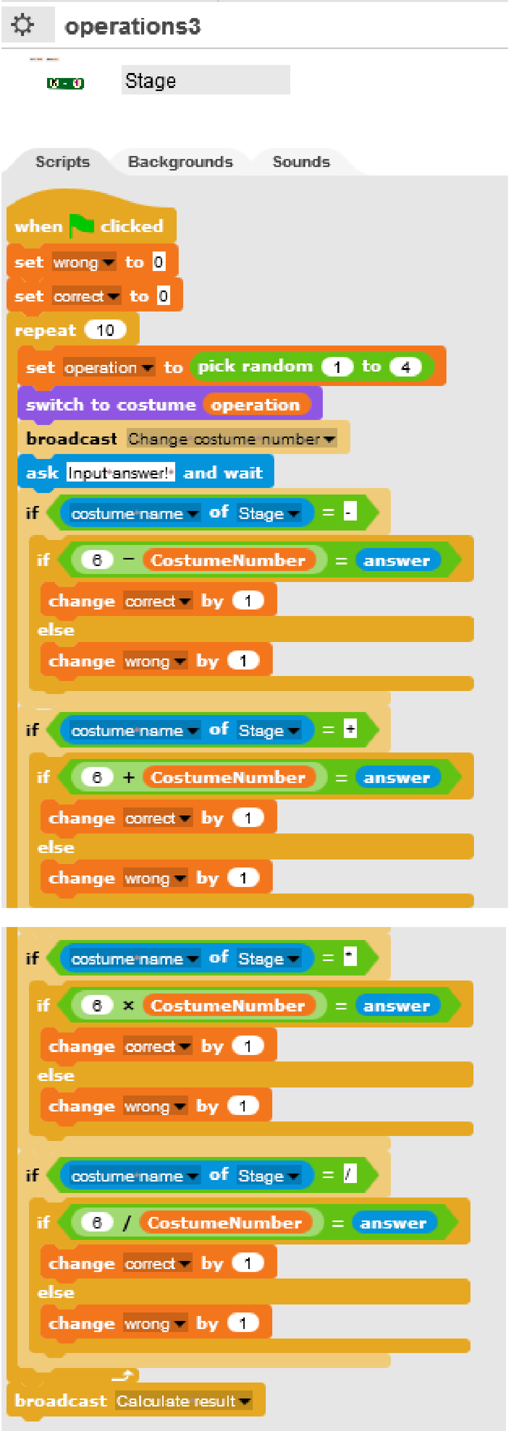
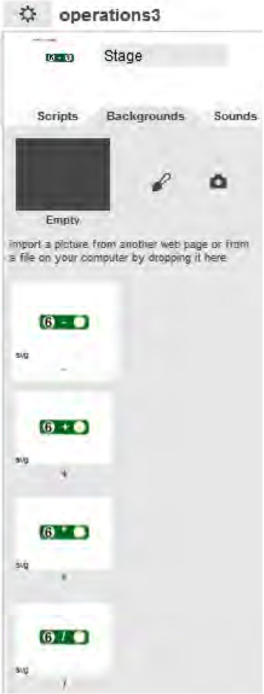


```
when I receive Change costume number
set CostumeNumber to pick random 1 to 3
switch to costume CostumeNumber

when I receive Calculate result
say join Your total score is (correct - wrong) for 10 secs
```

,



operations3
1
number2
draggable
Scripts
Costumes
Sounds
Turtle
Import a picture from another web page or from a file on your computer by dropping it here
1
2
3

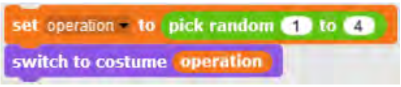
Code to the scene	Scenery/stage décor/
	

```
operations3
Stage
Scripts   Backgrounds   Sounds

when [flag] clicked
set wrong to 0
set correct to 0
repeat 10
    set operation to pick random 1 to 4
    switch to costume operation
    broadcast Change costume number
    ask Input answer! and wait
    if costume name of Stage = -
        if 6 − CostumeNumber = answer
            change correct by 1
        else
            change wrong by 1
    if costume name of Stage = +
        if 6 + CostumeNumber = answer
            change correct by 1
        else
            change wrong by 1
    if costume name of Stage = *
        if 6 × CostumeNumber = answer
            change correct by 1
        else
            change wrong by 1
    if costume name of Stage = /
        if 6 / CostumeNumber = answer
            change correct by 1
        else
            change wrong by 1
broadcast Calculate result
```

operations3
Stage
Scripts
Backgrounds
Sounds
Empty
import a picture from another web page or from a file on your computer by dropping it here
6 - O
6 + O
6 * O
6 / O

The scene code contains the initializations for the right and wrong answer variables.

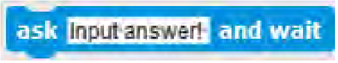
To select an operation the following commands are used:



```
set operation to pick random 1 to 4
switch to costume operation
```

The choice of costume for the sprite is made by broadcast to Number Sprite. The selected costume number is stored in the Costume Number variable, which is defined for all objects in the project and can therefore be used in the stage code.

Once the scenery /stage décor/ and the sprite costume have been selected at random, a question is asked to the player to enter the correct response for the operation with the following command:



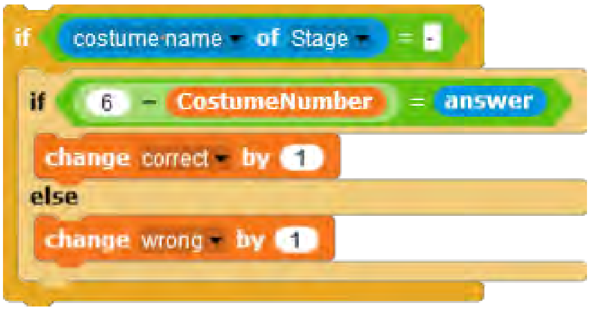
```
ask Input answer! and wait
```

The entered response is compared with the result of the selected operations. The following command is used:

if (conditional)

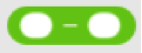
else6

If operation “-“ is selected, then a check is performed whether the result of 6 - "Sprite’s costume number" matches the answer. If they match, the *correct* variable increases, otherwise the variable for the count of incorrect answers increases.



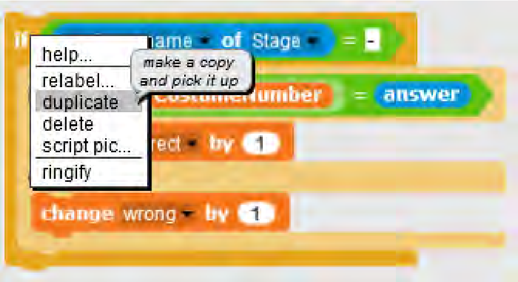
```
if costume name of Stage = -
  if 6 - CostumeNumber = answer
    change correct by 1
  else
    change wrong by 1
```

For the rest of the commands the script is similar, the difference is in the selected operation.

To avoid repeated code ordering for the rest of the operations, students may be taught how to copy part of the code and change the arithmetic operation in :

Code copying:

1. Click with the right mouse button on the script
2. Choose duplicate

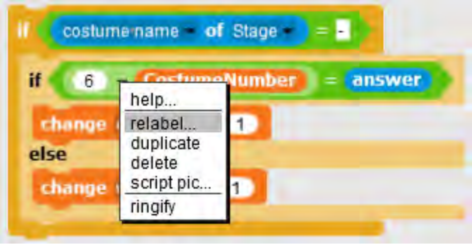


3. Use the mouse to place the duplicated script at the corresponding location.

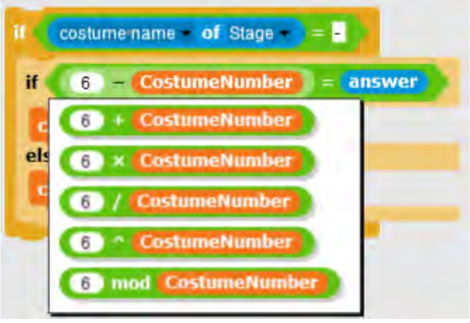
   At the teacher's discretion, students may be tasked with figuring out how to copy some of the code themselves.

Changing the operation.

1. Click with the right mouse button on the operation sign. Context menu will appear.



2. Choose relabel. A list of operations will appear.

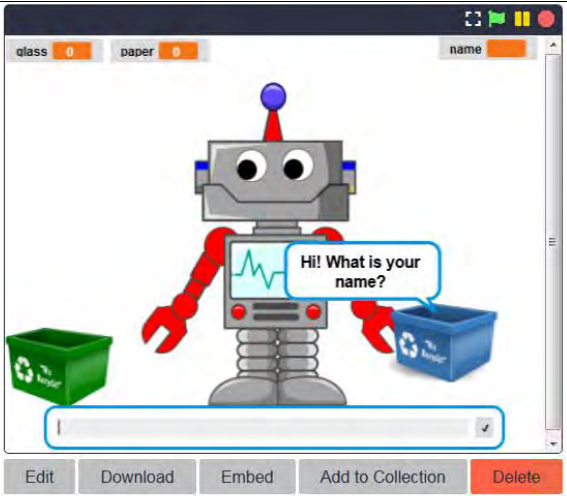
	 if costume name of Stage = if 6 − CostumeNumber = answer 6 + CostumeNumber 6 × CostumeNumber 6 / CostumeNumber 6 ^ CostumeNumber 6 mod CostumeNumber 3. Choose operation Note: If students' age and knowledge of arithmetic operations allow the task may be expanded with operations, exponentiation (^) and modulo operation (mod). 4. Students work in teams creating their own scenery/stage décor/ and costumes for the sprite. If there are time constraints, a "half-backed" project can be used that contains the stage and the sprite.
Tools and Resources for the Teacher	Whole activity in Snap!: https://snap.berkeley.edu/project?user=ddureva&project=operations3 Whole activity in Scratch: - Дурева Д., М. Касева, Г. Тупаров, Компютърно моделиране, 4. клас, Просвета, 2018, София (Dureva, D., M. Kaseva, G. Tuparov, Kompyutarno modelirane, 4. klas, Prosveta, 2019, Sofia)
Resources/materials for the Students	- Half-baked activity in Snap! https://snap.berkeley.edu/project?user=ddureva&project=operations_half - Instructions for student (C4G17_InstructionsForStudent.docx)

Learning Scenario 18 - Recycling

Learning Scenario Title	Recycling
Previous programming experience	Showing and hiding a sprite Using variables for counting points Using loop forever Using conditionals Using operations for comparison Using Sensing of colors
Learning Outcomes	General learning outcomes: ● Variables ● Conditionals ● Loop ● Point in direction ● Sensing blocks ● Code refactoring Specific learning outcomes oriented to algorithmic thinking: ● Students use *wait until and logical operations* to end the game ● Students use *wait until,* and *block* to change the stage ● Students use variables to count points ● Students use conditionals and logical operations ● Students compare the codes of the similar sprites. ● Students make code refactoring ● Students use positioning of sprites (in an additional task use random positioning)
Aim, Tasks and Short Description of Activities	Short description: Someone has dumped garbage in front of the school. The player is asked to help separate garbage collection by sorting it for recycling paper and glass. When the garbage is placed in the correct container, the garbage is hidden. If the garbage is placed in the wrong container, the relevant message - *"This is not a paper container"* or *"This is not a glass container"* appears and the garbage returns to its original

Co-funded by the Erasmus+ Programme of the European Union

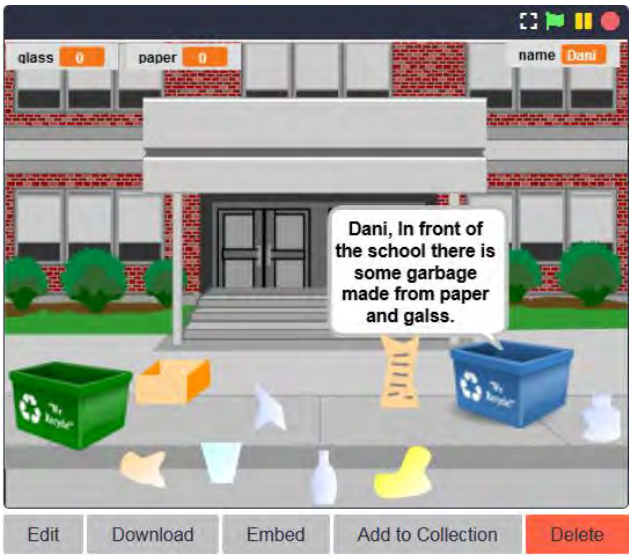
	position. The game ends when all the garbage is put in the right containers. Task: Students have to explore the codes of the stage and sprites, compare codes of waste-paper and waste-glass type of sprites, add new sprites and scripts, and change the script in the stage with respect to newly added sprites. Additional tasks could be to: ● change position of the waste sprites with random choice of coordinates of the sprites; ● decrease number of stages and extract robot as a separate sprite. (The robot is part of the background of the stage). **Aim: Students will improve their previously acquired knowledge and will extend the game scenario with new objects, code and changing code with respect to new sprites. They will be trained in code refactoring.**
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Active learning (discussions, experiment with a previously prepared game), game-design based learning, problem-solving,
Teaching Forms	Individual work /Work in pairs/Frontal work with the whole class
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) 1. The teacher poses the problem of separate garbage collection and comments on the colors of the bins for different types of garbage - blue for paper, green for plastic. 2. It sets the students to play the game and describe in words: How many scenes do they watch and how many sprites (characters)? How does the game begin? Which sprite asks for the player's name? How many variables are used and how are they named? What happens when paper is placed in a glass container and what when it is placed in a paper container?



Hi! What is your name?

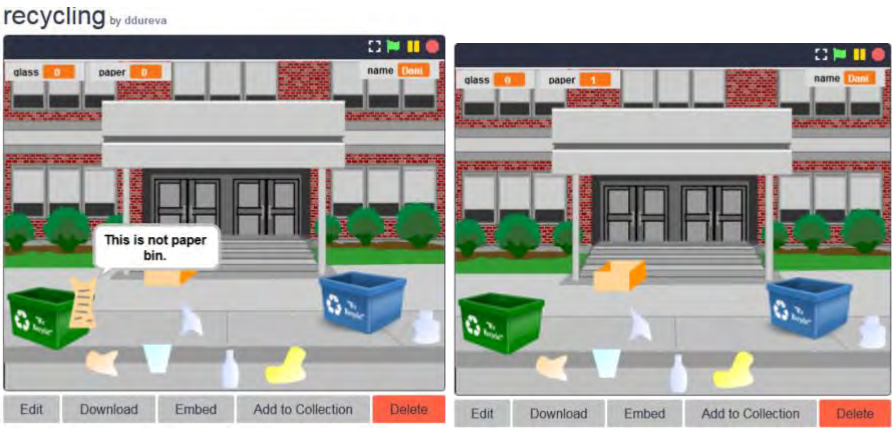
1. Updating the studied commands

Commands for engaging in dialogue with the user are recalled. A comment is made about changing scenes - Scene 1 with the Robot, Scene 2 with the school and junk and Scene 3 with the Robot and the caption Bravo!. Possible scene change commands are discussed.



Dani, In front of the school there is some garbage made from paper and galss.

It is discussed that checking for the proper placement of garbage in a container should be carried out with a conditional block and blocks with touch conditions of the Sensing group. A verbal description is given: If a piece of paper garbage touches the paper waste bin, the garbage is hidden (placed in the correct bin) and the points for collected paper waste are increased by 1. If a piece of paper garbage touches the glass bucket, it "says" - *"This is not a paper container."* The same happens with glass garbage.

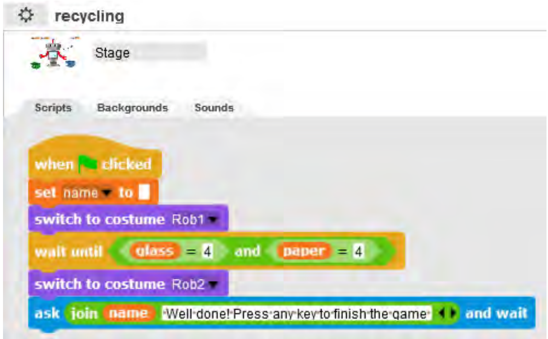


2. Examining scene and character codes.

After discussing the possibilities for solving the problem, the codes for the scene and the characters are discussed.

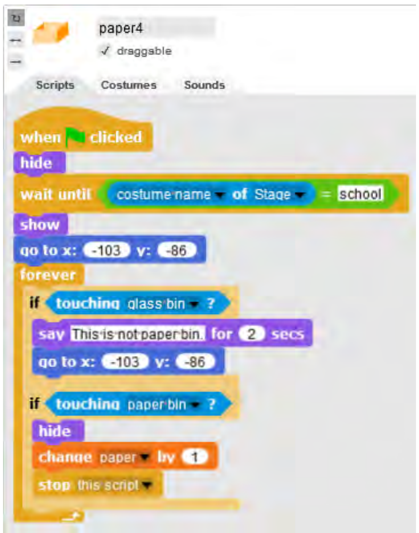
The scene code is commented with the emphasis on:

- Setting the initial value of the *name* variable and using it in a dialogue with the user;
- changing the stage scenery (costumes) and the condition for finishing the game.



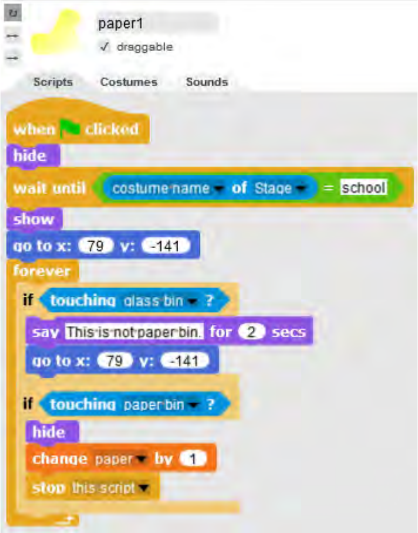
```
when ⚑ clicked
set name to
switch to costume Rob1
wait until glass = 4 and paper = 4
switch to costume Rob2
ask join name Well done! Press any key to finish the game ◂▸ and wait
```

When looking at character codes, it is advisable to show them on a single slide or to give in print pieces of junk paper and junk glass two codes each. A comparison is made between common and different elements in the codes.



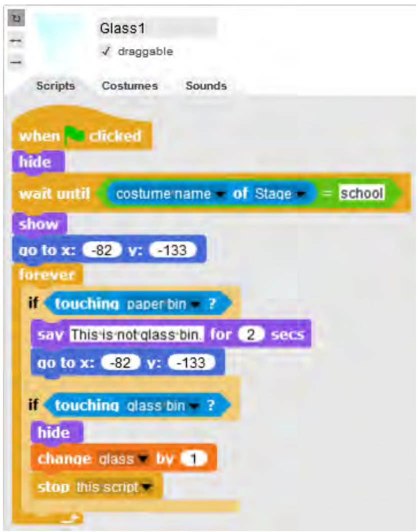
paper4 ✓ draggable

```
when [flag] clicked
hide
wait until (costume name of Stage) = school
show
go to x: -103 y: -86
forever
    if touching glass bin ?
        say This is not paper bin. for 2 secs
        go to x: -103 y: -86
    if touching paper bin ?
        hide
        change paper by 1
        stop this script
```



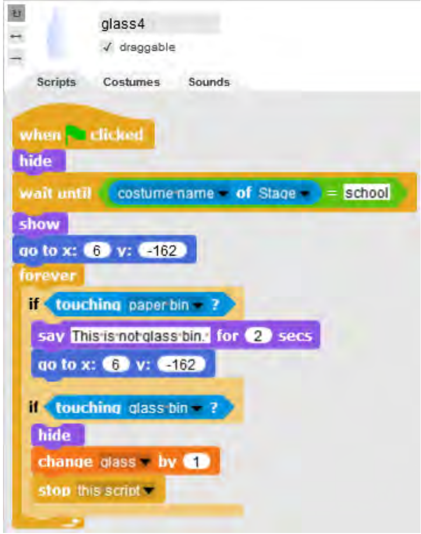
paper1 ✓ draggable

```
when [flag] clicked
hide
wait until (costume name of Stage) = school
show
go to x: 79 y: -141
forever
    if touching glass bin ?
        say This is not paper bin. for 2 secs
        go to x: 79 y: -141
    if touching paper bin ?
        hide
        change paper by 1
        stop this script
```



Glass1 ✓ draggable

```
when [flag] clicked
hide
wait until (costume name of Stage) = school
show
go to x: -82 y: -133
forever
    if touching paper bin ?
        say This is not glass bin. for 2 secs
        go to x: -82 y: -133
    if touching glass bin ?
        hide
        change glass by 1
        stop this script
```



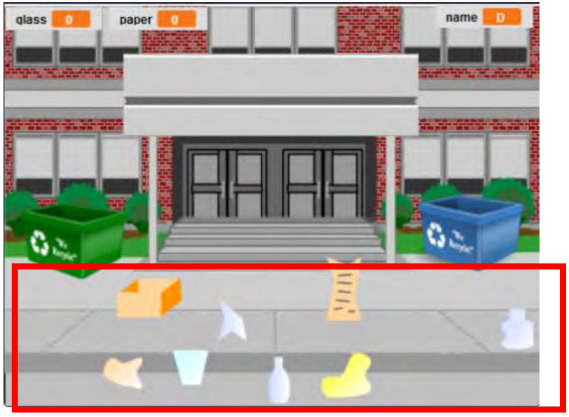
glass4 ✓ draggable

```
when [flag] clicked
hide
wait until (costume name of Stage) = school
show
go to x: 6 y: -162
forever
    if touching paper bin ?
        say This is not glass bin. for 2 secs
        go to x: 6 y: -162
    if touching glass bin ?
        hide
        change glass by 1
        stop this script
```

3. Setting up a task to complete the game with two new sprites - paper garbage and glass garbage, assigning a code to them and changing the scene and garbage container codes.

How to create the two new sprites is discussed. Options - Duplicate existing ones and edit in *Snap!*, create new ones in a graphics editor, or search for freely distributed images on the Internet and import them into the game.

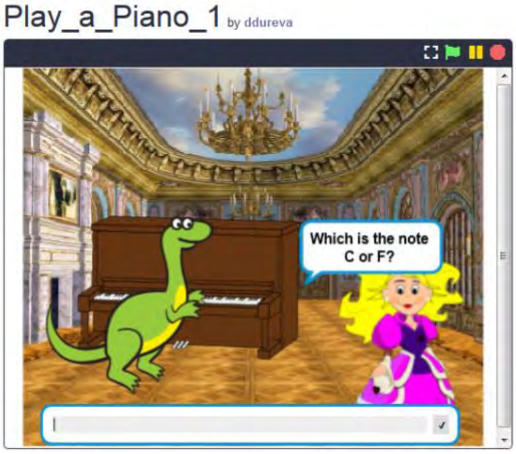
It is also necessary to comment on the changes to the scene code regarding the game's completion.

	Whether it is possible to set the initial values of the variables not in the code of the two containers, but in the code of the scene and make an adjustment accordingly should be discussed as well. At the teacher's discretion, the condition of the task can be complicated: ● the garbage should be spread at any suitable place when starting the game. It is good to note here that the coordinates in which the garbage can be dispersed should be limited so that it is in a realistic place. For example, bounded by the coordinates of the red rectangle.  ● Introduce a new Robot Sprite and reduce the number of scenery elements on the stage. Write the appropriate code to the Robot so that it engages in dialogue with the player rather than a blue container sprite.
Tools and Resources for the Teacher	Whole activity in Snap!: https://snap.berkeley.edu/project?user=ddureva&project=recycling Whole activity in Scratch: ● Дурева Д., М. Касева, Г. Тупаров, Компютърно моделиране, 4. клас, Просвета, 2018, София (Dureva, D., M. Kaseva, G. Tuparov, Kompyutarno modelirane, 4. klas, Prosveta, 2019, Sofia)
Resources/materials for the Students	● Half-baked activity in Snap! https://snap.berkeley.edu/project?user=ddureva&project=recycling ● Instructions for student (C4G18_InstructionsForStudent.docx)

Learning Scenario 19.1 - Play a piano

Learning Scenario Title	Play a piano
Previous programming experience	Using variables for counting points; Using event *When I am pressed*; Using repeat loop; Using conditionals; Using broadcast events to change scenery/stage decor/ and to manage sprite's activities;
Learning Outcomes	General learning outcomes: ● Variables; ● Conditionals; ● Loop; ● Broadcast events; ● Sounds; ● Programming music; Specific learning outcomes oriented on algorithmic thinking: ● Students use *variables for points counting;* ● Students initialise variables for points counting; ● Students use conditionals to estimate achieved points; ● Students use *broadcast event* for changing of the scenery/stage decor/and for sprites' activities; ● Students use blocks from the *Sound* group to compose melodies; ● Students identify need of repeat *loop* to decrease number of blocks in the scripts; ● Students extend the functionality of the game;
Aim, Tasks and Short Description of Activities	Short description: Let's get into the wonderful world of Queen Mary. She invites the player to her palace to listen to some music. In the ballroom, her little dinosaur friend Dino plays the piano. In the game Dino plays a few musical tones and the players must recognise which tone it is. If they guess right, they get one point for the right answer, if they don't know, they get point reduction for the wrong answer. After identifying the tones, a more

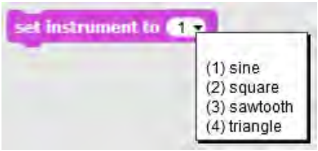
	complex task is set: Dino plays a tune, and the player must recognise which song it is. For a properly identified tune, the player gets 5 points. Task: Students use a half-backed file with scenery /stage decor/ and sprites' costumes. They need to plan the necessary variables, determine what blocks they need; get acquainted with the blocks of the *Sound* group and the way to play the notes. Create scripts to play several tunes. **Aim: Students will learn about melodies coding and playing and will improve their previously acquired knowledge about variables, loops, conditional, broadcast and other events.**
Duration of Activities	90 minutes
Learning and Teaching Strategy and Methods	Active learning (discussions, experiment with a previously prepared game), game-design based learning, problem-solving,
Teaching Forms	Individual work / Work in pairs/ Frontal work with whole class
Teaching summary	(Motivation-Introduction, Implementation, Reflection and Evaluation) 1. The teacher sets the task of creating the game. The means by which the task can be completed are discussed. It is concluded that they are not currently aware of the available code-writing resources to program a tune. 2. The teacher demonstrates part of the game by programming a tune. https://snap.berkeley.edu/project?user=ddureva&project=Play_a_Piano_1



3. The teacher shows the code and explains how the *Sound* group commands can be used.

In Snap! sounds from the built-in library can be used, as well as files from the computer, or music tones played on various instruments.

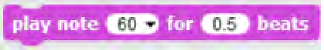
To select a tool, use the command:



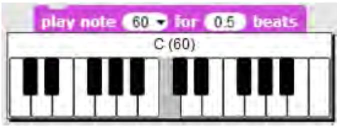
, Note: There are many more tools in Scratch.

Students test the sound of various instruments.

4. The teacher explains the way to set the musical notes:

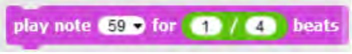
The command  is used. In it, the first number sets the note, and the second number describes how long the note is to be played.

When you click on the arrow next to the first number, a piano keyboard appears and a note can be selected from it. This piano keyboard spans two octaves.

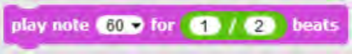


C	C#	D	Eb	E	F	F#	G	G#	A	Bb	B	C
60	61	62	63	64	65	66	67	68	69	70	71	72

The duration of each note is set by the numbers 1 - whole note, 0.5 - half, 0.25 – a quarter. (For students who have not studied real numbers, the decimal fractions may be presented in the form of ordinary fractions: ½, ¼, 1/8, etc.)

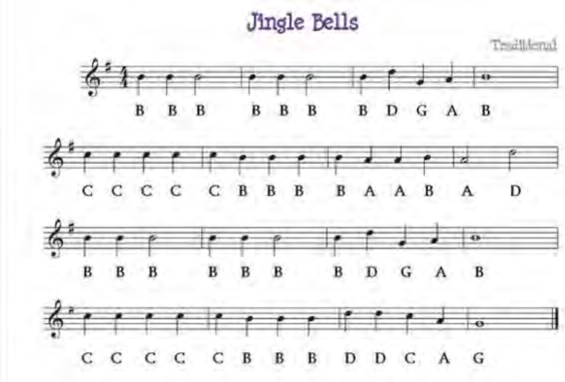

play note 59 for 1 / 4 beats

,


play note 60 for 1 / 2 beats

At the teacher's discretion, students can experiment with the commands and establish the dependencies themselves.

5. The *Jingle Bells* tune script is discussed using the musical notification as well.

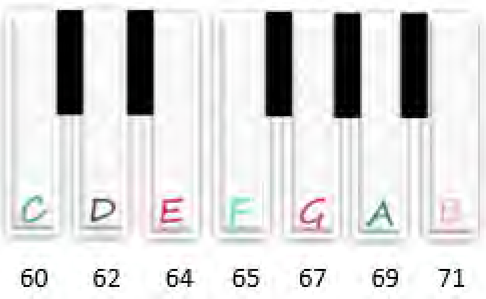
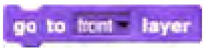


6. The task is set to reduce the number of lines in the code that are repeated. The command to be used (*repeat loop*) is discussed. Students are divided into teams that are required to create the game, set at the beginning of the lesson. Each team discusses the game scenario and describes the game plan in the description sheet (Attached SNAP_Program_Design_and_Planning Worksheet.docx) Tables can be added to the description for a detailed description of actions in the stages and sprites. A condition may be added for the dinosaur to dance while playing. (The dinosaur has several costumes in the pre-prepared file).

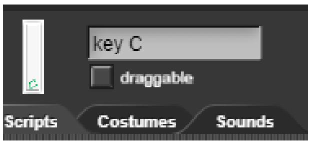
	7. The teacher can display some parts of scenarios from the file. https://snap.berkeley.edu/project?user=ddureva&project=PlayAPiano
Tools and Resources for the Teacher	Whole activity in Snap!: https://snap.berkeley.edu/project?user=ddureva&project=Play_a_Piano_1 https://snap.berkeley.edu/project?user=ddureva&project=PlayAPiano
Resources/materials for the Students	● Half-baked activity in Snap!: https://snap.berkeley.edu/project?user=ddureva&project=Play_a_Piano_Half_backed ● Instructions for student (C4G19.1_InstructionsForStudent.docx)

Learning Scenario 19.2 - Play a piano

Learning Scenario Title	Play a piano
Previous programming experience	Using loop repeat Using variables Using conditionals
Learning Outcomes	General learning outcomes: ● Conditionals ● Loops Specific learning outcomes oriented on algorithmic thinking: ● Student uses loop repeat for playing music ● Student uses code to make sprites react to input ● Student adds sounds to a sprite ● Student uses code to change a sprite's costume
Aim, Tasks and Short Description of Activities	Short description: Student has to play a song on a piano according to the given notes. Tasks: Students should program the piano keys - each key needs to play a particular tone. On the stage, two different buttons have to be shown, one to display the notes and the other to play the melody. **Aim: Students will learn how to play music and change costume by clicking on a sprite.**
Duration of Activities	45 minutes
Learning and Teaching Strategy and Methods	Active learning, game-design based learning, problem solving
Teaching Forms	Individual work / Work in pairs

Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) At the beginning, a piano is shown on a stage. Next to the piano should be two buttons. Click on the first button should display the notes and words of the song, and click on the second button should play the melody that needs to be repeated. Additionally, next to the piano should be the "X" button, which will restart the project. [Step 1] Open the program *Play a piano.* The program contains all the backgrounds and sprites needed for this task. The background is given, and also the sprite for key C and one black key. Students have to duplicate the key C, move it to the right position, and rename it. The keys should be in the following order: C, D, E, F, G, A, B. The keyboard should look as in the following picture and reproduce tones written below the keys:  C D E F G A B 60 62 64 65 67 69 71 Duplicate sprite “black_key” 4 times to get 5 black keys, and name them black key 2 to black key 5. Place new black keys between keys D and E, F and G, G and A, and A and B. If the black key is hidden behind the white keys, use the following code  go to front layer Do the same for the key B and place them at the end of the keyboard.

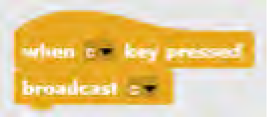
Uncheck the "draggable" button, so the key sprites cannot be moved while playing.



key C
□ draggable

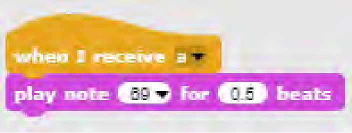
[Step 2]

Enable playing tones by pressing sprites. For the key "C", add hat block "When I am clicked" and allow it to broadcast the "c" message.



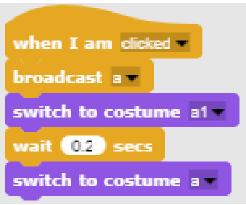
```
when c key pressed
broadcast c
```

To produce sound when a key is pressed, add hat block "When I receive c", and add a play note 60 for 0.5 beats.



```
when I receive a
play note 69 for 0.5 beats
```

To highlight which key is pressed, the costume of that sprite should be temporarily changed. Import c1 costume to the sprite C. In the "When I am clicked" block, change the costume to c1 for 0.2 seconds, then return to costume c.



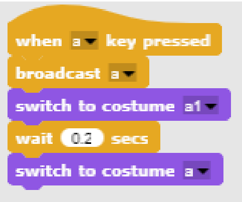
```
when I am clicked
broadcast a
switch to costume a1
wait 0.2 secs
switch to costume a
```

[Step 3]

Repeat step two for all remaining white keys.

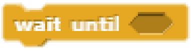
[Step 4]

To play the piano using the keyboard, add a "When c key pressed" block to key c sprite, and copy the rest of the code from the "When I am clicked" block.



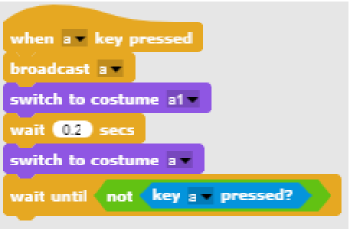
```
when a key pressed
broadcast a
switch to costume a1
wait 0.2 secs
switch to costume a
```

Notice if the c key on the keyboard is held down, the sound will be repeated as long as the key is pressed. This happens because the message "a" is repeatedly broadcasting. To stop broadcasting a message, on the end of code add a "wait until" block from Control pallet.



```
wait until
```

To finish broadcast a message, use the "not" operator and add to them a block "key a pressed".

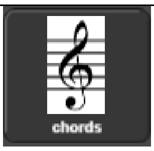


```
when a key pressed
broadcast a
switch to costume a1
wait 0.2 secs
switch to costume a
wait until not key a pressed?
```

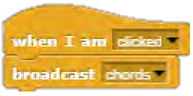
Do the same for all remaining white keys.

[Step 5]

Create a new sprite and import a picture of a violin key as a costume. This will be a button for displaying the words and notes to be played.

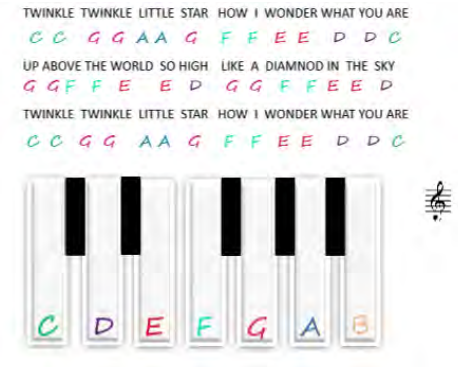


To display the notes, enable the broadcast of the "chords" message when the button is clicked.

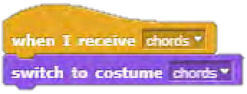


```
when I am clicked
broadcast chords
```

Import a new costume "chords" for the stage.


TWINKLE TWINKLE LITTLE STAR HOW I WONDER WHAT YOU ARE
C C G G A A G F F E E D D C
UP ABOVE THE WORLD SO HIGH LIKE A DIAMNOD IN THE SKY
G G F F E E D G G F F E E D
TWINKLE TWINKLE LITTLE STAR HOW I WONDER WHAT YOU ARE
C C G G A A G F F E E D D C
C D E F G A B

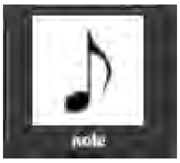
Add a code that enables the stage to change the costume to "chords" when it receives a message "chords".



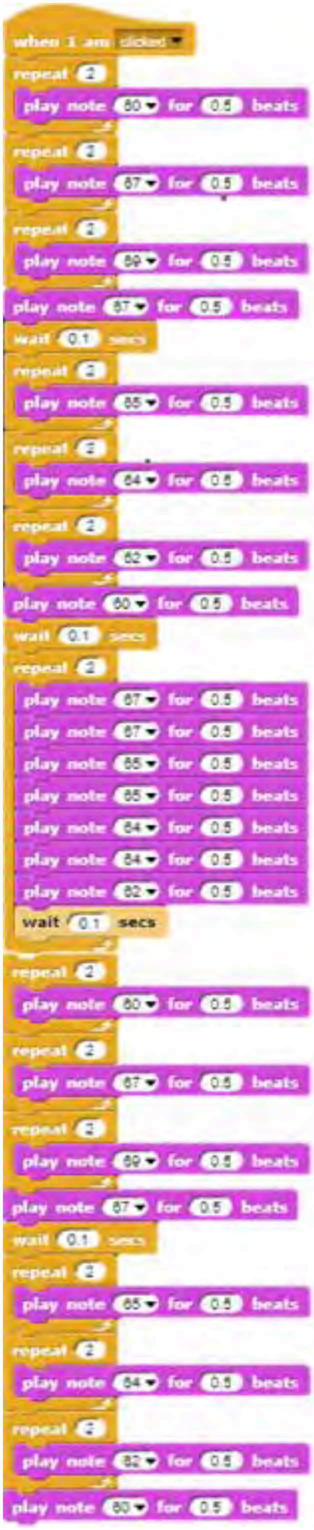
```
when I receive chords
switch to costume chords
```

[Step 6]

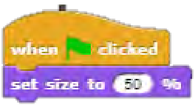
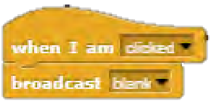
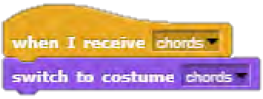
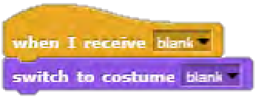
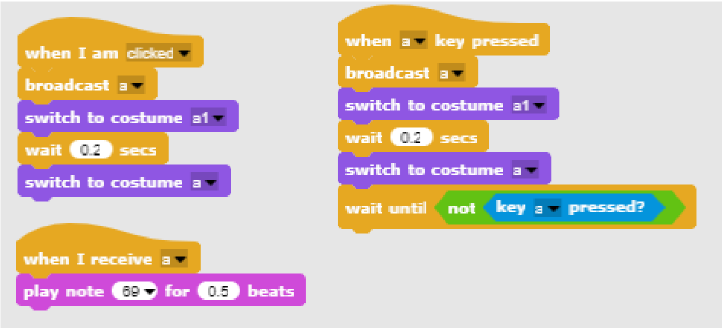
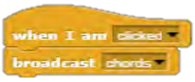
Find the sprite with a note as a costume. This will be a button for playing the song that needs to be repeated.



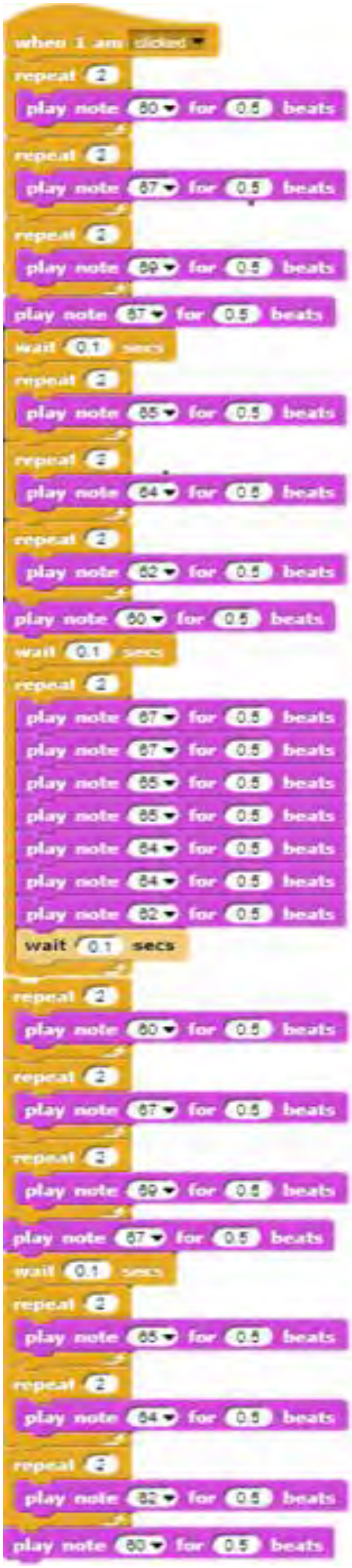
The code is written for the first two verses of the song, and you have to write the code for the other verses. It is the same song as displayed in the sheet music.



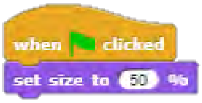
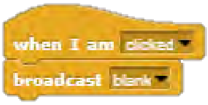
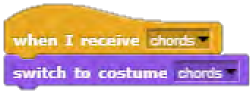
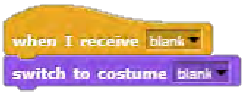
```
when I am clicked
repeat 2
    play note 60 for 0.5 beats
repeat 2
    play note 67 for 0.5 beats
repeat 2
    play note 69 for 0.5 beats
play note 67 for 0.5 beats
wait 0.1 secs
repeat 2
    play note 65 for 0.5 beats
repeat 2
    play note 64 for 0.5 beats
repeat 2
    play note 62 for 0.5 beats
play note 60 for 0.5 beats
wait 0.1 secs
repeat 2
    play note 67 for 0.5 beats
    play note 67 for 0.5 beats
    play note 65 for 0.5 beats
    play note 65 for 0.5 beats
    play note 64 for 0.5 beats
    play note 64 for 0.5 beats
    play note 62 for 0.5 beats
    wait 0.1 secs
repeat 2
    play note 60 for 0.5 beats
repeat 2
    play note 67 for 0.5 beats
repeat 2
    play note 69 for 0.5 beats
play note 67 for 0.5 beats
wait 0.1 secs
repeat 2
    play note 65 for 0.5 beats
repeat 2
    play note 64 for 0.5 beats
repeat 2
    play note 62 for 0.5 beats
play note 60 for 0.5 beats
```

	[Step 7] Create a new button X that will reset the project (without the notes). Create a new sprite - reset, choose the costume “X” and set its size to 50%. Enable the broadcast of the "blank" message when the button is pressed.  when ⚑ clicked set size to (50) %  when I am [clicked ▾] broadcast [blank ▾] Ad a hat block “When I receive” to the stage to change the costume to “blank” after receiving the "blank" message.  when I receive [chords ▾] switch to costume [chords ▾]  when I receive [blank ▾] switch to costume [blank ▾] [Additional tasks] Students can add additional tasks according to their wishes or they can follow the tasks below: ● Duplicate the sprite Note (and change its position on the background) and write a program for another song. ● Add a background with chords for the new song. [Final code] A key  when I am [clicked ▾] broadcast [a ▾] switch to costume [a1 ▾] wait (0.2) secs switch to costume [a ▾] when I receive [a ▾] play note (69 ▾) for (0.5) beats when [a ▾] key pressed broadcast [a ▾] switch to costume [a1 ▾] wait (0.2) secs switch to costume [a ▾] wait until < not < key [a ▾] pressed? > > The violin key  when I am [clicked ▾] broadcast [chords ▾]

Note



```
when I am clicked
repeat 2
    play note 60 for 0.5 beats
repeat 2
    play note 67 for 0.5 beats
repeat 2
    play note 69 for 0.5 beats
play note 67 for 0.5 beats
wait 0.1 secs
repeat 2
    play note 65 for 0.5 beats
repeat 2
    play note 64 for 0.5 beats
repeat 2
    play note 62 for 0.5 beats
play note 60 for 0.5 beats
wait 0.1 secs
repeat 2
    play note 67 for 0.5 beats
    play note 67 for 0.5 beats
    play note 65 for 0.5 beats
    play note 65 for 0.5 beats
    play note 64 for 0.5 beats
    play note 64 for 0.5 beats
    play note 62 for 0.5 beats
    wait 0.1 secs
repeat 2
    play note 60 for 0.5 beats
repeat 2
    play note 67 for 0.5 beats
repeat 2
    play note 69 for 0.5 beats
play note 67 for 0.5 beats
wait 0.1 secs
repeat 2
    play note 65 for 0.5 beats
repeat 2
    play note 64 for 0.5 beats
repeat 2
    play note 62 for 0.5 beats
play note 60 for 0.5 beats
```

	X  when [flag] clicked set size to (50) %  when I am [clicked] broadcast [blank] The stage  when I receive [chords] switch to costume [chords]  when I receive [blank] switch to costume [blank]
Tools and Resources for the Teacher	Snap! project "Play a Piano": https://snap.berkeley.edu/project?user=ifrankovic&project=Play%20a%20Piano
Resources/materials for the Students	Half-baked activity in Snap!: https://snap.berkeley.edu/project?user=ifrankovic&project=Play%20Piano (27.1.2020) Images: ● Sprite images: ● a.png, a1.png ● b.png, b1.png ● violin_key.png ● Backgrounds: notes.png

Learning Scenario 20 - Test

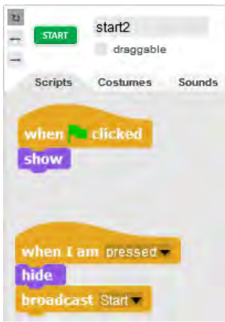
Learning Scenario Title	Test
Previous programming experience	Showing and hiding sprite Using variables for counting points Using loop forever Using conditionals Using operations for comparison Using Sensing of colors Change stage
Learning Outcomes	General learning outcomes: ● Variables ● Conditionals ● Loop ● Sensing blocks Specific learning outcomes oriented on algorithmic thinking: ● Students use conditionals to estimate answer – Right or Wrong ● Students use blocks for stage's costume change ● Students use variables for points counting ● Students use logical operations ● Students use external graphical editor for preparing complex backgrounds of the stages.
Aim, Tasks and Short Description of Activities	Short description: Help Your Teacher Test Your Snap! knowledge by creating a Quest Based Game to test the commands used in Snap Task: Students have to explore example game, choose from the "half-backed" game, find or design their own sprite that will set questions, choose from the "half-backed" game or design initial stage background and stage backgrounds with appropriate questions, modify and extend scripts in test with respect to questions.

	Aim: Students will improve their previously acquired knowledge and will extend the game scenario with new background, code and changing code with respect to new stages.
Duration of Activities	90 minutes
Learning and Teaching Strategy and Methods	Active learning (discussions, experiment with a game prepared in advance), game-design based learning, problem solving,
Teaching Forms	Individual work / Work in pairs/ Frontal work with whole class
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) 1. The teacher raises the problem of the need to create a game-test to test programming knowledge. 2. Assigns students to play the game and describe in words: How many stage decorations do they observe and how many sprites (characters)? How does the game begin? How many variables are used, how are they named, what are they used for? What happens when the answer is right/wrong? How are the questions presented in the test? /individual work or work in pairs at the teacher's discretion/ 3. Comment on the algorithm for asking and answering questions. /frontal activity/ • moving to a stage costume (contains the question); • assigning Abby a costume for asking a question; • Abby says - Answer Yes or No; • The player enters an answer - Yes or No; • If the answer is correct, Abby says "*Correct*" and the number of correct answers increases; Otherwise, Abby says "*You're wrong*" and the number of wrong answers increases. 4. Comment on what happens after answering all the questions. /frontal activity/

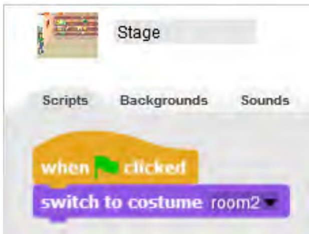
- change of costume/background on stage;
- Abbey indicates the number of right and wrong answers and gives an estimate

5. Examining the codes in the game /Updating old knowledge/individual and frontal activity/

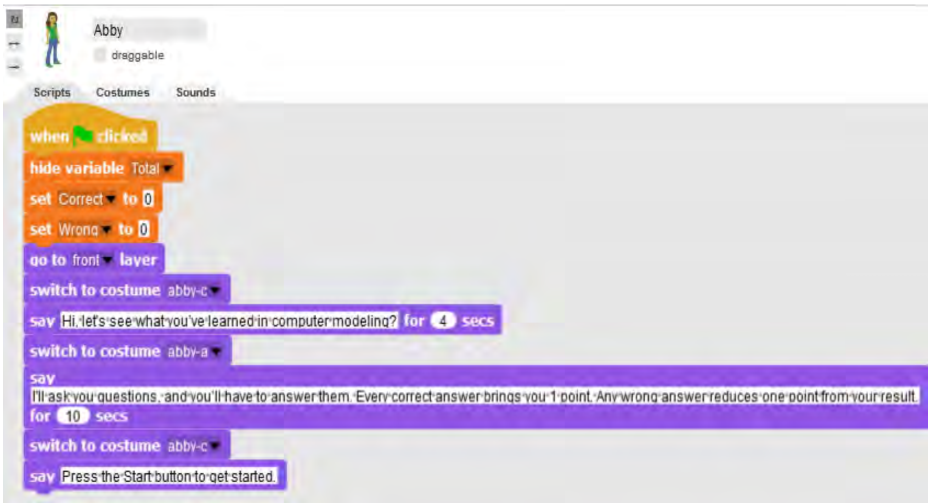
The commands for engaging in a dialogue with the user, for changing the stage decor and character costume, conditional commands are commented. The codes of each character are examined. Creating a variable is commented on.



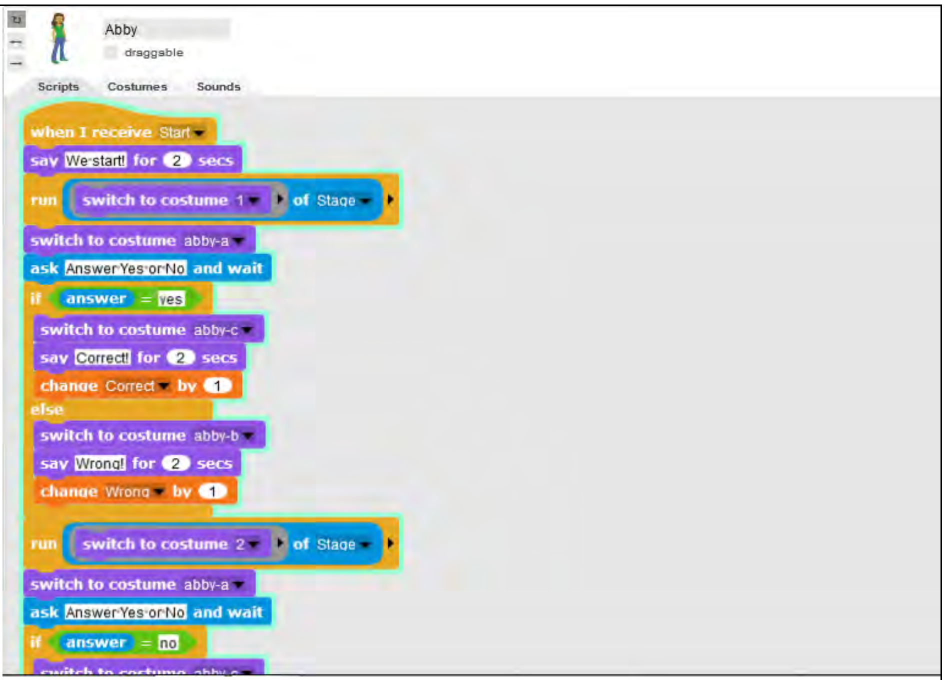
```
when ⚑ clicked
show

when I am pressed
hide
broadcast Start
```



```
when ⚑ clicked
switch to costume room2
```



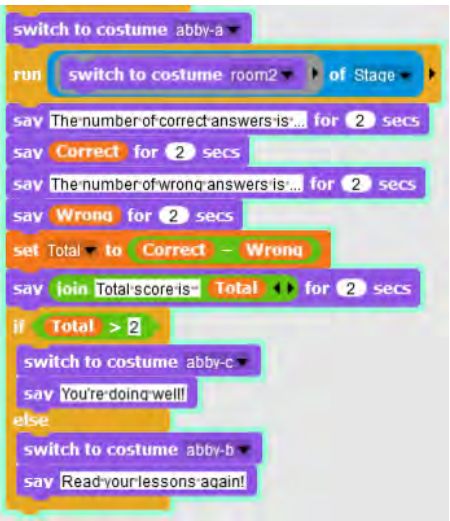
```
when ⚑ clicked
hide variable Total
set Correct to 0
set Wrong to 0
go to front layer
switch to costume abby-c
say Hi, let's see what you've learned in computer modeling? for 4 secs
switch to costume abby-a
say I'll ask you questions, and you'll have to answer them. Every correct answer brings you 1 point. Any wrong answer reduces one point from your result. for 10 secs
switch to costume abby-c
say Press the Start button to get started.
```



```
when I receive Start
say We start! for 2 secs
run (switch to costume 1) of Stage
switch to costume abby-a
ask Answer Yes or No and wait
if answer = yes
    switch to costume abby-c
    say Correct! for 2 secs
    change Correct by 1
else
    switch to costume abby-b
    say Wrong! for 2 secs
    change Wrong by 1
run (switch to costume 2) of Stage
switch to costume abby-a
ask Answer Yes or No and wait
if answer = no
    switch to costume abby-c
```

Situations when the correct answer is *yes* and when the correct answer is NO are commented.

The code for grading is discussed in detail as well as why the *Total* variable is used.



```
switch to costume abby-a
run (switch to costume room2) of Stage
say The number of correct answers is ... for 2 secs
say Correct for 2 secs
say The number of wrong answers is ... for 2 secs
say Wrong for 2 secs
set Total to Correct − Wrong
say join Total score is  Total for 2 secs
if Total > 2
    switch to costume abby-c
    say You're doing well!
else
    switch to costume abby-b
    say Read your lessons again!
```

The way of designing of the stage scenery for individual questions is discussed.

Because in Snap! it is not possible to write text in costumes and scenery, it is necessary to use an external graphic editor. Another option is to use

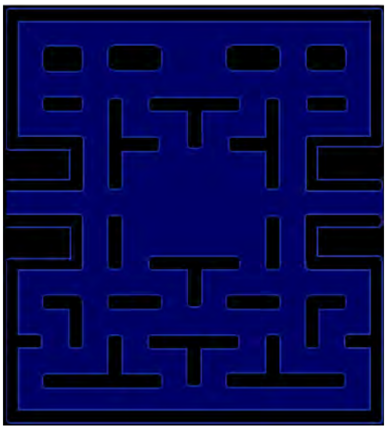
	MS Powerpoint to create the question and export the corresponding text box in graphical format. Inserting a costume in Snap! May be revised. 1. Dividing the group into teams of 2 or 3 students. 2. Posting the topic for the test questions. For example - Using variables; Loops; Mooving, Sensing, Arithmetic and Logical Operations. 3. Designing the scenes with questions on a topic by the respective team. If necessary, the teacher advises the students on the content of the questions. Questions are discussed and each team creates a scene for at least two questions. 4. Creating the code. A half-baked file of costumes of stage and sprites is provided for students to use. They can also create a file of their own if they wish. Work is done by analogy with the model test.
Tools and Resources for the Teacher	Whole activity in Snap!: https://snap.berkeley.edu/project?user=ddureva&project=test2 Whole activity in Scratch: ● Дурева Д., М. Касева, Г. Тупаров, Компютърно моделиране, 4. клас, Просвета, 2018, София (Dureva, D., M. Kaseva, G. Tuparov, Kompyutarno modelirane, 4. klas, Prosveta, 2019, Sofia)
Resources/materials for the Students	● Half-baked activity in Snap!: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_20_test_en_tmp ● Instructions for student (C4G20_InstructionsForStudent.docx)

Learning Scenario 21 - Simplified PACMAN game

Learning Scenario Title	Simplified PACMAN game
Previous programming experience	● conditionals, ● coding multiple objects, ● single color sensing, ● loops (forever, repeat until), ● event based object movement, ● random numbers
Learning Outcomes	General learning outcomes: ● cloning an object, ● defining the behaviour of a clone, ● broadcasting messages, ● Boolean value readings in logical expressions, ● defining, differentiating, dynamically checking and responding to two different game states, Specific learning outcomes oriented on algorithmic thinking: ● student implements object movement with arrow keys using events and takes into account restrictions, ● student uses clones to make instances of the original object, ● student know how to code a behavior of each clone, ● students knows the meaning of sending messages, ● student implements sending a message from clone to increment counter, ● student knows how to detect the message was received by the object and makes an appropriate response
Aim, Tasks and Short Description of Activities	Short description: Program game in which the main character will pick up randomly positioned stars and be chased by a ghost. Tasks: Students have to program the movement of the main character so she will move inside a labyrinth. They have to implement movement restrictions so that the main character cannot move through the walls.

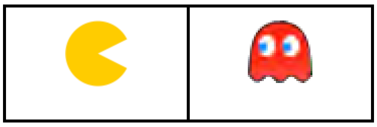
	Next they have to program a star object that will clone itself when the game starts and then on a random new location each time a character will collect it. They have to store the value of collected stars and finish the game when the player collects 20 stars. To make the game more interesting they can program evil ghost that will randomly move throughout the labyrinth. If a player touches the ghost, the game is over. **With this activity students will review their knowledge about movement inside a labyrinth with the use of sense color block that they learned in previous activities. They will be introduced to the concept of cloning the object with position restrictions and how to create a very simple nonplayer character with its own random movement.**
Duration of Activities	90 minutes
Learning and Teaching Strategy and Methods	active learning, collaborative learning, problem solving
Teaching Forms	frontal teaching individual work/working in pairs/group work
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) Player is collecting randomly positioned stars while being chased by a red ghost. If a player and ghost collide, the game is over. If a player collects the 20 stars he wins. [Step 1] We instruct students to design a labyrinth in which area where the player is allowed to move is of one color (e.g. blue) and walls that stop player movement that are colored in some other color (e.g. black). To

save time we can prepare the background picture of the labyrinth beforehand.



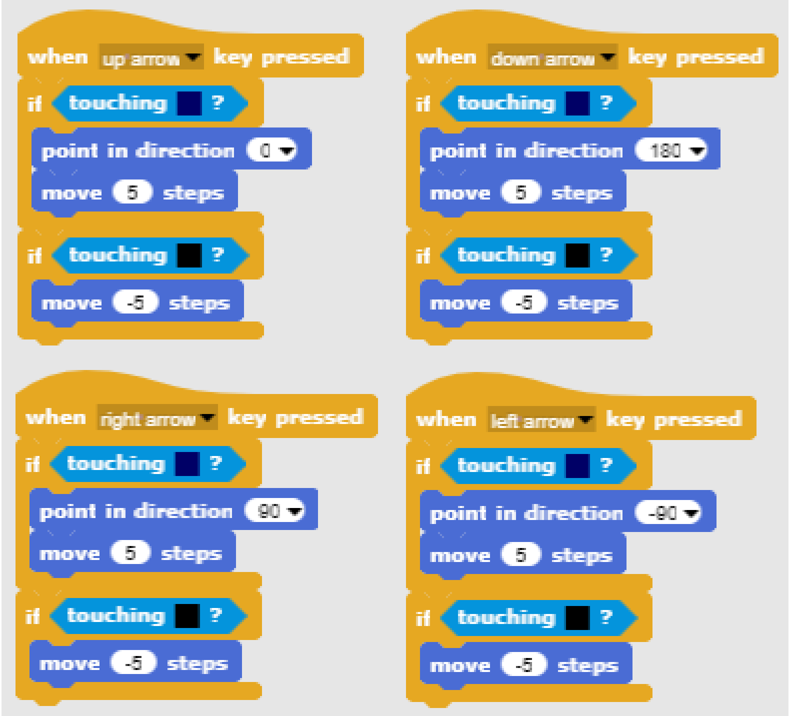
[Step 2]

They have to draw the pacman and the red ghost. For a star we can simply draw a circle inside Snap!:



[Step 3]

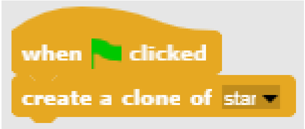
In order to make pacman move, we can use different possibilities. The sample below is one of them. In it we use an event system for detecting which key is pressed, left, right, up or down. After each of these events happen, we have to test if he is touching the color of the area he is allowed to move. If this is the case, he first turns into that direction and makes the move. But if he touches the color of the walls, he must move back, because otherwise he would get stuck at the wall because of the first condition.



```
when up arrow ▾ key pressed
if touching 🟦 ?
    point in direction 0 ▾
    move 5 steps
if touching ⬛ ?
    move -5 steps

when down arrow ▾ key pressed
if touching 🟦 ?
    point in direction 180 ▾
    move 5 steps
if touching ⬛ ?
    move -5 steps

when right arrow ▾ key pressed
if touching 🟦 ?
    point in direction 90 ▾
    move 5 steps
if touching ⬛ ?
    move -5 steps

when left arrow ▾ key pressed
if touching 🟦 ?
    point in direction -90 ▾
    move 5 steps
if touching ⬛ ?
    move -5 steps
```

[Step 4]

Next task is to program the stars. Stars will be all the same but there will be many of them. In this case it is better than making multiple identical objects (in our case 20), to make one object and then create its clones. At the beginning of the game the first clone will appear randomly inside the labyrinth, then when the player collects it it will disappear and a new one will be created on a different random location. In order to create the first clone at the beginning of the game we put this code on a Scene script.

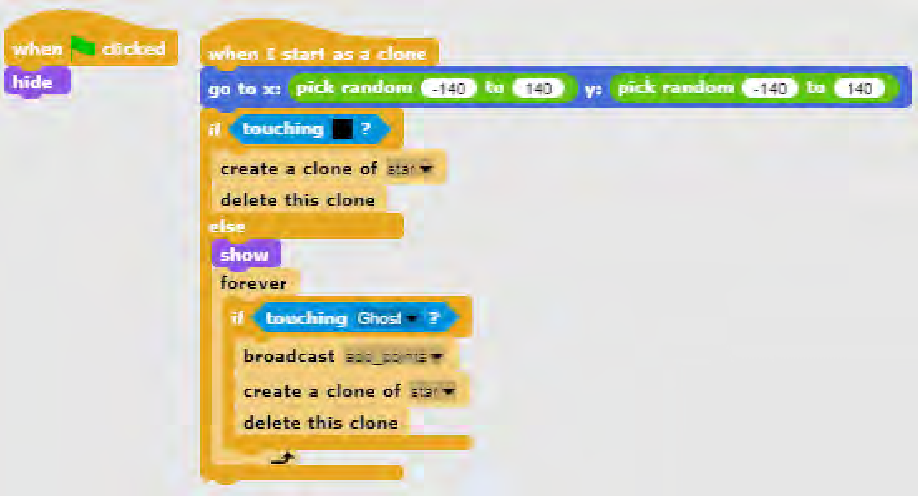


```
when 🏁 clicked
create a clone of star ▾
```

In order to hide an original object and only show clones, we have to do this at the start of the program.

In order to find suitable random locations we have to observe certain restrictions. If a star is created on a wall, a player cannot reach it, meaning we cannot place it there. Strategy for doing so is as follows:

1. We have to find the random x,y position of the star clone. Both x and y coordinates are on the same interval [-140, 140]. So we choose a random number from that interval for both.
2. Next we check if this clone is touching the color of the wall. In this case its location is not legal.
3. If location is ok, we have to show the clone (remember, the original is hidden and the clone would also be hidden if we didn't use show block) and in forever loop check if the collision with the player occurs.
4. If location is not ok, we create a new clone (hoping that for the new one, random numbers will be chosen so he would be placed in a legal location) and delete this one.
5. In order to count collected clones we have to inform a total counter of stars that must be defined outside the clone, e.g. on the player. This can be done by broadcasting a message that the collision occurred. Then we can delete it.



```
when ⚑ clicked
hide

when I start as a clone
go to x: (pick random -140 to 140) y: (pick random -140 to 140)
if <touching ■ ?>
    create a clone of star ▾
    delete this clone
else
    show
    forever
        if <touching Ghost ▾ ?>
            broadcast add_points ▾
            create a clone of star ▾
            delete this clone
```

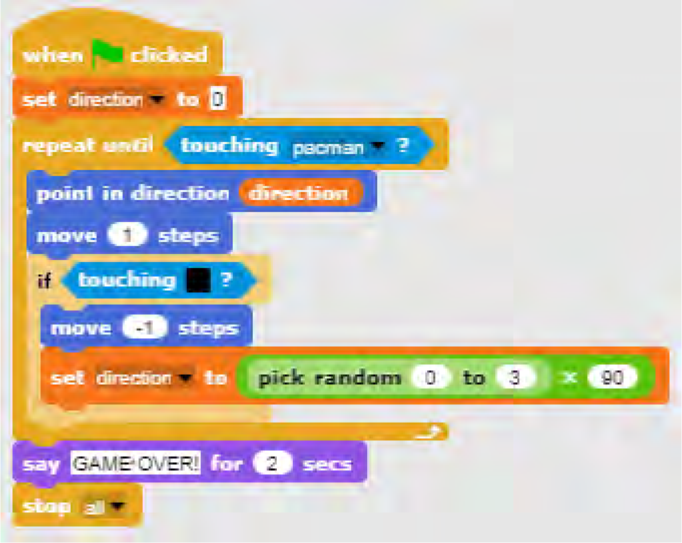
[Step 5]

Next we program a ghost. He has to randomly move throughout the labyrinth and has to change direction when he bumps into the wall. In order to make his movement random we want him to move in a random direction after the bump. In Snap! directions are expressed with degrees:

1. 0 degrees - UP
2. 180 degrees - DOWN
3. 90 degrees - RIGHT
4. 270 degrees - LEFT

In other words if we randomly choose the number from 0 to 3 and multiply it by 90 we get a random direction!

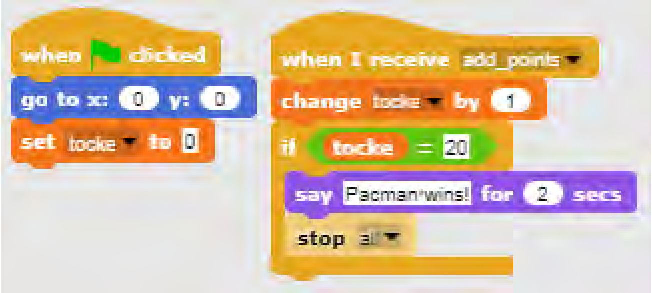
He has to move until he collides with a pacman. Then the game is over.



```
when ⚑ clicked
set direction to 0
repeat until touching pacman ?
    point in direction direction
    move 1 steps
    if touching ■ ?
        move -1 steps
        set direction to pick random 0 to 3 × 90
say GAME OVER! for 2 secs
stop all
```

[Step 6]

Now we have to program when the player will win the game. This will be when she collects 20 stars. We have a star counter inside pacman script. At the beginning we initialize it to 0, and then increase its value by 1 each time the clone sends a message that the player has collected

	it. If the counter comes to 20, pacman wins and we have to stop the game.  when 🏁 clicked go to x: 0 y: 0 set tocke to 0 when I receive add_points change tocke by 1 if tocke = 20 say Pacman wins! for 2 secs stop all
Tools and Resources for the Teacher	- Whole activity in Snap!: https://snap.berkeley.edu/project?user=zapusek&project=pacman_clone - Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena. - Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.
Resources/materials for the Students	- Template in Snap!: https://snap.berkeley.edu/project?user=zapusek&project=pacman_template - Instructions for student (C4G21_InstructionsForStudent.docx)

APPENDIX 2. LEARNING SCENARIO CODES

BASIC LEARNING SCENARIOS

ENGLISH Scenarios

N.	TITLE	CODE
1	**Introduction to Snap! Interface** Students add a new sprite, add a costume to the sprite, edits the costume, and deletes one of them. Student creates a new background to the stage, edits it, and deletes unwanted ones.	**starting**
2	**Discover Snap! : move a spite** Help the students to discover the Snap! interface and code their first sprite so that it moves and speaks.	**dispubeng**
3	**Moving around the stage** Students learn how to move the sprite in x and y direction on the stage, code an easy program to solve the tasks given, learn how to turn the sprite in a different direction.	**monkey**
4	**Changing costumes and turning** Students learn how to change the sprite's costume to make an animation by changing between different types of rotation.	**Dancer**
5	**Sounds of the farm** Students learn how to program a simple game in which player can recognize the sounds of animals by pressing certain keys.	**farm**
6	**Chameleon's summer vacation** Program a simple game in which an object will change its costume based on the color of the background	**chameleoneng**
7	**Helping Prince and Princess to find their animals** A girl has to help the Princess find her cat and the Prince find his dog by avoiding that the animals can meet each other during their path.	**finding**
8	**Drawing with a chalk** Students will be introduced into drawing different shapes with a code. They will learn to use loop repeat for shorten the code and to change a background.	**chalk**
9	**Picking up trash and cleaning the park** Students will learn how to use variables and how to duplicate a block of code or even a whole sprite.	**cleaning**
10	**Feeding the cats** Students will be introduced to the concept of multiple variable random value assignment inside a loop and how it is different from when we do it outside a loop. They will also learn about how to get, test and count correct player inputs.	**feeding**

11	**Guessing the number of cats in a shelter** Students will be introduced to "repeat until" loop and how to set the condition to implicitly track the condition that stops the game. They will also learn how to use variable in a different situations: to set a random value, as a counter or to get the players input.	**shelter**

SLOVENIAN Scenarios

N.	TITLE	CODE
1	**Uvod v Snap!** Učenec doda nov lik, doda obleko, uredi obleko in izbriše obleko/lik. Učenec ustvari novo ozadje, ga uredi, zbriše odvečna ozadja.	**uvod_slo**
2	**Razišči Snap!: premikaj lik** To uro boš spoznal/a, kako z bloki likom naročimo naj se premikajo po odru in govorijo.	**razisci_slo**
3	**Premikanje po odru** Učenec se nauči, kako premikati lik v x in y smeri, kako napisati preprost program ter kako obrniti lik v drugo smer	**premikanje_slo**
4	**Menjava obleke in obrat** Učenec se nauči, kako liku zamenjati obleko in kako narediti animacijo z obračanji lika	**obleka_slo**
5	**Zvoki na kmetiji** Učenec se nauči programiranja preproste igre, v kateri lahko igralec s pritiskom na določene tipke prepozna zvoke živali	**kmetija_slo**
6	**Kameleon na počitnicah** Izdelaj preprosto igro, v kateri bo glavni objekt spreminjal svojo obleko glede na barvo ozadja na njegovi trenutni poziciji.	**kameleon_slo**
7	**Pomagaj princu in princeski najti svoje živali** Dekle mora pomagati princesi poiskati svojo mačko in princu svojega psa. Pri tem se mora izogniti srečanju med živalmi.	**iskanje_zivali_slo**
8	**Risanje s kredo** Učenec se seznani z risanjem različnih oblik s kodo. Nauči se uporabljati zanko ponovi za krajšanje kode ter kako zamenjati ozadje.	**kreda_slo**
9	**Pobiranje smeti in čiščenje parka** Učenec se nauči uporabljati spremenljivke in kako podvojiti blok kode ali celotno kodo lika.	**ciscenje_parka_slo**
10	**Nahrani mucke** Učenec se seznani s konceptom dodeljevanja več spremenljivk z naključno vrednostjo znotraj zanke in kakšna je razlika, če vrednost dodelimo zunaj zanke. Spoznali bodo tudi, kako pridobiti in preveriti igralčev odgovor ter kako šteti pravilne odgovore.	**mucke_slo**
11	**Mačje zavetišče** Učenec se seznani z zanko "ponavljaj dokler" in kako ustvariti pogoj za implicitno sledeje pogoju, ki konča z igro. Nauči se tudi, kako uporabljati spremenljivko v različnih situacijah: nastaviti naključno vrednost, kot števec ali kot igralčev vnos.	**zavetisce_slo**

ITALIAN Scenarios

N.	TITLE	CODE
1	**Snap!: introduzione** Lo studente impara ad aggiungere un nuovo sprite, a modifica il costume e ad eliminarne uno di essi. Impara a crea un nuovo sfondo per il palco, lo modifica e cancella quelli indesiderati.	**Primipassi**
2	**Scoprire Snap!** Imparare a programmare un gioco semplice dove l'oggetto cambia il proprio colore secondo il colore dello sfondo	**Snap!**
3	**Muoversi sullo "stage"** Il discente impara a spostare il suo Sprite in direzione x e y sullo "stage"; a preparare un semplice programma per risolvere i compiti assegnati; a far girare il suo Sprite in diverse direzioni.	**stage**
4	**Cambiare il costume e creare rotazioni** Lo studente impara a cambiare il costume dello sprite per realizzare un'animazione.	**ballerina**
5	**I suoni di una fattoria** Gli studenti imparano come programmare un gioco semplice in cui il giocatore può riconoscere i suoni degli animali premendo determinati tasti.	**fattoria**
6	**Vacanze estive di un Camaleonte** Imparare a programmare un gioco semplice dove l'oggetto cambia il proprio colore secondo il colore dello sfondo	**camaleonte**
7	**Alla ricerca degli animali del Principe e della Principessa!** La ragazza deve aiutare la principessa a trovare il suo gatto e il principe a trovare il suo cane evitando che gli animali possano incontrarsi durante il loro.	**labirinto**
8	**Disegnare con un gesso** Gli studenti impareranno a disegnare forme diverse con un codice e ad usare la ripetizione ciclica per abbreviare il codice e cambiare lo sfondo.	**gesso**
9	**Raccogliere la spazzatura e pulire il parco** Gli studenti impareranno ad usare le variabili e a duplicare un blocco di codice o anche un intero Sprite.	**spazzatura**
10	**Dare da mangiare ai gatti** Gli studenti impareranno il concetto di assegnazione di più valori casuali, variabili all'interno e al di fuori di un ciclo. Impareranno anche come ottenere, testare e contare gli input corretti immessi dal giocatore.	**gatto**
11	**Indovinare il numero di gatti in un rifugio** Gli studenti impareranno ad utilizzare il ciclo "ripeti fino a" e ad impostare la condizione per interrompere il gioco. Impareranno anche ad usare la variabile in situazioni diverse.	**rifugio**

CROATIAN Scenarios

N.	TITLE	CODE
1	**Uvod u sučelje alata Snap!** Učenik dodaje nove objekte, dodaje kostime objektima, uređuje kostime te ih briše. Učenik stvara nove pozadine na pozornici, uređuje ju, te neželjene briše.	**start_hr**
2	**Otkrijte Snap!: kretanje sprite** Učenici otkriva gdje pronaći programske blokove i povezati ih u niz. Učenici će naučiti kako micati objekt i omogućiti da objekt govori.	**disc_cro**
3	**Kretanje po pozornici** Učenici će vidjeti kako izraditi program u kojem će se objekt kretati u smjeru x, te izraditi program u kojem će se objekt kretati u smjeru y.	**kretanje**
4	**Mijenjanje kostima i okretanje** Učenik uči kako promijeniti kostim objekta te kako napraviti animaciju. Također uči kako se između njih može mijenjati različite vrste rotacije objekta.	**ples**
5	**Zvukovi s farme** Učenici će se upoznati kako programirati jednostavnu igru u kojoj igrač može prepoznati zvukove životinja pritiskom na određene tipke.	**farma**
6	**Kameleonov ljetni odmor** Učenici će biti upoznati s blokom naredbi osjeta boja i kako ga koristiti u logičkim izrazima da bi razlikovali dinamično promjenjiva stanja igre i dali prave odgovore.	**chameleon_cro**
7	**Pomaganje princu i princezi u pronalasku njihovih životinja** Učenici će se upoznati s crtanjem pokretom tipke. Uz to će naučiti kako koristiti uvjete kojima će spriječiti da se objekt kreće po cijelome ekranu.	**finding_hr**
8	**Crtanje s kredom** Učenici će se upoznati s crtanjem različitih oblika pomoću kôda. Naučit će koristiti petlju ponavljaj kako bi smanjili veličinu kôda i promijeniti pozadinu.	**kreda**
9	**Skupljanje otpada i čišćenje parka** Učenici će se upoznati kako koristiti varijable i kako kopirati blok kôda ili čak cijeli objekt.	**park**
10	**Hranjenje mačaka** Učenici će biti upoznati sa konceptom dodjele slučajne vrijednosti varijabli unutar petlje te će razlikovati kada navedeno napravimo van petlje. Naučiti će kako dobiti, testirati i izbrojati ispravne unose igrača.	**hranjenje**

11	**Pogađanje broja mačaka u skloništu** Učenici će se upoznati s petljom ponavljaj dok i kako postaviti uvjet koji zaustavlja igru. Također će naučiti kako koristiti varijable u različitim situacijama: za pohranjivanje nasumične vrijednosti, kao brojač ili za pohranjivanje vrijednosti koju upiše igrač.	**pogodi**

BULGARIAN Scenarios

N.	TITLE	CODE
1	**Увод в Snap!** Ученикът добавя нов спрайт, добавя костюм към спрайта, редактира костюма и изтрива едино от двете. Ученикът създава нов фон на сцената, редактира го и го изтрива.	**start_bg**
2	**Да разучим Snap!: Движещ се спрайт** Помогнете на учениците да разучат интерфейса на Snap! и да съставят код за първия им спрайт, така че той да се движи и говори.	**dissnap_bg**
3	**Движейки се по сцената** ченикът се научава как да движи спрайта в х и у посока на сцената, създава лесна програма за решаване на задачите и се научава как да завърти спрайт в различна посока.	**monkey_bg**
4	**Смяна на костюми и завъртане** Ученикът се учи как да промени костюма на спрайт при завъртане, за да направи анимация.	**dancer_bg**
5	**Звуци от фермата** Учениците научават как да програмират игра, в която играчът може да разпознава звуците на животните чрез натискане на определени клавиши.	**sounds_bg**
6	**Лятната ваканция на хамелеона** Програмирайте проста игра, в която обект да променя костюма си в зависимост от цвета на фона.	**cham_bg**
7	**Помогнете на принца и принцесата да намерят своите домашни любимци** Момиче трябва да помогне на принцесата да намери котката си, а на принцът да намери кучето си, като трябва да избегне възможността животните да се срещнат по пътя.	**find_bg**
8	**Рисуване с тебешир** Учениците ще бъдат запознати с рисуването на различни фигури(форми) използвайки код. Те ще се научат да използват цикъл с повторение за намаляне обема на кода и за промяна на фона.	**draw_bg**
9	**Събиране на боклук и почистване на парка** Учениците ще научат как да използват променливи и как да копират блок с код или дори цял спрайт.	**clean_bg**
10	**Нахранете котките** Учениците ще бъдат въведени в концепцията за присвояване на случайно число на няколко променливи в цикъл, както и каква е разликата при използването им вътре и извън цикъл. Те също така ще научат как за да зададат, тестват и изброят правилно въведеното от играча.	**cats_bg**

11	**Познайте броя на котките в приюта** Учениците ще бъдат запознати с цикъл "repeat until" и как да зададат условието, при което спира играта. Те също така ще се научат как да използват променливи в различни ситуации: за задаване на произволна стойност, като брояч или като стойност въведена от играча.	**shelter_bg**

GREEK Scenarios

N.	TITLE	CODE
1	**Εισαγωγή στο Snap!** Ο μαθητής προσθέτει ένα νέο στοιχείο, προσθέτει ένα κοστούμι στο στοιχείο, επεξεργάζεται το κοστούμι και διαγράφει ένα από αυτά. Ο μαθητής δημιουργεί ένα νέο φόντο στη σκηνή, το επεξεργάζεται και διαγράφει τα ανεπιθύμητα.	--
2	**Ανακαλύψτε το Snap! : μετακινήσε μία εικόνα** Βοήθησε τους μαθητές να ανακαλύψουν την πλατφόρμα προγραμματισμού Snap! και να κωδικοποιήσουν την πρώτη τους εικόνα ώστε να κινέιται και να μιλάει.	**SampleCourse1Greek**
3	**Ας μετακινηθούμε γύρω από τη σκηνή** Ο μαθητής μαθαίνει πώς να μετακινεί ένα στοιχείο σε κατεύθυνση x και y στη σκηνή, δημιουργεί ένα εύκολο πρόγραμμα για την επίλυση των καθηκόντων του, μαθαίνει πώς να γυρίζει το στοιχείο σε διαφορετική κατεύθυνση.	--
4	**Αλλαγή κοστουμιών και στροφή** Ο μαθητής μαθαίνει πώς να αλλάζει το κοστούμι του στοιχείου για να κάνει ένα κινούμενο σχέδιο αλλάζοντας μεταξύ διαφορετικών τύπων περιστροφής.	--
5	**Ήχοι φάρμας** Οι μαθητές μαθαίνουν πώς να προγραμματίζουν ένα απλό παιχνίδι στο οποίο ο παίκτης μπορεί να αναγνωρίζει τους ήχους των ζώων πατώντας συγκεκριμένα κουμπιά.	--
6	**Καλοκαιρινές διακοπές του Χαμαιλέοντα** Προγραμματίστε ένα απλό παιχνίδι στο οποίο ένα αντικείμενο θα αλλάξει το κοστούμι του με βάση το χρώμα του φόντου	**SampleCourse2Greek**
7	**Βοηθώντας τον Πρίγκιπα και την Πριγκίπισσα να βρουν τα ζώα τους** ο κορίτσι πρέπει να βοηθήσει την Πριγκίπισσα να βρει τη γάτα της και τον Πρίγκιπα να βρει τον σκύλο του, αποφεύγοντας τα υπόλοιπα ζώα που μπορεί να συναντήσουν κατά τη διάρκεια της πορείας τους.	--
8	**Ζωγραφίζοντας με κιμωλία** Οι μαθητές θα μάθουν να ζωγραφίζουν διαφορετικά σχήματα με κώδικα. Θα μάθουν να χρησιμοποιούν βρόχους επανάληψης για να συντομεύσουν τον κώδικα και να αλλάξουν φόντο.	--

9	**Μαζεύοντας σκουπίδια και καθαρίζοντας το πάρκο** Οι μαθητές θα μάθουν πώς να χρησιμοποιούν μεταβλητές και πώς να αντιγράφουν ένα κομμάτι κώδικα ή ακόμα και ένα ολόκληρο στοιχείο.	--
10	**Ταΐζοντας τις γάτες** Οι μαθητές θα μάθουν την έννοια της πολλαπλής ανάθεσης τυχαίας τιμής σε μεταβλητές μέσα σε ένα βρόχο και πώς αυτό είναι διαφορετικό αν το κάνουμε εκτός βρόχου. Θα μάθουν επίσης για τον τρόπο λήψης, δοκιμής και μέτρησης των σωστών εισόδων από τον παίκτη.	--
11	**Μαντέψτε τον αριθμό των γατιών στο καταφύγιο** Οι μαθητές θα μάθουν το βρόχο «επανάληψη έως» και πώς να ρυθμίσουν τη συνθήκη ώστε να παρακολουθούν τη συνθήκη που σταματά το παιχνίδι. Θα μάθουν επίσης πώς να χρησιμοποιούν τη μεταβλητή σε διαφορετικές καταστάσεις: για να ορίσουν μια τυχαία τιμή, ως μετρητή ή για να πάρουν είσοδο από τον παίκτη.	--

PORTUGUESE Scenarios

N.	TITLE	CODE
1	**Introdução à interface Snap!** Os alunos adicionam um novo sprite, adicionam um traje ao sprite, editam o traje e excluem um deles. O aluno cria um novo plano de fundo para o palco, edita-o e exclui os indesejados**.**	starting_PT
2	**Descubra o Snap! : mover um Sprite** Os alunos descobrem a interface do Snap! e codificam o seu primeiro sprite para que ele se mova e fale.	dispubeng_PT
3	**Movendo-se pelo palco** Os alunos aprendem como mover o sprite nas direções x e y do palco, codificam um programa fácil para resolver as tarefas dadas e aprendem a virar o sprite em uma direção diferente.	monkey_PT
4	**Mudando de traje e girando** Os alunos aprendem a mudar o traje do sprite para fazer uma animação e alternam entre os diferentes tipos de rotação.	dancer_PT
5	**Sons da quinta** Os alunos aprendem a programar um jogo simples no qual o jogador pode ouvir os sons dos animais pressionando certas teclas.	farm_PT
6	**Férias de verão do camaleão** Os alunos programam um jogo simples em que um objeto muda de traje de acordo com a cor do fundo	chameleoneng_PT
7	**Ajudando o príncipe e a princesa a encontrar seus animais** Uma menina tem que ajudar a Princesa a encontrar seu gato e o Príncipe a encontrar seu cachorro, evitando que os animais se cruzem durante o caminho.	finding_PT
8	**Desenho com giz** Os alunos aprendem a desenhar formas diferentes com um código. Aprendem a usar ciclos para encurtar o código e alterar um plano de fundo.	chalk_PT
9	**Recolher o lixo e limpar o parque** Os alunos aprendem a usar variáveis, a duplicar um bloco de código e um sprite.	cleaning_PT
10	**Alimentando os gatos** Os alunos aprendem o conceito de atribuição de valores aleatórios a múltiplas variáveis dentro de um loop e como é diferente de quando o fazemos fora de um loop. Eles também aprendem a obter, testar e contar as entradas corretas do jogador.	feeding_PT

11	**Adivinhando o número de gatos em um abrigo** Os alunos aprendem o ciclo "repetir até" e como definir a condição para rastrear a condição que interrompe o jogo. Aprendem ainda a usar variáveis em diferentes situações: definir um valor aleatório, como um contador ou para obter dados dos jogadores.	**shelter_PT**

TURKISH Scenarios

N.	TITLE	CODE
1	**Snap! ara yüzünü tanıyalım** Öğrenciler yeni bir karakter (sprite/kukla) ekler, karaktere bir kostüm ekler, kostümü düzenler ve bunlardan birini siler. Öğrenciler sahne için yeni bir arka plan oluşturur, düzenler ve istenmeyenleri siler.	**başla**
2	**Snap!'i keşfet - Karakteri hareket ettirme** Öğrencilerin Snap! arayüzünü keşfetmelerine ve ilk kodlarını oluşturarak hareket eden ve konuşan bir karakter oluşturmalarına yardımcı olun.	**keşfet**
3	**Sahnede hareket etme** Öğrenciler sahnede karakteri x ve y yönünde hareket ettirmeyi öğrenir, verilen görevleri çözmek için basit bir program programlar, karakteri farklı yöne çevirmeyi öğrenir.	**hareket**
4	**Kostüm değiştirme ve dönüşler** Öğrenci, farklı rotasyon türleri arasında geçiş yaparak bir animasyon yapmak için karakerin kostümünü nasıl değiştireceğini öğrenir.	**dans**
5	**Çiftlikteki sesler** Öğrenciler, oyuncunun belirli tuşlara basarak hayvanların seslerini tanıyabileceği basit bir oyun programlamayı öğrenirler.	**çiftlik**
6	**Bukalemenun yaz tatili** Bir nesnenin kostümünü arka planın rengine göre değiştireceği basit bir oyun programlayın	**bukalemun**
7	**Prens ve Prensese evcil hayvanlarını bulmaları için yardım et** Kız, seyahatleri sırasında hayvanların birbirleriyle karşılaşmasını engelleyerek Prenses'in kedisini bulmasına ve Prens'in köpeğini bulmasına yardım etmelidir.	**prenses**
8	**Tebeşir ile çizim yapmak** Öğrencilere bir kodla farklı şekiller çizme tanıtılacaktır. Kodu kısaltmak ve arka planı değiştirmek için döngü tekrarını kullanmayı öğrenecekler.	**çizim**
9	**Çöpleri toplamak ve parkı temizlemek** Öğrenciler değişkenleri nasıl kullanacaklarını ve bir kod bloğunu veya hatta bütün bir hareketli grafiği nasıl kopyalayacaklarını öğrenecekler.	**çöpler**
10	**Kedileri besleme** Öğrencilere, bir döngü içinde çok değişkenli rastgele değer atama kavramı ve bir döngü dışında yaptığımızdan nasıl farklı olduğu anlatılacaktır. Ayrıca doğru oyuncu girdilerinin nasıl alınacağını, test edileceğini ve sayılacağını da öğrenecekler.	**kediler**
11	**Barınaktaki Kedi Sayısını Tahmin Etmek** Öğrencilere "----e kadar tekrar et" döngüsü ve oyunu durduran koşulu örtük olarak izlemek için koşulun nasıl ayarlanacağı tanıtılacaktır. Ayrıca değişkeni; rastgele bir değer belirlemek, sayaç olarak veya oyuncuların girdisini almak gibi farklı durumlarda nasıl kullnacaklarını da öğrencekler.	**barınak**

ADVANCED LEARNING SCENARIOS

ENGLISH Scenarios

N.	TITLE	CODE
12	**Catching healthy food** Students will learn how to randomly move for X steps and choose a position and also how to use variables and conditionals for preventing other event.	**food**
13	**Storytelling** Students will learn how to plan storytelling, how to use broadcast messages for synchronisation of the activities of sprites and stage changes.	**storytelli ng**
14	**Drawing** Student will learn to draw in Snap!, to change the colour and the pen thickness, and how to use variables and conditionals that cause a new event.	**drawing**
15	**Catch the mouse** Student will be introduced to the concept of multiple variable random value assignment. They will learn how to use the Operators/pick random[x]to[y] block.	**mouse**
16	**Buying food for a picnic** Students will learn how to work with variables: setting different starting values, using conditionals to compare variables' value, changing variables' value, using variables for counting (un)healthy food. In addition, they will repeat adding text, joining texts and if statement.	**picnic**
17	**Operations** Students will improve their knowledge on variables, random numbers, loops, broadcast.	**operatio n**
18	**Recycling** Students will improve their knowledge and will extend the game scenario with new objects, code and changing code with respect to new sprites. They will be trained in code refactoring.	**recycling**
19.1	**Play a piano_1** Students will learn about melodies coding and playing and will improve their knowledge about variables, loops, conditional, broadcast and other events.	**music**
19.2	**Play a piano_2** Students will learn how to play music and change costume by clicking on a sprite.	**piano**
20	**Test** Students will improve their knowledge and will extend the game scenario with new background, code and changing code with respect to new stages.	**test**

21	**Simplified PACMAN game** Students will review their knowledge about movement inside a labyrinth with the use of sense color block. They will learn the concept of cloning the object with position restrictions and how to create a very simple non player character with its own random movement.	**pacman**

SLOVENIAN Scenarios

N.	TITLE	CODE
12	**Lovljenje zdrave hrane** Učenec se nauči naključnega premikanja za X korakov in naključne izbire pozicije ter uporabo spremenljivk in pogojnih stavkov za preprečevanje drugih dogodkov.	**lovljenje_hrane_slo**
13	**Sestavi zgodbo** Učenec se nauči načrtovanja pripovedovanja zgodb, uporabe pošiljanja obvestil za sinhronizacijo dejavnosti likov ter zamenjave ozadij.	**zgodba_slo**
14	**Onesnažen zrak** Učenec se nauči risanja, spreminjanja barve, debeline svinčnika ter uporabe spremenljivke in pogojev za izvedbo drugega dogodka.	**zrak_slo**
15	**Ulovi miš** Učenec se seznani s konceptom dodeljevanja naključnih vrednosti in se nauči, kako uporabiti operator naključno število od_[x]_do_[y].	**mis_slo**
16	**Kupovanje hrane za piknik**	**piknik_slo**
17	**Računanje** Učenec izboljša svoje znanje o spremenljivkah, naključnih številkah, zankah ter pošiljanju obvestil.	**racunanje_slo**
18	**Recikliranje**	**recikliranje_slo**
19.1	**Zaigraj na klavir 1** Učenec se nauči sestavljanja melodij ter izboljša svoje znanje o spremenljivkah, zankah, pogojih, pošiljanju obvestil ter drugih dogodkov.	**ples_slo**
19.2	**Zaigraj na klavir 2** Učenec se nauči predvajati glasbo in liku spremeniti obleko s klikom nanj.	**klavir_slo**
20	**Test** Učenec izboljša svoje znanje in igro razširi z novimi ozadji, doda in spremeni kodo.	**test_slo**
21	**Enostavni Pacman** Učenec ponovi s pomočjo zaznavanja barve liku omejiti gibanje, spozna koncept kloniranja z omejitvami gibanja in se nauči ustvarjanja lika, ki se samostojno nakljčno premika po labirintu.	**pacman_slo**

ITALIAN Scenarios

N.	TITLE	CODE
12	**Raccogliere i cibi salutari** Gli studenti impareranno a muoversi casualmente per X passi, a scegliere una posizione, ad usare variabili e condizioni per prevenire altri eventi.	**cibo**
13	**Alice nel Paese delle Meraviglie** Gli studenti impareranno a pianificare una storia, ad utilizzare i messaggi inviati e ricevuti per la sincronizzazione delle attività degli Sprite e per i cambiamenti di scena.	**alice**
14	**Disegno** Lo studente imparerà a disegnare con Snap!, a cambiare il colore e lo spessore della penna e ad usare le variabili e le condizioni che determinano nuovi eventi.	**disegno**
15	**Alla caccia di un topolino!** Gli studenti comprenderanno il concetto di assegnazione di più valori casuali variabili. Impareranno come usare gli Operatori / scegliere il blocco casuale da [x] a [y].	**topolino**
16	**Acquistare cibo per un picnic** Gli studenti impareranno a lavorare con le variabili: impostare diversi valori iniziali, usare i condizionali per confrontare il valore delle variabili, cambiare il valore delle variabili, usare le variabili per contare alimenti sani (e non). Inoltre, aggiungeranno il testo, useranno l'unione di testi e la condizione "if".	**picnic_1**
17	**Operazioni** **Gli studenti miglioreranno le loro conoscenze su variabili, numeri casuali, cicli e broadcast.**	**operazioni**
18	**Raccolta differenziata** Gli studenti miglioreranno le loro conoscenze per estendere lo scenario di gioco con nuovi oggetti, codice e cambiando codice rispetto ai nuovi Sprite.	**riciclo**
19.1	**Suonare il piano_1** Gli studenti impareranno a codificare le melodie e miglioreranno le loro conoscenze su variabili, loop, condizionale, trasmissione e altri eventi.	**music_1**
19.2	**Suonare il piano_2** Gli studenti impareranno a suonare e a cambiare costume facendo clic su uno Sprite	**piano_2**
20	**Test** Gli studenti miglioreranno le loro conoscenze ed estenderanno lo scenario del gioco con nuovi background, codici e sue modifiche.	**test_1**

21	**PACMAN** Gli studenti rivedranno le loro conoscenze sul movimento all'interno di un labirinto con l'uso dei blocchi dei sensori del colore. Impareranno il concetto di clonazione dell'oggetto con restrizioni di posizione e a creare un personaggio "non giocatore".	**pacman_1**

CROATIAN Scenarios

N.	TITLE	CODE
12	**Hvatanje zdrave hrane** Učenici će naučiti kako nasumično pomicati za X koraka i odabrati položaj i također kako koristiti varijable i uvjete za sprječavanje drugih događaja.	**hvatanje**
13	**Pričam ti priču** Učenici će naučiti kako ispričati priču, kako koristiti poruke i kako promijeniti pozadinu pozornice.	**prica**
14	**Crtanje** Učenici će naučiti kako se crta pomoću Snap!-a, promijeniti boju i debljinu olovke te kako koristiti varijable i uvjete koji uzrokuju novi događaj.	**crtanje**
15	**Uhvati miša** Učenik će se upoznati s konceptom dodjeljivanja više varijabli slučajnim vrijednostima. Naučit će kako koristiti blok Operatori / slučajni broj od [x] do [y].	**miš**
16	**Kupnja hrane za piknik** Učenici će naučiti kako raditi sa varijablama: postavljanje različitih početnih vrijednosti, korištenjem uvjeta za usporedbu vrijednosti varijabli, promjenom vrijednosti varijabli, korištenjem varijabli za brojanje (ne) zdrave hrane. Osim toga, ponovit će dodavanje i spajanje teksta te naredbu ako.	**piknik**
17	**Operacije** Učenici će poboljšati svoje znanje o varijablama, slučajnim brojevima, petljama, emitiranju.	**operacije**
18	**Recikliranje** Učenici će poboljšati prethodno stečeno znanje te će proširiti scenarij igre sa novim objektima, kodom i promjenom koda s obzirom na nove objekte. Učenici će moći restrukturirati kod.	**recikliranje**
19.1	**Sviranje klavira_1** Studenti će upoznati kodiranje i sviranje melodija te će poboljšati svoje prethodno stečeno znanje o varijablama, petlji, uvjetnim, emitiranim i ostalim događajima.	**glazba**
19.2	**Sviranje klavira_2** Učenici će naučiti svirati glazbu i mijenjati kostim klikom na sprite.	**klavir**
20	**Test** Učenici će poboljšati svoje znanje i proširiti scenarij igre s novom pozadinom, kodom i promjenom koda u odnosu na nove pozornice.	**test_cro**

BULGARIAN Scenarios

N.	TITLE	CODE
12	**Подбор на здравословна храна** Учениците ще се научат как да се придвижат на случаен принцип с X стъпки и да изберат позиция, както и как да използват променливи и условности за предотвратяване на друго събитие.	**food_bg**
13	**Разказвач** Учениците ще научат как да планират разказването на истории, как да използват излъчвани съобщения за синхронизиране на дейностите на спрайтове и промени на сцената.	**story_bg**
14	**Рисуване** Ученикът ще се научи да рисува в Snap !, да променя цвета и дебелината на писалката и да използва променливи и условия, които причиняват ново събитие.	**drawing_bg**
15	**Хванете мишката** Учениците ще бъде запознати с концепцията за присвояване на случвайни стойности на променливи. Ще се научат как да използват блока Operators/pick random[x]to[y] block.	**mouse_bg**
16	**Купуване на храна за пикник** Учениците ще се научат как да работят с променливи: задаване на различни начални стойности, използване на условия за сравняване на стойностите на променливите, промяна на стойностите на променливите, използване на променливи за броене на (не) здравословна храна. В допълнение, те ще разучат добавяне на текст, съединяване на текстове и if оператор.	**picnic_bg**
17	**Операции** Учениците ще подобрят знанията си за променливи, случайни числа, цикли, Broadcast.	**oper_bg**
18	**Рециклиране** Учениците ще подобрят знанията си и ще разширят сценария на играта с нови обекти, код и променлив код по отношение на новите спрайтове. Те ще бъдат обучени как да редактират кодове.	**recycling_bg**
19.1	**Посвири на пиано Версия 1** Учениците ще се запознаят с кодирането и свиренето на мелодии и ще подобрят знанията си за променливи, цикли, условия, излъчване и други събития.	**music_bg**

Co-funded by the Erasmus+ Programme of the European Union

19.2	**Посвири на пиано Версия 2** Учениците ще се научат как да свирят музика и да сменят костюма, като кликнат върху спрайт.	**piano_bg**
20	**Тест** Учениците ще подобрят знанията си и ще разширят сценария на играта с нов фон, код и променлив код по отношение на новите сцени.	**test_bg**
21	**Опростена игра PACMAN** Учениците ще приложат своите знания за движение в лабиринт с помощта на сензорен цветен блок. Те ще научат концепцията за клониране на обект с ограничения за позицията и как да създадат много прост персонаж, който не е играч, с опция за произволно движение.	**pacman_bg**

GREEK Scenarios

N.	TITLE	CODE
12	**Πιάνοντας υγιεινά τρόφιμα** Οι μαθητές θα μάθουν πώς να κινούνται για Χ τυχαία βήματα και να επιλέγουν μια θέση και επίσης πώς να χρησιμοποιούν μεταβλητές και συνθήκες για την πρόληψη άλλου συμβάντος.	**food_gr**
13	**Διήγηση ιστορίας** Οι μαθητές θα μάθουν πώς να σχεδιάζουν την αφήγηση μιας ιστορίας, πώς να χρησιμοποιούν τα μηνύματα μετάδοσης για συγχρονισμό των δραστηριοτήτων των στοιχείων με τις αλλαγές στη σκηνή.	**story_bg**
14	**Ζωγραφική** Ο μαθητής θα μάθει να ζωγραφίζει με το Snap!, να αλλάζει το χρώμα και το πάχος του στυλό και πώς να χρησιμοποιεί μεταβλητές και συνθήκες για ένα νέο συμβάν.	**drawing_gr**
15	**Πιάσε το ποντίκι** Ο μαθητής θα μάθει την έννοια της πολλαπλής ανάθεσης τυχαίας τιμής. Θα μάθει πώς να χρησιμοποιεί τους τελεστές / επιλέγει τυχαία [x] έως [y].	**mouse_gr**
16	**Αγοράζοντας φαγητό για πικνίκ** Οι μαθητές θα μάθουν πώς να δουλεύουν με μεταβλητές: ανάθεση διαφορετικής αρχικής τιμής, χρήση συνθήκης για σύγκριση της τιμής των μεταβλητών, αλλαγή της τιμής των μεταβλητών, χρήση μεταβλητών για την καταμέτρηση (μη) υγιεινών τροφίμων. Επιπλέον, θα επαναλάβουν την προσθήκη κειμένου, την ένωση κειμένων και τη δήλωση εάν.	**picnic_gr**
17	**Λειτουργίες** Οι μαθητές θα βελτιώσουν τις γνώσεις τους σχετικά με μεταβλητές, τυχαίους αριθμούς, βρόχους και μετάδοση.	**oper_gr**
18	**Ανακύκλωση** Οι μαθητές θα βελτιώσουν τις γνώσεις τους και θα επεκτείνουν το σενάριο του παιχνιδιού με νέα αντικείμενα, κώδικα και ανακατασκευή κώδικα σε σχέση με τα νέα στοιχεία. Θα εκπαιδευτούν στην ανακατασκευή κώδικα.	**recycling_gr**
19.1	**Παίξτε πιάνο_1** Οι μαθητές θα μάθουν να γράφουν κώδικα και να παίζουν μελωδίες και θα βελτιώσουν τις γνώσεις τους σχετικά με μεταβλητές, βρόχους, συνθήκες, μετάδοση γεγονότων και άλλα γεγονότα.	**music_gr**
19.2	**Παίξτε πιάνο_2**	**piano_gr**

	Οι μαθητές θα μάθουν πώς να παίζουν μουσική και να αλλάζουν κοστούμι κάνοντας κλικ σε ένα στοιχείο.	
20	**Δοκιμαστικό μάθημα - Αλλαγή εμφάνισης και στροφή** Μάθετε πώς να αλλάζετε την εμφάνιση του στοιχείου για να κάνετε ένα κινούμενο σχέδιο, αλλάζοντας μεταξύ διαφορετικών τύπων περιστροφών.	**test_gr**
21	**Απλοποιημένο παιχνίδι PACMAN** Οι μαθητές θα επανεξετάσουν τις γνώσεις τους σχετικά με την κίνηση μέσα σε ένα λαβύρινθο. Θα μάθουν την έννοια της κλωνοποίησης του αντικειμένου με περιορισμό θέσης και πώς να δημιουργήσουν έναν πολύ απλό χαρακτήρα με τη δική του τυχαία κίνηση.	**pacman_gr**

PORTUGUESE Scenarios

N.	TITLE	CODE
12	**Comendo comida saudável** Os alunos aprendem a mover-se aleatoriamente X passos e a escolher uma posição. Aprendem a usar variáveis e instruções condicionais para prevenir um evento.	**food_PT**
13	**Narrativa** Os alunos aprendem a planejar a narração de histórias e como usar mensagens para sincronização das atividades dos sprites e mudanças de palco.	**storytelling_PT**
14	**Desenhando** Os alunos aprendem a desenhar no Snap!, a alterar a cor e a espessura da caneta e a usar variáveis e condicionais que geram um novo evento.	**drawing_PT**
15	**Apanhar o rato** Os alunos aprendem o conceito de atribuição de valores aleatórios de múltiplas variáveis. Aprendem ainda como usar o bloco de operadores de escolha aleatória [x] a [y].	**mouse_PT**
16	**Comprando comida para um piquenique** Os alunos aprendem a trabalhar com variáveis: valores iniciais, operadores condicionais para comparar o valor das variáveis, alterar o valor das variáveis, usar variáveis para contar alimentos saudáveis. Além disso, eles vão adicionar e juntar textos.	**picnic_PT**
17	**Operações** Os alunos vão melhorar os seus conhecimentos sobre variáveis, números aleatórios e ciclos.	**operation_PT**
18	**Reciclagem** Os alunos vão aumentar um cenário do jogo com novos objetos, código e código em relação a novos sprites. Aprendem ainda sobre refatoração de código.	**recycling_PT**
19.1	**Tocar piano_1** Os alunos aprendem sobre codificação e reprodução de melodias e melhoram os seus conhecimentos sobre variáveis, ciclos, instruções condicionais e outros eventos.	**music_PT**
19.2	**Tocar piano_2** Os alunos aprendem a tocar música e a mudar de traje clicando num sprite.	**piano_PT**
20	**Teste** Os alunos irão melhorar os seus conhecimentos alargando o cenário do jogo com um novo fundo, código e código mutável em relação a novos palcos.	**test_PT**

21	**Jogo PACMAN simplificado** Os alunos irão rever os seus conhecimentos sobre o movimento dentro de um labirinto com o uso do bloco de reconhecimento de cores. Aprendem o conceito de clonar o objeto com restrições de posição e como criar um personagem não-jogador muito simples com o seu próprio movimento aleatório.	**pacman_PT**

TURKISH Scenarios

N.	TITLE	CODE
12	**Sağlıklı yiyeceği yakalamak** Öğrenciler, belirli bir adım atmak için rastgele hareket etmeyi, bir konum seçmeyi ve ayrıca diğer olayları önlemek için değişkenleri ve koşulluları nasıl kullanacaklarını öğreneceklerdir.	**yemek**
13	**Hikaye Anlatma** Öğrenciler hikaye anlatımını nasıl planlayacaklarını, karakterin aktiviteleri ile sahne değişikliklerinin senkronizasyonu için yayın mesajlarının nasıl kullanılacağını öğrenecekler	**hikaye**
14	**İklimi İyileştirmek-Çizim Yapma** Öğrenci, Snap!'te çizim yapmayı, rengi ve kalem kalınlığını değiştirmeyi ve yeni bir olaya neden olan değişkenler ile koşullu ifadeleri nasıl kullanacağını öğrenecek.	**iklim**
15	**Fareyi yakalamak** Öğrencilere çok değişkenli rastgele değer atama kavramı tanıtılacaktır. Operatörleri nasıl kullanacaklarını ve rastgele [x] ila [y] bloğu seçmeyi öğrenecekler.	**fare**
16	**Piknik için yemek almak** Öğrenciler değişkenlerle nasıl çalışılacağını öğrenecekler: farklı başlangıç değerleri belirleme, değişkenlerin değerini karşılaştırmak için koşullu ifadeler kullanma, değişkenlerin değerini değiştirme, sağlıklı yiyecekleri saymak (olmayan) için değişkenler kullanma. Ek olarak, metin eklemeyi, metinleri birleştirmeyi ve Eğer (if) ifadesini tekrarlayacaklar.	**piknik**
17	**İşlemler** Öğrenciler değişkenler, rastgele sayılar, döngüler, yayın hakkındaki bilgilerini geliştireceklerdir.	**işlem**
18	**Geri Dönüşüm**	**dönüşüm**

	Öğrenciler bilgilerini geliştirecek ve oyun senaryosunu yeni karakterlere göre yeni nesneler, kodlar ve değişen kodlarla genişletecekler. Kod yeniden düzenleme konusunda eğitilecekler	
19.1	**Piyano Çalmak** Öğrenciler melodileri kodlama ve çalma hakkında bilgi edinecek ve değişkenler, döngüler, koşullu, yayın ve diğer olaylar hakkındaki bilgilerini geliştireceklerdir.	**Piyano1**
19.2	**Piyano Çalmak** Öğrenciler bir karaktere tıklayarak müzik çalmayı ve kostümü değiştirmeyi öğrenecekler.	**piyano2**
20	**Test- Kostümleri değiştirmek ve Dönüşler** Farklı döndürme türleri arasında geçiş yaparak bir animasyon yapmak için karakterin kostümünü nasıl değiştireceğinizi öğrenin	**test**
21	**Basitleştirilmiş PACMAN oyunu** Öğrenciler duyu renk bloğu kullanarak bir labirent içindeki hareketin nasıl kodlanması gerektiği hakkındaki bilgilerini gözden geçireceklerdir. Nesneyi konum kısıtlamaları ile rastgele hareket edecek şekilde basitçe kodlayacakar ve karakterleri klonlamayı öğrenecekler.	**pacman**

REFERENCES

A project of the K-12 Initiative at Stanford University. (2009, 05 06). *Taking Design Thinking to School: Approaches to Integrating the Design Process in K-12 Teaching and Learning.* Retrieved 09 10, 2020, from Stanford Graduate School of Education: https://web.stanford.edu/dept/SUSE/taking-design/presentations/Taking-design-to-school.pdf

Alserri, S., Zin, N., & Wook, T. (2018). Alserri, S. A., Zin, N. A. M., & Wook, T. S. M. T.Gender-based Engagement Model for Serious Games. , , 8(4). . *International Journal on Advanced Science, Engineering and Information Technology, 8*(4), 1350-1357.

Baptista, R., & Vaz de Carvalho, C. (2010). Role Play Gaming and Learning. *Learning Technology, 12*(1).

Combéfis, S., Beresnevičius, G., & Dagiene, V. (2016). Learning Programming through Games and Contests: Overview, Characterisation and Discussion. *Olympiads in Informatics, 10*.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: A 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges, 15*(5), 107-116.

Doorley, S., Holcomb, S., Klebahn, P., Segovia, K., & Utley, J. (2018). *Design Thinking Bootleg.* Retrieved from https://static1.squarespace.com/static/57c6b79629687fde090a0fdd/t/5b19b2f2aa4a99e99b26b6bb/1528410876119/dschool_bootleg_deck_2018_final_sm+%282%29.pdf

Eurostat. (2020, 09 25). *ICT specialists in employment.* Retrieved 11 12, 2020, from eurostat Statistics Explined: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=ICT_specialists_in_employment#Number_of_ICT_specialists

Feldgen, M., & Clua, O. (2004). Games as a motivation for freshman students to learn programming. *34th ASEE/IEEE Frontiers in Education Conference*, (pp. 1079–1084).

Frankovic, I., Hoic-Bozic , N., & Nacinovic-Prskalo, L. (2018). Serious Games for Learning Programming Concepts. *8th Conference edition The Future of Education.* Florence, Italy.

Garris , R., Ahlers, R., & Driskell, J. E. (2002). Games, motivation and learning. Simulation & Gaming. *An Interdisciplinary Journal of Theory, Practice and Research, 33*(4), 43-56.

Gee, J. P. (2003). *What Video Games Have to Teach Us About Learning and Literacy.* New York: Palgrave/Macmillan.

Gee, J. P. (2007). Are video games good for learning? *Curriculum & Leadership Journal, 5*(1).

Geist, E. (2016). Robots, Programming and Coding, Oh My! *Childhood Education, 92*(4), 298-304. doi:10.1080/00094056.2016.1208008

Georgieva, R., & Tuparova, D. (2019). Educatiomnal Computer Game-Based Environments for Teaching Programming and Developmnt of an Algorithmic Thinking - Comparative AnalysisITHMIC THINKING – COMPARATIVE ANALYSIS. *Mathematics and Education in Mathematics, Forty-eighth Spring Conferenceof the Union of Bulgarian Mathematicians* (pp. 150-156). Union of Bulgarian Mathematicians. Retrieved from http://www.math.bas.bg/smb/2019_PK/tom/pdf/150-156.pdf

Georgieva, R., & Tuparova, D. (2020). Design Thinking - helps in school eucation. *Anniversary International Scientific Conference “Synergeticsand ReflectioninMathematics Education”*, (pp. 341-347). Pamporovo.

Grivokostopoulou, F., Perikos, I., & Hatzilygeroudis, I. (2016). An Educational Game for Teaching Search Algorithms. *Proceedings of the 8th International Conference on Computer Supported Education (CSEDU 2016)* (pp. 129–136). Setubal. Portugal: SCITEPRESS - Science and Technology Publications, Lda.

Gross, B. (2003). The impact of videogames in education. 8(7). *First Monday, 8*(7).

Hijon-Neira, R., Velazquez-Iturbide, A., Pizarro-Romero, C., & Carrico, C. (2015). Serious games for motivating into programming. *Frontiers in Education Confere.*

IDEO Design thinking. (2008, 09 7). *Definitions of design thinking*. Retrieved 11 20, 2020, from https://designthinking.ideo.com/blog/definitions-of-design-thinking

Innovation Starter. (2015, 06 04). *What is design thinking?* Retrieved 11 20, 2020, from http://innovationstarterbox.bg/resources/kakvo-e-dizain-mislene/

Jemmali , C. (2016). May's Journey: A serious game to teach middle and high school girls programming. Worcester Polytechnic Institute. Retrieved from https://digitalcommons.wpi.edu/etd-theses/455

Johnson, C., & all, a. (2016). Game Development for Computer Science Education. *the 2016 ITiCSE Working Group Reports.*

Johnston, R. T., & de Felix, W. (1993). Learning from video games. *Computer in the Schools* , 199-233.

Kafai, Y. (1995). Making game artifacts to facilitate rich and meaningful learning. *Annual Meeting of the American Educational Research Association*, (pp. 1–20). San Francisco.

Karakehayova, M. (2019). Opportunities for Using Design Thinking in School Education. *Education and Development, 3*.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012). A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. *Procedia - Social and Behavioral Sciences, 47*, 1991-1999.

Kelleher , C., Pausch, R., & Kiesler, S. (2007). Storytelling Alice motivates middle school girls to learn computer programming. *Proc. CHI 2007.*, (pp. 1455-1464).

Luka, I. (2014). Design thinking in pedagogy. 2014, 5,. *The Journal of Education, Culture, and Society, 5*, 63-74.

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2014). CMX: Implementing an MMORPG for learning programming, 8th European Conference on Games Based Learning. *8th European Conference on Games Based Learning - ECGBL 2014.* Berlin.

Malone, T. (1981). What makes computer games fun? *Byte, 6*(12), 258-277.

Malone, T. W. (1981b). Toward a theory of intrinsically motivating instruction. *Cognitive Science, 4*.

Mathrani, A., Christian, S., & Ponder-Sutton, A. (2016). PlayIT: Game Based Learning Approach for Teaching Programming Concepts. *Educational Technology & Society, 19*(2), 5-17.

Meerbaum-Salant , O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with Scratch. *Computer Science Education, 23*(3), 239-264. doi:10.1080/08993408.2013.832022

Michael, D. R., & Chen, S. (2006). *Serious Games: Games That Educate, Train, and Inform.* Boston: Course Technology PTR.

Miljanovic, M., & Bradbury, J. (2016). Robot on!: a serious game for improving programming comprehension. *Proceedings of the 5th International Workshop on Games and Software Engineering.* Austin, Texas, USA.

Miljanovic, M., & Bradbury, J. (2018). A Review of Serious Games for Programming. *4th Joint International Conference on Serious Games.* Darmstadt, Germany.

Naiman, L. (2019, 06 10). *Design Thinking as a Strategy for Innovation*. Retrieved 11 21, 2020, from Creativity at work: https://www.creativityatwork.com/design-thinking-strategy-for-innovation/

Nančovska Šerbec, I., Kaučič, B., & Rugelj, J. (2008). Pair programming as a modern method of teaching computer science. *International journal on emerging technologies in learning, 3*, 45-49.

Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning Basic Programming Concepts by Creating Games with Scratch Programming Environment. , 191, . *Procedia - Social and Behavioral Sciences, 191*, 1479-1482. doi:doi:10.1016/j.sbspro.2015.04.224

Paavola, S., & Hakkarainen, K. (2005). The Knowledge Creation Metaphor – An Emergent Epistemological Approach to Learning.14. *Sci Educ, 14*, 535-546.

Phan, M., Jardina, J., Hoyle, S., & Chaparro, B. (2012). Examining the role of gender in video game usage, preference, and behavior. *Human Factors and Ergonomics Society 51*, (pp. 1496–1500.).

Pivec, M., & Kearney, P. (2007). Games for Learning and Learning from Games. *Informatica, 31*, 419-423.

Pivec, M., Koubek, A., & Dondi, C. (2004). *Guidelines for Game-Based Learning. Lengerich.* Pabst Science Publishers.

Prensky , M. (2001). *Digital Game-Based Learning.* New York: Mc-Graw-Hill.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond , E., Brennan , K., . . . Kafai, Y. (2009). Scratch: Programming for All. *Communication of the ACM, 52*, 60-67.

Rieber , L. P., Smith , L., & Noah, D. (1998). The value of serious play. *Educational Technology, 38*(6), 29-37.

Rugelj, J. (2016). Serious computer games design for active learning in teacher education. (C. Carvalho , P. Escudeiro, & A. Coelho, Eds.) *Serious games, interaction, and simulation : revised selected papers. Lecture notes of the institute for computer sciences, social informatics and telecommunications engineering, 161*, 94-102.

Rugelj, J., & Lapina, M. (2019). Game design based learning of programming. In J. Rugelj, & M. Lapina (Ed.), *Proceedings of the International Scientific Conference Innovative Approaches to the Application of Digital Technologies in Education and Research (SLET-2019), May 20-23, CEUR workshop proceedings. 2494.* Stavropol-Dombay, Russia: Aachen: CEUR-WS.

Shabanah, S., Chen, J., Wechsler, H., Carr, D., & Wegman, E. (2010). Designing Computer Games to Teach Algorithms. *Seventh International Conference on Information Technology: New Generations, ITNG 2010*, (pp. 1119-1126). Las Vegas, NV.

Shute, V. J., & Ke, F. (2012). Games, Learning, and Assessment. In D. Ifenthaler, D. Eseryel, & X. Ge, *Assessment in Game-Based Learning: Foundations, Innovations, and Perspective.* New York, USA: Springer.

Shute, V. J., Rieber, L. P., & Van Eck, R. (2012). Games ... and ... Learning. In R. A. Reiser, & J. V. Dempsey, *Trends and issues in instructional design and technology (3rd ed.).* (pp. 321-332). Boston: Pearson Education.

Sykes, E. (2007). Determining the Effectiveness of the 3D Alice Programming Environment at the Computer Science I Level. *Journal of Educational Computing Research, 36*(2), 223-244. doi:10.2190/J175-Q735-1345-270M

Tuparova, D., Nikolova, E., & Tuparova, E. (2020). Integrated game-based learning in an informatics secondary course: Is there a difference between girls' and boys' achievements? *CSEDU 2020 - Proceedings of the 12th International Conference on Computer Supported Education*, *1*, pp. 702-709. Retrieved 10 12, 2020, from https://www.scitepress.org/Link.aspx?doi=10.5220/0009818407020709

Tuparova, D., Tuparov, G., & Veleva, V. (2019 b). Girls’ and boys’ viewpoint on educational computer games. *European Conference on Games-based Learning*, (pp. 757–766).

Vahldick, A. (2017). Aperfeiçoamento das Competências de Resolução de Problemas na Aprendizagem Intodutória de Programação de Computadores usando um jogo sério digital. Faculdade de Ciências e Tecnologia da Universidade de Coimbra. Retrieved from http://hdl.handle.net/10316/79528

van Eck, R. (2006). Digital Game-Based Learning: It's Not Just the Digital Natives Who Are Restless. *EDUCAUSE Review, 41*(2).

Vermeulen, L., Van Looy, J., Courtois, C., & De Grove, F. (2011). Girls will be girls: a study into differences in game design preferences across gender and player types. *Under the mask: perspectives on the gamer conference*, (pp. 1-21).

Weintrop, D., & Wilensky, U. (2015). To Block or not to Block, That is the Question: Students' Perceptions of Blocks-based Programming. *Interaction Design and Children*, (pp. 199-208).

Whitton, N. (2009). *Learning with Digital Games: A Practical Guide to Engaging Students in Higher Education.* New York: Routledge.

Whitton, N., & Moseley, A. (2012). *Using Games to Enhance Learning and Teaching: A Beginner's Guide.* New York: Routledge.

Wolz, U., Barnes, T., & Bayliss, J. (2009). Girls do like playing and creating games. *ACM SIGCSE Bulletin, 41*(1), 199–200.

Wolz, U., Maloney, J., & Pulimood, S. (2008). 'Scratch' Your Way to Introductory CS. *SIGCSE Bulletin, 40*, 298-299.

Zapušek, M., & Rugelj, J. (2013). Learning programming with serious games. *EAI Endorsed Trans. on Game-Based Learning, 13*, 1-12.