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

O1/A6 GAME-BASED DESIGN-THINKING LEARNING FRAMEWORK FOR ENHANCING PROGRAMMING SKILLS IN SECONDARY EDUCATION

Elaborated by University of Thessaly (Greece) and EU-Track (Italy)

Document Data

Deliverable: O1/A6 – Game-based Design-Thinking Learning Framework for enhancing programming skills in secondary education

Intellectual Output No - Title: O1 – Methodological Learning Framework

Elaborated by: University of Thessaly (Greece) and EU-Track (Italy)

Disclaimer

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

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

TABLE OF CONTENTS

TABLE OF FIGURES

1. THE METHODOLOGY ADOPTED FOR THE LEARNING FRAMEWORK DEVELOPMENT

1.1 Waterfall vs Agile development methodology

Before selecting a model, the following questions should answered (Balaji & Murugaiyan, 2012):

- How stable are the requirements?
- Who are the end users for the system?
- What is the size of the project?
- Where the project teams are located?

As a methodology for the learning environment development the so called “Agile” and “Waterfall” methods were considered. While the Waterfall model adheres to a plan-driven approach, Agile pursues an adaptive approach. To justify the selection of an appropriate methodology, below the general ideas that lie behind both methods are shortly elaborated and comparatively summarized.

1.2 Waterfall

Basically, waterfall model, also known as a Linear Sequential Life Cycle Model (Unhelkar, 2016; Weisert, 2003), is characterized by the sequential dependability on the previous deliverable. A dependability which holds back system design when the analysis model is still to be signed off, and holds back coding if the design is still to be signed off. In other words, project development team only moves to the next phase of development or testing if the previous step is completed successfully. Just like natural waterfalls where once the water has flowed through the edge of the cliff and began flowing downwards, never turns back to reach the top of the hill (Jhajharia, kannan, & Verma, 2014).

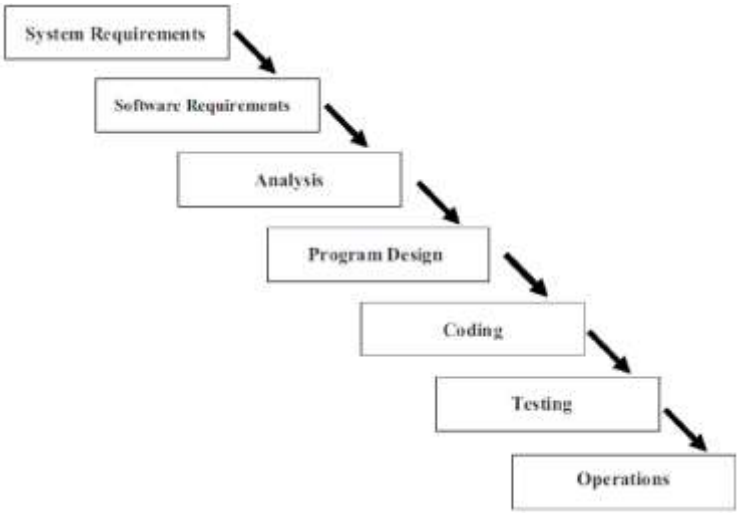

Figure 1: The Waterfall Model (Royce, 1987)

In a waterfall process, normally the documents are the output of each phase which serves as the input to the nest phase (Balaji & Murugaiyan, 2012). Team members cannot change the outputs that the project has certified. Nevertheless, requirements are subject to change. Thus, there is a need for a mechanism which ensures the modifications are done in a controlled manner without affecting the final product and its progress. Therefore, waterfall model is likely to be unsuitable if requirements are not well understood/defined or are likely to change in the course of the project. Authors (Petersen, Wohlin, & Baca, 2009) associate the Waterfall model with high costs and efforts. The numbers of documents to be approved in every phase, the difficulty to make changes, the problems that arise only in later phases confirm this belief.

1.3 When to use waterfall

The waterfall model works appropriately where the product development mainly consists of adding limited functionalities to an existing set functionality (Aitken & Ilango, 2013; Balaji & Murugaiyan, 2012; Jhajharia et al., 2014). It also scales well in the cases where changes in design can be introduced in a controlled way and the development can be continued without the requirement of customers and competitors. However waterfall model fails when there is so much new content or so many uncertainties to resolve that they become inevitable.

As such, authors (Jhajharia et al., 2014) provide an example of the *PC software market, where both hardware technologies and customer requirements change very rapidly, thus*

provoking a difficulty in capturing the specifications of a project at the very beginning. Therefore, the development cannot occur in a linear fashion. As consequence: "Any change in any part of the product for e.g., due to feedback from customers or evolution in particular hardware or software technologies, or even just to add a feature that a competitor has just introduced, may end up with pieces that no longer are compatible. " The testing fails. Due to the necessity of the pieces reworking much of the developed product may seem a waste. As an alternative, a classical contribution (Royce, 1987) discusses the iterative approaches (see Picture 2) between the preceding and succeeding steps but rarely with the more remote steps in the sequence. This type of development in iteration is what many developers undergo in projects in an unplanned manner (Jhajharia et al., 2014). But, again the implementation is risky and invites failure (Royce, 1987).

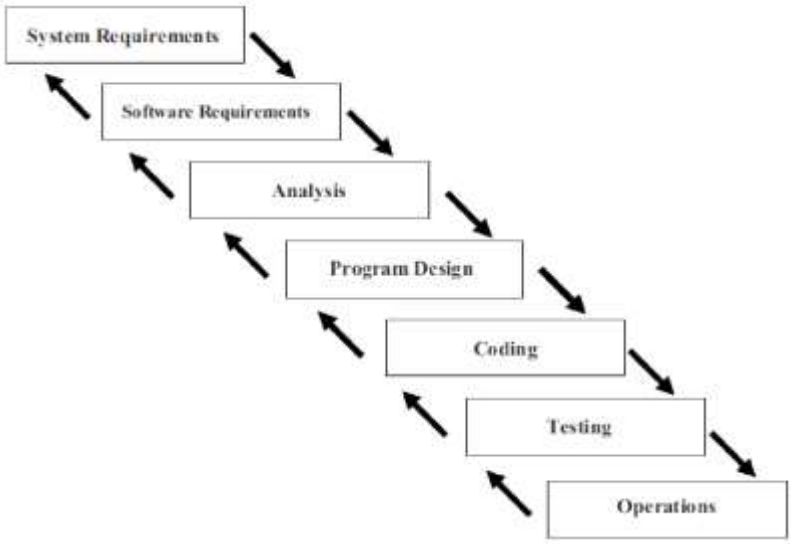

Figure 2: The Waterfall Model: iterative relationship between successive phases (Royce, 1987)

1.4 Agile

Agile development is by itself a huge umbrella term that includes other agile methodologies also, such as: Scrum, XP, Crystal, FDD and DSDM (Martin, 2002). Agile models were specifically designed keeping the adaptability of *changing requirements* in mind (Jhajharia et al., 2014). An agile method is a combination of iterative and incremental process models with an accent towards the flexibility and the timely delivery of the software. Considering every development process as specific and the method assumes that the existing methods

need to be personalized to suit best the project requirements. Agile provides methods to assess the development and risks and also the direction throughout the development lifecycle.

The product is developed in a series of rounds known as iterations. Each of the iteration involves various teams working simultaneously on various areas/phases (e.g. see the Waterfall phases). Each iteration guarantees an enhancement in features of the product of the previous iteration and the final iteration product involves all the features demanded by the customer (Ow, 2009).

1.5 When to use Agile

Basically, the use of Agile methodology is reasonable in the following cases. The requirements of the software are not well specified or when the requirements are expected to change during later phases of the development process. The condition is adequately satisfied due to the frequency of new increments produced and, as consequence, a very low cost of new changes implementation (Jhajharia et al., 2014).

Secondly, agile is appropriate when freedom of options and time is required by both developers and stakeholders. In this case it is possible to leave important decisions until more or better data are available and the project can be continued without the fear of it being a failure (Balaji & Murugaiyan, 2012; Jhajharia et al., 2014; Martin, 2002).

To summarize, agile methodology is a practice that helps continuous iteration and concurrency of development and testing in the software development process. Being focused client process, agile method guarantees the continuous involvement of the clients (students and teachers) during every stage. In addition, generally organized in this way, agile teams are extremely motivated and self-organized which at the end provides a better result from the development projects. The method assures that the quality of the development is maintained. And being based on the incremental progress the approach enables the client and team know exactly what is complete and what is not. The last consequently reduces risk in the development process.

To conclude, below the table outlining the basic differences between Agile and Waterfall Methods is provided (Ahmad, Soomro, & Naqvi, 2016).

Agile	Waterfall
Separates the project development lifecycle into sprints, following an incremental approach.	Software development process is divided into distinct phases following sequential design process.
Agile methodology is known for its flexibility, that allows changes to be made in the project development requirements even if the initial planning has been completed.	Being a structured software development methodology (SDM), most times it can be quite rigid, so there is no scope of changing the requirements once the project development starts.
Can be considered as a collection of many different projects.	Software development is completed as one single project.
Follows an iterative development approach, thus, planning, development, prototyping and other software development phases may appear more than once.	All the project development phases like designing, development, testing, etc. are completed once in the Waterfall model.
Test plan is reviewed after each sprint	The test plan is rarely discussed during the test phase.
A process in which the requirements are expected to change and evolve.	Requirements and changes are defined once for all.
Testing is performed concurrently with software development.	The “Testing” phase comes after the “Build” phase
Introduces a product mindset where the software product satisfies needs of its end customers and changes itself as per the customer’s demands.	Shows a project mindset and places its focus completely on accomplishing the project.
Works exceptionally well with Time & Materials or non-fixed funding. It may increase stress in fixed-price scenarios.	Reduces risk in the firm fixed price contracts by getting risk agreement at the beginning of the process.
Prefers small but dedicated teams with a high degree of coordination and synchronization.	Team coordination/synchronization is very limited.
Test team can take part in the requirements change without problems.	It is difficult for the test to initiate any change in requirements.
Description of project details can be altered anytime during the system development life cycle process (SDLC).	Detail description needs to implement waterfall software development approach.
Team members are interchangeable, as a result, they work faster. The projects are managed by the entire team, so there is no need for project managers.	The process is always straightforward so, project manager plays an essential role during every stage of SDLC.

Figure 3: Basic differences between Agile and Waterfall methods

1.6 The methodology adopted

On base of the analysis presented above the choice fell on agile methodology due to the following reasons:

1. The requirements and functions are subject to change
2. The product needs to be developed in a limited time.
3. A quick prototype with only certain functionalities is needed before the final product is available.
4. Agile requires the active participation of stakeholders.
5. The previewed work of the team is highly collaborative and self-organizing. The last ensures that the team members are actively planning and estimating their own work.

1.7 Quick lexicon

The abbreviation C4G and the full name Coding4Girls are used indifferently in this document.

A Lobby represents a course for a teacher. For example if a teacher wants to do a course about Elements of Programming, it means they will need to create a lobby Elements of Programming in the C4G software.

A Challenge corresponds to a chapter of the course; it is a subdivision of a lobby. If we continue our last example about a course concerning Elements of Programming, then elements of the course concerning conditional statements or loops or conditional loops would each be part of their own challenge.

An exercise or level corresponds to an elemental decomposition of a chapter of a course. If we continue our last example, asking the students to write a certain conditional statement would be an exercise in the challenge about conditional statements in the lobby titled Elements of Programming.

2 SPECIFICATIONS DRAFT

2.1 The selected platform for coding

As the platform for Coding Snap! (Fig 4) was chosen. The platform is an HTML/Javascript evolution of the proven and ubiquitous Scratch with some additional features. Snap!, similarly to Scratch, is web-based, but also has the possibility to be run offline through a browser.

Apart from the features of Scratch, Snap! adds first class lists, first class procedures, first class sprites, first class costumes, first class sounds and first class continuations, thus making it more suitable for older audiences and as an introduction to computer science than Scratch. Snap! is available in over 40 languages, including Bulgarian, Croatian, English, Greek, Italian, Portuguese, Slovenian and Turkish.

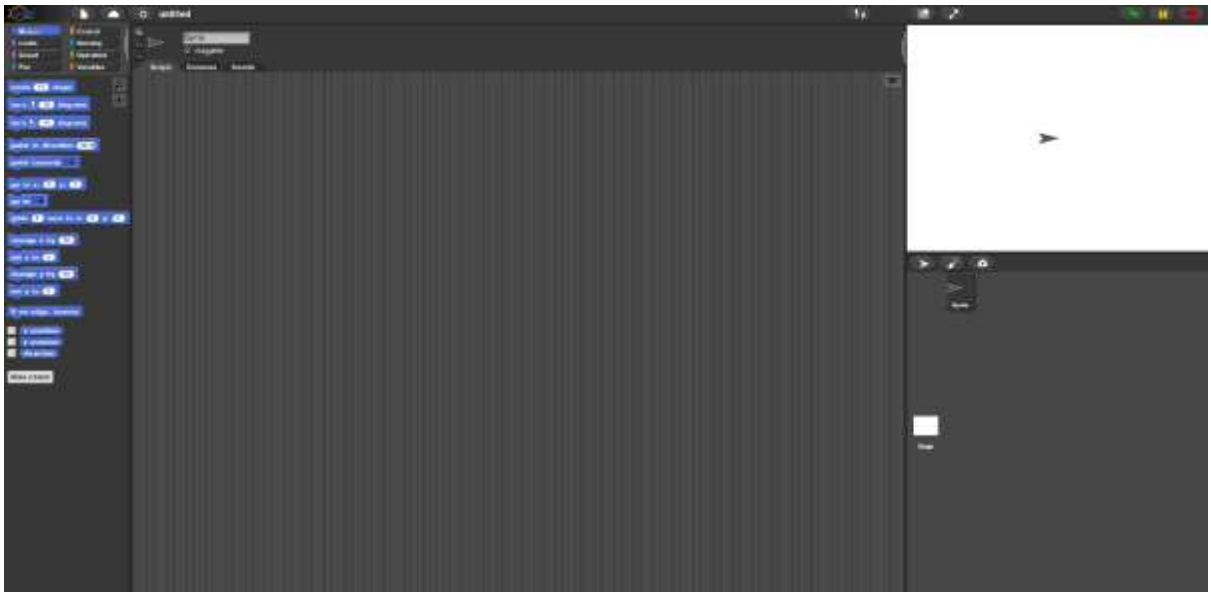

Figure 4: Snap! User Interface

A study was conducted by Weintrop and Wilensky (2015) with high school students using Snap! and Java. The students found the blocks-based approach to be easier than Java – thus, it is in accordance with the view that blocks-based programming (like Scratch and Snap!) is more accessible to novice programmers. According to the study findings, this was due to the fact that blocks are easier to read, due to the visual nature of the blocks that provide cues on how they can be used, they are easier to compose, and serve as memory aids. (Weintrop & Wilensky, 2015).

2.2 First ideas and experimentations

Following the Agile methodology a first version of the platform was very quickly designed and implemented in order to test the implementation of the specifications.

A platform hosting the course was organized and designed, in many instances drawing on the concept of a Learning Management System (LMS), which will follow and motivate the students' progress in programming and also will be furnished by the instruments to manage the courses. There are six major indicators, constituting the concept of LMS itself that caused the selection of this type of platform organization: interoperability, accessibility, reusability, durability, maintenance ability and adaptability (Long, 2004).

At the current stage of development two options for course web-platform organization were considered: basic and advanced (see Fig 4). The basic option was to have a full LMS style implementation for the project, with Snap! exercises illustrating the points of the course, the more advanced option was making use of 3D games inside the LMS via a WebGL implementation of games developed in Unity.

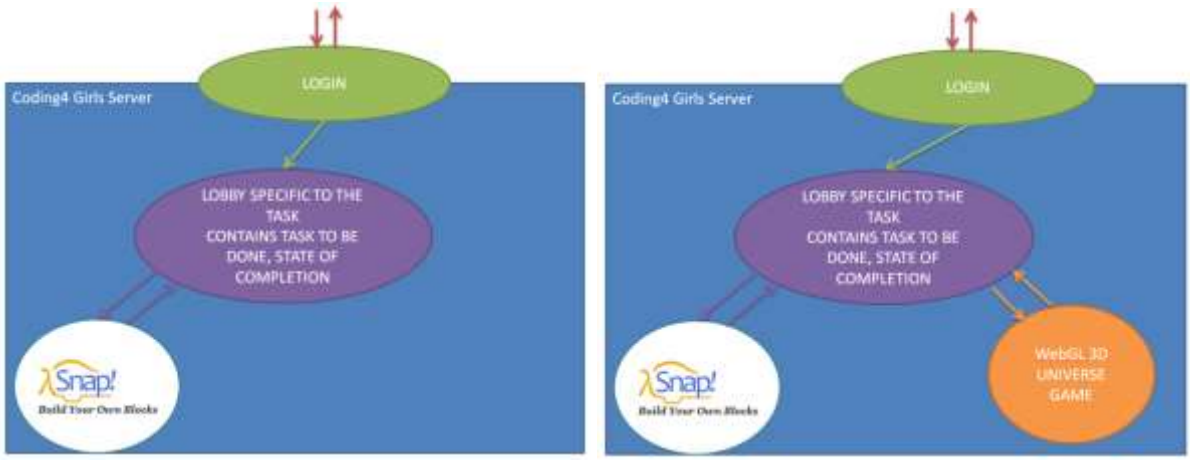

Figure 5: Basic (left) and advanced (right) organizations of the course web-platform

The general underlying principles of the educational platform were the following:

1. The platform is supposed to be fully Web-based for teachers and students
2. Constant internet connection is required;
3. The platform contains both students/teachers accounts and holds the work to be done;

4. Work (coding exercises) are opened by the students via a code given by the teacher (basic modality);
5. Exercises/challenges are chained together thematically; for as long as pedagogically needed;
6. Unity WebGL game, integrated to the platform unlocks exercises/hints according to player progression can be divided in chapters according to the work plan chapters;
7. Gatekeeping the coding with the game;
8. The narrative of the game corresponds to the text given by the teacher;

The first testing of the solution showed some flaw with the design. The WebGL component, although very attractive as it immersed students in a 3D environment was also very hungry of computer resource which led to issues performance-wise especially given the variety of available browser for end users. Furthermore, the volume of data to be downloaded at each use was excessive. Quickly an improved version of this design was drafted.

2.3 Second implementation

The main change of this implementation was to completely separate the students' and teachers' implementation. Teachers would only deal with the online platform; students would only play with a desktop Unity game.

From the student's point of view (Fig 5) the interaction principle with the platform is contained within the following basic steps:

1. Log in to the Coding4Girls game;
2. Get acquainted with the tasks in a LMS style environment;
3. Basic option: coding exercises are opened by the students via a code given by the teacher.
4. Advanced option: play the game to unlock instructions/hints;
5. Code using Snap!

As one can see (Fig 5), due to the organizational principles chosen the permanent internet connection is required to provide to continuous communication with Coding4 Girls Server.

Whereas teacher's interaction, as one can see from Fig 5, consists in Log in to the Coding4 Girls Server and work with either courses access/creation or student management (e.g. assignments, analytics).

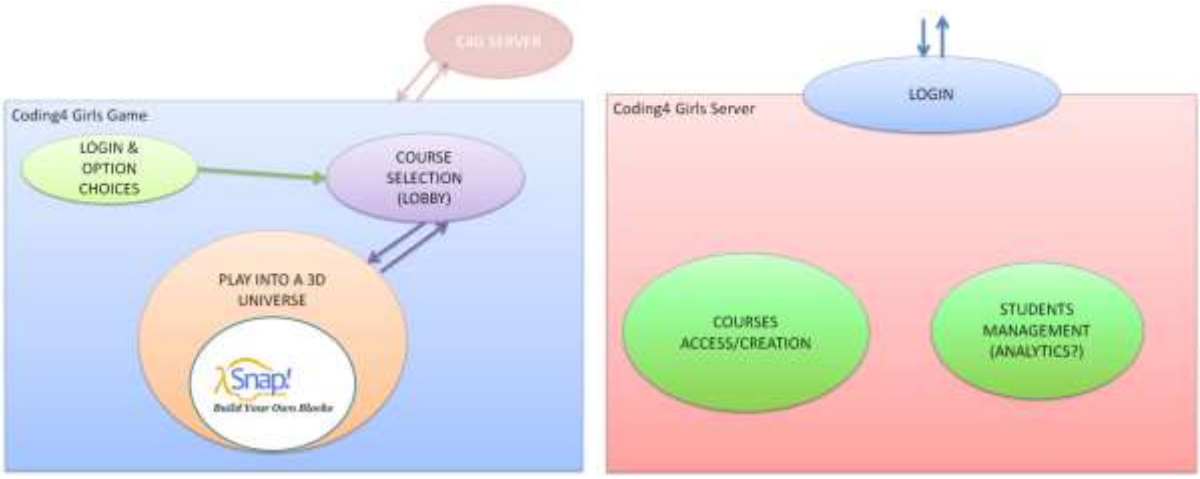

Figure 6: Students' view (left), Teachers' view (right)

The Coding4Girls game would start with the students arriving in a room where they would login to a server. From there, their progress and all personal information would be transferred t the game. They would then proceed to a room full of portals, each of them corresponding to a game lobby, as shown in Fig 7 and Fig 8.

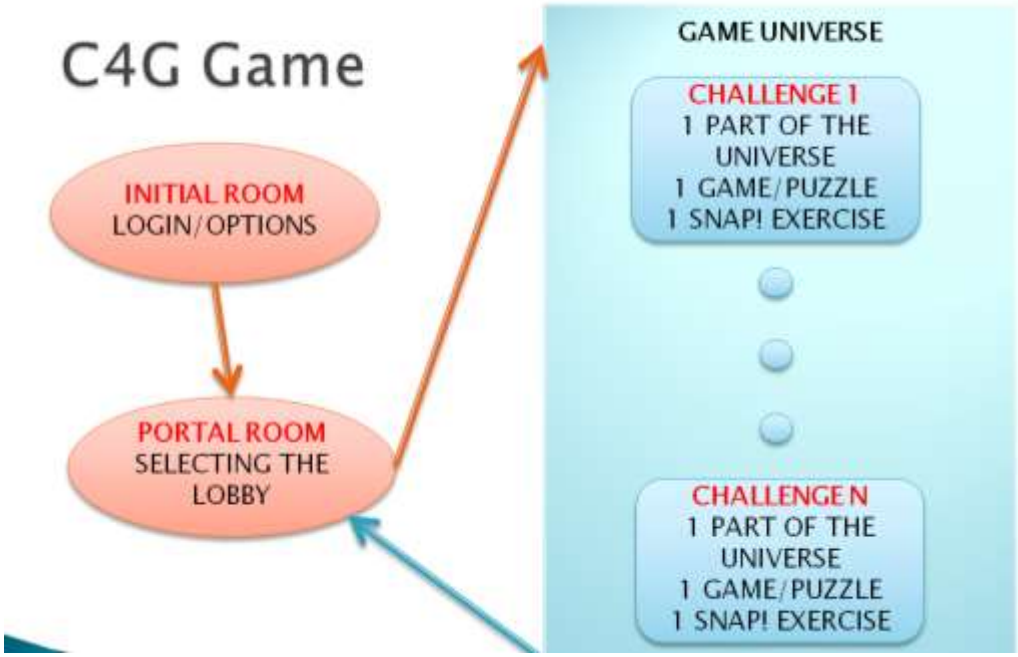

Figure 7: The C4G game gameplay

Once the students have chosen their lobby, they can choose the challenge inside the lobby corresponding to their progression in the course and start the game. They will be propelled into a 3D universe in which they will need to solve puzzles or play some games to unlock the access to an instance of Snap! where they will do the required coding.

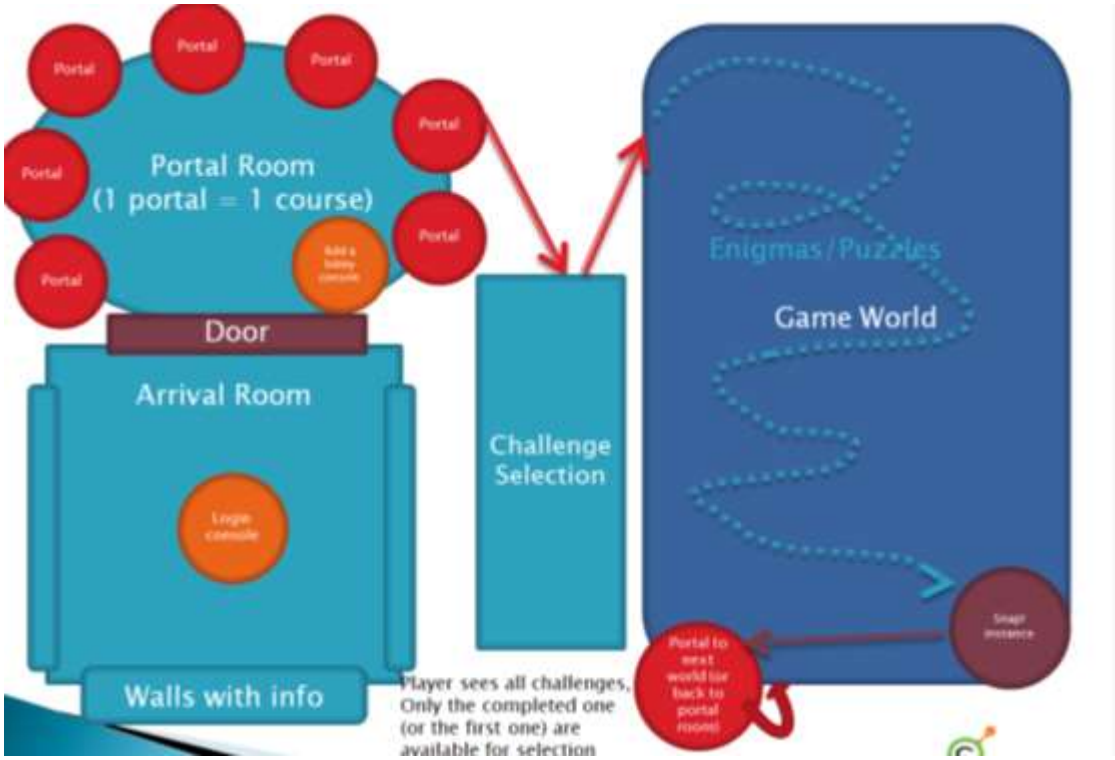

Figure 8: C4G game detailed gameplay

The general underlying principles of the educational platform were the following:

1. The platform is supposed to be fully Web-based for teachers, but for the students the solution will be fully desktop;
2. Constant internet connection is required;
3. The platform contains both students/teachers accounts and holds the work to be done;
4. Work (coding exercises) are opened by the students via a code given by the teacher (basic modality);
5. Exercises/challenges are chained together thematically; for as long as pedagogically needed;
6. Unity desktop game, integrated to the platform unlocks exercises/hints according to player progression can be divided in chapters according to the work plan chapters;

7. Gatekeeping the coding with the game;
8. The narrative of the game corresponds to the text given by the teacher;
9. All in one integrated game for the students, no other interface than the game;
10. The assignments of existing games/puzzles/universes take place automatically according to students' progress.
11. Download once use forever model will be adopted.

2.4 Final Implementation of the Coding4Girls platform

When the second implementation was presented to the partners, a lot of feedback was gathered and a new and improved design (Fig 9) was created. This new design added to the previous one two main key points:

- Framing the process into a design thinking methodology
- Strongly linking the 3D games and puzzles to the content of the course

In order to integrate the C4G platform into a design thinking pattern, it was decided that at the beginning of each lobby, all the students of a class would be given a certain real life problem, inside the game. All the students would then exchange ideas about the problem, what they understand of it and how they think they could solve the issue via a shared space in which they can place multimedia message via virtual post-its.

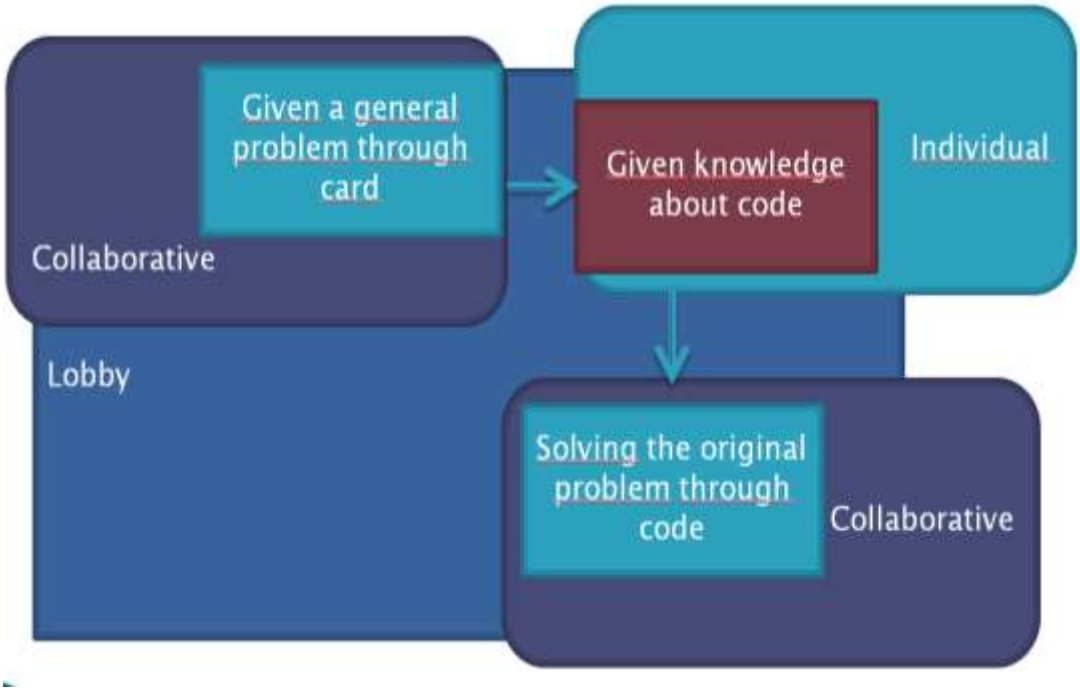

Figure 9: Global design

Once this brainstorming phase is achieved, each of the students would then start the loop of challenges and Snap! Exercises described before, where all of them are geared toward illustrating concepts necessary to solve the overarching real life problem given to everyone. The loop (see Fig 10) can be decomposed as such:

- The student gets selects where they are in the lobby progression wise and jump to the corresponding challenge
- The students are put into a 3D world where they will play a certain game, thematically linked to current elements of courses. This step is optional.
- After playing the game, the students are shown some course principle expanding on what the game showed them and given some instructions through an HTML page containing multimedia content.
- The students are given a Snap! Interface where they need to solve the given exercise or elaborate on already existing snippets of code
- The students go to the next challenge/chapter of the course
- Once the entire course has been done, students go back to the original problem at hand and will code a solution for it
- The students will be able to see all the other solutions elaborated by their class comrades and compare it to their own.

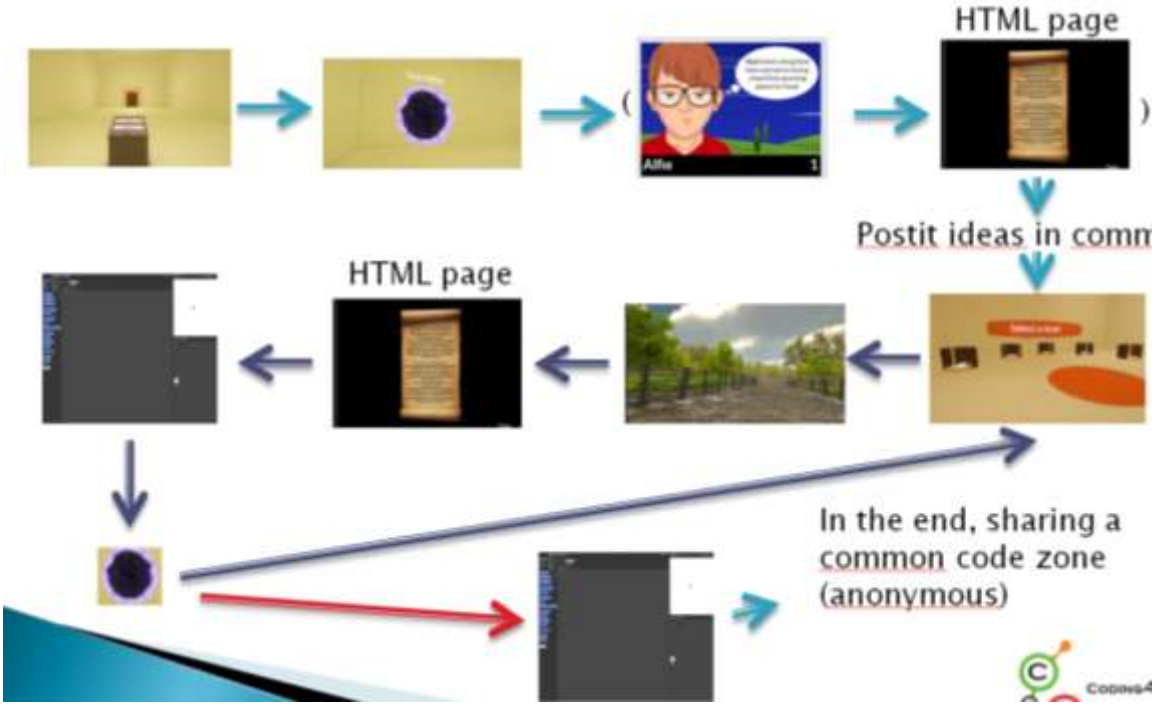

Figure 10: Updated gameplay loop

3 A FOCUS ON GIRLS

3.1 Girls Gaming Preferences

Serious games are games that aren't meant only for entertainment purposes, but also for education purposes: to engage, educate, and motivate students in the learning process. However, the degree in which one wants to play a game varies across gender and the gaming industry has made few attempts at studying female-preferred games (Alserri, Zin, & Wook, 2018).

According to the "Gamer Consumer Insights", 46% of gamers across thirteen countries are women, showcasing a growing trend in which women have been becoming more interested in gaming. Likewise, interest in gaming and its connection with gender issues has increased rapidly in recent years. While across all platforms, men tend to prefer strategy, sports, action/adventure, and shooter games, women tend to enjoy more action/adventure, puzzle, strategy and arcade games (Osborn, 2017).

Women tend to not be fond of direct competition (conflict or unjust violence settings) and prefer problem resolution (Vermeulen, Looy, Courtois, & Grove, 2011). There is also a preference for puzzle games, social games (with a rewarding system), collaborative and exploration games, and virtual life and party games. In regard to adventure games, there is a preference for observing first and playing after the act of observation (Alserri et al., 2018).

It's also important to note the difference in preferences when it comes to platforms: while men prefer to play on the PC or the console (48% and 37%), only 35% and 23% of the women inquired prefer to do so, with this last group playing more mobile games (48%) (Osborn, 2017).

3.2 Approaches to Teaching Programming to Girls

Adopting a constructionist approach to games is beneficial for education, because there is not only a focus on providing the game to students, but also providing this group with means and knowledge to develop their own games (Kafai, 2006).

A study conducted by Carmichael (2008) showcases the beneficial outcomes of combining computer science concepts with video games specifically to a young female target-group. The goal of the one-week course towards twelve girls in grades eight and nine was to teach basic concepts on Computer Science and also to disperse negative stereotypes associated with it. One crucial point to bear in mind when choosing the coding creation software is that educators should conform to a series of requirements: the academic year of the students, familiarity of the instructor(s) and capability of the student in creating a coding project according to the amount of time they will be spending learning the know-how to do it. Some teaching methods used ranged from group activities, brainstorming, reading of relevant articles to interactive demonstrations. An important aspect that Carmichael (2008) emphasizes is that during lab time, more than one instructor would've been helpful to, in turn, help all the girls with their doubts about the gaming development phase.

Lastly, Alserri et al. (2018) have developed a conceptual model for gender-based engagement in Serious Games, consisting of five elements:

1) Learning Elements: these are the elements that distinguish entertainment games from educational games (Alserri et al., 2018);
2) Female Preferences for digital games: these are the preferences specific to girls, that have to be incorporated into the design in order to motivate and engage them. According to the literature review conducted by the author, these preferences consist in exploration, character customization, storyline, social interaction, collaboration, challenges, fun, control and feedback (Alserri et al., 2018);
3) Flow state theory: some of these elements are also female preference elements. These elements should also be incorporated in order to obtain engagement and motivation: challenges, fun, control, feedback, concentration, clear goals, skill and immersion (Alserri et al., 2018);

4) Female game types and genres: according to the authors, these would be fantasy and role-playing games.
5) Social gender factors: parental, peers and teacher influence.

3.3 Platform adaptation for girls

In accordance with Girls Gaming preferences elaborated earlier, in particular, such as action/adventure, puzzle, strategy and arcade games (Osborn, 2017) the games genres used will be adapted correspondingly. Following the adaptation strategy, games will be focused more on the problem solving than on the enforced/violent resolution.

The study paths are supposed to be entirely translated into game format, thus the content is believed to become more appealing. According with the recent findings (Hosein, 2019), 13-14 year old girls classed as 'heavy gamers' - those playing over nine hours a week - were three times more likely to pursue a PSTEM degree compared to girls who were non-gamers, the coding part is placed inside a video game.

No avatars and built-in multi-players components are foreseen. Nevertheless, strategy allows the work of the players in teams for Snap! part.

4 TEACHERS' LEVELS OF INVOLVEMENT

The platform will offer to the teachers a high level of involvement into the customization of the students' experience.

In order for the teachers to be able to illustrate their courses concepts in the game, it was decided to create a list of keywords, called meta-tags which would link existing coding concepts with the available library of 3D games.

	A	B
1	**List of Tags**	**Minigame associated**
2		
3	Loops	Match3
4	Conditionals	Find your path
5	Variables	N/A
6	Variables - Data types	Inventory/Marble
7	Variables - Data structures	Inventory/Marble
8	Statements	N/A
9	Statements - Sequence of statements	Stepping Game
10	Statements - Sounds	Sound Game
11	Statements - Movement	Stepping Game
12	Statements - Looks (appearance of a sprite in snap)	Snake game
13	Statements - Drawing	Witness Game
14	Parallelism	N/A
15	Parallelism - Simultaneous sounds/movements/characters/interactions	Donald Game
16	Operators	N/A
17	Operators - 4 basic operations	Donald Game
18	Operators - Modulo, roundings	Donald Game
19	Operators - Advanced, sin, cos, sqrt, power	Donald Game
20	Operators - AND/OR/NOT, booleans, ==	Donald Game
21	Operators - String operators	Find your path
22	Operators - Random	Slot Machine
23	Events	Variation of the Stepping Game

Figure 11: Meta-tags and mini-games

Those meta-tags will allow the teachers to create a lobby/lesson seamlessly as they will only select the relevant meta-tags linked to their course and the C4G platform will automatically transform those into a certain game which will then be offered to the students.

For example if a teacher creates a lobby which will present a course about the concept of data type and data structure, they will only have to select the data type meta-tag in the authoring tool. The students will then automatically be given to play a game related to this concept, one where they need to fill an inventory by combining colored marbles.

In the scope of this project a limited amount of meta-tags and 3D games will be produced (see Fig 11 for the full ist), corresponding to all the basic notion of coding with Snap!. But

the list of met-tags and corresponding mini-games could be extended infinitely without any issue.

From the teachers point of view, the lesson (or lobby) will be seen (see Fig 12) as a list of chapters (or challenges) linked the one to the other, each of them illustrated by a meta-tag and each containing detailed instructions and illustrations for the students given through an HTML page.

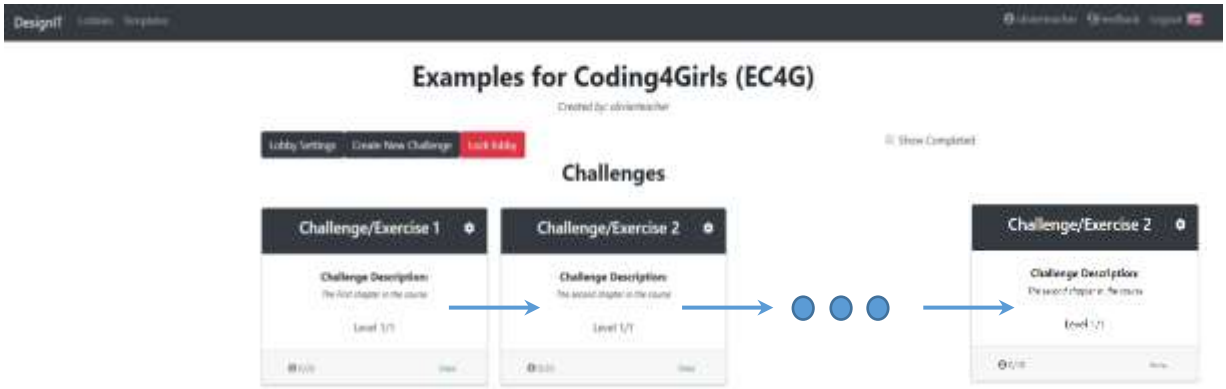

Figure 12: A lobby seen from the teachers' point of view

REFERENCES

Ahmad, G., Soomro, T., & Naqvi, S. M. (2016). *AN OVERVIEW: MERITS OF AGILE PROJECT MANAGEMENT OVER TRADITIONAL PROJECT MANAGEMENT IN SOFTWARE DEVELOPMENT* (Vol. 10).

Aitken, A., & Ilango, V. (2013). *A comparative analysis of traditional software engineering and agile software development.* Paper presented at the 2013 46th Hawaii International Conference on System Sciences.

Alserri, S. A., Zin, N. A. M., & Wook, T. S. M. T. (2018). Gender-based Engagement Model for Serious Games. *International Journal on Advanced Science, Engineering and Information Technology, 8*(4), 1350-1357. doi: 10.18517/ijaseit.8.4.6490

Balaji, S., & Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management, 2*(1), 26-30.

Carmichael, G. (2008). Girls, computer science, and games. *ACM SIGCSE Bulletin, 40*(4), 107-110. doi: 10.1145/1473195.1473233

Hosein, A. (2019). Girls' video gaming behaviour and undergraduate degree selection: A secondary data analysis approach. *Computers in Human Behavior, 91*, 226-235. doi: https://doi.org/10.1016/j.chb.2018.10.001

Jhajharia, S., kannan, v., & Verma, S. (2014). *Agile vs waterfall: A Comparative Analysis* (Vol. 3).

Kafai, Y. B. (2006). Playing and Making Games for Learning: Instructionist and Constructionist Perspectives for Game Studies. *Games and Culture, 1*(1), 36-40. doi: 10.1177/1555412005281767

Long, P. D. (2004). Encyclopedia of Distributed Learning. Thousand Oaks

Thousand Oaks, California: SAGE Publications, Inc.

Martin, R. C. (2002). *Agile software development: principles, patterns, and practices*: Prentice Hall.

Osborn, G. (2017). Male and Female Gamers: How Their Similarities and Differences Shape the Games Market., from https://newzoo.com/insights/articles/male-and-female-gamers-how-their-similarities-and-differences-shape-the-games-market/

Ow, S. (2009). *Review of Agile Methodologies in Software Development* (Vol. 1).

Peteranetz, M. S., Flanigan, A. E., Shell, D. F., & Soh, L. K. (2017). Computational Creativity Exercises: An Avenue for Promoting Learning in Computer Science. *Ieee Transactions on Education, 60*(4), 305-313. doi: 10.1109/te.2017.2705152

Petersen, K., Wohlin, C., & Baca, D. (2009). *The waterfall model in large-scale development.* Paper presented at the International Conference on Product-Focused Software Process Improvement.

Royce, W. W. (1987). *Managing the development of large software systems: concepts and techniques.* Paper presented at the Proceedings of the 9th international conference on Software Engineering.

Unhelkar, B. (2016). *The art of agile practice: A composite approach for projects and organizations*: Auerbach Publications.

Vermeulen, L., Looy, J. V., Courtois, C., & Grove, F. D. (2011). *Girls will be girls : a study into differences in game design preferences across gender and player types*. Paper presented at the Under the mask: perspectives on the gamer, Luton, UK. http://hdl.handle.net/1854/LU-1886961

Weintrop, D., & Wilensky, U. (2015). *To block or not to block, that is the question: students' perceptions of blocks-based programming*. Paper presented at the Proceedings of the 14th International Conference on Interaction Design and Children, Boston, Massachusetts.

Weisert, C. (2003). There's no such thing as the Waterfall Approach!(and there never was)'. *Information Disciplines, Inc*.