O3 – Instructional Support Content

COLLECTION OF GAME DESIGN BASED LEARNING SHEETS TARGETING TEACHERS

Document Data

Deliverable: O3/A1 - Collection of game design based learning sheets targeting teachers

Intellectual Output No - Title: O3 – Instructional Support Content

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

Partners involved: University of Ljubljana (Slovenia), University of Rijeka (Croatia)

TABLE OF CONTENTS

INTRODUCTION

Leading psychologist of the last century identified play as one of the most important activities for the development of important life skills, regardless of age or stage of development. Child through the play quick adopts to new circumstances and handles change with ease. When he plays, he discovers basic concepts from real word and first fundamental relationships between them are made.

Nowadays, games are more commonly used in the earliest stages of a child's development at home and in kindergarten. Learning in school is still too often based on traditional transmission of knowledge within a teacher-centered model with passive students. On the other hand, learning theories, developed in the last century, promote new approaches to teaching and learning that are student centered, problem based, directed to higher ordered educational goals on higher taxonomic levels, motivational and often supported by ICT.

CODING4GIRLS approach will encourage participation in programming activities through a "low entry high ceiling approach" that has low knowledge requirements in the beginning while not limiting problem-solving challenges for more advanced learners. Learners will be encouraged to finish partially completed solutions by adding missing building blocks of code or to create their own solutions. Activities are planned in sequence, from basic ones with only one programming concept to more advanced with multiple programming concepts. As we were preparing learning activities in Snap!, we focused on the identified characteristics of games preferred by girls and on the activities related to the real-world problems.

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.

21 learning sheets corresponding to learning activities have been prepared. Teachers can use the scenarios and games in the proposed sequence or can select them freely according to their preferences and needs. 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.
The learning sheets are available in English as well as the national languages of project partners – Bulgarian, Croatian, Greek, Italian, Portuguese, Slovenian and Turkish.

LEARNING SHEETS

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.

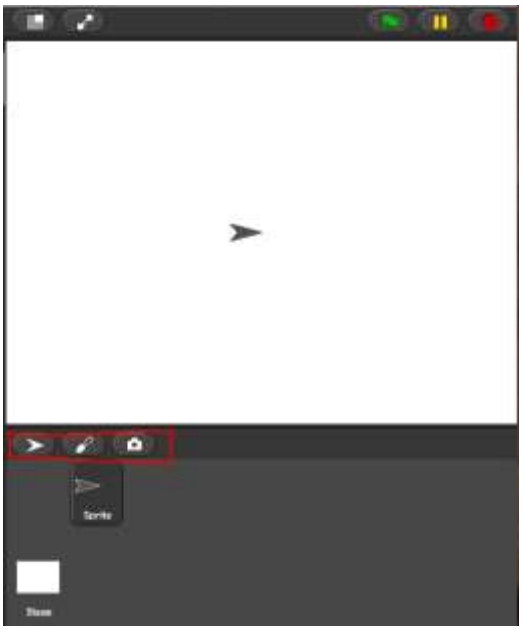
BASIC LEARNING SCENARIOS		
1	**Introduction to Snap! interface** Getting familiar with Snap! visual programming environment	UL
2	**Time to bring your sprite to life** Finding programming blocks, connect them, move a sprite, make sprite say something	UL
3	**Moving around the stage** Making a meaningful 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** Using random values, variables input, conditionals, comparison operators, counter	UL
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

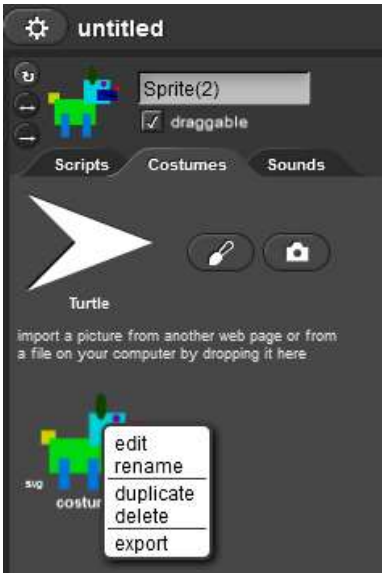
20	**Test**	SWU
21	**Simplified PACMAN game** Using event based object movement, color sensing, Boolean values, checking and responding to two different game states	UL

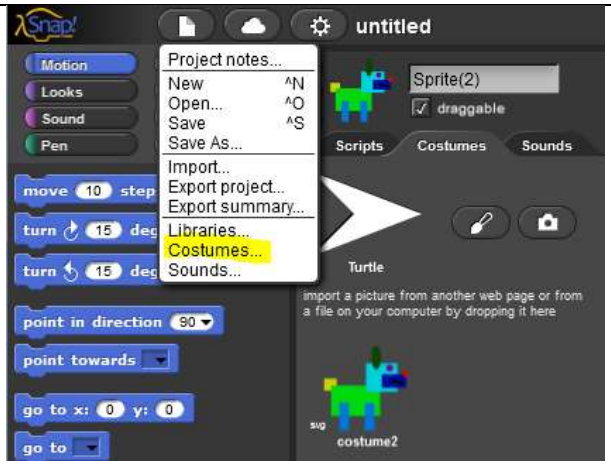
BASIC LEARNING SCENARIOS

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
Teaching Forms	Frontal work Individual work

Teaching summary	(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: 

	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.  [Step 4] To import an already existing costume, click on the icon with a piece of paper drawn on it, and choose Costumes…

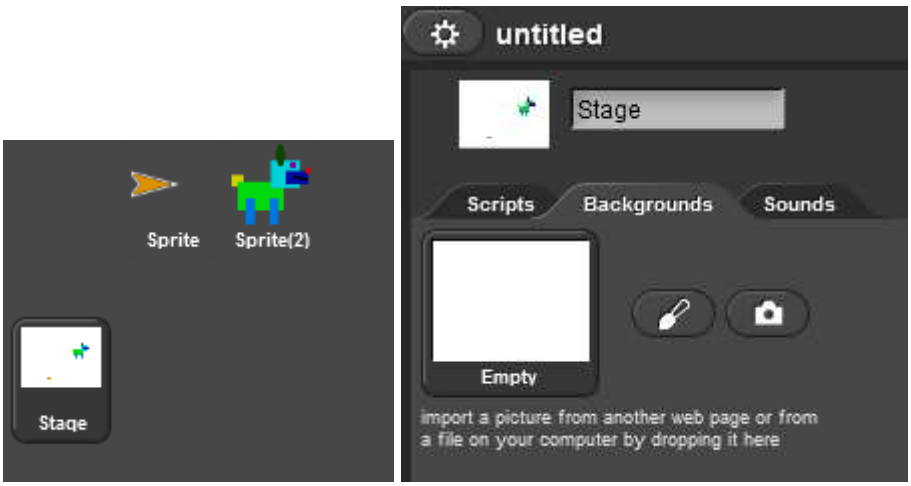


Again, this option will only be shown, when your sprite is clicked on under the stage.

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:



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 

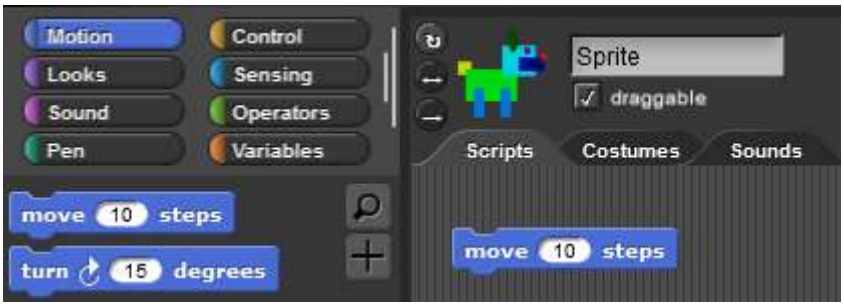
blocks.

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.



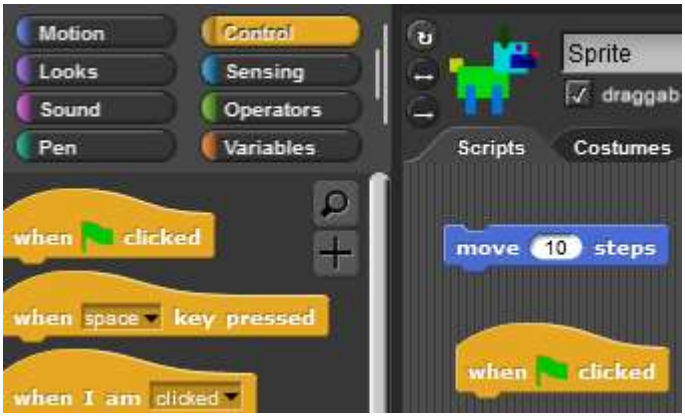
You can double-click on the block inside Scripts tab to execute the code.

[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:



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;

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 

and 

, and explain it to your neighbour.

[Step 6]

You found both say commands in Looks category. The main difference is that with 

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 

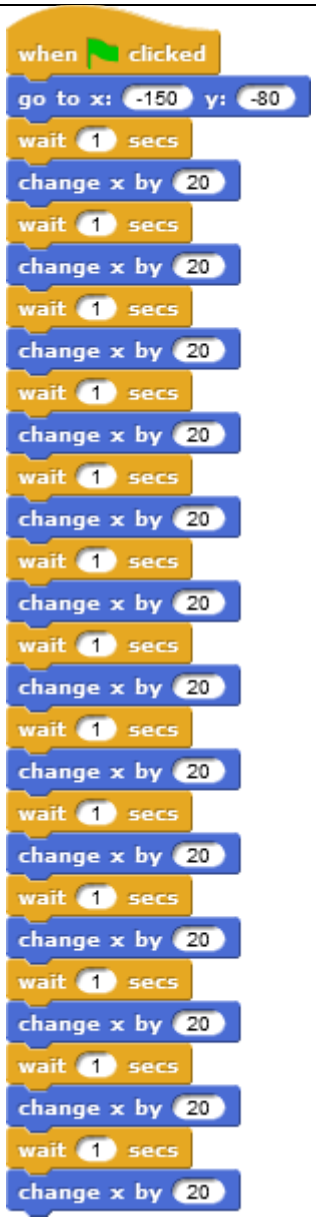
from its position on the left to the right side of the stage. After each move, the character should say

	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  and  blocks to make an animation of a dog moving towards the ball. A possible solution to the task:



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.

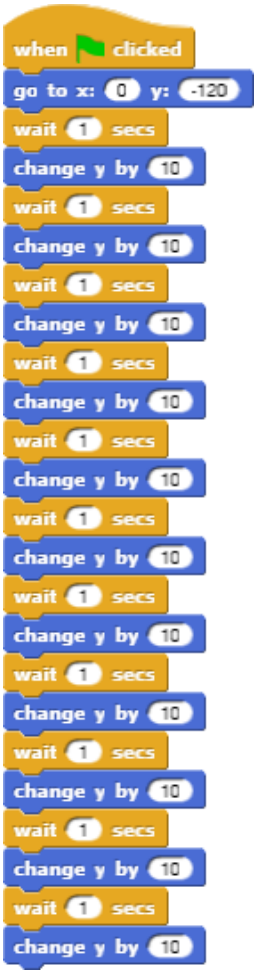
[Task 2]

Open Help monkey climb the tree, and add code to the monkey to fetch the bananas. Use 

and 

blocks, to make an animation of a monkey climbing on the palm tree.

A possible solution of the task:



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 

.

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]

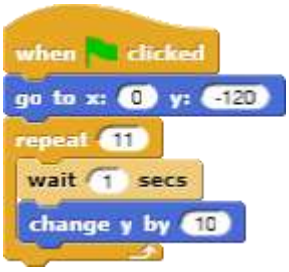
In both tasks you had to interchangeably use two blocks. How many times did you have to **repeat the code?**

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 

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.

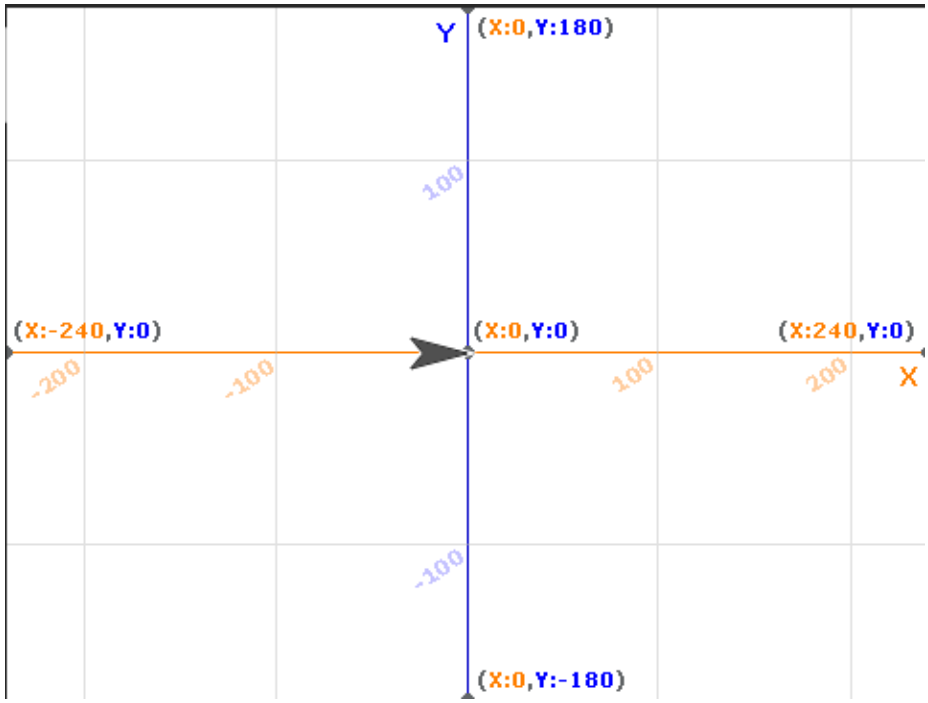
Code for the dog:

Code for the monkey:



Task: Try to make the dog run to the ball and back.

Task: Try to make monkey climb the tree and back down.

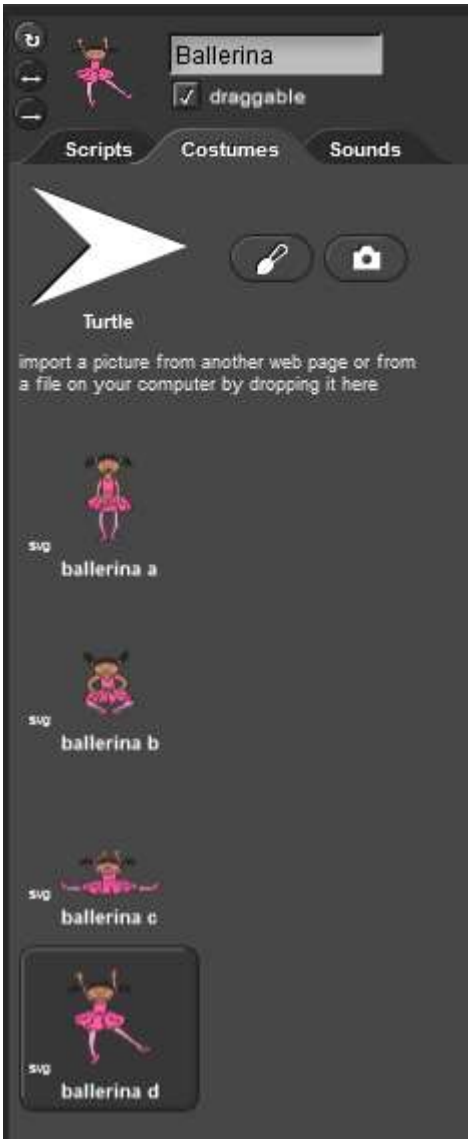
	What did you like the most? You can help yourself with x and y position of the sprite by using XY Grid background in Snap: 
Tools and Resources for the Teacher	• A possible solution to Catch the ball: https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_moving_x • A possible 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:



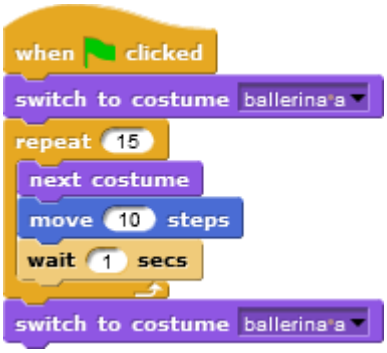
Now go back to Scripts tab and try to create a code, that will start when the green flag is clicked, and 15 times change every second change the appearance of the Ballerina. You will need to use next costume block. Make sure our Ballerina starts and finishes her dance with both legs on the floor. Start and end position are not part of her dance.

Solution:

[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:



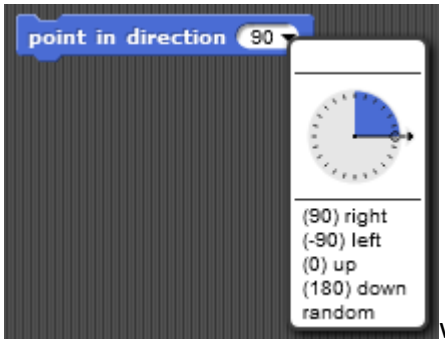
[Step 3]

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:

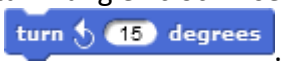
[Step 4]

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

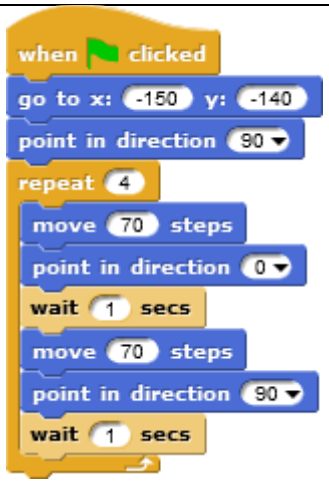
b) you can tell her to turn for a certain angle clockwise 

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.

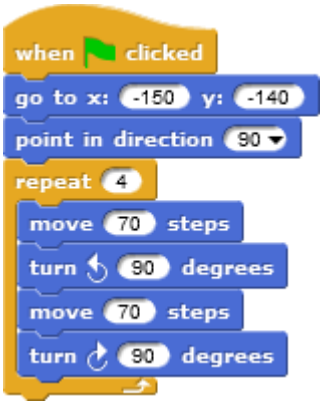
Open https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Find_cheese. Write a program that mouse has to follow to reach the cheese if she has to walk only on the green area. Make mouse point in the direction she is heading and move __ steps block. To see how the mouse moves,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

Solution:



Now try to write a program with turn 90 degrees.

Solution:



[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.  [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.

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.

[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.

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.

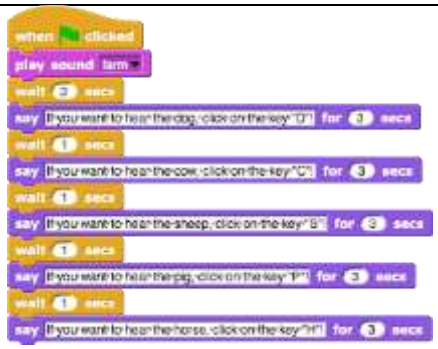
[Step 6]

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.

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



[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

The background

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).

[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 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.

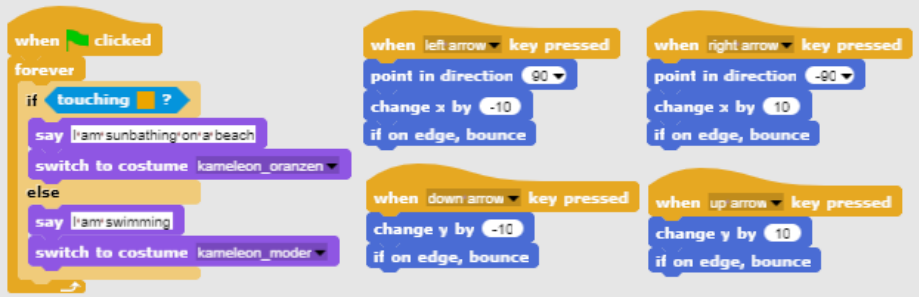
We can choose which color are we going to check and implicitly other color will fall into “else” case. In the sample code we chose sandy color:

[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]



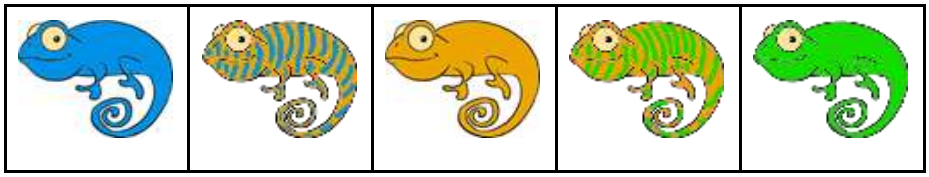
[Full version]

[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).

[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:

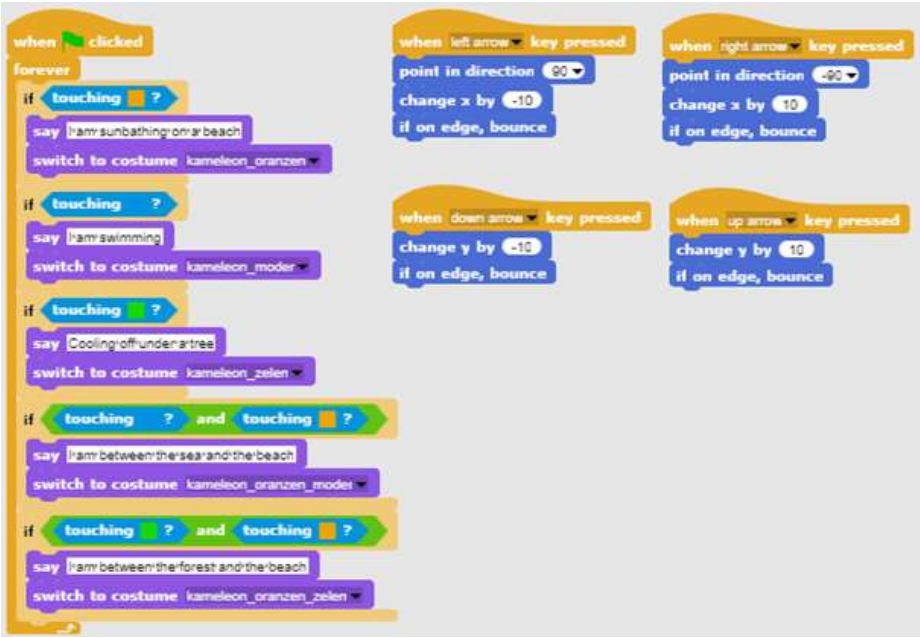
Next we form a logical expression with the use of logical operator AND, because we want to verify if chameleon is touching two colors at the same time:

If 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]



[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).

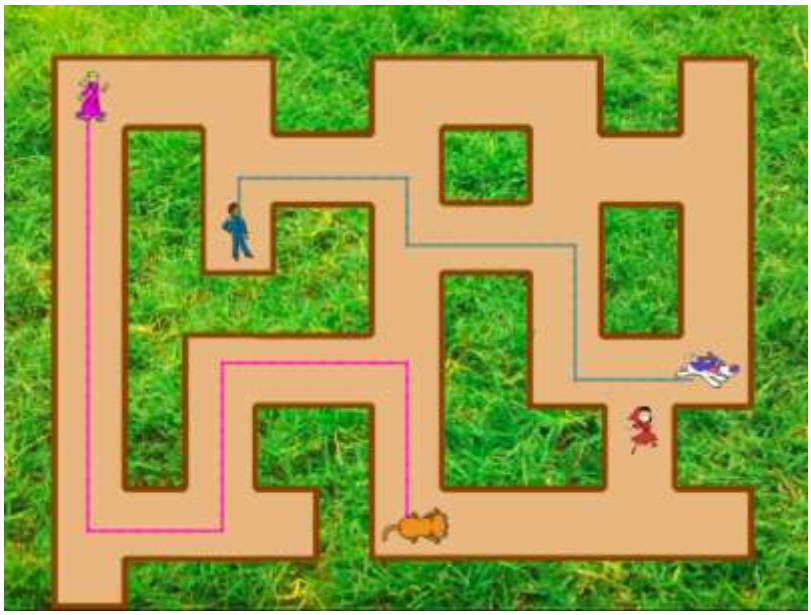
To help them understand the concept of forever loop and how to use it for detecting background color we can include code for detecting two situations: 1) the object is entirely on one color, 2) the object is touching two colors at the same time. We instructed them to complete the code for every case.

Suggested code template:

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:

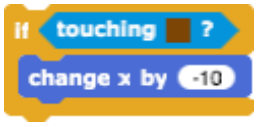
Recommended size for the girl is 8%, other sprites can be bigger.

[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.

[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.



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

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

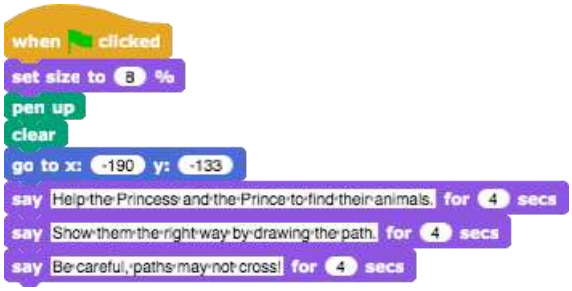
Similar they set pen color by pressing the key.

[Step 6]

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

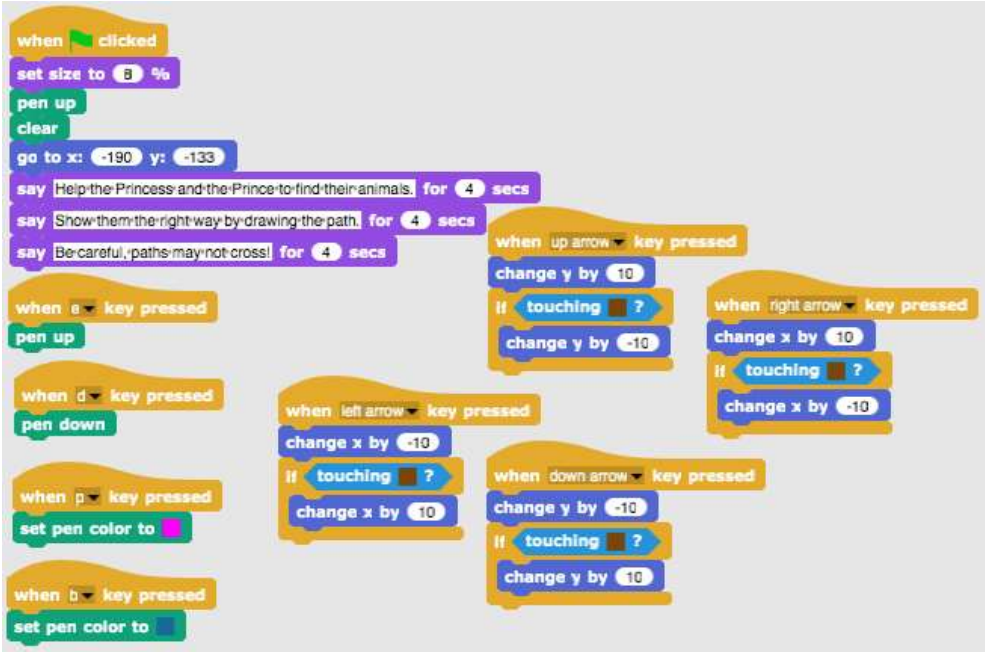
When playing the game, stop it and play it again, students will see that is good to add following blocks: *pen up* (in case it stayed down from the previous playing), *clear* (clears the path from previous playing) and *go to x, y* (the girl always starts at these coordinates, which are inside the path 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.



[Final Code]

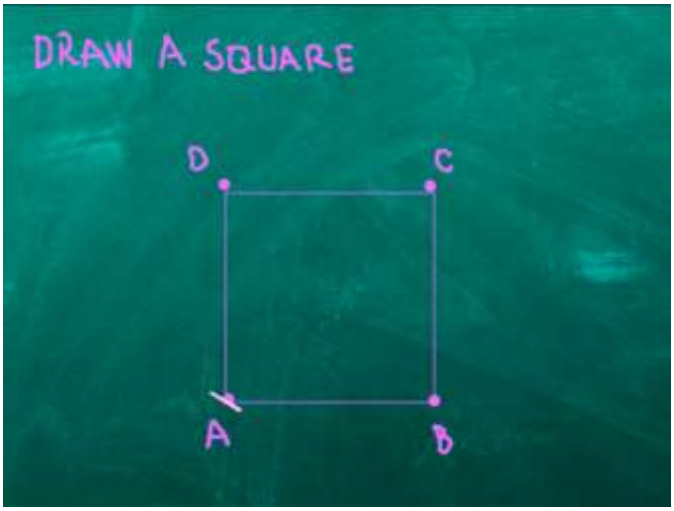
Girl



	E.g. Princess  [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).

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*.

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

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

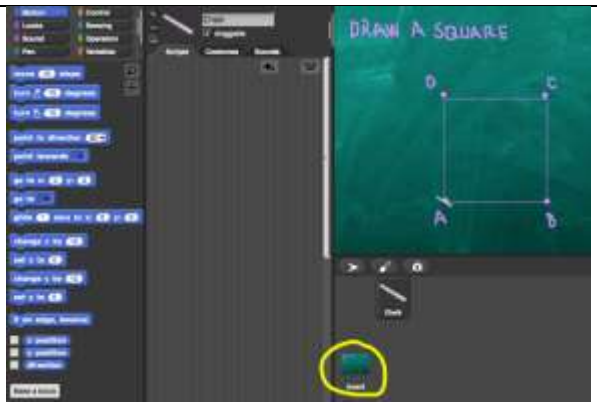
[Step 2]

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*.

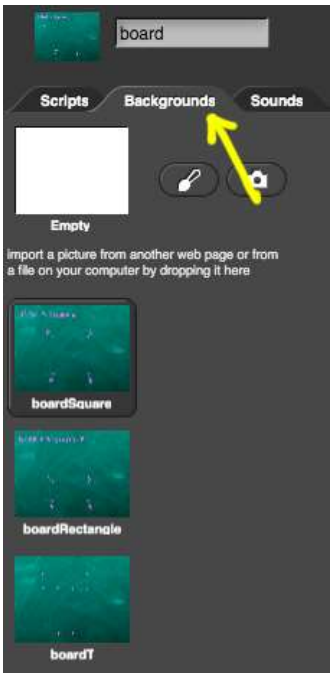
[Step 3]

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:

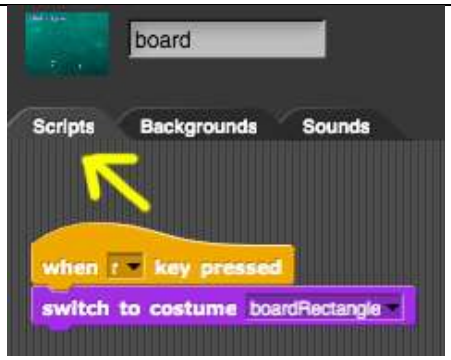
a) We click on the background (named *board*, on the right side of the screen).



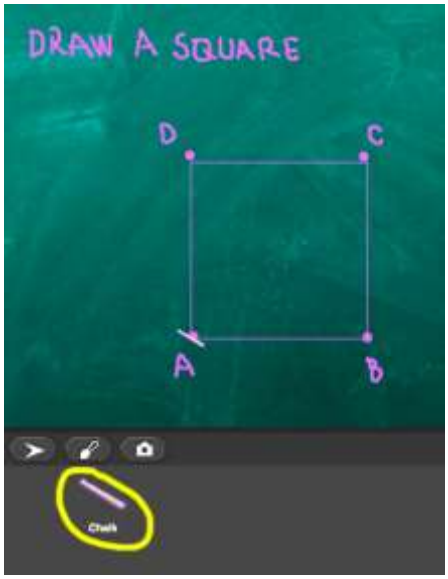
Clicking on *Backgrounds* we can see all three needed backgrounds (*boardSquare, boardRectangle, boardT*), already prepared for this activity.



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*.

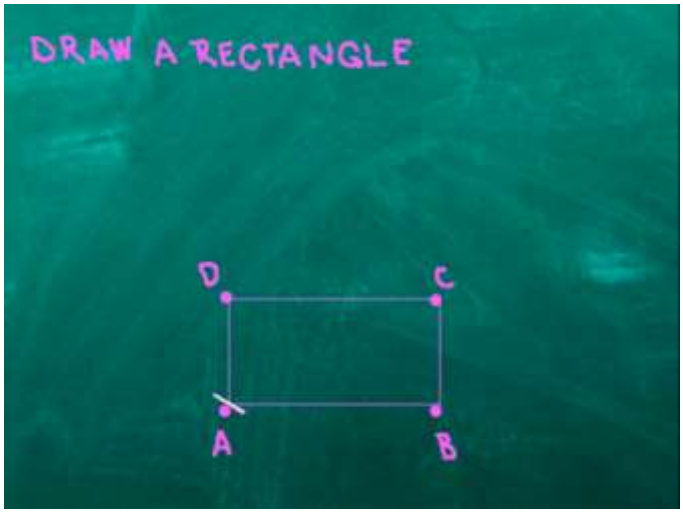


b) We click back on the chalk.



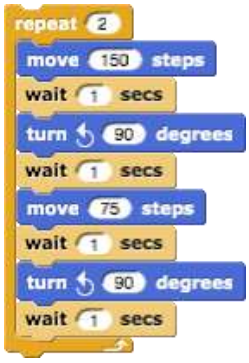
Under the code from [Step 2] students add a block, where they'll tell a player what to do to change the background, which is, press the key "R".

[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.

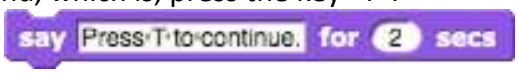


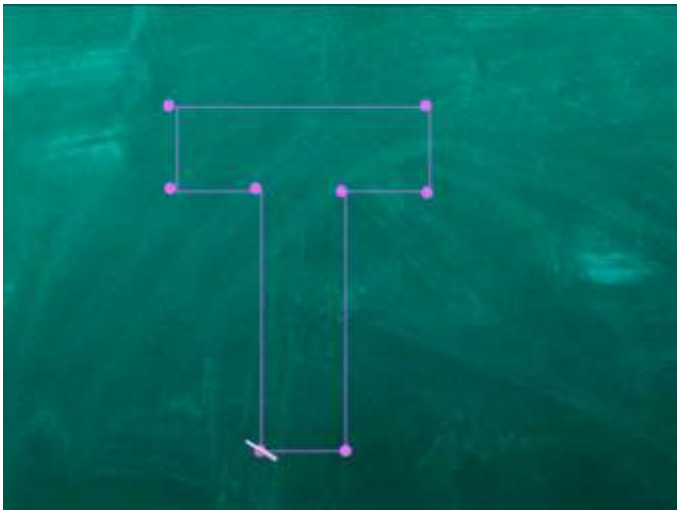
[Step 5]

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*.

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”.



	[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:

[Step 7]

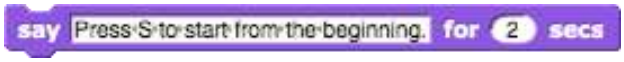
Since we changed the background, we can not return to the first background to draw a square. So students will have to add one last code.

They repeat [Step 3/5].

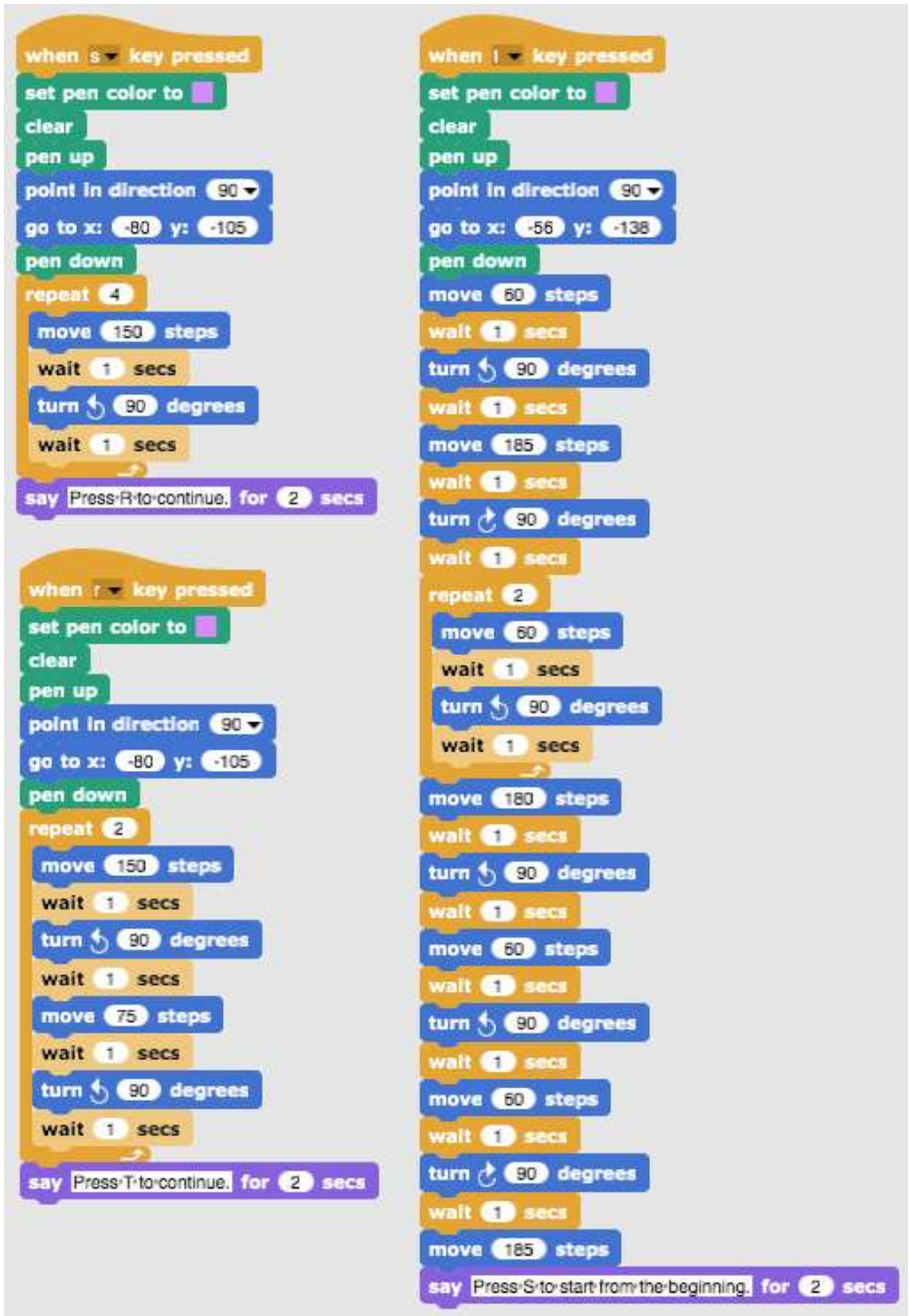
a) They click on the background (named *board*, on the right side of the screen), where they write a code for changing background. They will do this with *when S key pressed* and then *switch to costume boardSquare*.

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”.



[Final Code]



[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

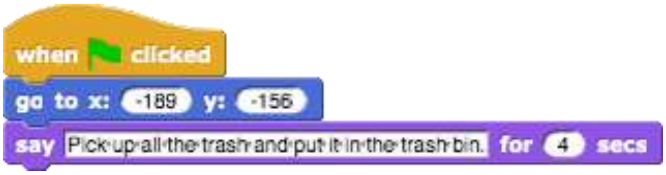
Learning Scenario Title	Picking up trash and cleaning the park
Previous programming experience	Setting starting coordinates Setting size for sprite Adding text for sprite Object movement with arrow keys using events Using conditional *object is touching* for object state
Learning Outcomes	General learning 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.

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.:



[Step 2]

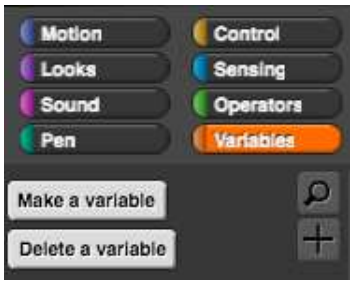
For counting number of trash the girl picked up, we will use variables.

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?



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*.

[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.

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*.

[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.

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.

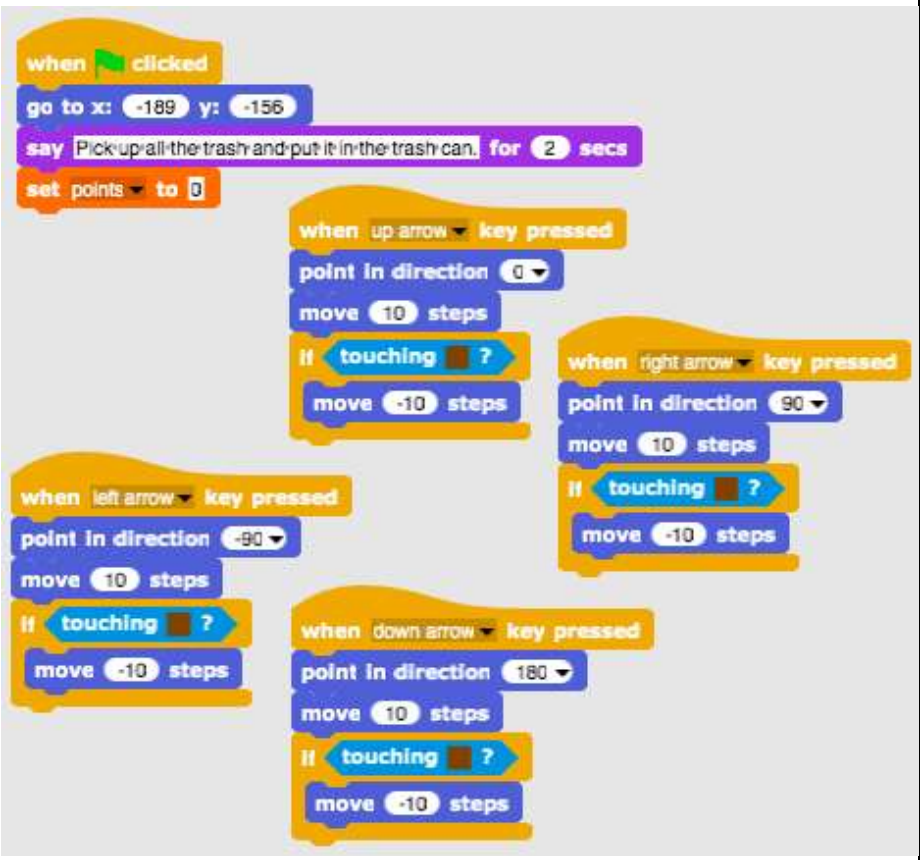
Also this code will activate when the girl touches it.

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.

[Final Code]

Girl



Bottles / Papers

Trash can

	 [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=Pick

	ing%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]

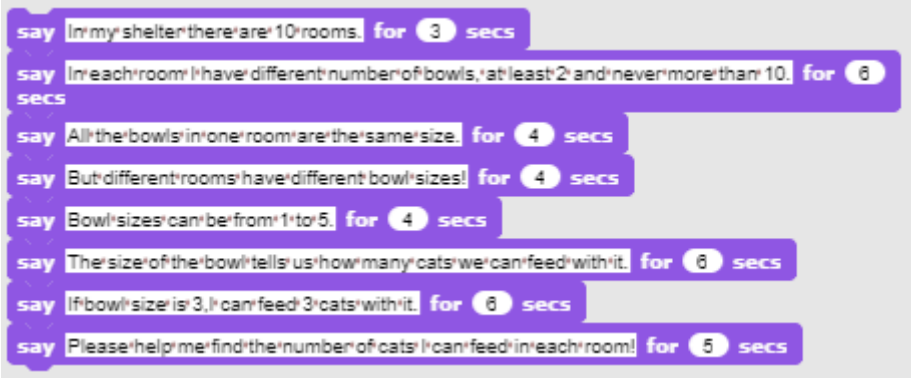
We have to select a new costume for the default turtle sprite that will represent cat shelter keeper.

[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.

[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.



[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]

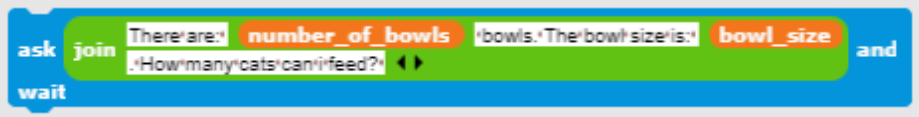
To randomly assign the values for the number of bowls and their size we will use Variables/set [options] value with Operators/pick random [n] to [m].

[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.

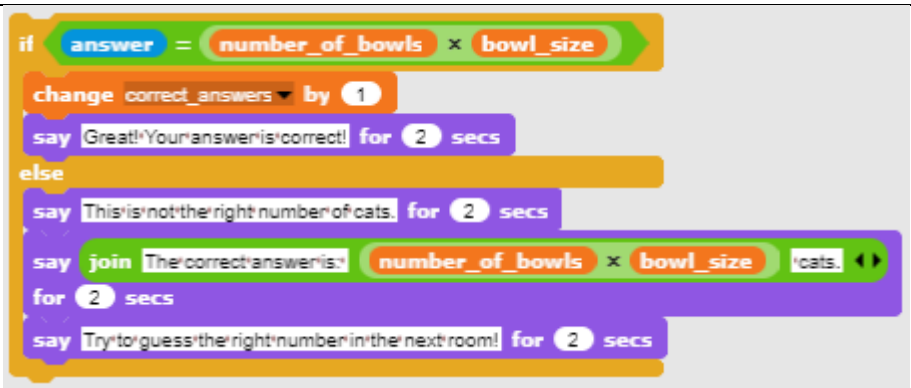
[Step 8]

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



[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.



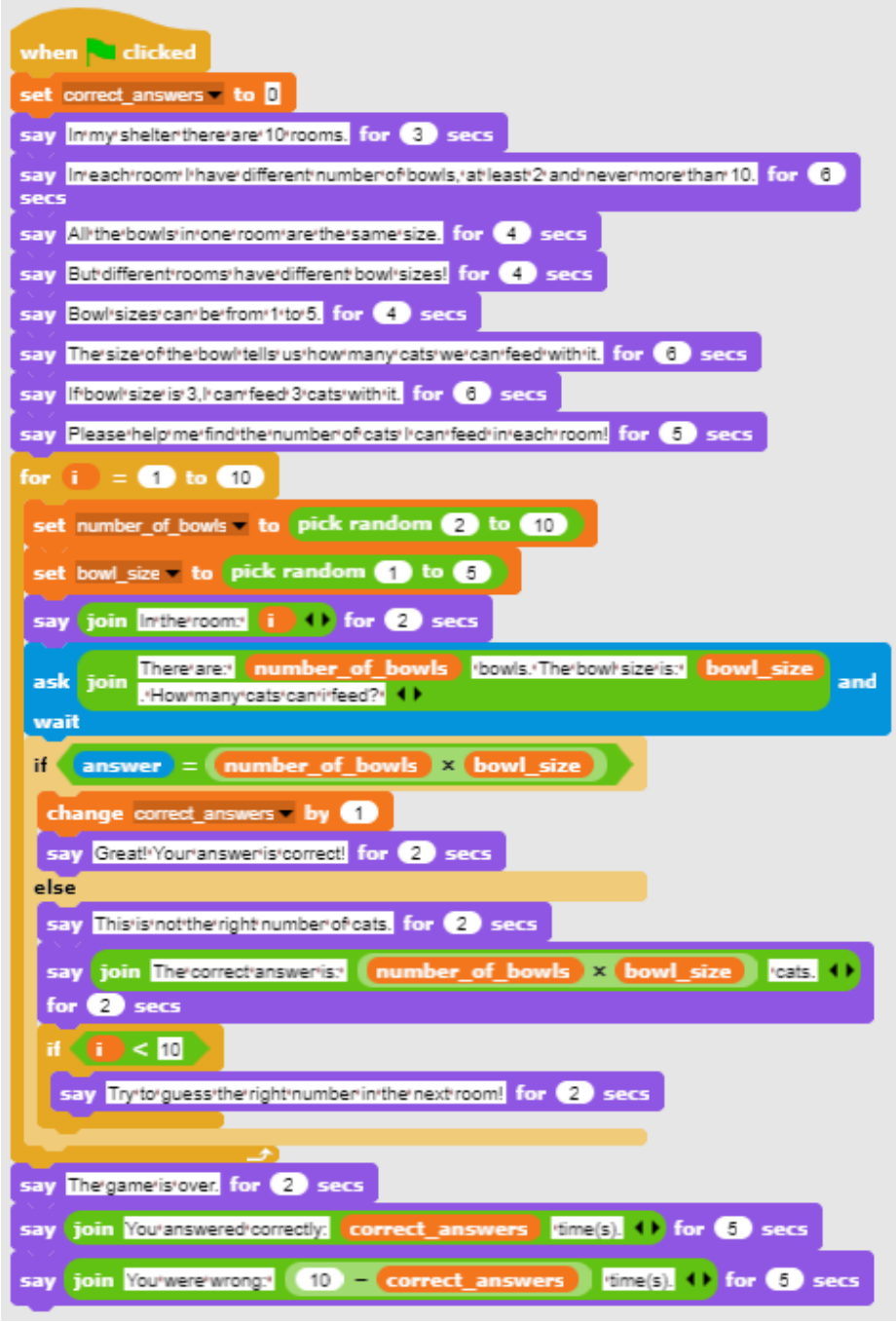
[Step 11]

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]

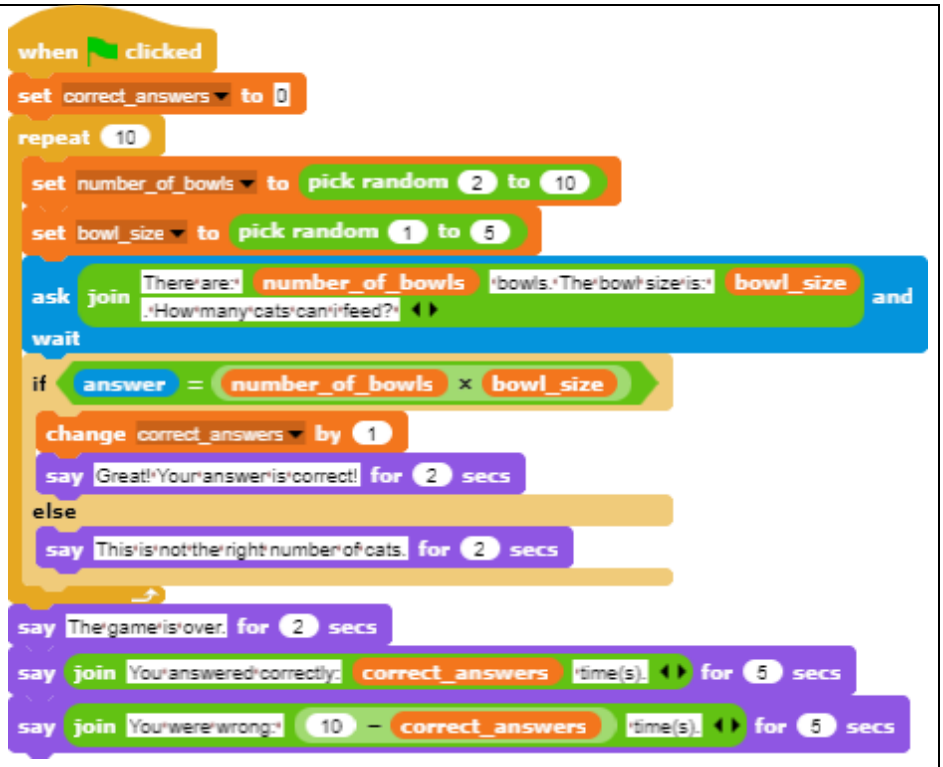
When the loop stops, the game is over. We provide the information about player achievement. Number of correct answers is stored in the correct answer counter; the number of wrong answers can be calculated.

[Final code]



[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> 
</td></tr>
<tr><td>Tools and Resources for the Teacher</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>Resources/materials for the Students</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).

[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.

[Step 5]

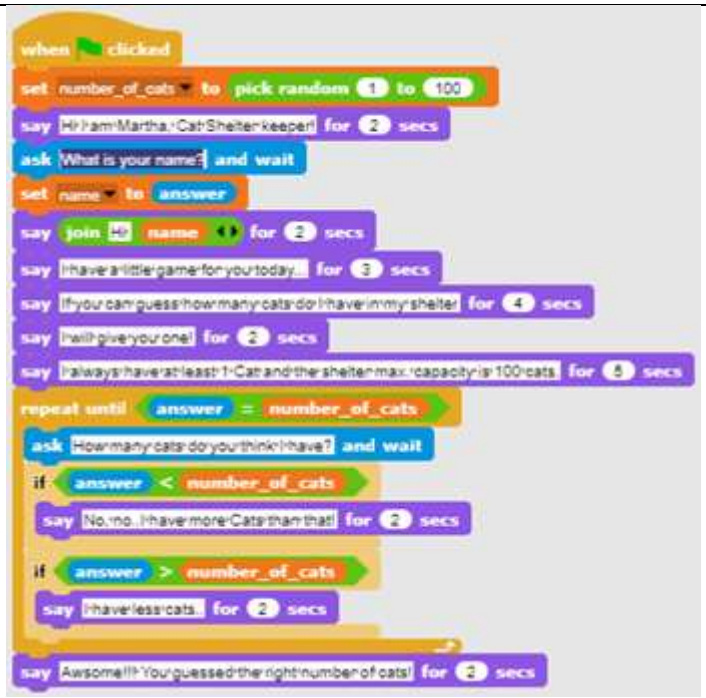
We discuss with students that it is not possible to predict how many times players will have to guess in order to find the right number. She can get very lucky and guess it in her first try, maybe it will take her 5 guesses, or even more, we cannot tell! This is the reason we have to choose the right 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.

[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.

[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.



[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.

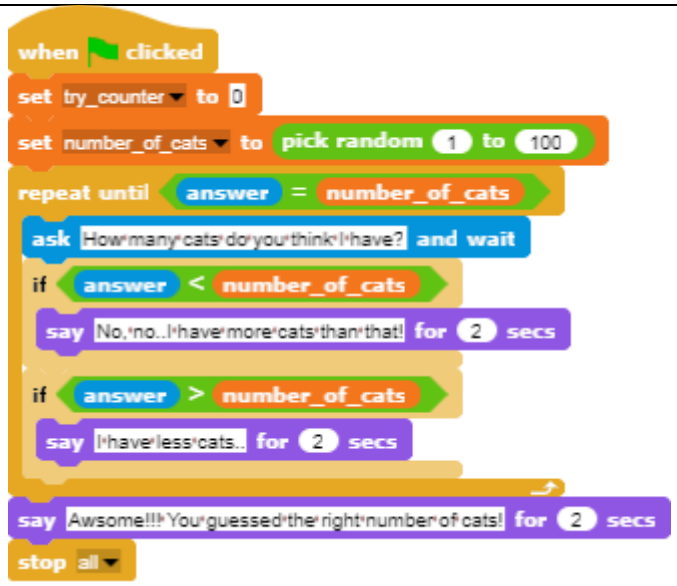
[Step 10]

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]

[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.

	
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:

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:

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.:

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:

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).

[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).

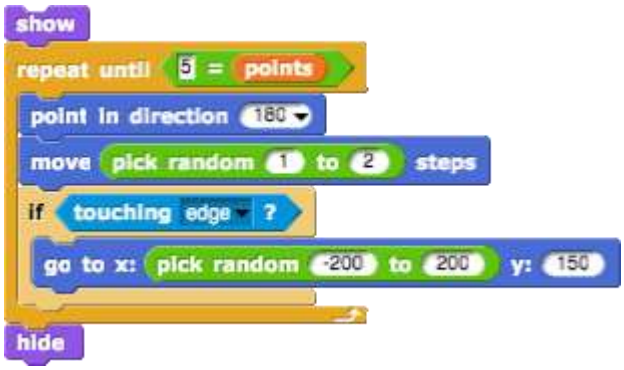
[Step 4]

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.

[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:



[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.

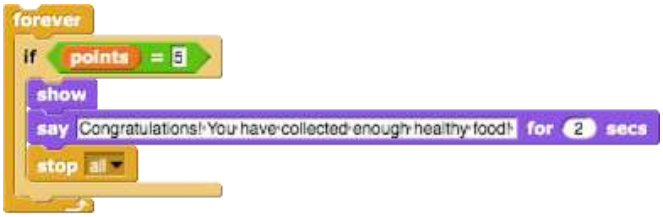
[Step 7]

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).



[Step 8]

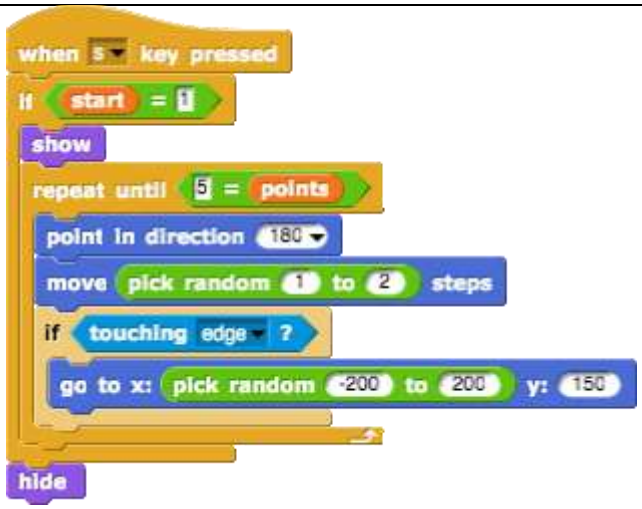
When playing the game again, when students will already know all the instructions (from [Step 1]) and they will surely want to skip them. They can press the "S" before so the game will begin, but the girl will be still talking.

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.

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:

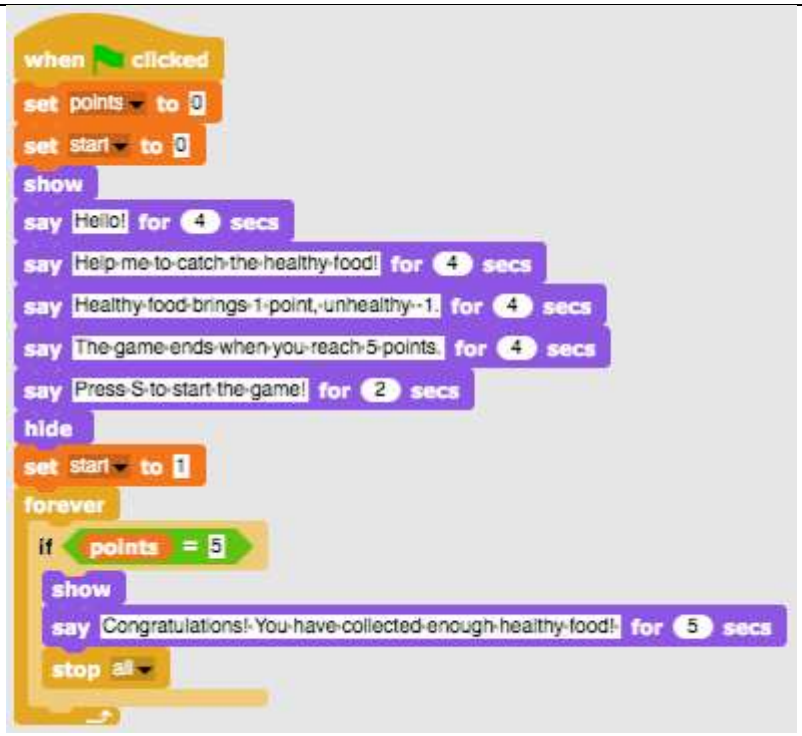


[Step 9]

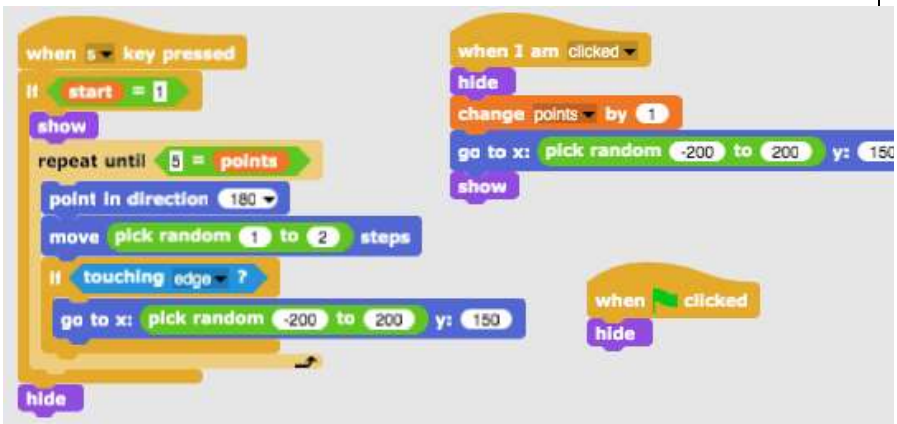
Students can now duplicate the apple sprite many times and change them costume (if they want). The code will be the same. The only change is with unhealthy food, where they will lose 1 point by clicking it.

[Final Code]

Girl



Apple



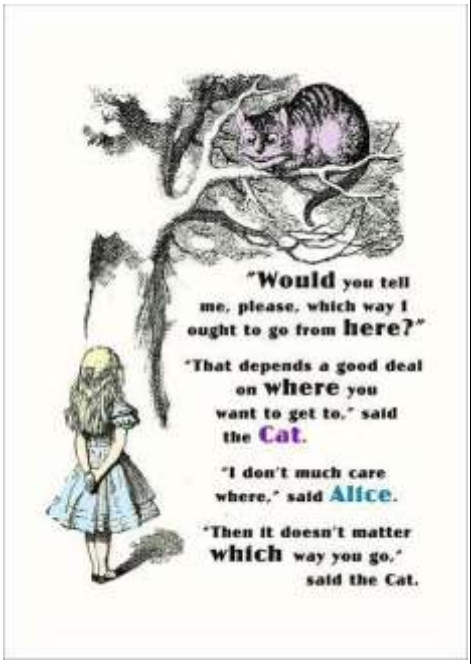
[Additional tasks]

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.  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:

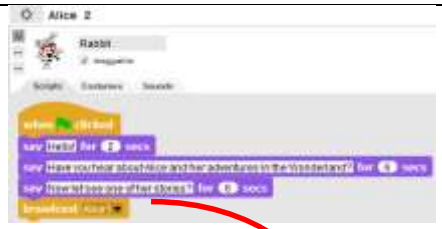
Sprite	Activity	Start from beginning	End time	Duration
Rabbit	Say: Hello! Have you heard about Alice and her adventures in Wonderland? Now let's see one of her stories.	0	14	14
Alice	Say: Would you tell me please, which way I ought to go from here?	9	21	12
Cat	Say: That depends a good deal on WHERE you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

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:

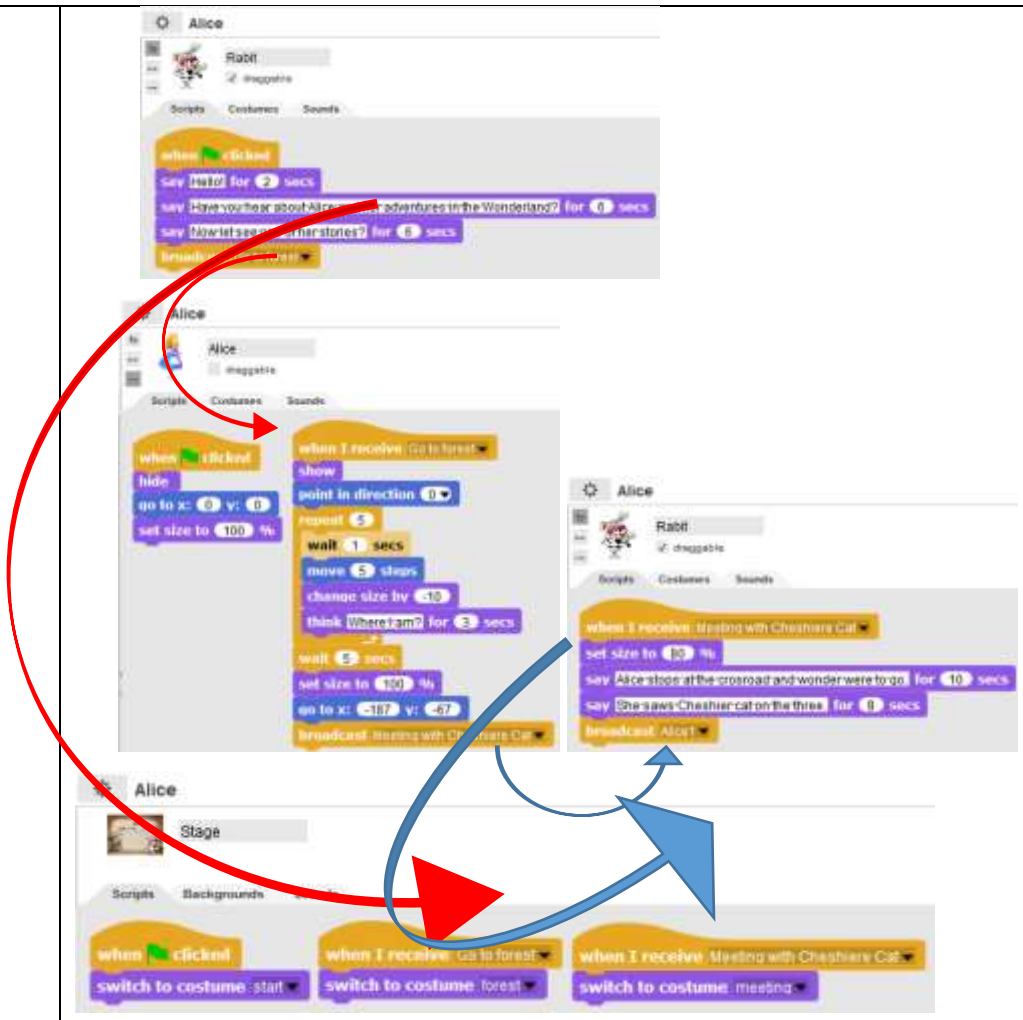
	1.  , 2. The message that should be received by the sprite is selected from the list. 3. The group discusses how to complete the story in the picture. How to name messages: e.g. The message from Cat to Alice to be Alice2 and from Alice to Cat - Cat1. 4. Students complete the story in pairs. 5. The teacher 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

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?"

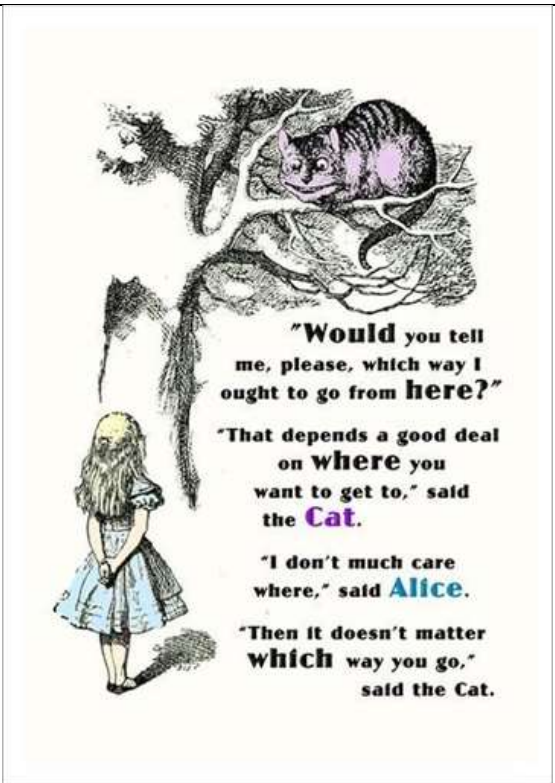
The scenes in the Alice_2 project are discussed. There are 3 scenes, one already used. Which scene to use to get started? What should be done to prevent the sprites of Alice and the Cat from being displayed at the start? How to change the stage décor? A *broadcast* can be used to change the stage set by the Rabbit after his introductory words. Alice appears when the scene is changed with the message *Go to forest.*



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.

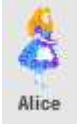
	3. The teacher comments that in order to tell a story, one must first invent a plot. An additional table may be used to describe the scenario of the story. (Appendix 1) At the teacher's discretion, the finished table may be given or partially completed and students, guided by the illustration, may complete 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	 • 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)

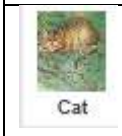
Appendix 1. Story plots/Scenarios

Name	Design	Actions	Notes

1. Start		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."	

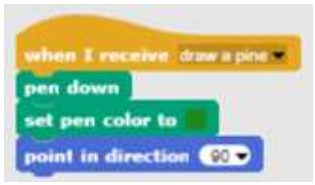
Learning Scenario 14 - Drawing

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	45 minutes

Activities	
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.  [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.



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

This movement should be repeated three times.

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.

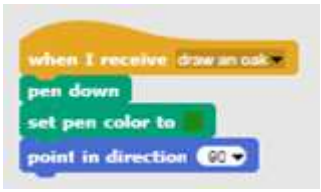
Turn 90 degrees to the right, and then move 10 steps.

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.

[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.



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

This movement should be repeated 120 times.

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.

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

This part is repeated three times.

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.

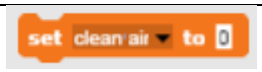
After the oak is drawn, the pen should be moved to the random position.

[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.

[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.



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.

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.

[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.

Then, add operator >.

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

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

[ADDITIONAL TASK]

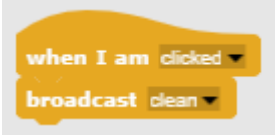
You can upgrade the game by adding animals that they appear when the air is not polluted anymore.

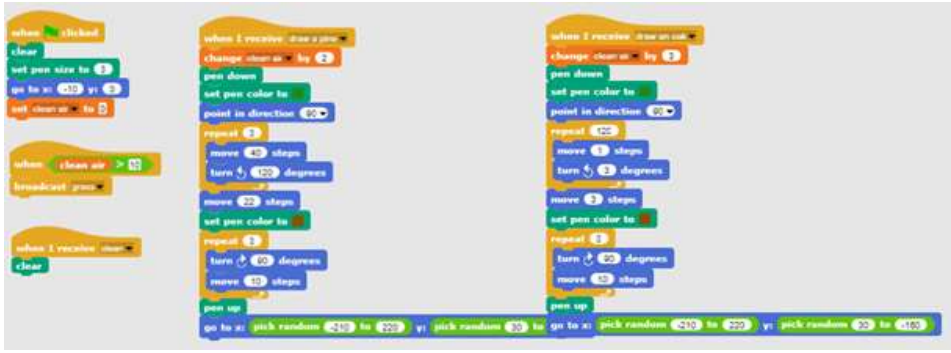
[Final code]

Pine

Oak

X



	Pencil  Stage 
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.

	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.

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

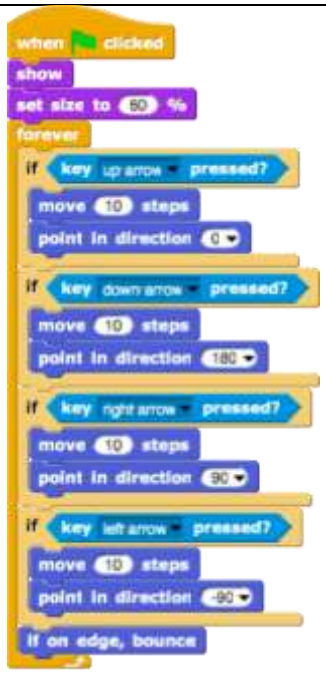
Implementation

[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.

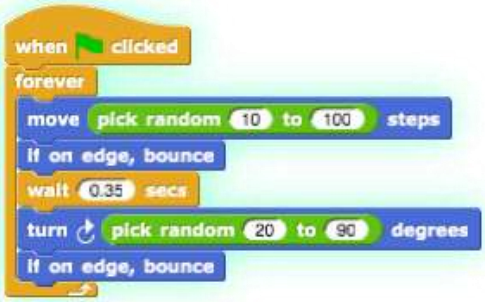
[Step 2]

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



[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.



[Step 4]

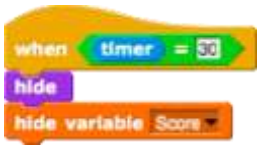
Next step is to program the mouse to hide when it touches the cat. The idea is that the mouse hides and appears in a random location when it touches the cat. In this case, the game does not end at the first catch of the mouse. Students can add their own rule here. In any case, they have to use the *pick random[x]to[y]* operator.

[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]*).

[Step 6]

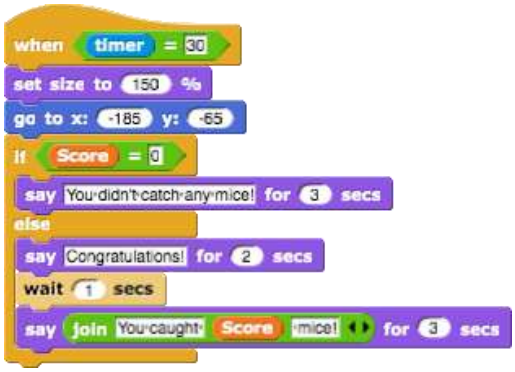
Students determine when the game ends. They do this with adding the timer. After some time (e.g. 30 seconds) the mouse and the cat disappear, the variable *Score* is hidden and the game is over.



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

[Step 7]

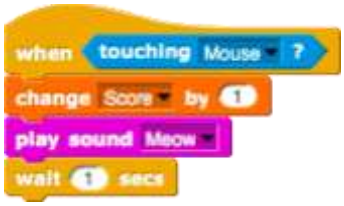
Students have to program the girl to summarize how successful player was. If the player doesn't catch any mice, the girl says: "You didn't catch any mice!". Else she says: "Congratulations! You caught *x* mice!"



[Additional tasks]

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

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



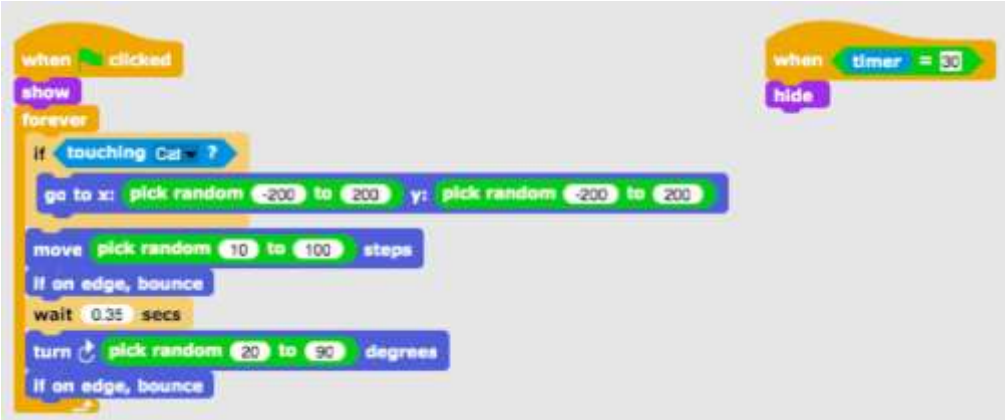
Reflection and evaluation

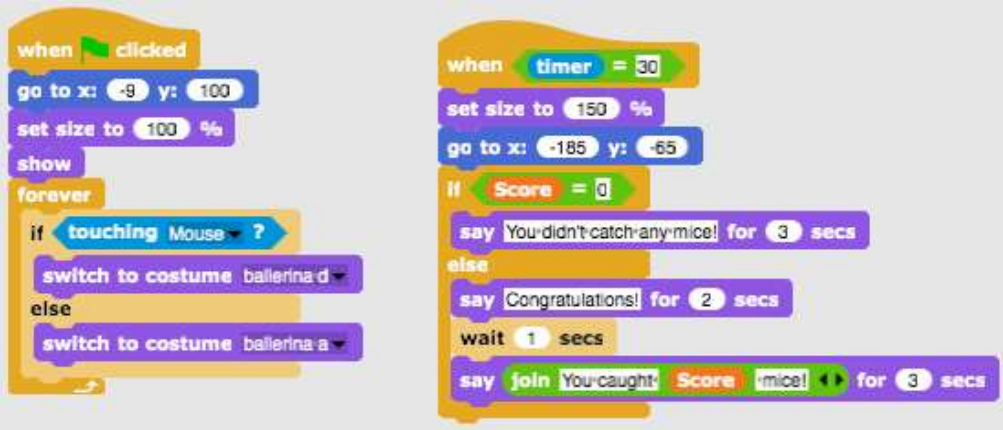
Students adjust the code:

- 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



	The cat  *The girl*  *The background* 
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.:

[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].

[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).

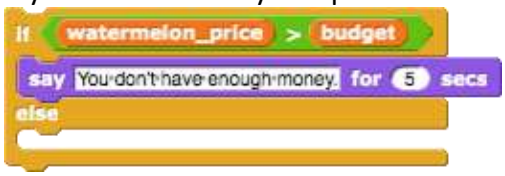
b) *When mouse-entered:* to tell the player how much does the product costs.

Students can use *Looks* – *thinking* block with use of joining text –

variable's value – text, e.g.:

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.



   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.

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

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

One possible solution is:

[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:

[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.

[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.

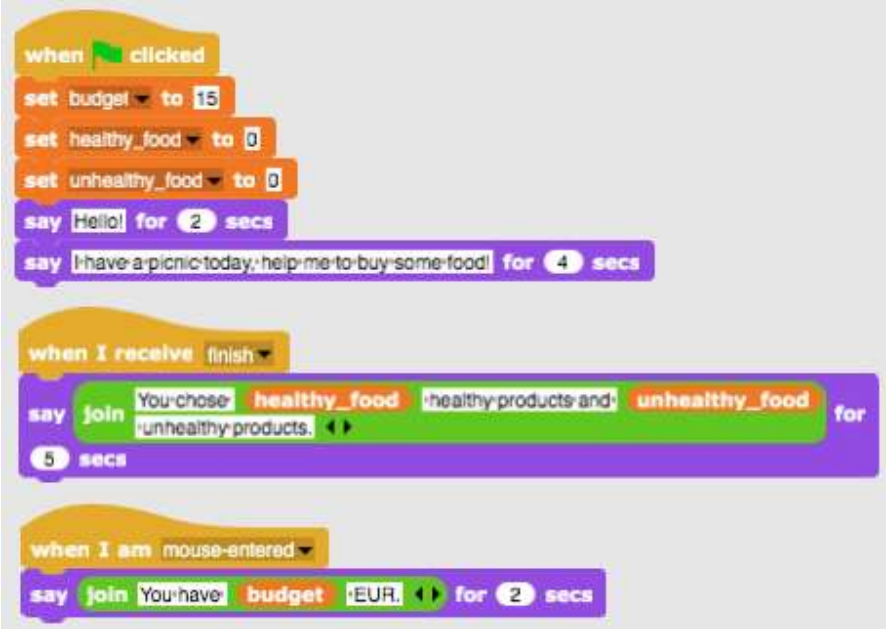
When the girl receives the message *finish*, she tells, e.g. "You chose X healthy products and Y unhealthy products".

[Step 7]

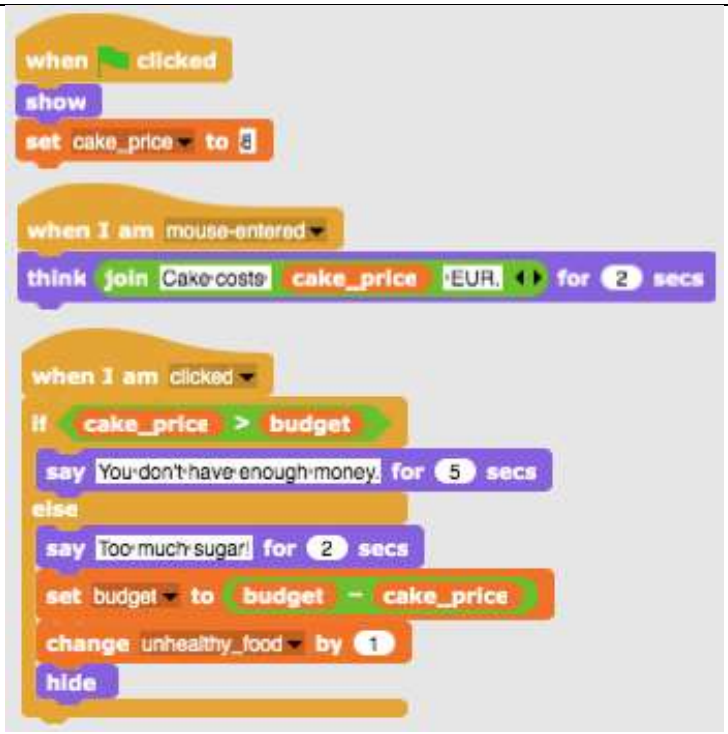
Anytime during the game, the player can check his budget by mouse-enter the girl. E.g., she can say / think something like:

[Final Code]

Girl



Food



Finish Button

[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

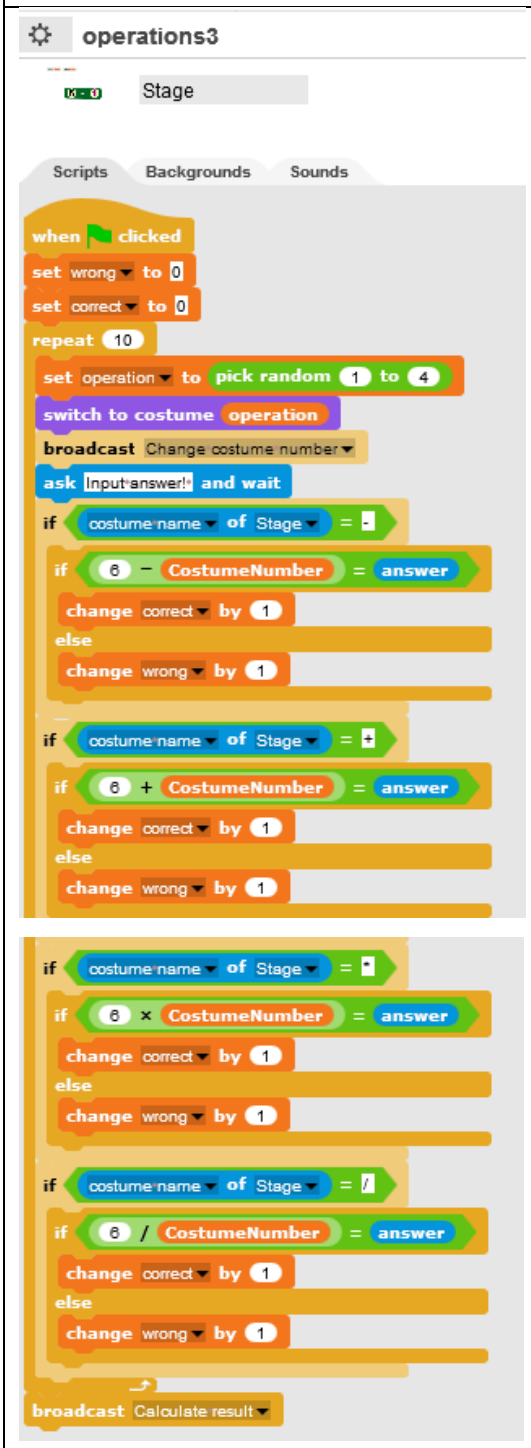
Learning Scenario Title	Operations
Previous programming experience	Using variables for counting points and to choose costume of the stage and of the sprite; Using random number to choose stage décor and costume for the sprite. Using repeat loop Using conditionals Using operations for comparison Using 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 .  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.

,


Code to the scene	Scenery/stage décor/
	

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

To select an operation the following commands are used:

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:

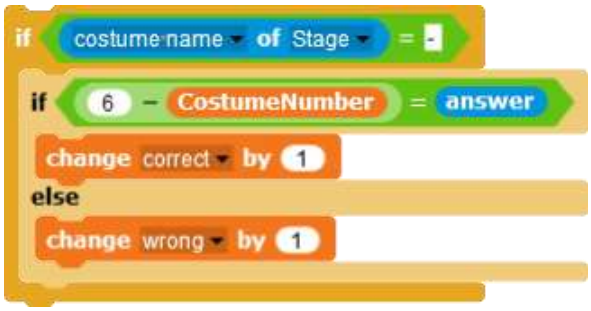
The entered response is compared with the result of the selected operations.

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.

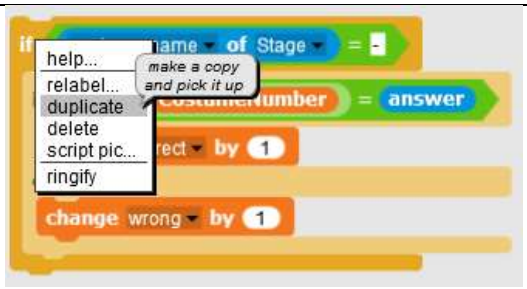


For the rest of the commands the script is similar, the difference is in the selected operation.

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.

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 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?

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.

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 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.

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

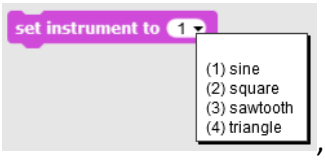
Learning Scenario Title	Play a piano
Previous programming experience	Using variables for counting points; Using event *When I am pressed*; Using repeat loop; Using conditionals; Using broadcast events to change scenery/stage decor/ and to manage sprite's activities;
Learning Outcomes	General learning 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.

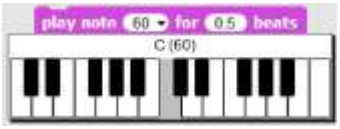
Students test the sound of various instruments.

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.



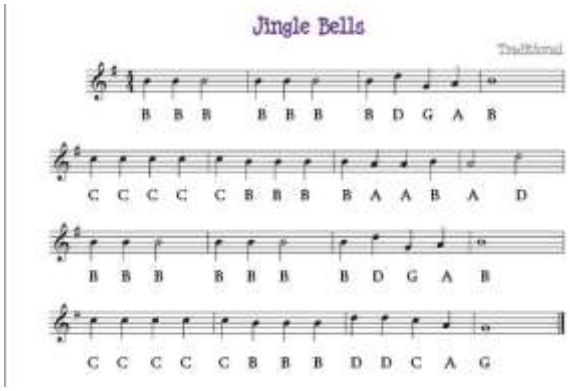
C	C#	D	Eb	E	F	F#	G	G#	A	Bb	B	C
60	61	62	63	64	65	66	67	68	69	70	71	72

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.) 

,

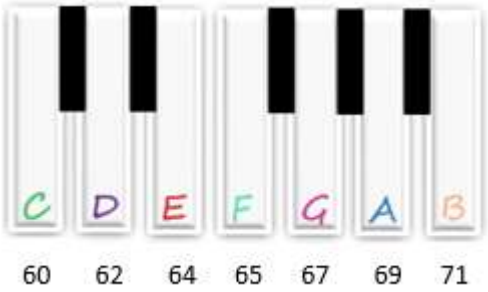
At the teacher's discretion, students can experiment with the commands and establish the dependencies themselves.

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

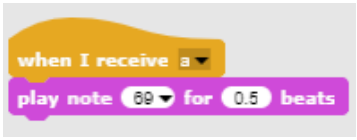
Teaching summary	(Motivation-Introduction, Implementation, Reflection and evaluation) At the beginning, a piano is shown on a stage. Next to the piano should be two buttons. Click on the first button should display the notes and words of the song, and click on the second button should play the melody 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:  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  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.

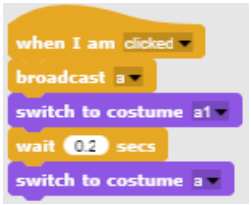
[Step 2]

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.

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.



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.

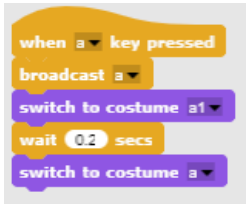


[Step 3]

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.



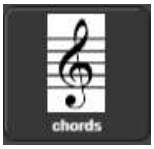
Notice if the c key on the keyboard is held down, the sound will be repeated as long as the key is pressed. This happens because the message "a" is repeatedly broadcasting. To stop broadcasting a message, on the end of code add a "wait until" block from Control pallet.

To finish broadcast a message, use the "not" operator and add to them a block "key a pressed".

Do the same for all remaining white keys.

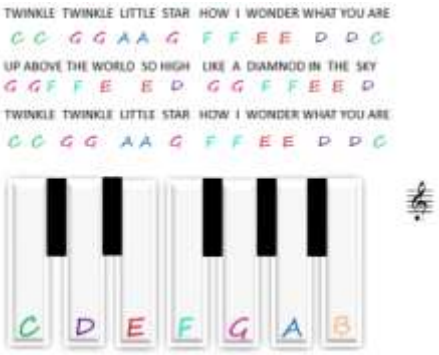
[Step 5]

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.

Import a new costume "chords" for the stage.



Add a code that enables the stage to change the costume to "chords" when it receives a message "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.

Ad a hat block “When I receive” to the stage to change the costume to “blank” after receiving the "blank" message.

[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

The violin key

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

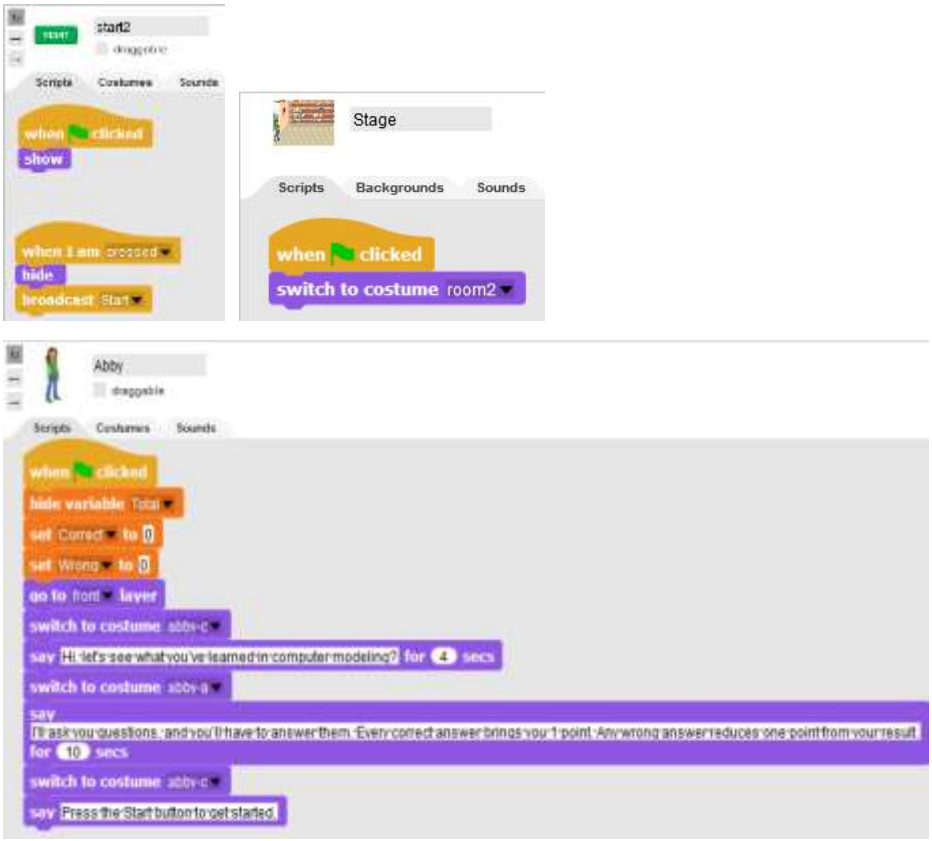
	X  The stage 
Tools and Resources for the Teacher	Snap! project “Play a Piano”: https://snap.berkeley.edu/project?user=ifrankovic&project=Play%20a%20Piano

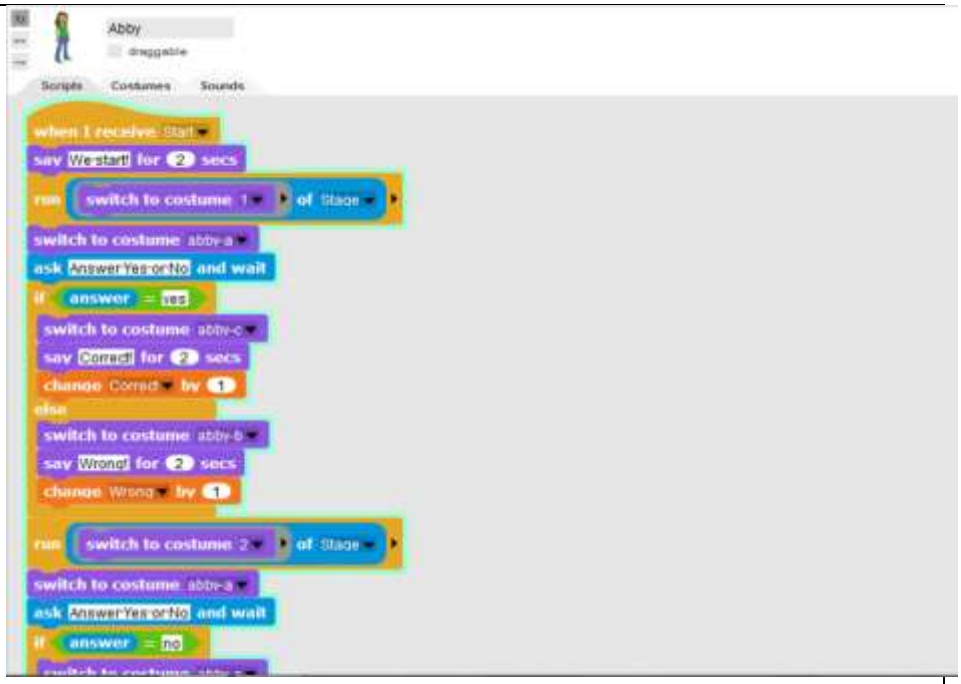
Resources/materials for the Students	Half-baked activity in Snap!:
	https://snap.berkeley.edu/project?user=ifrankovic&project=Play%20Piano (27.1.2020)
	Images:
	• Sprite images:
	  ◦ a.png, a1.png
	  ◦ b.png, b1.png
	  ◦ violin_key.png
	• Backgrounds: notes.png

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.**

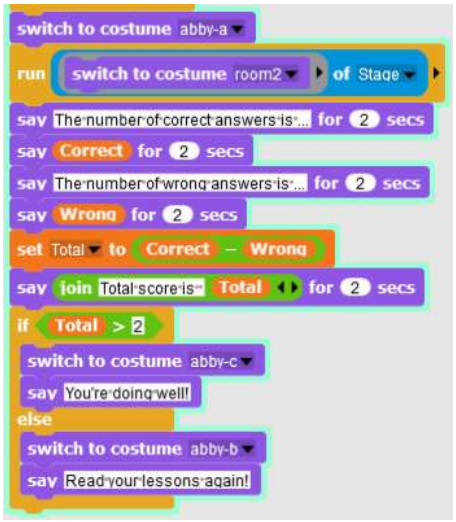
Duration of Activities	90 minutes
Learning and Teaching Strategy and Methods	Active learning (discussions, experiment with a game prepared in advance), game-design based learning, problem solving,
Teaching Forms	Individual work / Work in pairs/ Frontal work with whole class
Teaching summary	(Motivation-Introduction, 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. 



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.



The way of designing of the stage scenery for individual questions is discussed.

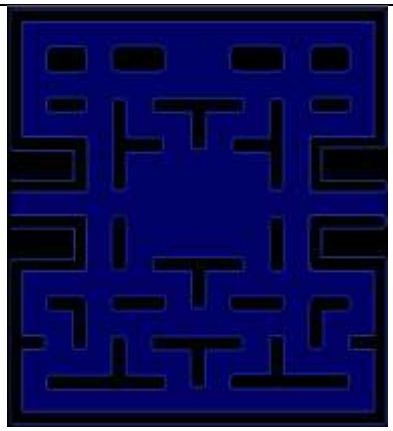
Because in Snap! it is not possible to write text in costumes and scenery, it is necessary to use an external graphic editor. Another option is to use MS Powerpoint to create the question and export the corresponding text box in graphical format.

	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.

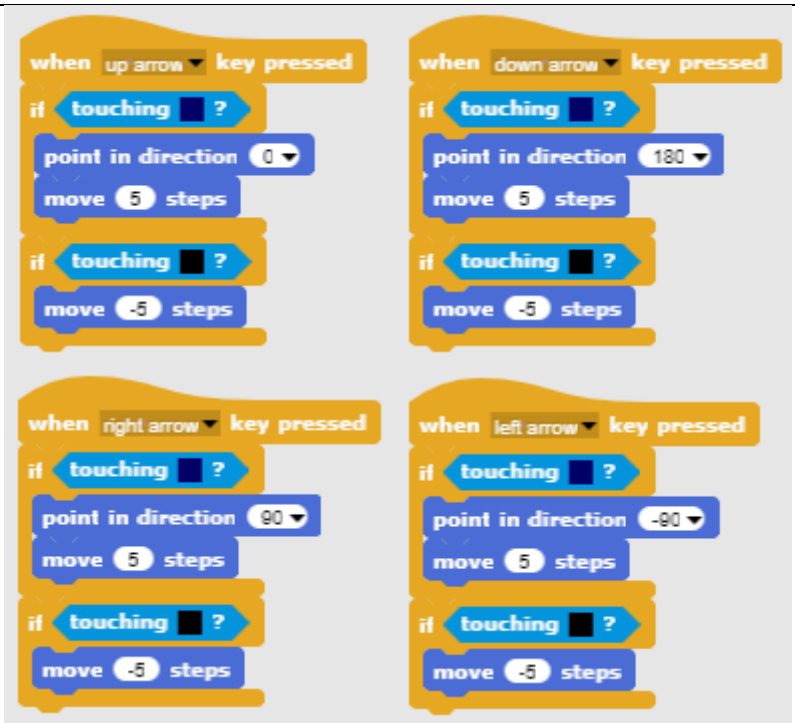


[Step 2]

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.



[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.

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.

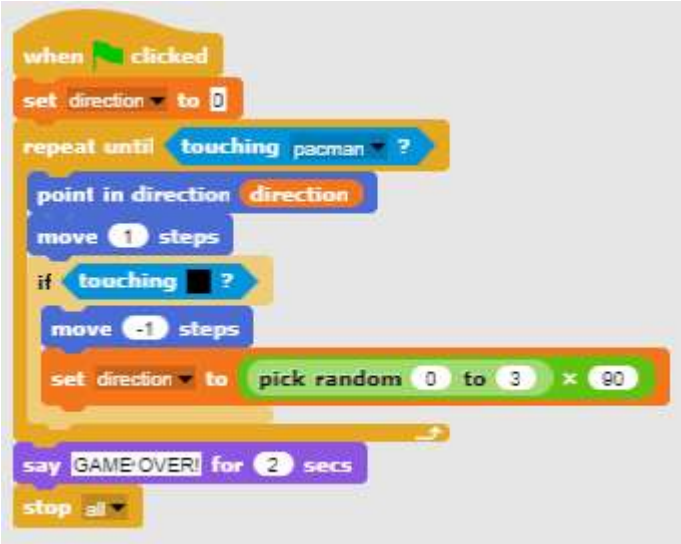
[Step 5]

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.



[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.

	
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)

REFERENCES

Lajovic, S. (2011). Scratch. *Nauči se programirati in postani računalniški maček*. Ljubljana: Pasadena.

Rugelj, J. (2019). Game design based learning of programming.

Vorderman, C. (2017). *Računalniško programiranje za otroke*. Ljubljana: MK.