Miloš Kojić, BSc

Procedural Content Generation in a Multidisciplinary Educational Mobile Game

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.Ing. Dr.techn. Christian Gütl

Dipl.Ing. Dr.techn. Johanna Pirker

Institute for Information Systems and Data Science

Head: Univ.-Prof. Dipl-Ing. Dr. Stefanie Lindstaedt

Graz, December 2017

Miloš Kojić, BSc

Prozedurale Generierung von Inhalten in einem Mobilen Multidisziplinären Pädagogischen Spiels

Masterarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium: Informatik

eingereicht an der

Technischen Universität Graz

Betreuer

Assoc.Prof. Dipl.Ing. Dr.techn. Christian Gütl
Dipl.Ing. Dr.techn. Johanna Pirker

Institut für Interaktive Systeme und Datenwissenschaft
Head: Univ.-Prof. Dipl-Ing. Dr. Stefanie Lindstaedt

Graz, December 2017

Procedural Content Generation in a Multidisciplinary Educational Mobile Game

Miloš Kojić

Advisers

Christian Gütl
Johanna Pirker
Graz University of Technology

Markos Mentzelopoulos
Daphne Economou
University of Westminster

Graz, December 2017

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

Datum	Unterschrift

Acknowledgments

I wish to express my deep gratitude to my advisors Assoc. Prof. Dipl.-Ing. Dr.techn. Christian Gütl and Dr. Johanna Pirker for providing me the opportunity to work on my thesis in London, and for their constant support throughout the entire project.

Many thanks to my twin brother for his support and advices which helped me to successfully complete the project.

For guidance, continuous input and inspiring ideas I would like to thank Markos Mentzelopoulos and Daphne Economou from the University of Westminster in London. My stay at the University of Westminster was a very valuable experience.

Finally, I want to thank my family and friends for their constant encouragement, patience, and love throughout my studies.

Abstract

Procedural content generation (PCG) in video games is a technique used to generate maps, assets, and sounds. This technique can provide players with a non-repetitive experience by creating endless maps with a large number of content variations. There is a number of different PCG algorithms and those which are of interest for this thesis are the methods used for generating dungeon-like map structures, terrains, trees, plants, and sounds. Cellular automata is an algorithm used for construction of map structures. It is useful for distinguishing between walkable and non-walkable map areas. Another important algorithm for creation of natural-looking terrains is Perlin noise, which is one of the most known noise functions.

A problem which educational games usually face is the complexity of level creation. Instead of focusing on educational content creation, educators have to invest a lot of effort in designing every single level. This thesis discusses solutions for this problem and the implementation of a multidisciplinary educational mobile game - ”sCool”, which is based on PCG techniques. The video game serves as a platform for conveying education in a way which is different from traditional teaching approaches and more attractive to high school students and first-year university students.

The thesis also discusses implementation details of a programming environment where players can learn programming principles, type and execute code on mobile devices, practice their skills, and progress by solving challenges within the game.

The evaluations of the game showed that the participants find the game very interesting and engaging, as well as that the game helped them to better understand the educational content and inspired them to learn more about STEM disciplines.

Kurzzusammenfassung

Prozedurale Inhaltsgenerierung (PCG) ist eine Technik, um Karten, Assets und Sounds zu generieren. Diese Technik kann Spielern eine einzigartige Erfahrung bieten, indem sie endlose Karten mit einer großen Anzahl von Inhaltsvariationen erstellt. Es gibt eine Reihe verschiedener PCG-Algorithmen. Für diese Arbeit sind jene Methoden interessant, mit denen „dungeon"-artige Kartenstrukturen, Gelände, Pflanzen und Klänge erzeugt werden. „Cellular Automata" wird verwendet um zwischen begehbaren und nicht begehbaren Kartenbereichen zu unterscheiden. Mithilfe der Rauschfunktion „Perlin Noise" werden möglichst natürlich aussehende Gelände erstellt.

Ein Problem, mit dem Lernspiele oft konfrontiert sind, ist die Komplexität der Level Erstellung. Die manuelle Erstellung von Levels ist ein sehr zeitaufwendiger Prozess. Anstatt sich auf die Erstellung von Inhalten zu konzentrieren, müssen Pädagogen viel Aufwand in die Gestaltung jeder einzelnen Herausforderung investieren. Diese Arbeit diskutiert Lösungen dafür und die Implementierung des multidisziplinären mobilen Lernspiels - „sCool", welches auf PCG-Techniken basiert. Das Spiel dient als innovative Plattform für die Vermittlung von Lerninhalten, die für Gymnasiasten und Studienanfänger attraktiv ist.

Die Arbeit behandelt die Implementierungsdetails einer Programmierumgebung für mobile Geräte innerhalb des Spiels. In dieser Umgebung können Spieler Programmierprinzipien erlernen, Code schreiben und ausführen und dabei ihre Fertigkeiten trainieren. Fortschritte im Spiel werden durch das Lösen von Herausforderungen erzielt.

Die ersten Evaluierungen des Spiels zeigten, dass die Teilnehmer das Spiel sehr interessant und ansprechend fanden. Das Spiel half ihnen, den Inhalt besser zu verstehen und inspirierte sie dazu, mehr über die MINT zu lernen.

Contents

List of Figures

List of Tables

Abbreviations

ASP Answer Set Programming. 16

FPS Frames per second. 61
FPS First-person shooter. 7

IDE Integrated development environment. 33

PCG Procedural content generation. xiii, 1, 2, 6

UX User Experience. 31

1. Introduction

In the past, people have been using different techniques for memorizing content since they did not have access to technology we have today (Ralby, Mentzelopoulos, & Cook, 2017). Nowadays, it is really hard to imagine teaching and learning processes without technology involved. Technological innovations allow quick access to different learning resources and have the potential to transform education (Maksym, 2017). Scientists agree that video games can be very useful for teaching as they can improve the motivation of students (Squire, 2003).

One way to design an educational video game is to implement a course curriculum into a commercial game design for entertainment, which could help enhance the motivation and engagement of players Squire (2011). In order to provide unique activities throughout the game levels, increase attention and interest of players, it is necessary to constantly offer a new content. That is the case where PCG plays a significant role. Procedural generation is a process of creating content based on the set of parameters provided by the programmer. This powerful approach allows programmers to create a number of variations of a game object, map or sound. Another important aspect of PCG is replayability, which basically means that every time players run a game, they will get a different levels and content. There is already a number of video games which are utilizing this concept, but these techniques are becoming more and more popular among game programmers (Smith, 2014, 2015). The main motivation is that it is possible to generate infinite maps, open worlds, endless music, and unique elements with very limited resources. In other words, it is not necessary to manually create every single element, but rather use a very basic input (a texture, a model or a sound) and generate a number of variations.

1.1. Project Aims and Objectives

The main objectives of this work are the design, implementation and evaluation of an educational video game based on PCG techniques. There are two main features which this educational game should support:

- Procedural content generation (PCG) to get educators focused on their curriculums rather than on level creation, to enhance engagement of players, and motivate players to improve their knowledge by playing the game.
- A programming environment where players can learn programming principles, type and execute code on mobile devices, practice their skills, and progress by solving challenges within the game.

The most challenging part in this work is to make a game which is going to be engaging and educational at the same time. The target audience are high school students and first-year university students, that is, students of age 11 to 19. As it is already known, this target audience is very demanding which makes it challenging to design an educational game that they will find interesting. There should not be the parts where a play stops and education comes in (Šisler & Brom, 2008). Education should blend in into a game so that the players are not able to tell the difference (Amory & Seagram, 2003).

Based on already existing video games, there is certainly a potential in using PCG for educational purposes, as it can motivate players to explore and engage more with the provided content (Nebel, Schneider, & Rey, 2016). Providing an exploration experience is a good approach for motivating users to acquire knowledge and critically think about the information they receive (Risi, Lehman, D'Ambrosio, Hall, & Stanley, 2016).

The name of this project is "sCool", and it represents the combination of words school and cool. The idea is to implement one course (programming) which will serve as a proof of concept and must show that other subjects like mathematics, physics, electrical engineering, etc. could be implemented in the same way without collapsing the structure. The reason why programming course is taken for the prototype, is mostly because the author of this paper comes from computer science background. In the programming

course, students can learn Python in an interesting way by programming a robot to move around the map. According to Zelle (2004), Python is near-ideal first language, and compared to other programming languages, Python is more readable and takes fewer lines of code to write the equivalent program. Building blocks concept allows players to use already predefined code snippets and drag-and-drop them into the desired place in the code. This helps players to write their code faster and reduces the number of syntax errors in the code. Also, this could be very helpful for beginners as they do not need to memorize code syntax in order to write their programs. An important component of this project is a web application which is used to help educators create their courses and invite students to join. The content created by educators is loaded into the game and presented to players. The web application, which works closely with the game, is implemented by Kojić (2017), as well as, the functionalities for fetching the content from the server and displaying it in the game. An educational content obtained from the server is presented in the forms of missions and challenges within the game.

1.2. Structure of the Thesis

This thesis consists of seven chapters, where every chapter discusses specific parts of the game. Chapter 1 is an introduction and gives a brief overview of the entire work. It also analyzes the motivation for this work and defines the thesis structure. Chapter 2 provides a literature review about existing educational video games and an overview of procedural content generation algorithms. Chapter 3 discusses about design and introduces a conceptual model. The basis for the educational video game is explained in this chapter with other important features such as playgrounds and building blocks. It shows what algorithms for procedural map and sound generation are going to be used for. Chapter 4 consists of the most important sections of the implementation and showcase scenario. It provides a detailed discussion how Cellular automata and Perlin noise can be used for generation of maps. There will be discussions about infinite maps, how to efficiently generate endless words, and run all of these on different mobile platforms. Chapter 5 analyzes the prototype evaluation and the final evaluation. It discusses

different setups and methodologies for evaluation and summarizes participants' feedback. There will be more details about initial testing of UI and UX, and how the feedback received from players who have tested the game helped to make important improvements and modifications. Also, the chapter covers the final evaluation of the system and the general feedback from the specific target audience. Chapter 6 discusses lessons learned while working on this project and summarizes different challenges and issues which appeared during the background and concept research, implementation, and evaluation. Chapter 7 provides ideas for future work and gives suggestions about the steps which have to be taken in order to make this game look and feel better. It also consists of a list of features which have to be improved and proposes features and modification which could make this game even better.

2. Background and Related Work

This chapter provides an explanation about procedural content generation and analyzes the content which can be procedurally generated. It shows an overview of most popular PCG algorithms, examines the way they work, as well as, what they can be used for. Also, there is a discusses about some of successful educational video games and more details about components which make those games outstanding. This chapter also provides an overview of PCG algorithms used by developers in those games. Minecraft[1], Spelunky[2], No man's sky[3] are some of the games which are based on PCG principles and are going to be explained in greater detail further on. Minecraft is one of the game examples which widely used in education.
It is essential to reflect all the benefits and drawbacks of procedural generation, as well as, to explain which one is giving better results and when. Finally, once the content is generated, it is important to perform an evaluation of the created content, understand is the content satisfying proposed criteria, and can it be more successful.

2.1. Video Games in Education

Video games may help players develop different skills and enhance education, if they are designed in a manner which enforces the natural way of acquiring knowledge (De Aguilera & Mendiz, 2003). The idea is that one should be able to fail a number of times until understanding what was wrong and what caused the failure (Pirker, Guetl, & Astatke, 2015). The major benefit of video games in education is that they can capture the

[1]https://minecraft.net
[2]http://www.spelunkyworld.com
[3]https://www.nomanssky.com

attention of students so that the students want to keep playing the game and learn more through it (Gentile, Lynch, Linder, & Walsh, 2004).

New generations of students have different preferences when it comes to learning, mostly due to technological innovations which have significantly influenced learning and teaching strategies (Chang & Guetl, 2010). Video games should be able to immediately provide feedback, and reward or punish players based on what they are doing in the game is correct or wrong (Hanus & Fox, 2015). Researchers have been using them in teaching computer science, medicine, languages, and even criminal law, where video games helped with motivating students and enhanced learning effectiveness (Mentzelopoulos, Parrish, Kathrani, & Economou, 2016). Another great benefit of video games is the use of repetition. Repetition is very important in education as it probably the most intuitive principle of learning (Weibell, 2017). Video games are also based on repetition, as goals in a game are usually repeating throughout the game. For example, if a game is a First-person shooter (FPS), then one has to use guns and shoot at enemies. That usually repeats as a player progress and the only difference is that enemies are getting stronger and more advanced, but the player usually has better weapons. Course curricula are mostly designed in a similar way, as initial lectures are introductions to more advanced topics. One of the best ways to learn something is by practicing it (Schank, Berman, & Macpherson, 1999). When a content from a topic is explained with words using traditional (oral) approach, sometimes it can be too abstract to understand, but when one does something in practice, then student is not only understanding how presented matter works, but mastering that and is able to apply acquired skills in different settings. That is the main benefit of practical approach over theoretical, but theoretical part cannot be neglected, as before doing something one has to be able to understand how that works. It is important, however, that theoretical and practical tasks are combined so that they act together. (Dewey, 1904)

2.2. Procedural Content Generation (PCG)

Procedural Content Generation is a process of generating game content using algorithms with little or no user input (Togelius, Kastbjerg, Schedl, & Yannakakis, 2011). The content can be anything from maps and worlds, to assets and music. It is important to note that a content is not considered to be any game-specific functionality and behavior. One of the very important components, which has to be taken into account when using PCG techniques, is a set of constraints, which ensures that the generated output is valid. The reason why PCG is so useful, is that it can replace humans in content creation. Also, it can speed up that process and generate a number of different outputs in a matter of seconds. Another benefit is that the gameplay can be adapted to how a specific player is playing the game. This means the new game rules and goals can be created based on the previous player's progress and the way that player was interacting with the game. In the early 80s, the limited capabilities of computers forced developers to look for other methods for loading and storing the content. That inspired developers to try to overcome those limitations by developing PCG methods. The most challenging part is to generate the content which reaches the quality of manually crafted one. It should be possible to control how the generated content will look like and it is important to ensure that there is enough diversity, so that the outputs do not look alike. Finally, the PCG generators should be fast and reliable so they can generate the content according to specified criteria (Shaker, Togelius, & Nelson, 2016).

2.3. History of PCG Games

A game category which has the longest tradition of using PCG techniques are roguelike games (Hatfield, 2017). In the early 1980s, game developers started using PCG to randomly define the maps with rooms and hallways, as well as, to generate certain enemy units and collectibles.

An example of a game where pretty much everything is generated with PCG algorithms is .kkrieger (first person shooter which size is only 96KB). This means that every texture, mesh, and sound has been generated with

Figure 2.1.: Screenshot taken from GamaSutra website (Wen, 2017).

procedural methods. The game is considered to be one of the smallest PC FPS games (Wen, 2017).

No Man's Sky[4] is an action-adventure survival video game where the geometry, textures, and animations were built by using procedural methods. The game is based on the procedurally generated multi-planetary scale with 18 quintillions of planets, unique creatures, ships and other models which are built by using PCG principles. Every planet features unique flora and fauna. The game was marked as one of the most anticipated games in 2016 (Gamasutra, 2017). Figure 2.2 shows a screenshot from the game.

Minecraft[5] is one of the most known PCG games. Worlds in the game are created by using Perlin noise (see Figure 2.3). The game is mostly about building a world with textured blocks. Players can come up with their own goals which might be the reason why Minecraft is so popular (Walton, 2012). There are five different game modes: survival, creative, adventure, spectator, and hardcore mode (Wikipedia, 2017a), where every mode offers different gameplay and objectives. This means the game can attract different player

[4]https://www.nomanssky.com
[5]https://minecraft.net

Figure 2.2.: Screenshot from No Man's Sky game (MathChief, 2017).

types and target groups. According to (Bartle, 1996), there are four different gamer types, such as achievers, explorers, socializers and killers. This means that every player can find their matching Minecraft mode. Figure 2.4 shows a screenshot from Code builder - Minecraft education edition[6]. Students can use Scratch[7], Tynker[8] or MakeCode[9] to create their worlds in Minecraft through the code. Coding is not the only thing one can learn, as there are many other subjects like architecture, art, literacy, etc. One of the reasons why Minecraft is so well accepted by students is because they play Minecraft in their free time anyway, so if educators can adapt their curriculum and integrate it into Minecraft, then that can really help with presenting the curriculum in a way that is very interesting to students (Mojang, 2017a).

Another example is Terraria[10], which was characterized by many as a 2D version of Minecraft. Terrains are procedurally generated by using the

[6]https://education.minecraft.net
[7]https://scratch.mit.edu
[8]https://www.tynker.com
[9]https://makecode.com
[10]https://terraria.org

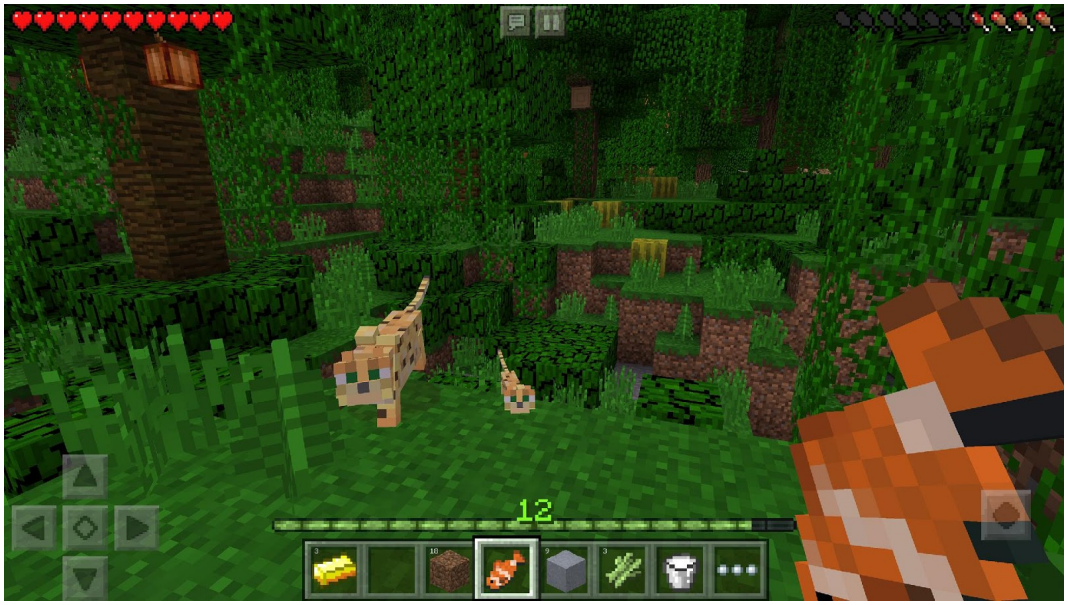


Figure 2.3.: Screenshot taken from Minecraft website (Mojang, 2017a).

Figure 2.4.: Screenshot taken from Minecraft education version website (Mojang, 2017b).

Figure 2.5.: Screenshot taken from Terraria website (Relogic, 2017).

constructive methods, such as Cellular Automata and noise functions. The game offers a rich gameplay as the core features are digging, gathering, crafting, exploring, fighting and building. Once some materials or items have been collected then one can combine them by crafting a new item, which is usually a weapon or a tool that can be used later on in fighting or building processes. This enhances the creativity of players as they often have to come up with solutions with limited resources at their disposal. There are many modes created by the community as everyone can create their own maps and make it publicly available so that others can try them as well (Relogic, 2017).

Spelunky[11] is an example of how procedural generation can make a game interesting if implemented in a way that follows all constraints defined in the game design process. The caves are procedurally generated which means that every time a level is repeated, the cave is going to be different and all pickups will be placed in different positions. There are four different

[11]http://www.spelunkyworld.com/

Figure 2.6.: Screenshot taken from Spelunky website (Spelunky, 2017).

cave types and every type features different enemy units. When players run out of hearts, they have to start the game from the beginning and this time the cave will look differently so the players will not be bored (Baghdadi et al., 2015; Wikipedia, 2017b). Figure 2.6 shows a screenshot from the game.

There is a number of mobile games which generate levels by randomly instantiating obstacles, power-ups, and pickups. Subway Surfers[12] is a mobile game with the most number of downloads on the mobile platforms and has more monthly active players than League of Legends[13]. It still generates 30 million new downloads per month and even though it has been released 5 years ago, the game is still at the top of the charts (Bogacheva, 2017). This game uses random number generator for generating a sequence of predefined map parts which are put together to form an endless map.

[12]http://subwaysurfers.com
[13]https://eune.leagueoflegends.com/en

Figure 2.7.: Yavalath game (Nestorgames, 2017).

2.4. Procedural Generation of Game Assets

When using PCG for generating a content, it is important to understand what one is actually expecting to have as an output. Procedural generation can be used not only for generating maps, worlds, content, and sound but for generating game rules as well. As an input, the algorithm can take a set of constrains which are going to be used when deriving the rules. The algorithm can determine whether the games should be turn-based or real-time. One of the most known examples of PCG in generating rules is Yavalath game, which has been invented by a computer program called LUDI written and by Cameron Browne (Browne, 2017). The game can be played by 2 or 3 players and the winner is a first player which manages to place four stones in a row without placing tree stones in a row first. A screenshot from Yavalath is shown in Figure 2.7. The game title was also generated, this time using Markovian process.

Procedural generation is mostly used for generation of maps and worlds.

There are many algorithms and techniques to successfully generate a map. Distribution methods are very useful for this purpose. Cellular automata is one of the most popular PCG algorithms. It works in a way that it generates zeros and ones at random positions. Then, it iterates a number of times until elements are grouped together and form a structure. When it comes to game content PCG techniques can generate different tree types, enemy types, rock types, etc. In No Man's sky, all species are procedurally generated, and there are many parameters which determine what kind of spicy is going to be generated, such as surrounding area, environment, location and player interaction (Gamepedia, 2017). Trees can be generated by using L-system algorithm or Fractals based on lines or squares (Pythagorean theorem). In upcoming chapters, this will be explained in greater detail. Procedural sound generation can help generate unique and endless music through the code. The main benefit is that it is not necessary to store many sound files which saves a lot of memory. Music can change based on the player's surrounding, health, and mood. It is important, in the end, to evaluate the quality of what is generated to check if produced rules are interesting enough to players.

2.5. Overview of PCG Algorithms

There is no standardized taxonomy of procedural content generation algorithms, so there is a number of different classifications.

Based on user input, procedural generation algorithms can be separated into two categories:

- Those who generate content based on input
- Those who try to create a structure out of a random noise

According to Shaker et al. (2016) all algorithm can be categorized as follow:

- Search-based approach
- Constructive generation methods for dungeons and level
- Fractals, noise and agents
- Grammars and L-systems
- Rules and mechanics

- Stories and quests
- Answer Set Programming (ASP)
- Representation for search-based methods
- Experience driven PCG
- Mixed-initiative content creation

Another categorization presented by Hendrikx, Meijer, Van Der Velden, and Iosup (2013), shows that taxonomy of common methods for the generation of a game content is:

- Pseudo-Random Number Generators
- Generative Grammars
- Image Filtering
- Spatial Algorithms
- Rules and mechanics
- Modeling and Simulation of Complex Systems
- Artificial Intelligence

There is a number of algorithms associated with every category in two previously described taxonomies. The further text discusses the algorithms which are related to this thesis and which can be used to generate a content for the multidisciplinary educational game.

2.5.1. Cellular Automata

The first three algorithms, described in further text, belong to constructive algorithms family (Shaker et al., 2016). The constructive generation methods are particularly used for generating dungeons and labyrinth-like structures. Their greatest advantage is that they are simple to understand and implement. Also, they are generally very quick and responsive. These methods, however, provide very limited control over what is being generated. In other words, they do not give programmers much freedom to control the final outcome.

Cellular Automata consist of a grid of cells, where every cell can be in the on-off or one-zero state. The algorithm defines how each cell changes based on the states of adjacent cells (Berto & Tagliabue, 2017). The best-known

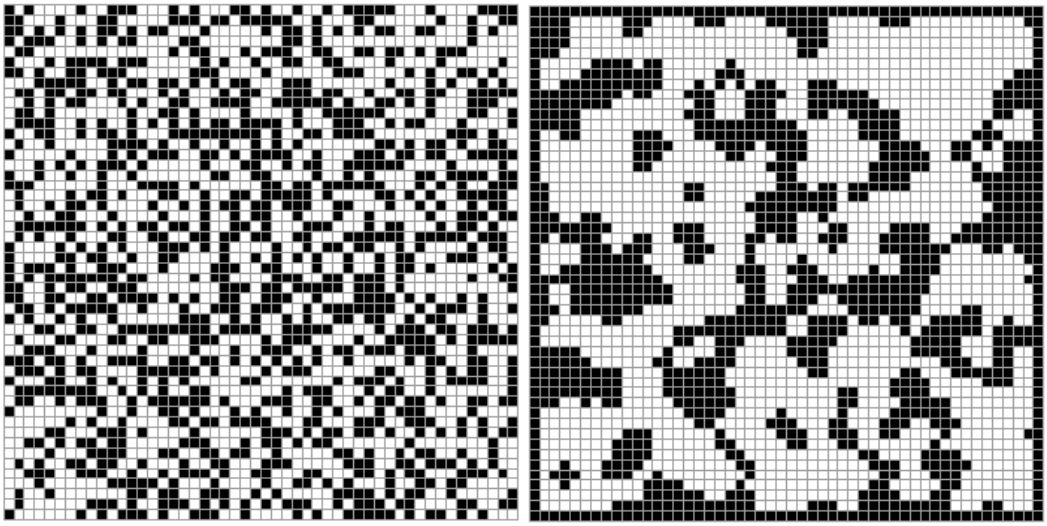


Figure 2.8.: An example of cellular automata which shows how 2D grid looks like when values are randomly generated and the outcome of iterations with natural cave-like shape (Kun, 2017).

example of Cellular automata is "The game of life" by John Horton Conway (Wikidot, 2017). Conway's Game of life has the following set of rules:

1. If a cell is alive (state 1) and has less than two live neighbors (with state 1), then it dies (becomes 0) - because of underpopulation
2. If a cell is alive and has three or more live neighbors, then it dies - because of overcrowding
3. If a cell is alive and has two or three live neighbors, then it stays alive
4. If a cell is dead and has three live neighbors, then it becomes alive

Figure 2.8 shows an example generated by using Cellular automata.

Pros and Cons

Cellular Automata is very useful for generating cave-like shapes. It is very often used for generating content in rouge like games, where it is necessary to generate different dungeons. However, sometimes the algorithm can produce isolated cave sections, that is, the rooms which are not reachable. Also,

creating the large maps might require additional tuning of the generated outputs as they sometimes might not have a natural look.

2.5.2. Agent-based Growing

The next method type, which can be used for generating maps, is agent-based growing. An agent is dictating how the generated map will look like. The agent's movement is based on stochastic process, therefore the output of this algorithm is unpredictable. It is very common to see that two or more rooms are overlapping when a map is generated. In order to improve results generated by this algorithm, one can use "loop-up" approach which is basically checking if overlaps are going to occur and prevents them. Figure 2.9) shows how a map generated by this algorithm looks like (Shaker et al., 2016).

Pros and Cons

One of the benefits is that this algorithm is capable of creating organic dungeons. Also, it is possible to modify the agents during simulations and inform them about the environment, so they can avoid potential overlapping of rooms and corridors. A drawback of this approach is that sometimes the outputs can be chaotic and unpredictable.

2.5.3. Space Partitioning Methods

Space partitioning methods divide space into subsets. The most popular method is binary space partitioning, where the surface is recursively divided into two parts which repeats until stopping criteria is met. The space can be represented as a binary tree. There are two types of trees which can be formed: a Quadtree and an Octree. A Quadtree is used for a 2D space and it splits the space into four quadrants. An Octree is useful for partitioning 3D spaces into eight octants. An example of quadtree can be seen in Figure 2.10 (Naylor, 1998).

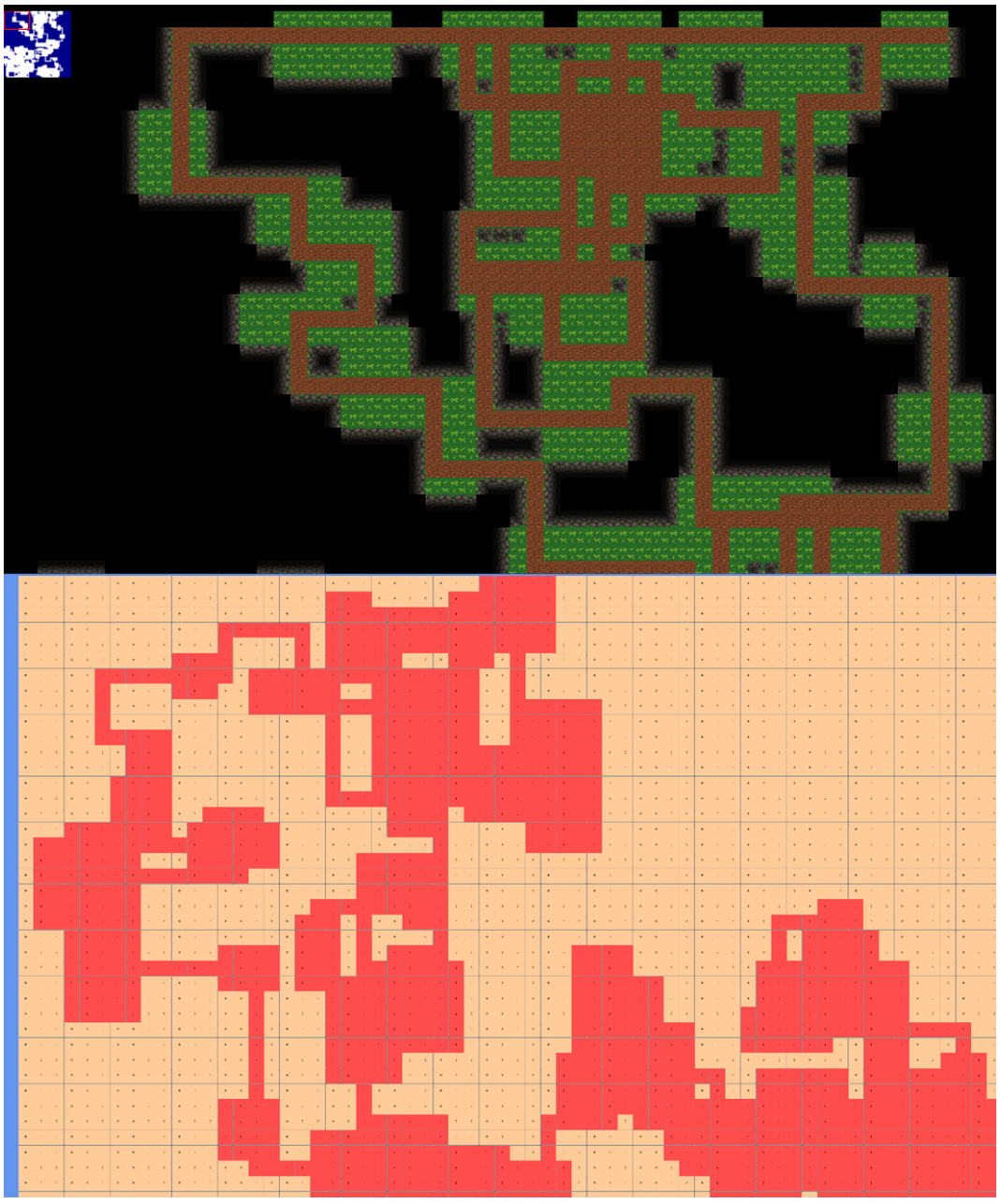


Figure 2.9.: An example of agent-based map generation (Johnson, 2017).

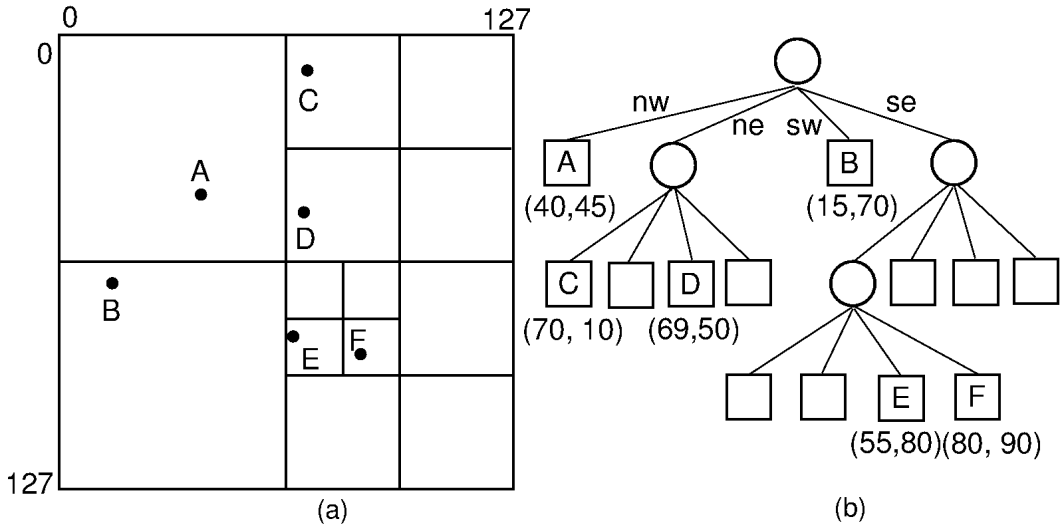


Figure 2.10.: Binary space partitioning - quadtree (Shaffer, 2017).

Figure 2.11 gives an example which can be used for generating dungeon-like maps. In this example, the first two iterations have been used for space partitioning. The next step is needed to draw a room in each cell. Finally, it is necessary to connect the rooms so that every room is reachable. Instead of a number of iterations, one can have some other stopping criteria. For example, partitions can be further divided, until width or height reaches a certain threshold.

Pros and Cons

Space partitioning method guarantees that there will be no overlapping and is useful for creating organized structures of dungeons. A lack of organic and more natural look of generated dungeons is the main disadvantage of this method.

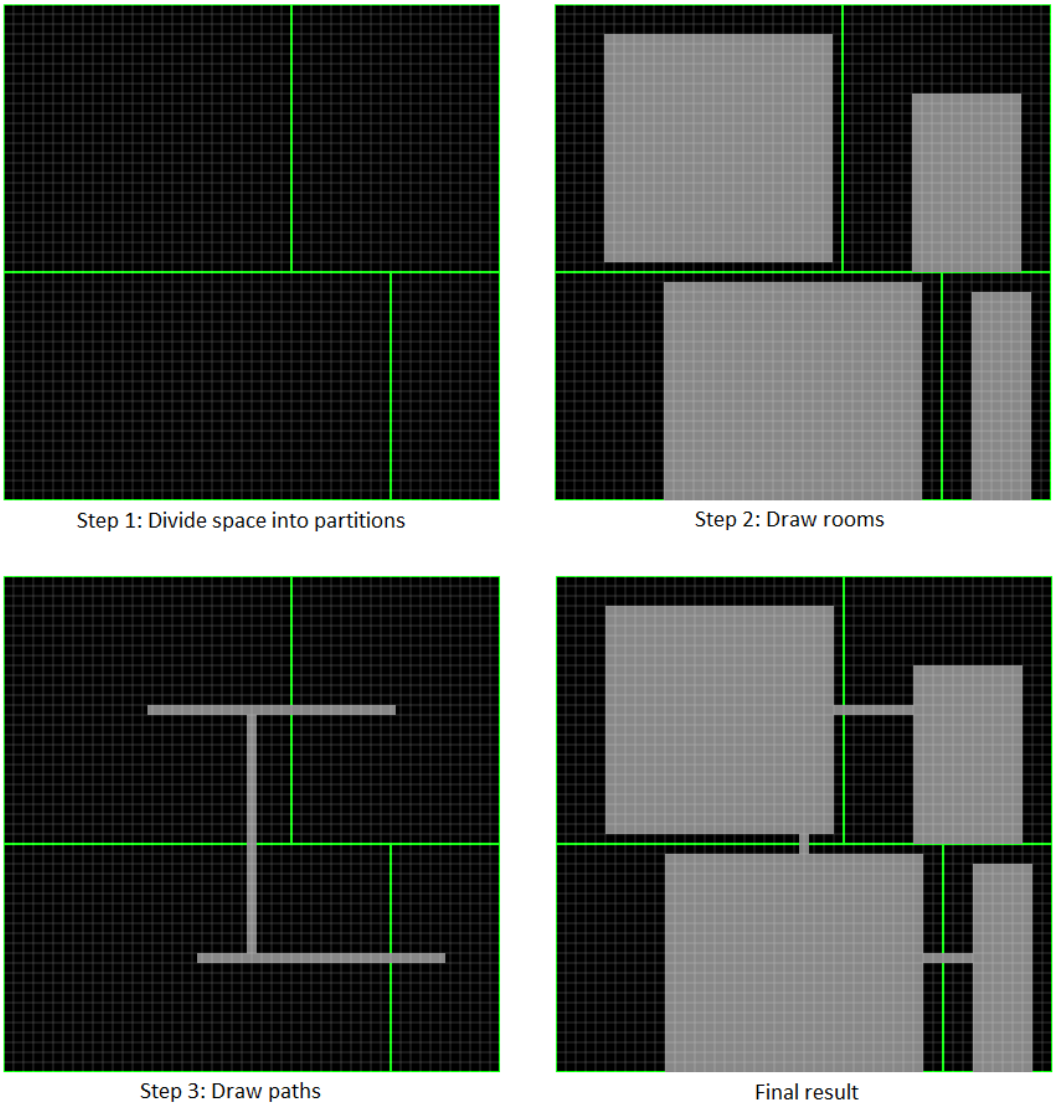


Figure 2.11.: The process of generating a dungeon using BSP (Eskerda, 2017).

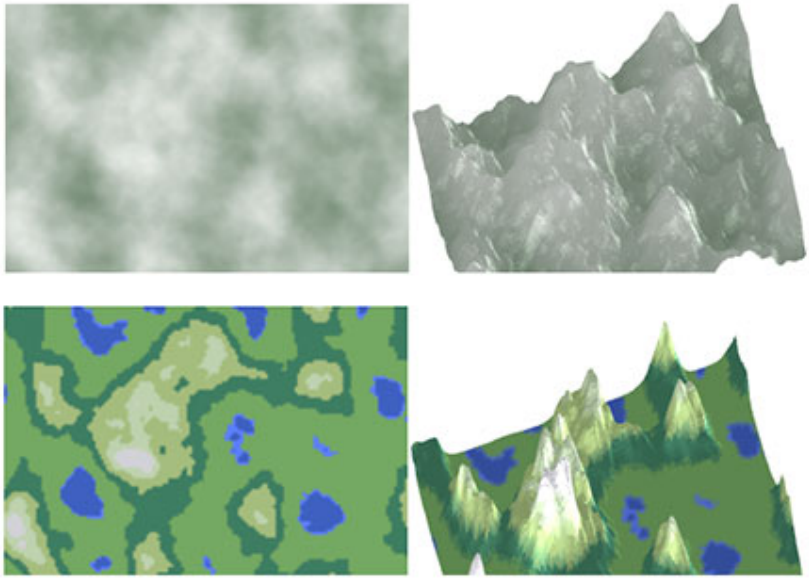


Figure 2.12.: An example of Perlin noise usage. Screenshot taken from (RedBlobGames, 2017) website.

2.5.4. Height Maps

Height maps are very useful for generating terrains and landscapes. There are different types of noises which can be used for this purpose. Perlin noise is the most used one, probably because it is not complex to understand how this algorithm works and to implement it. In order to generate a map using Perlin noise, it is needed to create a grid of random values between zero and one. The simplest way to create a height map is to use previously generated values for elevation. The terrain parts above specified thresholds can be colored differently, which helps to create a more realistic landscape. There are two important components: a frequency and an amplitude. Simple put, a frequency defines change over the x-axis, while an amplitude determines change over the y-axis. Octaves represent a set of frequencies, added as a supplement to the main frequency, in order to get natural-looking mountain shapes (RedBlobGames, 2017). Figure 2.12 shows an example of Perlin noise in action and what kind of outputs this algorithm can generate (Perlin, 2017).

Pros and Cons

Perlin noise is capable of generating natural terrain-looking maps. It is also used for increasing the appearance of realism in computer graphics. The complexity of Perlin noise is $O(N^2)$ which is improved and reduced to O(N) in Simplex noise.

2.5.5. L-System

L-system is a type of grammars which is mostly used for plants generation. Grammars simply said, are used to recursively make or derive objects from other objects. There is a number of examples of tree and weed generation, as well as, road and city constructions. Grammars consist of a nonterminal symbol set, terminal symbol set and a production rules set. The nonterminal symbols represent symbols which can be further derived according to production rules. The terminal symbols are finite results of the recursive process. In other words, they can not be changed (further derived) anymore. The production rules determine in what way nonterminal symbols will be derived. Generating fractal trees is one of the easiest and most popular examples of L-system. Figure 2.13 illustrates examples of tree generation using L-system (Marvie, Perret, & Bouatouch, 2005; Prusinkiewicz & Lindenmayer, 2012).

Pros and Cons

L-system is suitable for a tree, plant and city generation. It is also useful for generation of forests and vegetation. The weakness is that it requires some tunings for creation of desired forms and shapes.

2.5.6. Search-based Approach

An evolution computation is a basis for a search-based approach. It works in a way that it keeps generated solutions which satisfy given criteria and

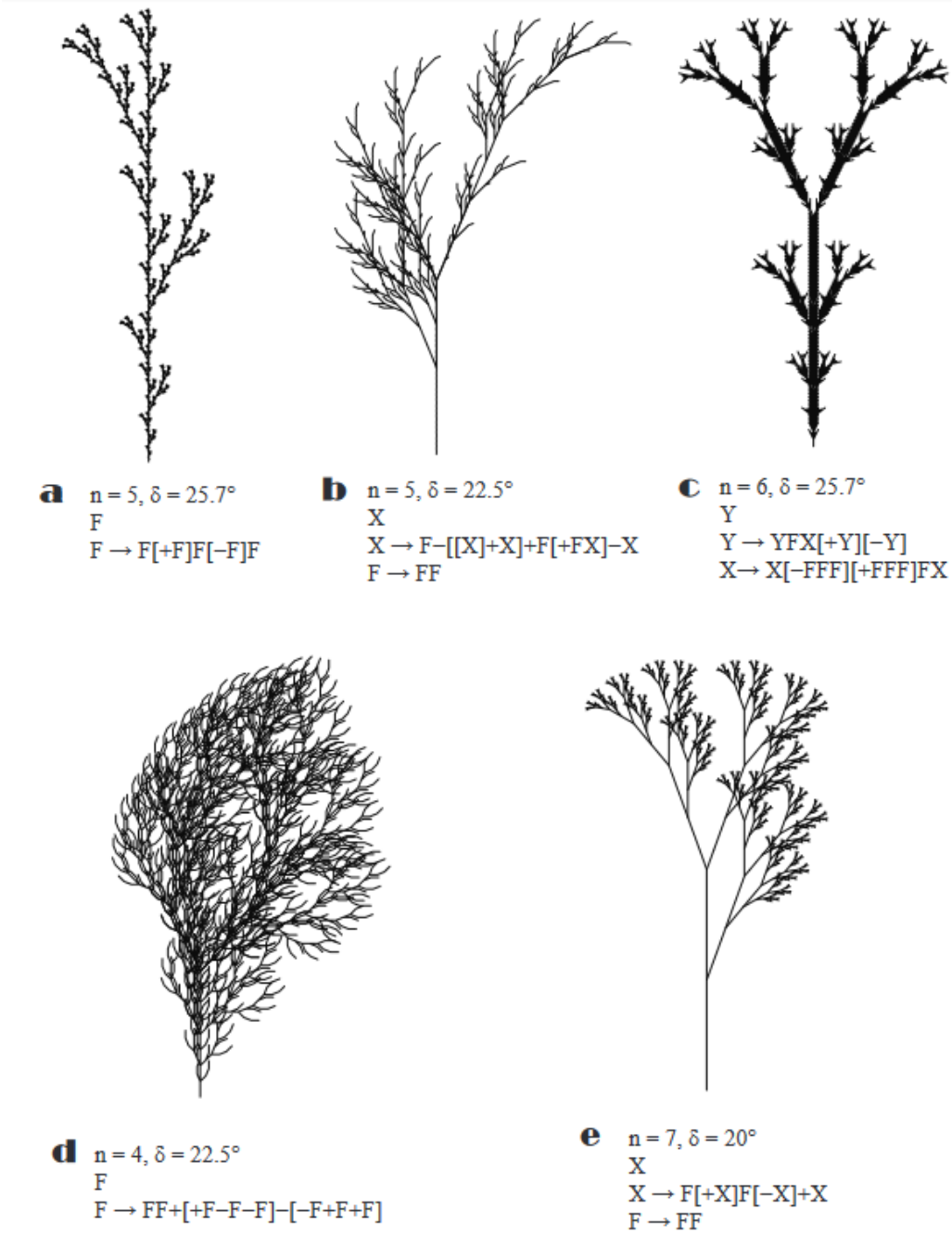


Figure 2.13.: Trees generated with L-system (Prusinkiewicz & Hanan, 2013).

discard those which do not (Eiben, Smith, et al., 2003; McPhee, Poli, & Langdon, 2008).

The main components of this approach are:

- **Search algorithm –** an evolutionary algorithm which generates candidate solutions using methods such as reproduction, mutation, recombination, and selection. There are other types of evolutionary algorithms which can be used as a search algorithm, such as genetic algorithm and stochastic optimization algorithms (J. Clune & Lipson, 2011; Goldberg, 1989).
- **Content representation –** necessary to determine what exactly is going to be generated. Genotypes, which represent the solutions from generation phase, are converted to phenotypes which represent evolved entities. The Content can be an element on a map, list of positions indicating where enemy units are located, list of collectibles one is going to have in a game, even some parameters like the distance between collectibles or enemy units.
- **Evaluation functions –** defines the quality of solutions, and it is one of the most important components as it can dramatically determine how the final result is going to look like. An example is to measure a fairness of parts of the map. Usually, only one evaluation function is not enough to ensure high quality of results, so it is necessary to use multi-objective evolutionary algorithms which will find elements with the best matching weights. There are three types of evaluation functions: direct, simulation-based, and interactive. The two major types of direct evaluation functions are theory driven and data-driven functions. The theory-driven functions take into account theoretical studies of fun in games, while the data-driven functions take collected data from the game and compare to player's feedback. In the simulation-based evaluation functions, AI agents are usually created to simulate player's behavior and test quality of generated maps, worlds or tracks. The interactive evaluation functions are based on a data collection which shows how a player is interacting with the game. An example of that would be to generate specific weapon (vehicle, spell, etc.) of a particular type if the data show that the player has been using a specific weapon very often (Togelius, Yannakakis, Stanley, & Browne, 2011).

Pros and Cons

The search-based method has a broad range of capabilities as it is possible to use this method for generation of different game elements, as well as, parameters which can be used to, for example, determine the position of enemies on the map and distances between them. If the search space is too large, then the algorithm might take some time to produce the result.

2.5.7. Voronoi Diagram

Voronoi diagram is mostly used for partitioning a plane into cells, where every cell has a set of points in the plane that are closer to each cell than to any other adjacent cell. It can be very useful in a case where there is a number of objects spread over a map and one wants to determine what is the closest object to the specific point on the map. Voronoi diagram is something which would really help with the previously explained problem. Every vertex in Voronoi diagram is of degree 3, which can be seen in Figure 2.14. A set of points of Voronoi diagram is dual to Delaunay triangulation. Delaunay triangulation is a special form of triangulation where no further straight line or edge can be added without crossing other edges (see Figure 2.15) (Aurenhammer, 1991; De Berg, Cheong, Van Kreveld, & Overmars, 2008). There is a number of applications of Voronoi diagram in video games development. It is mostly used for generating maps and textures. An outcome of using Voronoi diagram can be seen in Figure 2.16.

In order to reconstruct what is shown in Figure 2.16 it is necessary to pick random points on a grid and create Voronoi polygons. Once the map is ready, one can use any algorithm explained in previous sections to distinguish water polygons from grounds polygons. The next step is to elevate the ground polygons. This is usually achieved by using height maps. The elevation level can help to define different biome types. Moisture map sections and rivers can be useful to add more dynamics to maps and provide a more natural look and feel (Patel, 1991). Generating textures is another example where Voronoi diagrams can be very successful. Figure 2.17 shows how researchers have been using it for texture generation. It works in a way that users provide an input image. After a specific feature from the

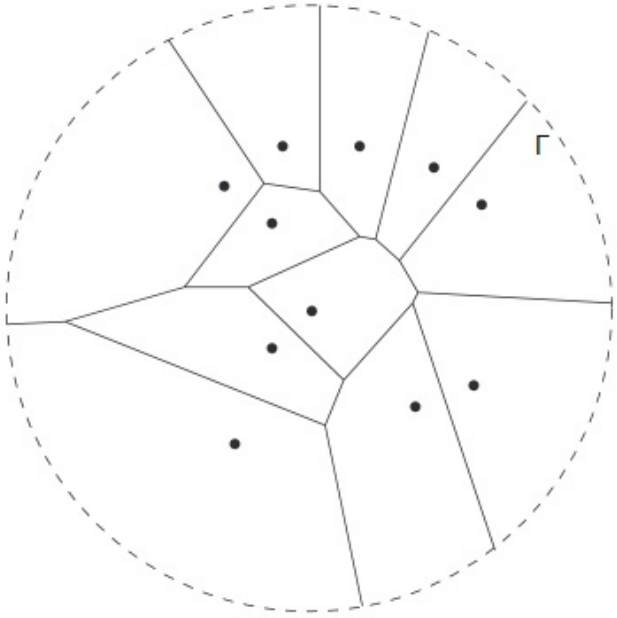


Figure 2.14.: An example of Voronoi diagram. Screenshot taken from (Aurenhammer, 1991).

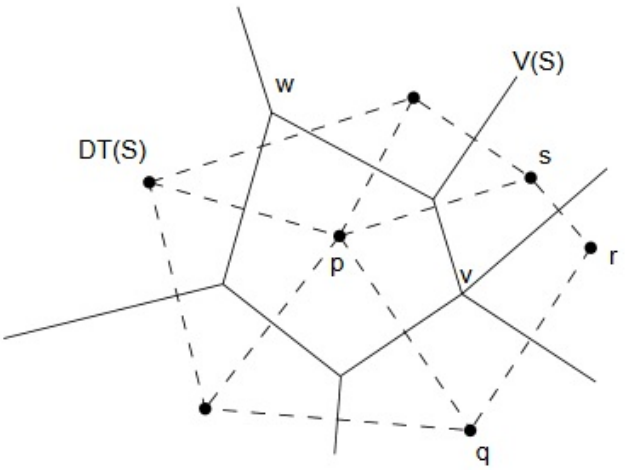


Figure 2.15.: An example of Delaunay triangulation. Screenshot taken from (Aurenhammer, 1991).

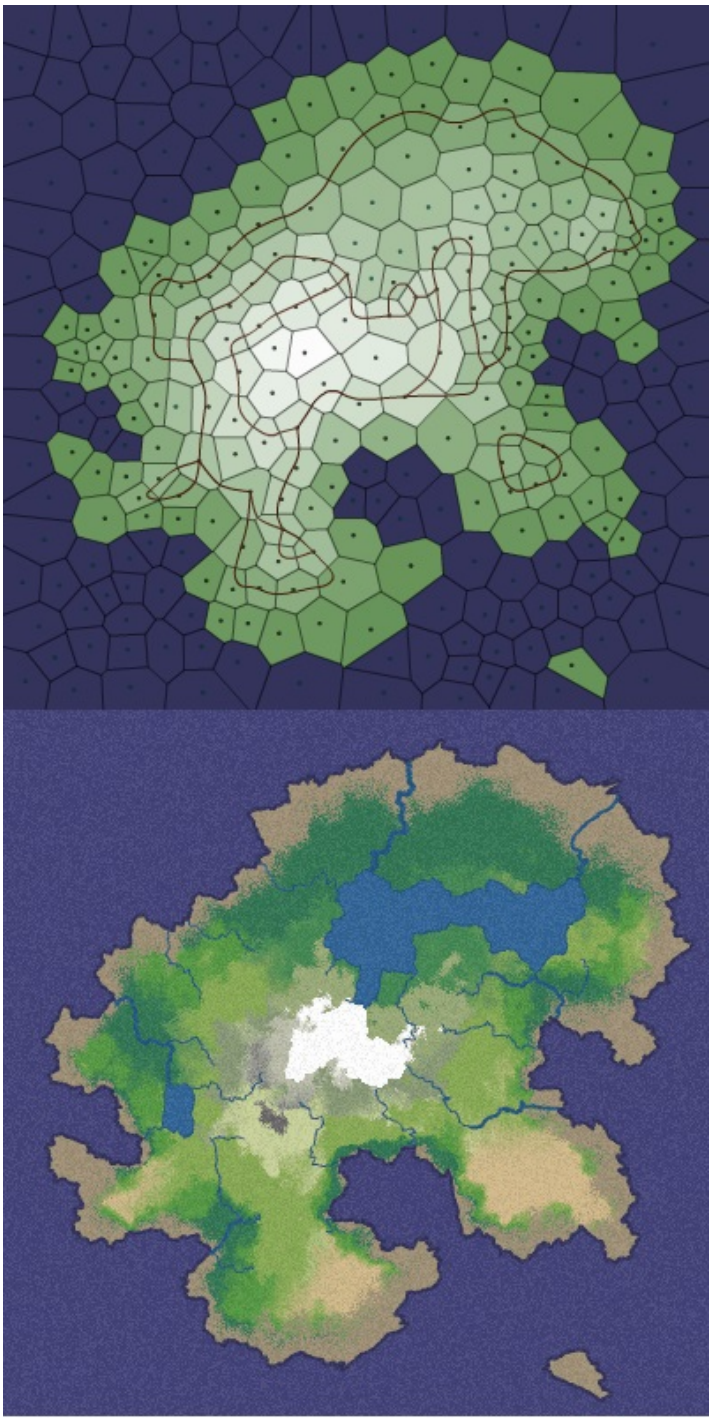


Figure 2.16.: An example of map generation using Voronoi diagrams. Screenshot taken from (Patel, 1991).

input has been selected, the algorithm finds positions of other occurrences of the selected feature and maps it into a Voronoi diagram (Sabha & Dutré, 2007).

Pros and Cons

Voronoi diagram principle is very effective in a map generation. By using this method, it is simple to modify specific sections of the map. A diagram can be easily calculated for small maps with a few points. However, in order to generate larger maps in real-time, an improved computational approach is required.

2.6. Evaluation of Content Generators

Sometimes PCG algorithms can produce pattern-like structures, and repetitions in the level design and map generation. Another problem could be that items are not spread out uniformly, resulting in positioning all pickups in one area only. It is very often that during map generation an algorithm creates maps with unreachable zones. In order to improve the quality of generated content, it is necessary to use evaluation functions. The evaluation function is an algorithm which judges the quality of the generated content. Some of the important things which the algorithm has to test in the map generation are the connectivity of every sector of a map. Also, there should be ranking and prioritization criteria. Results which do not satisfy these standards, should be thrown away and the process of map generation must be initialized again. The process repeats until one of the generated outputs fulfills given criteria. However, one has to be very careful with setting up constraints, as if constraints are too ”tight” then it might take some time until the algorithm produces the result which satisfies defined standards. It can happen that quality functions never accept anything, which can cause infinite loop, as the algorithm will try to generate new outputs over and over (J. E. Clune, 2007; Federoff, 2002; Horn, Dahlskog, Shaker, Smith, & Togelius, 2014).

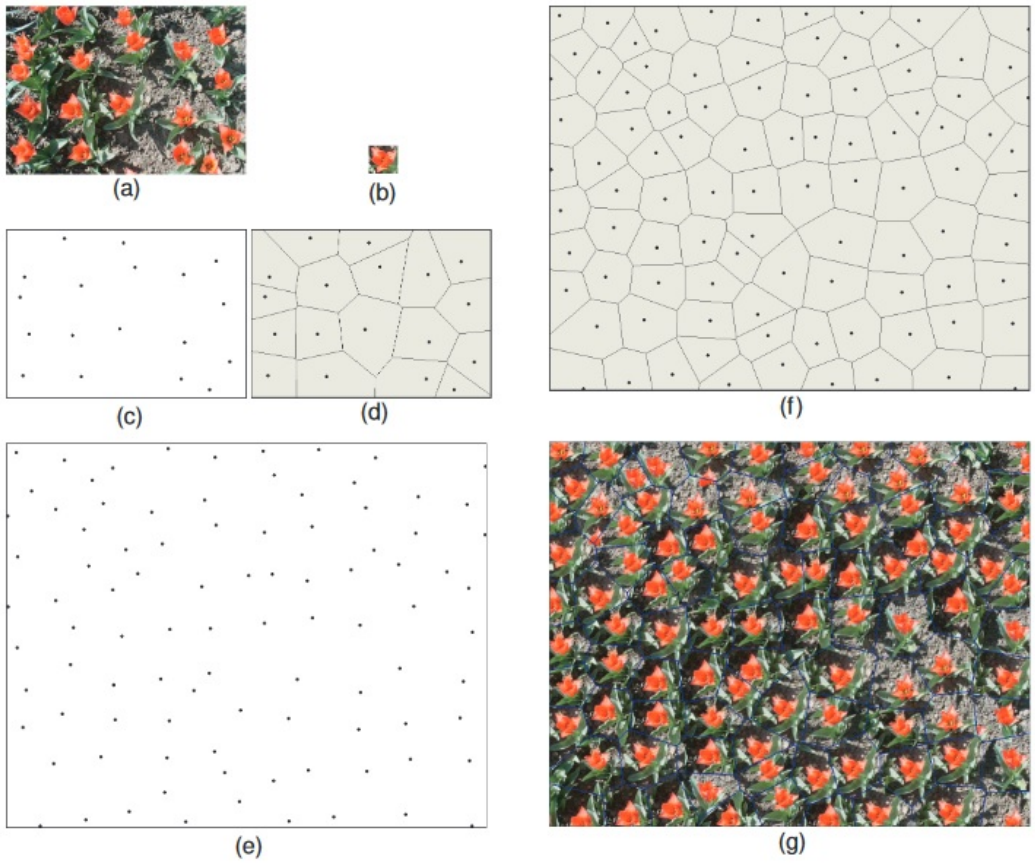


Figure 2.17.: An input image is shown in (a), while a feature which the user has selected is in (b). The algorithm detected positions of similar features (c) and their Voronoi diagram is constructed (d). A similar distribution to the source (c) is generated in (e). The distribution from (e) is used to generate Voronoi diagram in (f), similar shape cells are copied and stitched to the resulting texture (g). Screenshot taken from (Sabha & Dutré, 2007).

2.7. Summary

The first games based on PCG techniques have been created in the early 8os and since then developers have been using PCG to create games for different gaming platforms. Some games like Minecraft and No Man's Sky have reached global success and publicity as their developers have managed to get the most out of the PCG. Procedural generation can be used for generating worlds, rules, sound, and assets. There are different categorization for PCG algorithms, since there is a number of distinct techniques and approaches for procedural generation. Cellular Automata algorithm is widely used for generating map structures. It generates a grid filled with zeros and ones, which are further modified based on values in adjacent cells. Agent-based growing uses an agent which moves throughout the map based on stochastic processes. Binary space partitioning is the most popular technique of all space partitioning methods. It is based on dividing the space into subsets of different sizes. Perlin noise is one of the most used height maps methods. There are two components (frequency and amplitude), which determine how generated terrain will look like. When it comes to grammar methods, L-systems is one of the most popular techniques. It is based on recursive function calls which generate plant parts. Evolution computation uses genetic algorithm methods to create potential candidates which are later rated by evaluation functions and used for content generation. Voronoi diagrams are widely utilized for generating maps and textures. It is very useful for determining distances between the closest object to a specific point on the map. Evaluation of content generators is very important as they ensure that generated result will satisfy minimum requirements. If the generated result is not good enough then it is necessary to discard that result and go through the process of generation again, until a new result is fulfilling all required criteria. Given all pros and cons of PCG, one can conclude that PCG can really improve User Experience (UX), increase re-playability, if implemented in the right way, and can definitely pay off in the long run.

3. Design and Conceptual Model

This chapter discusses core game components, functionalities, technology requirements, and the conceptual architecture. After explaining why a web application is necessary in the "sCool" project, the first section discusses designs of both theoretical and practical aspects of different course curricula in the game. There are more details about techniques, concepts and requirements for procedural generation of maps, content and endless music. Custom Integrated development environment (IDE) section covers how code blocks work and why they are so useful. Also, it provides more details on integrating different courses within the game and proposes the implementation of the programming course. Main non-functional requirements explain core architecturally significant requirements like usability, modularity, configurability and appealing visuals. Conceptual architecture section shows the structure of the entire project. Also, it discusses the game concept and the main objectives of players in the game, and analyzes the key features which should be implemented in the game. Discussion points in technical requirements is the analysis of PCG algorithms that are most suitable for the given set of requirements, platform and technology selection, as well as decisions necessary for implementation of playgrounds for teaching practical aspects of different courses.

3.1. Functional Requirements

Educators usually teach curricula through two main educational components. Those two components are theory and practice, and this is something which is common for all courses (Banks, 1988). It is based on a model which is already used in schools and which expects students to understand both theoretical and practical usage of a teaching content. Exploration is

something which plays an important role in education and procedural map generation can enhance that experience (Roussou, 2004). This multidisciplinary educational video game is going to offer several different courses to players. It is required to fully implement the programming course which should serve as an example of how a course should be organized.

Most significant components which should be implemented in this project are:

- **Map generation (Theoretical mode)** - necessary for teaching theoretical aspects of a course. That is a mode where players have to explore a map and collect pickups which will reveal a pieces of information. Terrain and map structures are created based on PCG algorithms which ensure that on every run players will be faced with different worlds.
- **Challenge production (Practical mode)** - the mode where players have to apply their theoretical knowledge in practice, that is, test their knowledge against problems from the real environment. In the programming course, they have to program a robot to avoid all obstacles and collect the disk.
- **User Interface** - most significant components are: drag-and-drop blocks, virtual keyboard and fields. These components can be used in other courses, not only in the programming course. They serve as input enhancement tools, which should simplify the interaction on a mobile device.
- **Procedural Sound and Content Generation** - PCG sound provides never-ending and patternless music sequence which can change and adapt to players environment. Trees, plants, forests, and vegetation can be generated using PCG algorithms for content creation.
- **3D model creation** - Even though most of the content will be generated using PCG techniques, it is still necessary to create some 3D models manually like: a player, enemy units, obstacles, pickups, etc.
- **Player customization** - improves engagement of players by allowing them to customize their characters. It is possible to change the skin color, facial expression, and character's gear. They can also buy some items in the shop and improve the character's abilities.

3.1.1. Web Application

The idea of the project is to build a game which is not only supposed to work with courses created by the author of this master's thesis. It should serve as an educational platform for teaching different topics, and should be constructed in a way that educators can create their own courses and invite students. In order to support that, it is necessary to create a web application where educators can create knowledge trees out of their curricula. It is important to note that the web application described in the further text is implemented by Kojić (2017), as a part of his Master's theses. Every course module should consist of a number of fields which are called skills in the game. Every skill has a set of tasks which are used to present educational content and test players' knowledge. For every task, educators can specify the question text, incorrect answers, and correct answer. This is all presented in a form of radio buttons so players have to select one of many offered answers. The functionalities for fetching the data from the server, tracking players' progress, showing the content to players and retrieving their input, were some of the components implemented by Kojić (2017). The main benefit for educators is that they can navigate to statistics section and see how the students are performing. There should be an overview of how the entire class and each individual are progressing. For every student the educators can check the progress within every skill, and can get even more detailed information such as a number of attempts needed to provide the correct answer, session duration in minutes and average points per session. All these details can be very useful for educators, as analyzing them they can see the weak points of the students and spend more time covering those topics during classes, on which students did not perform well.

3.1.2. Theoretical Mode

As already discussed, enforcing exploration when teaching theoretical aspects of a course is a good approach to motivate students. Since exploration requires large terrains and maps, as well as, new contents to be explored, it is required to utilize one of a number of PCG techniques for map generation. Creating maps manually is definitely easier, but if one needs more than

several maps, then that approach is not feasible. Procedural generation can help with that issue as it is possible to generate an "infinite"[1] number of different maps. However, by using manual approach, one can easily create corridors, walkable areas and places for pick ups, while doing that with procedural generation might not be that easy. Another problem is that some generated maps can be unplayable as it could contain some parts of the map which are unreachable but must be visited by players. In order to fix that, it is needed to have evaluation functions which are going to discard all maps that do not fulfill the specified criteria. Procedural map generation is very useful for enhancing replayability experience as players will get different content when repeating the same level.

After understanding why procedural generation is important in the project, it is necessary to discuss about functionalities of different algorithms which could be used for map generation. The requirements for the map generation are the outputs with the clear distinction between walkable areas from non-walkable ares and natural-looking terrains. In other words, exploration maps in this game should have walkable areas surrounded by walls, hills or trees. In order to add more randomness to map generation, it is a good idea to introduce features which are slightly going to change the environment. One of the features could be a dynamic light which changes the lightings on a map. That could be based on random events, so that players can sometimes experience playing the game in dark environment (night) or bright environment (daylight). If the dark environment is generated, then there should be a number of additional point lights included as well, which should be there to enrich the experience and create more realistic surroundings.

3.1.3. Practical Mode

The exploration part of the game is to get players familiar with theoretical aspects of a topic and give them the freedom to explore maps in any way they want to, without forcing them to follow a specific path. The idea in the practical mode is to have a playground where students can test their practical skills. Every course should have different types of playgrounds, as

[1] Infinite, in this context, refers to a really large number, which will make players think that there is indeed an infinite number of content variations.

not every course can be based on the same set of features. The goal in this part is to understand the practical aspects of a certain topic, therefore, the player's objective in the practical mode of the game for the programming course is to program a virtual robot to avoid obstacles and collect a disk which carries a piece of information. That information is a part of the task which was designed previously through the web application. In other words, it represents a task which the educator specified in the curriculum.

Once the players are familiar with a topic presented in the theoretical mode they should test their knowledge in the practical mode. The process of programming the robot should be simple, given that the target platform is mobile, so there should be an UI interface which will simplify the process of writing the code. As a programming language, the best idea is to go with script languages as they do not require compilers and can be executed on mobile platforms. In order to create a rich user experience, players should be able to write valid code, observe that code in action and get error reports if their code contains any logical or syntax errors, just like in any IDE. To reach that, it is necessary to process the code provided by players. This is very important as the code provided by players has to be valid since it needs to be executed on the virtual robot.

User Interface

A user interface is a very important component of every game, especially in mobile games as space and input are very specific and different from other platforms. One of the most challenging tasks is to come up with an UI component which is going to simplify the process of coding, make it engaging and helpful enough for beginners, and motivate them to continue improving their coding skills. Users should be able to type the code, see line numbers and errors and navigate between console, working area and task description.

3.1.4. Procedural Sound and Content Generation

Even though the main focus of this master's thesis is to use the procedural generation algorithms for generating the terrains and maps, it is also possible to use PCG algorithms for generating objects which are not maps. Some algorithms which are not specifically designed for sound generation can be used for this purpose, but for better results it might be a good idea to use sound generation algorithms which are generating sine tones at certain frequencies. However, it is important to bare in mind the time and effort necessary to invest in order to successfully generate sound. Since there are no general algorithms for sound generation, the proposed solution should not go out of scope of the project.

Procedural generation algorithms can be also used for generating the game content like trees, roads, building, units, etc. Since the setting of this game is in the space, generating roads and buildings might not fit in well. The generation of friendly and enemy units usually requires massive worlds and complex animation mechanics, which again do not fit well in a mobile game. It is important to keep in mind that mobile devices have limited processing power, therefore complex calculations can drop the FPS down. The requirement for content generation in the project is fast production of patter structures which can be used for tree, plant or forest generation.

3.2. Non Functional Requirements

One of the main concerns with educational games is to make a game which is going to be interesting enough, so that it motivates players to keep playing the game. Educational games should combine engaging elements with aspects of instructional design in order to provide best possible experience (Amory, Naicker, Vincent, & Adams, 1999). Visualization, manipulation of objects, and competition are very important for education, as they enhance problem-solving and help with better understanding certain topics. (Betz, 1995; Leutner, 1993; Neal, 1990). Therefore, it is necessary to ensure that all these components are integrated in a way that they provide a rich learning

experience. Most important non-functional requirements in this project are:

- **Usability** - since the main platform for this project is mobile, it is necessary to construct a system which is going to be effective and easy to use. For the exploration part of the game, where a player has to move around the map, it is important to ensure that controls are adapted to mobile devices and that players can control the main character and interact with enemies and other objects with ease. Since there are many different mobile resolutions, the entire game should be responsive and adapt to different screen sizes. In the programming course, players have to type the code, thus it is needed to make that process as simple as possible. Also, the UI should be designed in a way that it supports most features of a regular IDE available on the desktop platform, so that players can use the custom UI just like if they were using an IDE on a PC.

- **Modularity and Scalability** - for the purpose of this theses, the goal is to create at least one course and design number of tasks for it. Creating more courses at the moment would be out of the scope of the project, however, the entire system should provide an interface for creating additional courses and assigning a number of tasks to every course. Core game features should work as modules and it should be possible to easily integrate new modules which can be used for different courses. Also, it should be possible to add new game types, enemy units and gear items to the existing structure.

- **Configurability** - one of the greatest advantages of procedural generation is that the content can be highly configurable. That means, it is possible to change parameters of the algorithm which is generating maps and produce maps with different dimensions, a density of the walkable and non-walkable area, different heights of mountains, etc. When it comes to asset generation, it should be possible to specify the kind of content which will be generated, quantity, shape and size. Finally, in procedural sound generation, one should specify what kind of music should be generated and when. Also, the content should be fully configurable by educators, as they should be able to specify how

the content should be presented, in which order, difficulty, and the type of rewards players will get for every task.

- **Appealing Visuals** - this is a very important aspect of application development, especially when it comes to making a first impression, higher satisfaction and perception (Collinge, 2017; Laja, 2017). There is a number of different art styles for video games and selection of art style is very significant. At the very beginning, developers have to choose whether their game is going to be 2D or 3D.
  If there are no specific requirements, most developers make this decision based on whether they have skilled graphic designers or 3d modelers on their team. This is a very important decision which can significantly determine further project flow. It does not only affect the art, it also affects the programming and game design as well. 3D for art mostly means that instead of using sprites, images and image editors, one has to create 3d models by using 3d modeling tools.

3.3. Conceptual Architecture

In order to make the game as good as possible, it is necessary to take into consideration existing solutions, methodologies, and approaches, so that good decisions can be made. It is important to design a game and evaluate it with the target audience so they confirm that the mechanics implemented in the game are interesting to them. It is important that educational content does not spoil the game and make it boring after it is integrated. The idea is to design an educational game so that players can learning without being aware of that. If education is too emphasized, then the players are more likely to stop playing the game at some point, therefore everything has to be balanced. The game has to be interesting and appealing, otherwise, the target audience will not be interested in playing the game (Klopfer, 2008).

Figure 3.1 demonstrates how conceptual architecture should look like. It is important to keep in mind that this is an educational video game designed for high-school students and first-year university students. This target group has very specific requirements as technology and how they interact with

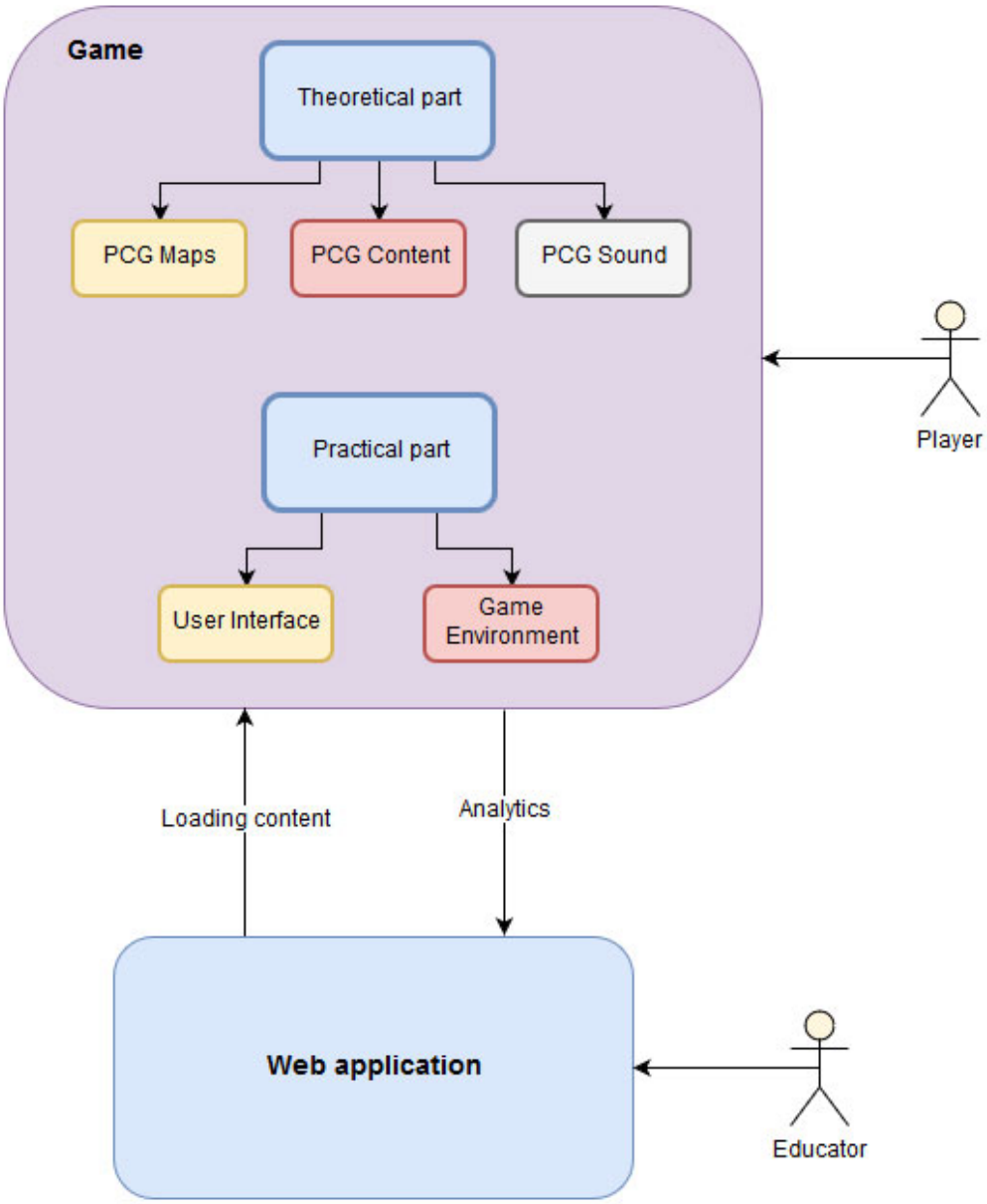


Figure 3.1.: Conceptual architecture of the project.

it, has special meaning for them (Davies & Eynon, 2013). It is important that every new course follows the same structure, that is, contains both theoretical and practical educational content. PCG techniques and interactive playground serve to enhance players' engagement and motivate them to learn more about certain topics.

3.4. Technology Decisions

This section discusses technology decisions made based on previously discussed approaches and methodologies which are important for design and implementation of the game. Since one of the goals in the project is to design a game for mobile devices it is important to understand limitations and advantages of mobile platforms. All that influences design and concept of the game. When it comes to selecting a proper technology, one needs to identify what technology is the best match for the given set of limitations and requirements.

3.4.1. Game Engines

When it comes to development of a game, it is possible to either build a game from scratch or to use one of game engines which are available on the market. Most popular game engines are Unity3d[2], Unreal Engine[3], CryEngine[4], Gamemaker[5], and Construct[6]. Gamemaker and Construct do not require strong programming background in order to make a game, as they both provide powerful editors with a number of features, so that people who are not technically skilled can still make games. However, working on more advanced features is not supported in the editor and it is necessary to write the code. Unity3d and Unreal are game engines which provide a very powerful APIs. There is a number of features which help developers

[2]https://unity3d.com
[3]https://www.unrealengine.com
[4]https://www.cryengine.com
[5]https://www.yoyogames.com/gamemaker
[6]https://www.scirra.com/construct2

to quickly build game prototypes in both engines. Both engines, provide the one-click-deployment tool which means that it is easy to deploy a game to any game platform, from mobile devices to desktops and consoles. Unity3d has a large community and there are many tutorials, instructions and documentation available online. Also, the authors of this Master's thesis has extensive experience in this engine, therefore, it is decided to build the game with Unity3d.

3.4.2. Procedural Content Generation

This sections discusses different procedural generation decisions. It covers the structure of the theoretical and practical modes and offers the implementation steps and strategy.

Procedural map generation

Cellular Automata is one of PCG algorithms which is a good candidate for the previously explained set of requirements. The algorithm is mostly used for creating cave-like structures. A result of Cellular Automata is a matrix with zeros and ones, where zeros could represent walkable map cells, while ones could depict non-walkable cells. In order to ensure that every sector of the walkable area is reachable, it is necessary to create corridors. The corridors keep all the map parts connected and allow the players to access and explore every segment of it. Once walkable and non-walkable areas are separated, it is important to add walls or obstacles over the non-walkable areas to keep players away from stepping onto them. This can be achieved by placing elements on non-walkable areas, however, that would result in creating a number of objects, which is not a good idea, as more objects on a scene mean more draw calls, which can cause FPS to drop down. In order to create large and complex terrains and use them on the mobile platform while also ensuring high FPS, it is mandatory to create a mesh. That is something which has to be implemented no matter what algorithm is going to be used. Figure 3.2 shows an output example of Cellular Automata.

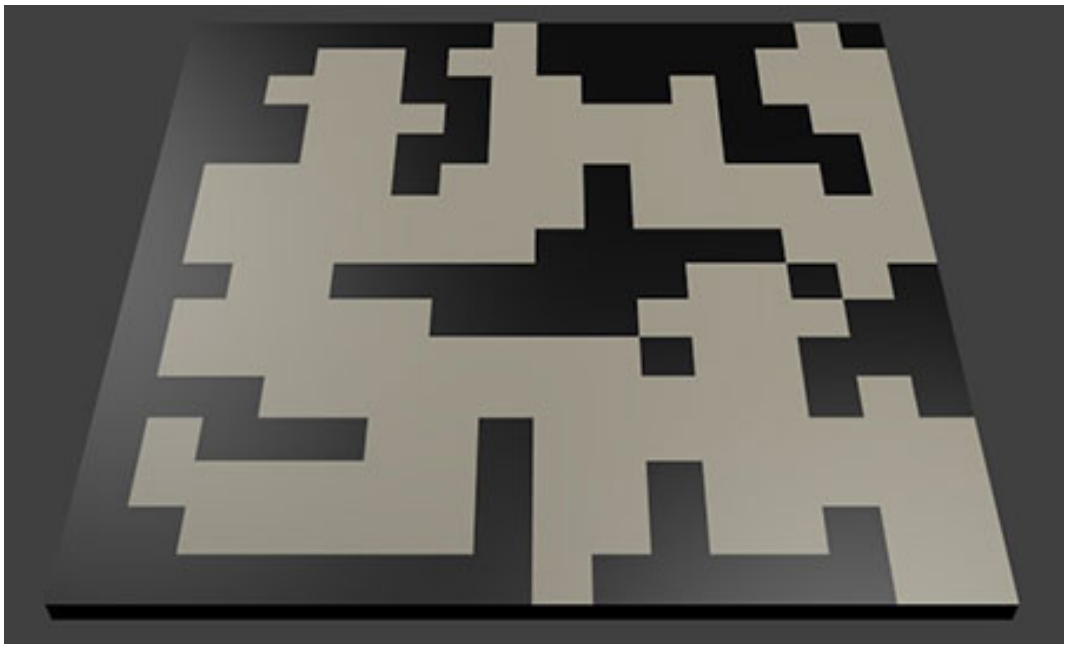


Figure 3.2.: An output example of Cellular Automata.

The best solution for non-walkable areas is to use height maps to elevate all map cells which have the value of one. If those map cells are elevated for a constant value, then that can create a strange effect as the generated map would look like a maze. In order to solve that issue a noise function will be used. Perlin noise is one of the most popular noise functions, and is very useful for generating terrain-like structures with hills and mountains. With Pelin noise, it is assured that non-walkable area will now have more realistic topology. Figure 3.3 shows an example of using noise to elevate specific parts of the map, in this case only non-walkable tiles. After all these steps, the result is a map with both walkable and non-walkable areas, where the non-walkable area has a natural-looking terrain structure. The next step is to do something with the walkable area. In order to utilize the generated map, it is necessary to add a player, pickups, enemies and other elements defined in the game design. In order to accomplish that, it is needed to detect walkable regions of a specific size so that items and units can be positioned. For that, it is necessary to implement an algorithm for detecting regions. Figure 3.4 shows the proposed structure of the theoretical game

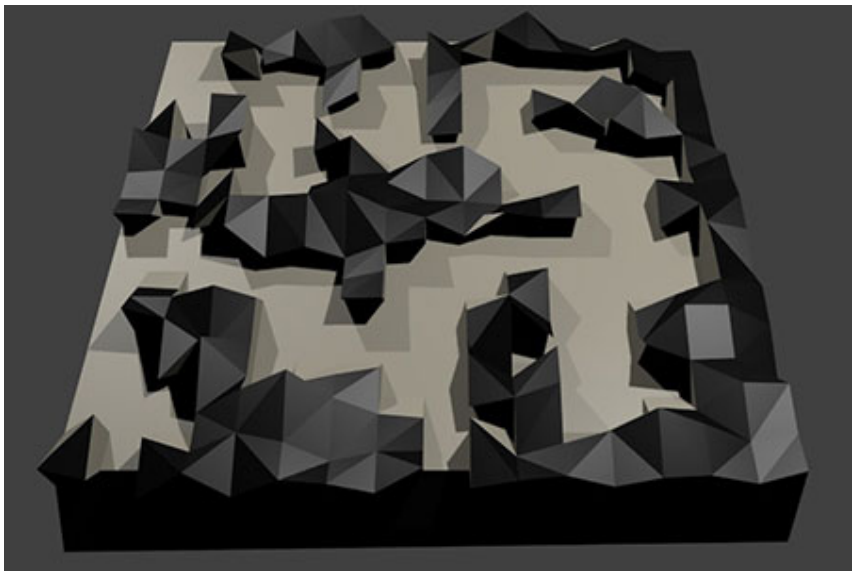


Figure 3.3.: An output example of noise functions used to elevate specific cells and create natural-looking terrains.

part, while Figure 3.5 shows how everything is supposed to look like at the end when Cellular Automata is combined with Perlin noise.

Sound Generation

The method for generating music which will be used in this project is based on producing compositions made out of basic tones. It takes a set of tones (C, D, E, F, G, A, B), as an input and generate endless music. There will be tones for several instruments provided (piano, violin and drums) but this principle can easily support more instruments. In order to provide infinite loops, it is required to call the same function again at the end of the sequence. First, it is necessary to create a matrix with certain dimensions (width and height). The next parameter could be fill percentage, which determines the ”density” of sounds. And finally, for every generated matrix, a seed could be generated as well, just like in the map generation so that those particular sequences could be reproduced. Failing to do so could lead to very strange results. For example, it is not a good idea to have tense music

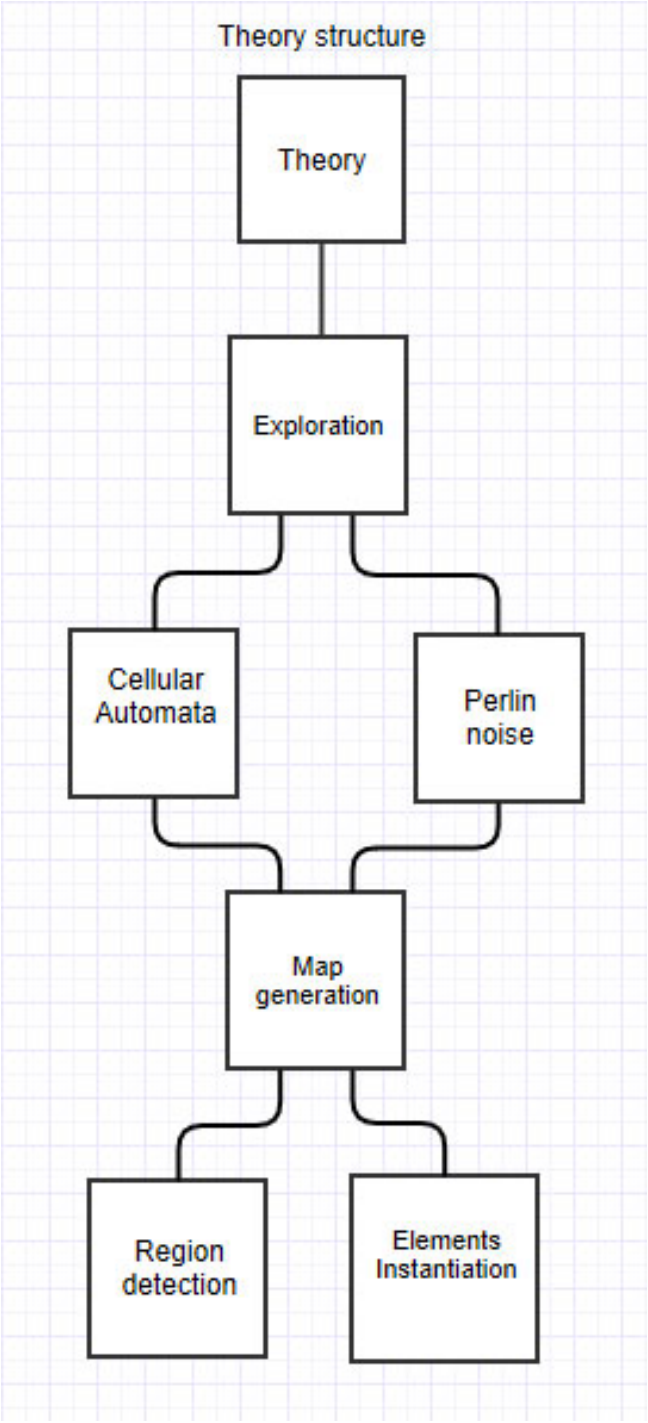


Figure 3.4.: The structure which shows how the theoretical game part should look like.

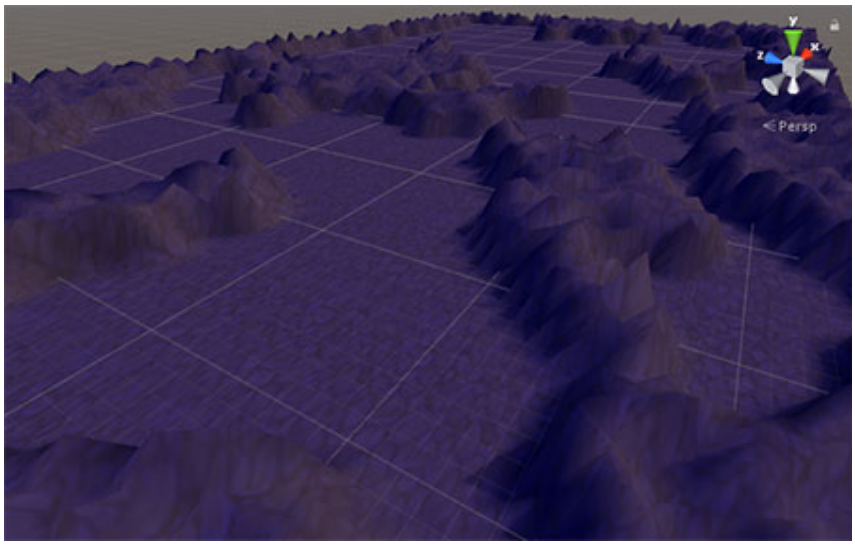


Figure 3.5.: Final output of combining Cellular Automata with Perlin noise.

when a player is not in a combat, has to solve a puzzle, or is performing something in the game which requires a lot of concentration.

Procedural Generation of Assets

Assets which have to be generated and placed in different levels are trees and forests. Procedural tree generation can be performed by using L-system algorithm. The algorithm will be implemented in a way that it supports two stopping criteria. It will either use number of iterations or length of branches. The number of iterations specifies the number of levels the tree is going to have. If the length of branches is used as the stopping criteria, then it requires that every time a new tree level is created, the length of branches for the next level reduces by specified value. The number of iterations in this case depends on the initial length of branches, as well as, the reduction parameter. If the reduction parameter has greater value, it will result in creating a tree with less levels, which also means it will take less iterations. In most cases this is acceptable, but if it is necessary to have really complex trees with many branch levels, it is a good idea to keep values of reduction parameter very small.

3.4.3. Playgrounds

The practical mode of every course has a playground where players can practice what they have previously learned. This section discusses decisions made for the custom UI and the environment for the programming course. As previously discussed, the game engine which is going to be used to create the game with is Unity3D. Unity3D[7] has UI functionalities which allow developers to easily create any kind of form and collect the user's input. One can add input fields, text, images, buttons, scrollbars, and panels, without any programming involved, however, Unity3d UI system behaves differently on other platforms, and if something works well on the desktop platform does not mean it will work in the same way on the mobile platforms. One of the issues on the mobile platform is that the input fields behave strangely on Android as the entire content is copied into an improvised input field and virtual keyboard blocks the entire screen after popping up. The problem is demonstrated in Figure 3.6. The same behavior occurs on iOS devices, however after ticking "Hide mobile input" box in input field settings, the problem disappears on iOS, but that still does not affect Android. Another problem is that virtual keyboard is still covering a large portion of the screen so it is necessary to come up with an effective solution.

This was one of the reasons for creating the custom UI system. The more important reason was providing rich programming experience on a mobile device. This means that it will be needed to create fully functional virtual keyboard and input fields, therefore the UI should consist of the following components:

- Virtual Keyboard
- Drag and drop code blocks
- Code elements - input fields
- Tabs and panels

The virtual keyboard needs to have all buttons which default mobile keyboard provides and support default functionalities for alphanumeric characters, symbols and special commands like enter, space, caps lock and

[7]https://unity3d.com

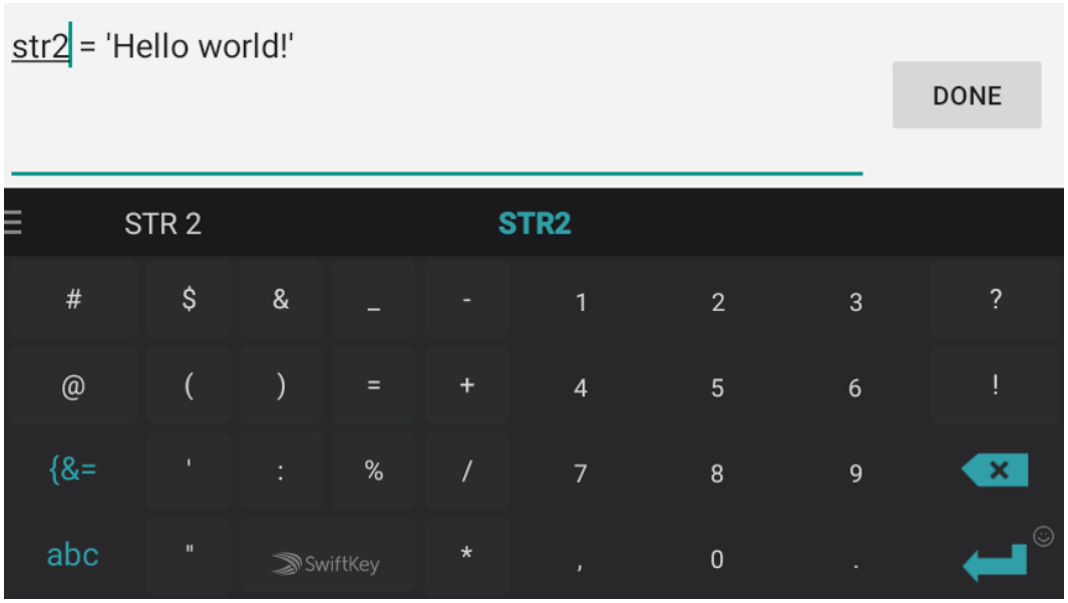


Figure 3.6.: Screenshot taken from the game after running the game on Android.

backspace. In order to include all the keys, it is needed to add the functionality for switching from letters to numeric characters and symbols. Since the target platform is mobile, it is also necessary to add the functionalities which will allow players to position a pointer between the characters in text.

That is why it is necessary to create code blocks, which are used as shortcuts for writing if statements, for loops, variable declaration, print outputs, etc. This is something which is very helpful for beginners as it will reduce the time necessary for writing the code and avoid potential syntax mistakes. In order to make this working, drag-and-drop functionalities have to be implemented. Another thing which has to be taken into consideration is nesting. That slightly complicates the structure, as it is important to specify which code blocks can contain more elements and which can not. Some components like if statement and for loop need to have nesting functionalities, as usually, there is more code inside these code elements, while declaring variables or function calls can not nest more elements.

Sorting of elements is something which also has to be taken into account.

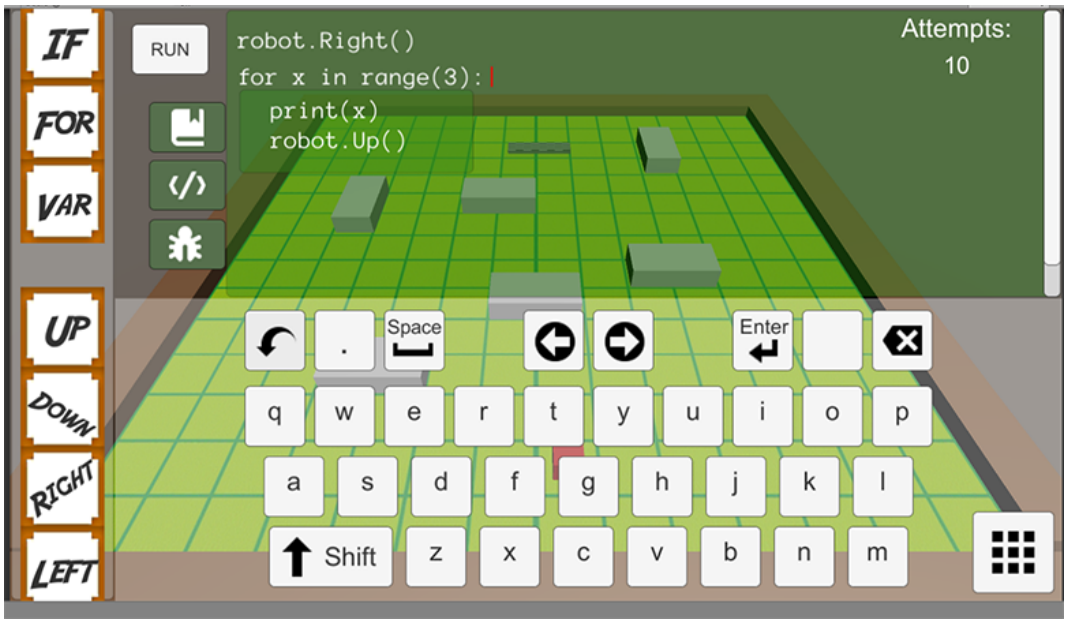


Figure 3.7.: Screenshot taken from the game which represents the early prototype of code components.

Once a component is dropped, it does not mean that it will remain where it is, as it should be possible to rearrange code elements and change their order. Therefore, sorting functionality is very important as it can allow sorting of elements inside an element, which supports nesting (like for loop), as well as, all the other elements (at the top of the hierarchy) which are not contained inside any other element. There is one more important feature which is based on sorting, and that is moving elements from one nested element to another, and to the root. Even though it looks like something which is very confusing, there is an example presented in Figure 3.7 which serves to describe the structure and relations between elements.

As could be seen from Figure 3.7, elements on the left-hand side are code blocks, which one can drag and drop on the main content area, which is located on the right side. Blocks on the left side should be fixed and dragging any of them will produce a copy which can be dropped onto any drop zone. This allows players to create as many elements of one "type" as they like. Once that is done, it is possible to drag and drop code elements

generated from code blocks and rearrange them in the desired order. As already mentioned, it should be possible to drag an element to and from any element which can nest other elements. Those areas must be visually emphasized, that is, painted in a different color so that players can easily notice them. The red line represents where the pointer is currently located and if any key on the virtual keyboard is pressed, it should make an effect on the current position of pointer whether it is adding a new character or removing existing ones. There are two arrow buttons on the virtual keyboard which have the ability to move the pointer to the left or right of the current position. When pointer moved all the way to the left until the last character, clicking on the left arrow button should keep the point in same position. The same should happen when the pointer is at the far right character. This can be further improved by moving pointer to one code block before or after the current code element, but that would require additional checking (it is mandatory to check if there are some elements around the current one, otherwise this action can cause errors), so due to simplicity, the boundaries of pointer can be kept within a code element.

Finally, once new code components are added, it is useful to have code line numbers left to the code, just like in any IDE. The code lines should be updated every time new code blocks are added or some code elements are removed. All previously described components form the custom mobile IDE for writing then code. The UI work along with the environment, as these two components form a playground. The environment consists of a virtual robot, a disk and obstacles, where the goal is to program the robot to reach the disk by avoid the obstacles. The UI system is used to enhance the process of programming the robot, that is why both components are important for the practical mode. Figure 3.8 shows how the structure of the practical mode for the programming course will look like. This is where custom UI - IDE system is utilized. Instead of writing the code, the players can use the specified code blocks, where every code block contains a code snippet with predefined functionality. For example, instead of typing a for-loop, one can just drag and drop the for-loop code block and the for-loop code will be automatically generated. Then, it is easy to modify a number of iterations and nest more code inside the loop. However, it should be possible to type the code as well, so using code blocks is not mandatory, but can speed up the process of development and avoid potential typing mistakes.

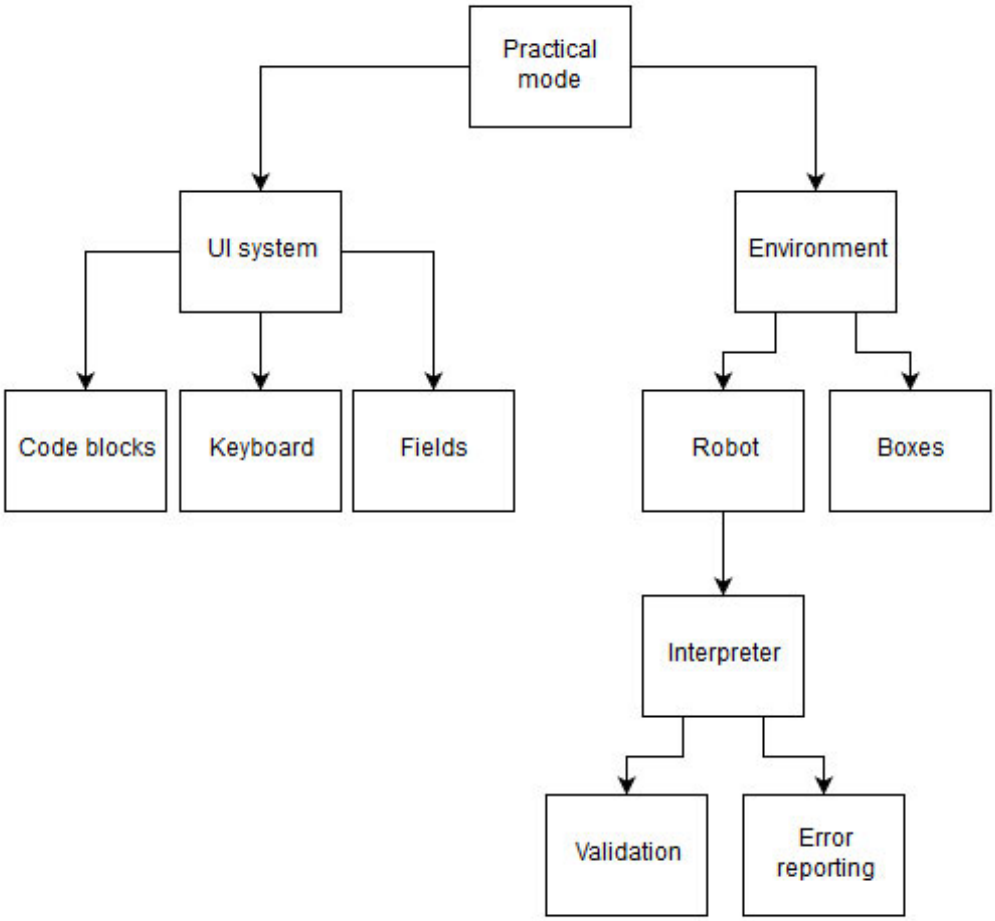


Figure 3.8.: Structure of the practical game part for the programming course.

It is decided that the programming language which will be used for teaching is Python. Many schools teach Python as a first language since it is a programming language with a very simple structure and syntax. It is easier for students to start with Python than, for example, with C or C++ mostly because students do not have to worry about more advanced concepts like the memory allocation and pointers (Elkner, 2000; Grandell, Peltomäki, Back, & Salakoski, 2006). That is why many schools teach Python as the first programming language. Python is used in many areas, from web development to building desktop applications (Hauser, 2017). IronPython[8] is a .NET implementation of Python programming language. It allows any .NET language to use Python code and it can help with interpreting and checking the validity of the code in this project. IronPython is not only providing validity of the code, it also gives error messages with more details about the problem. This is very useful for players as they are able to see the line number and the command which caused the error (Foord & Muirhead, 2009; Mueller, 2010). With all that, one can easily fix all the errors just like if he or she was using any IDE on a desktop computer. Technical aspects of Python interpreter and how it works will be explained in greater detail in Chapter 4.

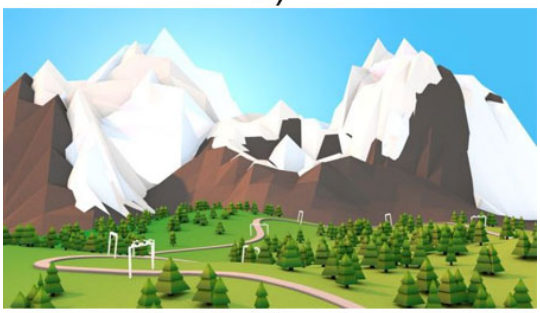
3.4.4. Other Decisions and Final Structure

Visuals are very important for attracting attention of players. Some popular 3D art styles for video games are Voxel, Low poly, Cartoon and Realistic. Figure 3.9 shows how different 3D art styles look like (Kerlow, 2004). Low poly art style is the best match for this project, as it is not too complicated and time-consuming to create, and it fits well with the gameplay and the game idea. Another reason are performances, as fewer vertices and triangles there are, the better performance will be.

Figure 3.10 shows implementation structure for the programming course.

[8] http://ironpython.net/

a)



b)

c)

d)

Figure 3.9.: These are the examples of different art styles. Voxel (a) (Oré, 2017), Low poly (b) (DesignContest, 2017), Cartoon (c) (PopCapGames, 2017) and Realistic (d) (EBGames, 2017).

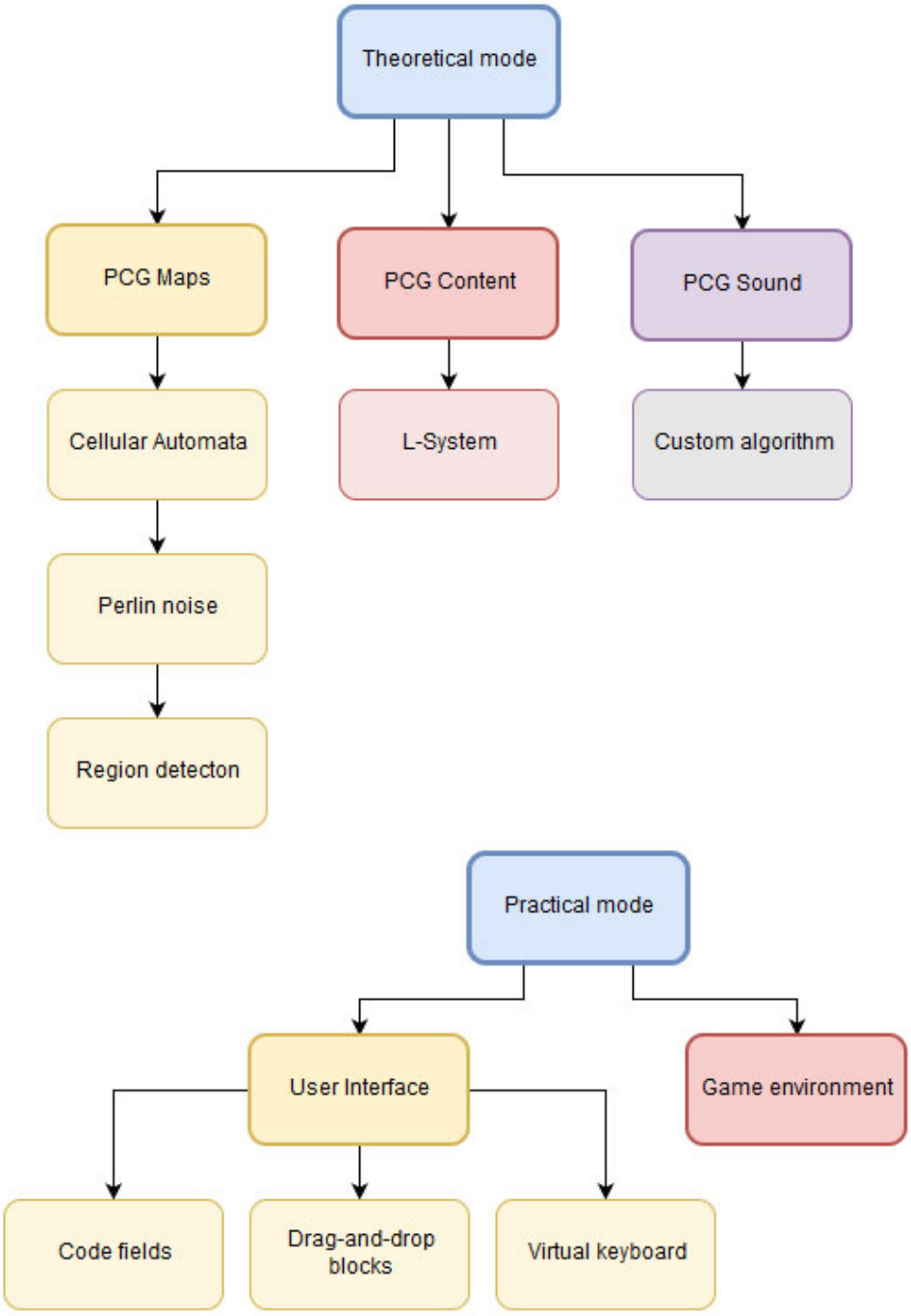


Figure 3.10.: Structure of the programming course.

3.5. Summary

Educational content in the ”sCool” is separated into two components: theory and practice. The web application provides educators an interface for creating lessons and tasks which are later obtained from the server and shown in the game. In the theoretical mode, players have to explore the map, defeat enemies and collect disks. This part is based on exploration where PCG algorithms for map generation can enhance exploration experience. The playground is a place where players have to apply the knowledge they previously acquired in the theoretical mode. In the programming course, the goal is to program the robot to avoid obstacles and reach the disk. Custom IDE, needed for the practical mode of programming course, is another functional requirement necessary to support interaction and speed up the process of forming the result. Non-functional requirements in the project are usability, modularity and scalability, configurability and appealing visuals. There are different game engines which can be used to create the game, but the Unity3D is the most popular game engine and simple to start working with.

4. Implementation Details and Showcase Scenario

This chapter covers different implementation aspects of the project. The first discussion point is the implementation flow and the analyses of issues which were faced in the early development stage. The implementation of exploration and coding game parts is based on the concepts presented in Chapter 3. PCG section analyzes implementation of procedural generation of maps, sounds and content, as well as, how PCG algorithms such as Cellular Automata and Perlin noise help making a natural-looking terrains and maps. The last discussion point in PCG implementation is the procedural asset generation and the use of the L-system algorithm for generation of trees. The drag-and-drop blocks and custom UI interface are the two main components of the playground mode in the programming course. The final section of this chapter presents an overview of the entire game, and shows how all parts of the game are put together.

4.1. Procedural Content Generation

The theoretical mode of the game is based on exploration and collection of artifacts which when collected reveal valuable information for players. The players have to explore the entire map, collect pickups, defeat enemies, read a provided lesson, and answer a quiz-based questions. Based on all of that, they will get a number of energy points which are very valuable at the end of a level. If the wrong answer is provided in the quiz, then the players might get more attempts if they performed well in the game. If that is not the case, they will have to repeat the level from the beginning. In this way, they should try to perform as good as possible in the exploration

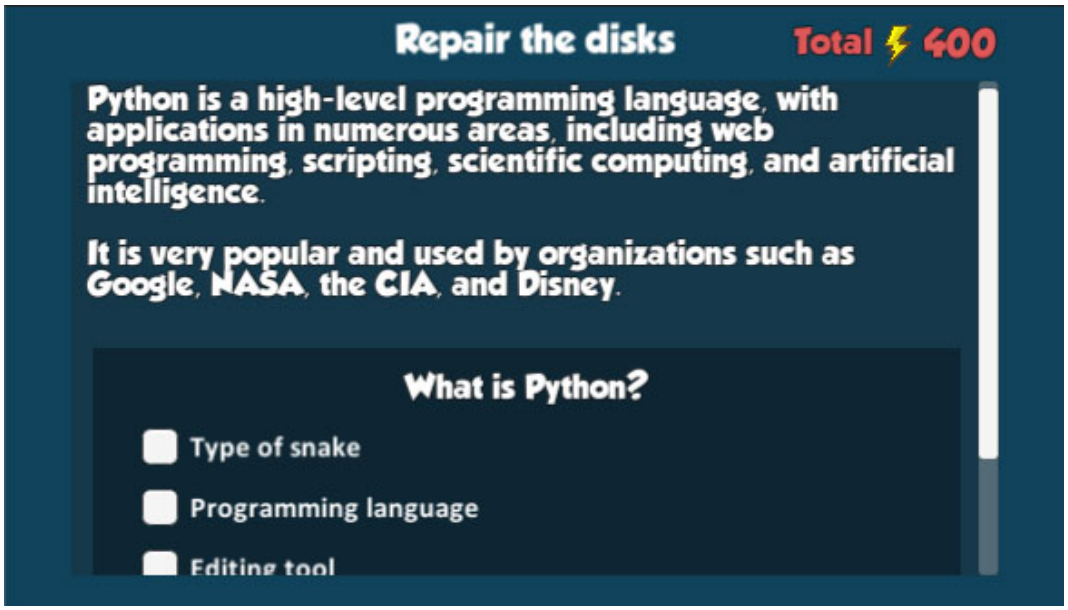


Figure 4.1.: A question which a player has to answer, when all the disks are collected, in order to move on.

part before they finish everything and get to the part where they are faced with questions. The theoretical content consists of a description of a specific educational matter with one theoretical question. That theoretical question has a number of offered answers, where only one is correct. Figure 4.1 illustrates an example of a question which players get once all the disks are collected in the theoretical mode.

The map generation is important for the exploration part as it provides exploration experience. The two essential procedural generation algorithms used for map generation are Cellular Automata and Perlin noise.

4.1.1. Cellular Automata

The Cellular Automata algorithm was used for generating cave-like structures and distinguishing walkable from non-walkable areas. Figure 4.2 shows the implementation of Cellular automata in this project. Red cells represent walls and gray cells are the walkable areas. The following rules

were used for determining whether a cell should be a wall or an open space:

1. If a cell was a wall (value of 1) and 4 or more of its neighbors were walls, then it remained unchanged.
2. If a cell was not a wall (value of 0) and 5 or more of its neighbors were walls, then it became a wall.

In order to create a "noiseless" map with a natural cave-like shape, it was necessary to repeat the entire process 5 times. It was concluded that the best results were obtained if the percentage of walls is in the range of 45% - 49%. Second, it was needed to create a two-dimensional array which is going to represent a map. There are two variables, the width and height which are created and used along with an array to specify the dimensions of the map. If random values are passed as the parameters, then the generated maps will have different sizes. Next, it was necessary to generate a seed value so that the same map can be regenerated again if needed. Also, on each run, a different seed was generated and passed to the map generation algorithm which gives a different map as the output. One of the parameters that was passed along the seed is a fill percentage, which determines the ratio of zeros and ones. Reachability of the every part of the map is ensured with the solution by Lague (2017), which is based on creating the corridors between the sectors on a map.

Since this is a 3D game, the map structure from Figure 4.2 was changed and adapted to the 3D environment. The simplest way to achieve that was to elevate wall areas for a specific value. In other words, the z values of wall cells were set to any scalar value, which determined the height of walls. However, changing the z value was not enough, as it was also necessary to rotate the entire map for 90 degrees over the x-axis. It is possible to rotate in both directions, however, that could lead to different orientation as in order to elevate wall areas it is needed to set the negative value for height. The environment was rotate clockwise so that positive values for elevation could be used. Figure 4.3 shows how 3D map looks like for different height values. In order to draw a map on a screen it was necessary to create a mesh. There were two requirements for mesh generation. First, it was necessary to position vertices on corresponding places and then, apply triangulation. There are two ways of forming triangles, discrete and continuous. Both of

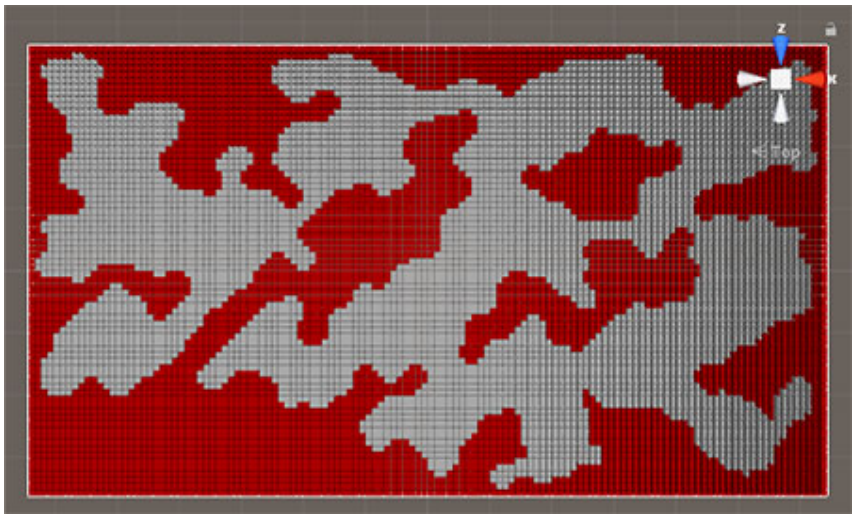


Figure 4.2.: Screenshot from the game showing the Cellular Automata implementation in the Unity3D engine.

them were implemented in order to conclude which approach is giving better results. In continuous triangulation adjacent faces shared the same vertices while in discrete triangulation every face had its own set of vertices. An advantage of the continuous triangulation was that there were fewer vertices in total, but one of disadvantages was that it was not possible to have steep transitions between adjacent faces. Also, this affected the coloring of faces, as there were some blurry transitions if two adjacent faces are colored in two very different colors. However, since performance was very important as the game is designed for mobile devices, then it was definitely a good idea to go with this approach. Event though the discrete triangulation provided a higher level of detail, the main drawback was that there were twice as many vertices as in the continuous triangulation. On the other hand, discrete triangulation provided a high level of detail and flexibility with creating different shapes of meshes. Maintaining high Frames per second (FPS) rate was more important than the level of detail, therefore, the discrete triangulation was discarded and continuous triangulation was used for building different meshes.

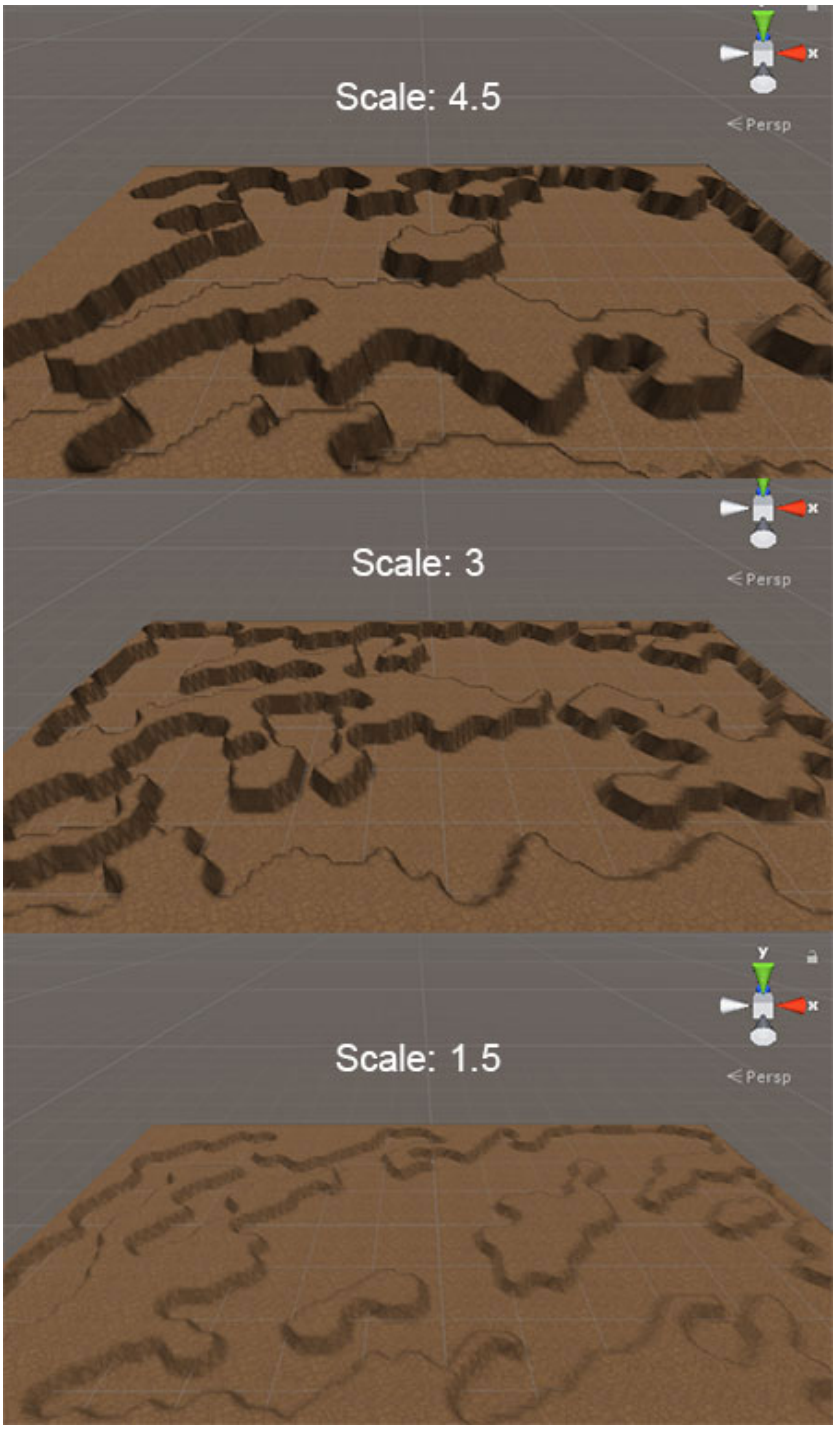


Figure 4.3.: Screenshot from the game showing the 3D transformation of Cellular Automata with different elevation values.

4.1.2. Perlin Noise

The generated map from Figure 4.3 already satisfies the criteria, as a 2D structure is converted into 3D and players can move around the walkable areas only. However, in order to make the walls look more realistic, that is, to have a terrain-like shape, it is needed to use height maps. For that purpose it was decided to use Perlin noise. There were three main parameters in implementation of Perlin noise and those parameters are octaves, lacunarity, and persistence. With these parameters, it was possible to modify the amplitude and frequency, which basically determine the height and distribution of terrain components. The lacunarity parameter was very important for determining the frequency for every octave. Octaves represented terrain components (base frequencies) which were put together to form the final terrain shape. A frequency for every octave was calculated as lacunarity to the power of i, where i started at 0 and was incremented before calculating the frequency of every octave. It is important to note that the lacunarity value is a scalar which can be modified to generate different outputs. The process of calculating the amplitudes was the same, and it was only necessary to replace the lacunarity with persistence. This provided natural looking shapes of terrains. Figure 4.4 shows how different octave, lacunarity, persistence values affected the map generation.

4.1.3. PCG Sound

Some of already existing sound generation methods are based on the sound processing and changing the source files of music sequences or creating files from scratch by generating frequencies. The implementation of the procedural sound generation in this project is different from all standard approaches. The solution implemented in the game is based on instantiating tones of different instruments and changing their parameters like pitch and volume levels, in order to create endless and unique musical compositions.

The core components in the map generation with Cellular Automata were walkable cells and wall cells. In the sound generation, the core components are music notes of instruments and instruments themselves. First, the algorithm generated a matrix with specific width and height and randomly

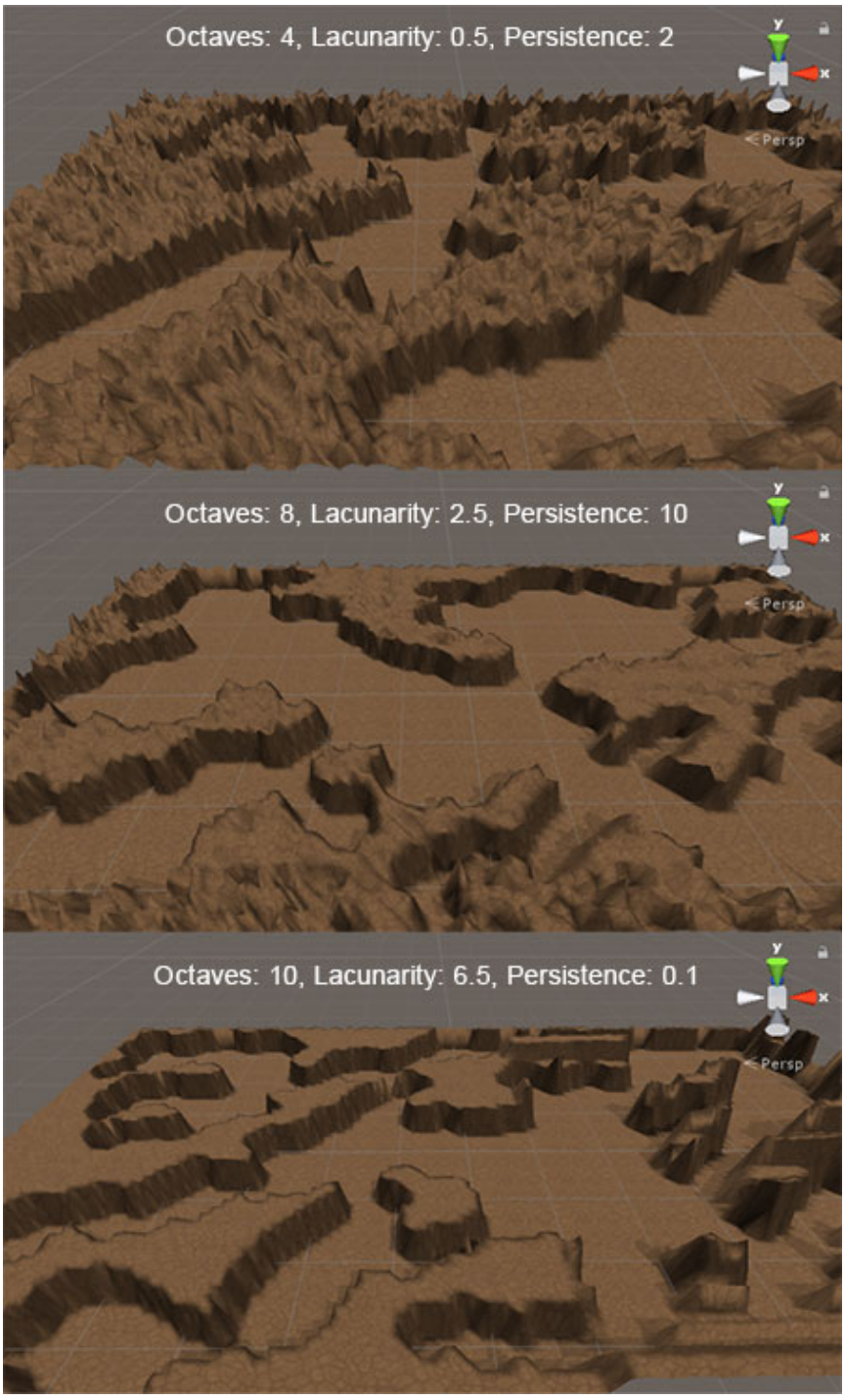


Figure 4.4.: Screenshot from the game showing how different parameter valued affected map generation. It is important to note that all three variations used the same height scale.

positioned notes. In the map generation, zeros and ones were used to make a difference between walkable cells and wall cells. In the sound generation, there was the same intuition, ones represent note cells, which is the moment when a note of an instrument is played, while zeros, in this case, represent pauses. One of core components is a set of note with different pitch levels distributed along rows, that is, the x-axis of the sound map. A different position in columns (the y-axis) determines volume level, so volume changes based on where exactly a note is placed. If it is located below middle row, then the note volumes down, otherwise it volumes up.

There were two essential scripts for the implementation, *InstrumentManager* and *MatrixGeneration*. *InstrumentManager* is a script which has a list of all instruments and is in charge of turning on and off certain instruments. In other words, this script is the one which is deciding what kind of music will be generated, so that it can be changed based on the environment and the activities in a game. In this project, there is one music type when players are exploring, the second type when players are in the combat, and the third when they are solving puzzles. Procedural sound generation is implemented in the way that the music adapts to player's behavior in the game.

The initial approach to the sound generation was to create some prefabs for different music moods, so that it is possible to activate a specific music theme which fits the game in that moment. This approach required to come up with several music sets of different instruments and activate them when it was needed. The second approach was to avoid using prefabs and to manually specify which instruments should be active. This approach was more flexible as any instruments could be combined and activated based on the player's action. However, this approach required more complex logic in the background, as it was not enough to simply replace one prefab (set of instruments) with another (like in the first approach). With this approach, it was necessary to turn on and off every instrument individually, otherwise, there could be overlapping between the sounds. Still, the second approach provided more variations and it was decided to use it in further implementation. Once the process of switching between instruments was ready, the next step was to implement a musical instrument itself, that is, matrix generation and distribution of notes for every instrument. Just like in the map generation, it was needed to generate a seed for creating a sound map with provided distribution.

Figure 4.5 shows how procedural sound generation script looks like and what parameters it takes. It can be noticed that a very similar intuition was used like in the map generation script and that the greatest difference is a list of notes. A number of elements in the list can be manually defined, but after many trials and experiments it was decided to go with seven elements, based on the note chromatic scale[1]. The rows of the matrix determine what note is going to be used, which basically defines the pitch level, while the position of that note in a column gives different volume value. All sound files used in the sound generation where obtained from Orchestra (2017). The sound generation process in the Unity3d engine is shown in Figure 4.6. For debugging purposes and in order to better understand how music is generated, every instrument was assigned with different color, where colors show which notes belong to which instruments. The idea was to play all the notes from one column at the time. Then the algorithm takes the notes from the next column and so on until the end of the matrix is reached. There is a pause of 0,5 seconds before moving on to the next column. When the end of the matrix is reached then the new matrix is generated using a different seed value, which means that on the next matrix iteration the music is going to be different. It is important to note that there is a shuffling process of notes prior to matrix generation, which is necessary to ensure randomness every time a new matrix is generated. In this way, it is ensured that the generated music is totally different and endless. Figure 4.7 shows a list of all instruments currently available to be used in the game.

An important aspect of the used method is that it was possible to generate different music for the main menu, the theoretical mode, and the practical mode. The instruments used in the sound generation, for all menu and level selection scenes, are cymbals and bells. With these two instruments, it was possible to create a music which is mysterious, fantasy and dramatic. The idea was to create a heroic emotion at the very beginning, as an introduction to the game, something which is going to motivate players to engage with the game.

In the theoretical mode, the music changes and different music is generated using tempo drums, bells, and shakers. This shows the full potential of the algorithm as it is very simple to generate different music with changing a

[1]https://en.wikipedia.org/wiki/Chromatic_scale

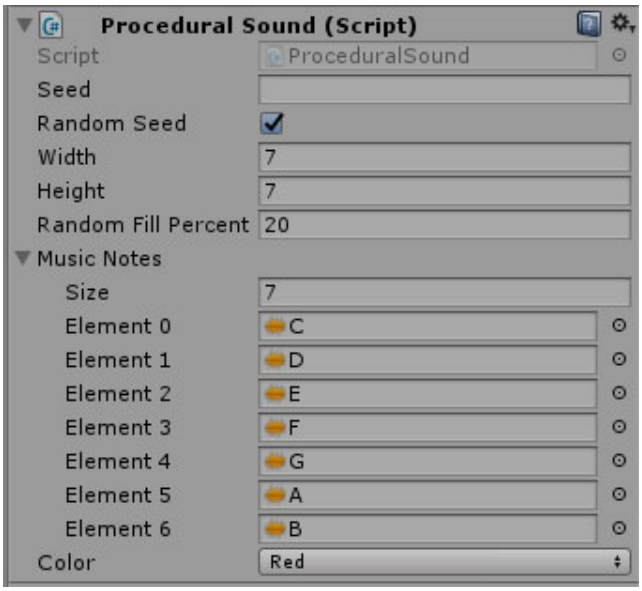


Figure 4.5.: Screenshot from the game showing the procedural sound generation script and parameters it takes.

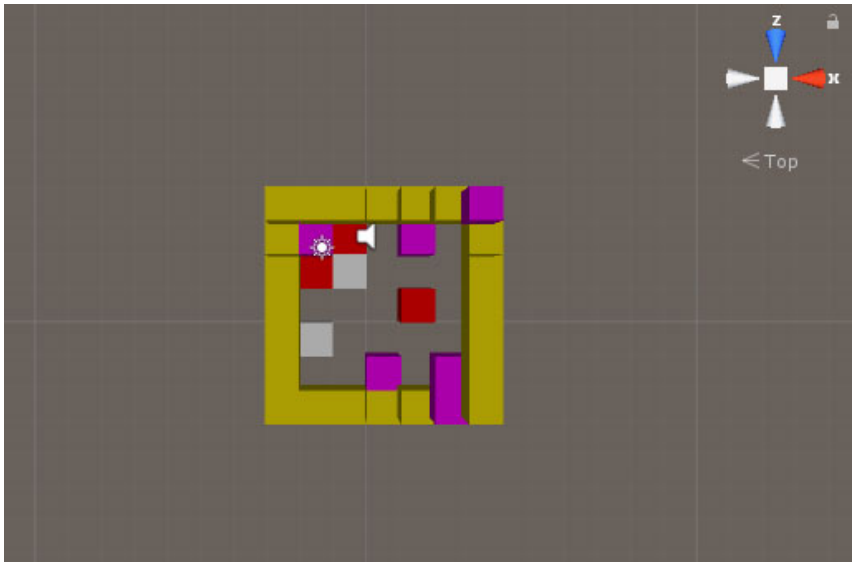


Figure 4.6.: Screenshot from the game showing the process of music generation, where one color represents one instrument.

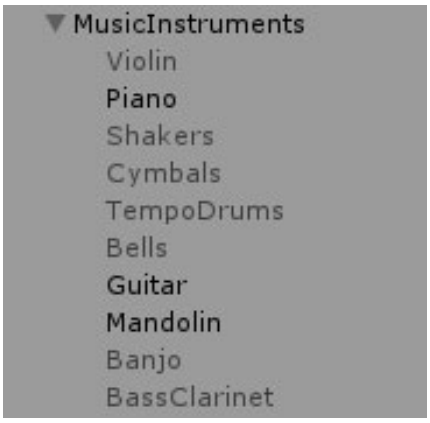


Figure 4.7.: List of all instruments available in the game.

few parameters only. For the theoretical mode, there is a lot of exploration and combats with monsters who protect disks, so the music is adapted to that environment. Finally, the instruments used in the practical mode are piano, guitar, and mandolin. The reason why these instruments were used is simply because with them it is not difficult to create an ambient, a relaxing and peaceful music, as this is the part where the players have to focus in order to complete the challenge. Important properties of the used method are that it is possible to set constraints. For example, there could be at least two out of three instruments activated, but it is also possible that the algorithm activates all three at the same time, based on a random process. Another property, is randomly distribution of notes, which means that in one moment a piano could have the highest distribution value and be the leading instrument, but in the next matrix generation process, a guitar or a mandolin can take that role. All this shows how adaptive this method is, as it generates completely unique music sequence in real-time and with only a few megabytes it creates endless music.

4.1.4. PCG Content

So far, it was shown how PCG techniques were used in this project for the map and sound generation. In this game, the L-system algorithm was used for generation of trees and forests. The algorithm is based on recursive

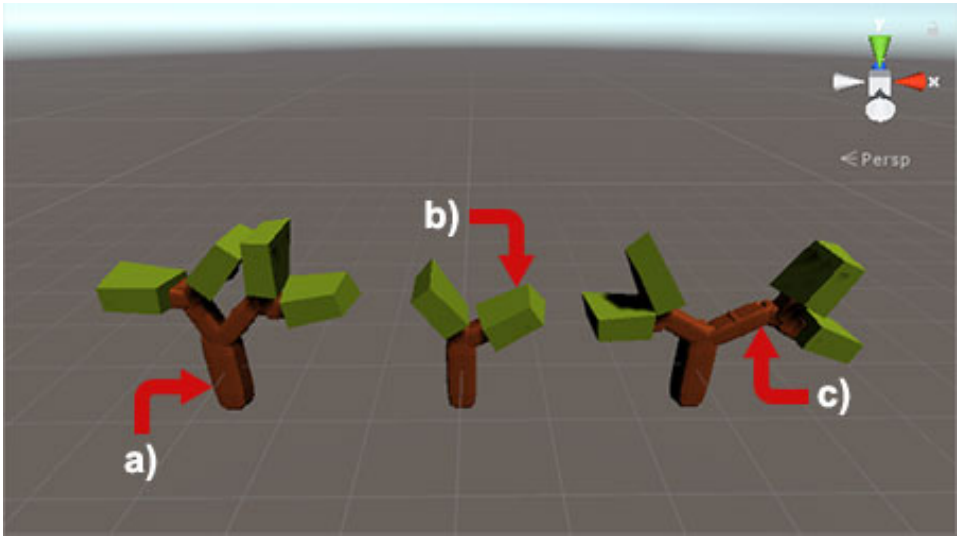


Figure 4.8.: Screenshot from the Unity project showing different tree structures generated with the core tree components, where (a) represents a tree trunk, (b) tree top and (c)a branch.

function calls for generating branches and forming trees. There are tree main components, which were used in the tree generation process, a tree trunk, branches, and a treetop. The algorithm starts with a tree trunk and then iteratively adds branches until stopping criteria are met. After that, it puts some leaves at the top of the branches and with that, the process of generating a tree is completed. Figure 4.8 shows different tree structures formed with these components. The created script can be applied to creating a forest, as random values for angles, branch sizes, and degree of branches will ensure that every time a different tree is generated which, is quite useful for providing natural forest structure, as trees in a forest are similar but still different.

The further text discusses the implementation of tree generation in the Unity3d engine. In order to implement the tree generation in Unity3d, it is important to start with instantiating a tree trunk, as that is going to be a basis for expanding the tree and creating branches. One important thing to note is that Unity3d draws objects from their center point. For obtaining the top point of the tree trunk, it is needed to add a half of the height to the y position value. That point will be denoted as the pivot point in the further

text. Once that is completed, the next step is creating branches.

Creating branches is the most complex part of the tree generation and it is necessary to create the left and right side branches in each iteration. The easiest way to do that, is to define static angle value (for example 45 degrees) and simply decrease the angle of the left side by that value, and do the opposite on the right side. In order to instantiate branches, it is necessary to use linear algebra and calculate the center of the hypotenuses for both branches, as those two points will show where exactly branches have to be positioned. After they are placed in the right spots, they will form ”V” shape which is exactly what is needed. It is not necessary to use the static values for angles, as it is possible to introduce some randomness and generate the branches with different angles, for example, one branch can be instantiated with 30 degrees and the other with 65 degrees.

Once the process of creating a pair of branches is completed, the next step is to check for the stopping criteria. If the stopping criteria are not fulfilled, then it is possible to continue with further tree expansion. It is needed to recursively call function two times (for each branch) and pass the new pivot points. Once the stopping criteria are reached it is important to make the final touch in the tree generation, which is adding leaves or in this case treetops to the top branches, that is, branches from the last iteration. With that, the process of the tree generation is completed and the final output is shown in Figure 4.9. The trees in Figure 4.9 form a forest which was generated by instantiating a number of game objects (trees) and shifting them for an offset. As can be noticed, every single tree is different and all trees together form a natural-looking forest. This can be used throughout a game, both as a group (forest) or individually (single tree). In the practical mode, it is possible to a number of trees spread out the map.

4.1.5. Region Detection

With the implementation of Perlin noise, the map generation process is completed. However, with Cellular automata and Perlin noise, it is ensured that on every run the generated map will be completely different, but there is no actual gameplay in the game. This indicates that it is necessary to

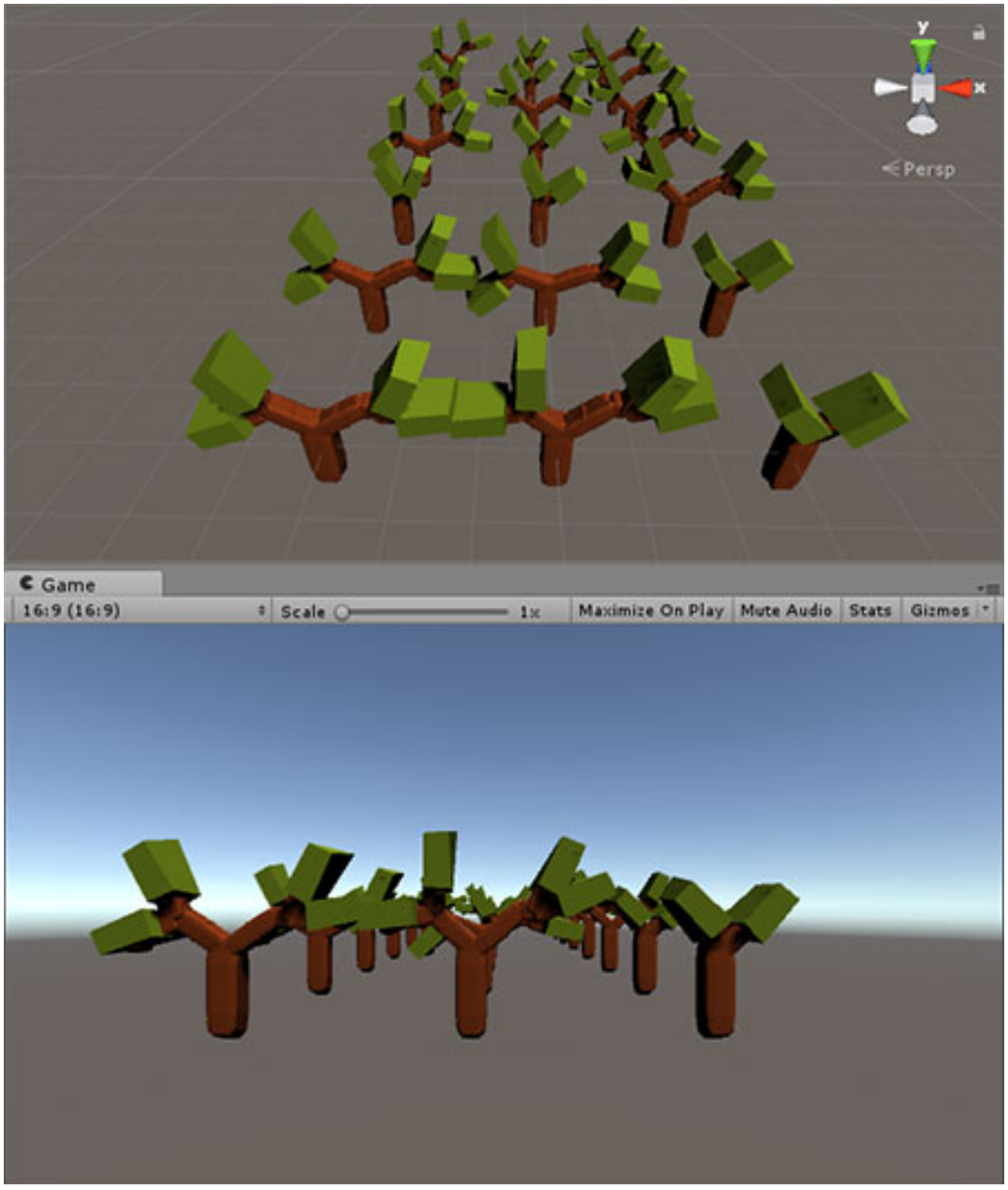


Figure 4.9.: Screenshot from the Unity project showing a forest generation, that is, a number of trees generated using the L-system algorithm.

have an algorithm which is going to detect regions or spots on the map where the elements like a player, enemies and pickups can be added. The first step in implementation was to determine the size of a region and pass it as a parameter along with a map. The algorithm searches through the map and looks for the regions of the specified size. Figure 4.10 shows the initial implementation of the region detection algorithm. The red squares represented areas where items could be placed. The region detection process was not completed with this as all the elements could be instantiated in one part of the map. Therefore, it was necessary to further improve the algorithm. The improved version introduced zones, where each zone contained a number regions. The algorithm worked in a way that it assigns elements to regions from different zones. In that way, it was ensured that the regions from different zones are selected. The examples of using different zones is shown in Figure 4.11. Regions from those zones were colored differently so that it was possible to say which one comes from which zone. The approach used for defining zones is straightforward. Basically, the entire map was divided into 6 parts (zones) where every zone had a list of regions. During region detection, zones' lists were populated so it was easy to color differently regions which belonged to different zones, as it was demonstrated in Figure 4.11. There are some more advanced approaches which could give even better results like binary space partitioning, which brakes up the map into zones of non-equal sizes. That can introduce even more randomness in the process of detecting the regions and positioning the elements, but that was left for future plants due to time constrains.

4.1.6. Gameplay and Elements

Once the process of determining which region belongs to which zone was completed, the next step was to place the elements in those zones. In order to do that it was necessary to pick regions from different zones, but not any regions, a region per zone which had the longest distance from the origin, that is, the center of the map. In that way, it was ensured that elements were not positioned in one zones only and very close to each other. If there are more elements which have to be instantiated than zones, then it is needed to place two or more elements in the same zone. Criteria to determine which

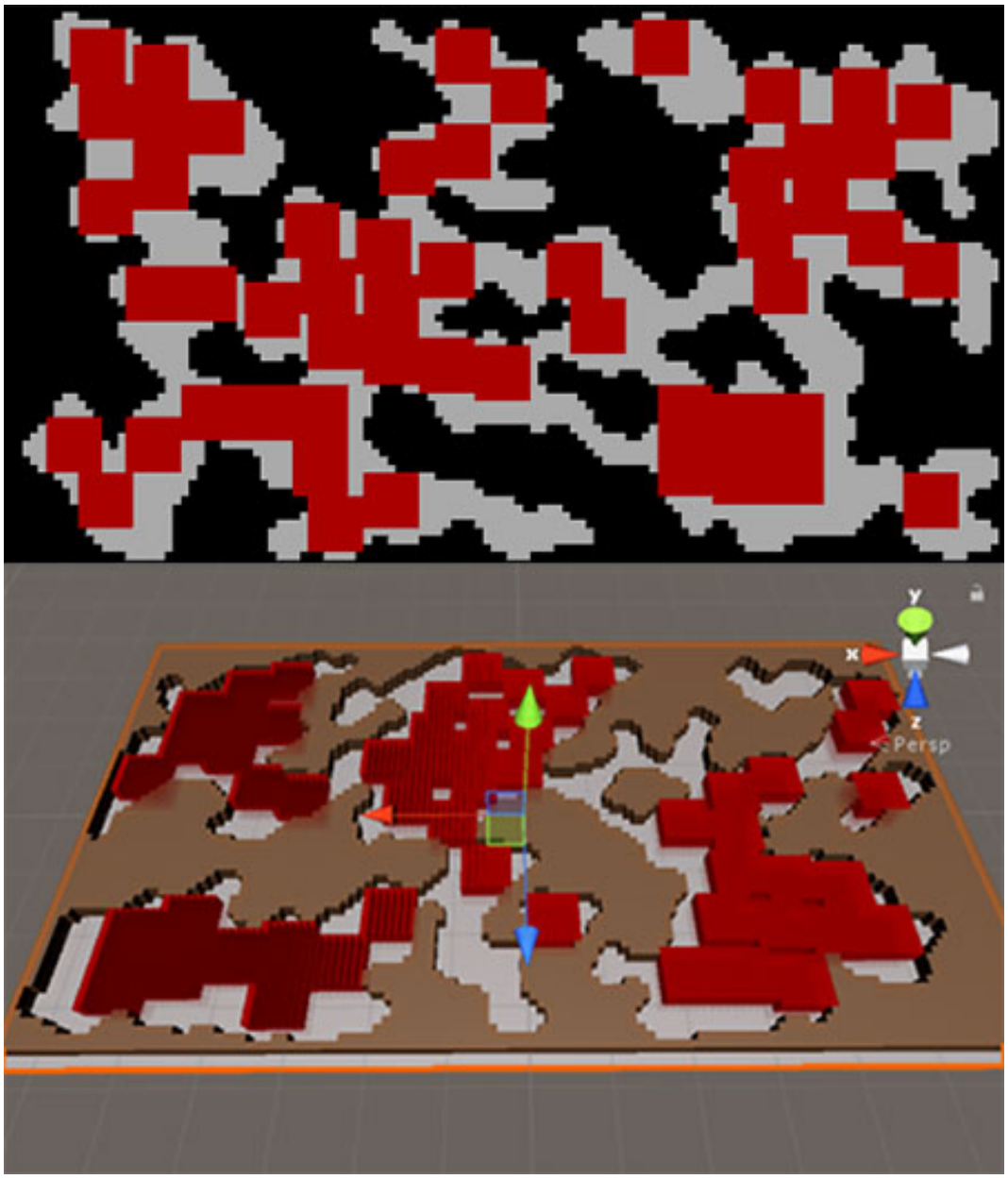


Figure 4.10.: Screenshot from the game showing regions from different zones. There are two map examples, 2D and 3D, generated with two different seeds.

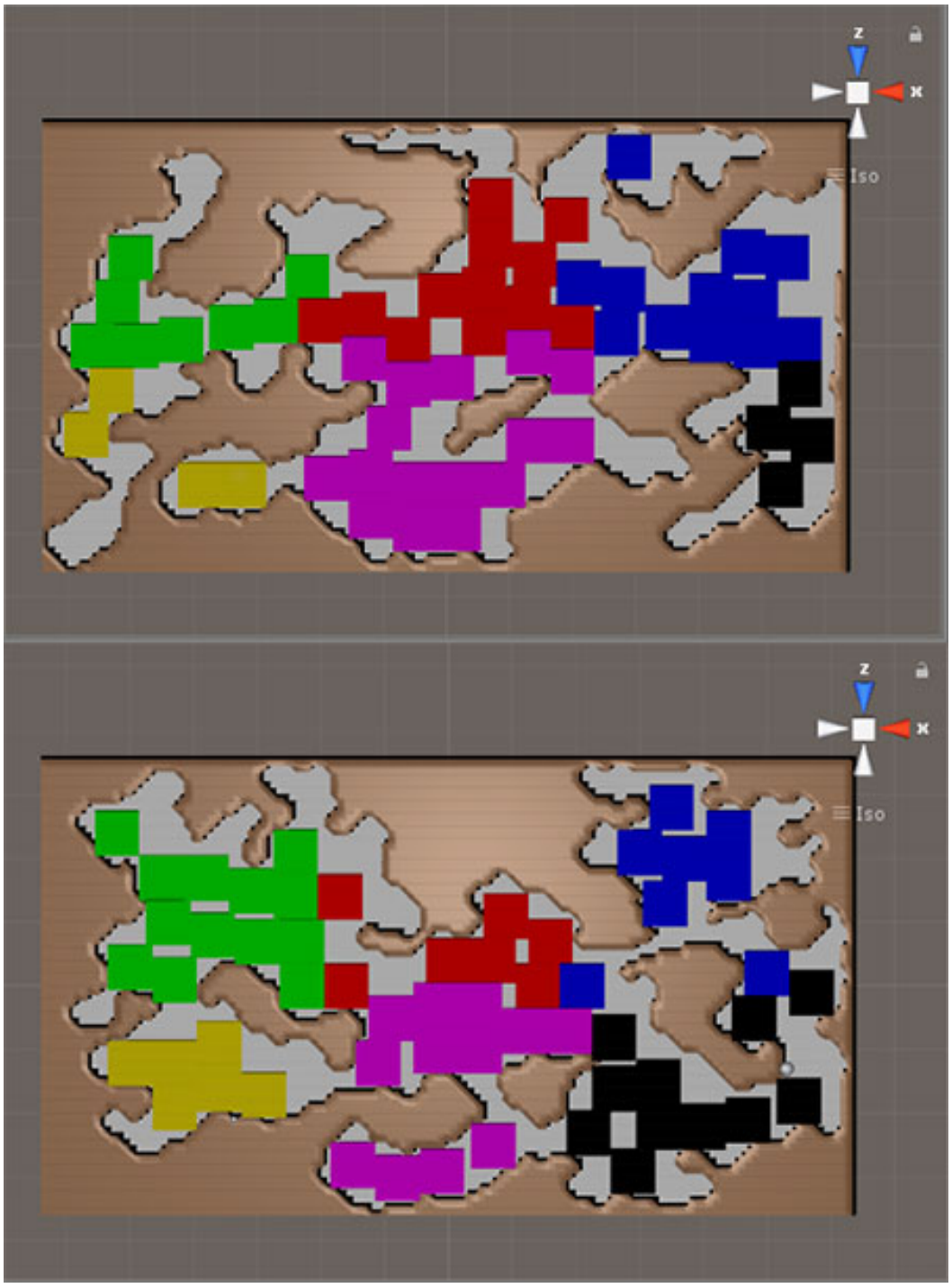


Figure 4.11.: Screenshot from the game showing outputs of region detection algorithm.

two should be put together depends on what kind of experience one wants the players to have. For example, there could be zones where there are only enemies who are surrounding an item which has to be collected. A map with zones like that would be very challenging and hard to complete, so placing many enemy units in one zone is probably not a good idea.

Therefore, there should be some checks of what elements are already positioned in the zone before new elements are added. One of the specified criteria for this game was that there could not be more than two groups of enemy units in one zone. In this way, it was guaranteed that there were no zones which players could find overwhelming or impossible to complete. It can happen that the player is in the same zone with an enemy group but only if all zones are populated with enemy groups. This is a very rare case but since the number of enemy groups is randomly defined, there is a probability for the already described scenario, however, that probability is very low. All of this, explains how important and hard it is to come up with the right ratio of enemy units and pickups, as well as to position all the elements.

In order to control the character, the players have to use a joystick which allows the character to move around, and a button which is used for shooting at enemies. Figure 4.12 shows how controls look like, along with all the elements positioned on the map.

4.2. Practical Mode

The practical mode of the game is where students can apply their knowledge which they have previously acquired in the theoretical mode. This part is going to be different for every course but the learning process is still the same. Two most important components implemented in practical mode for the programming course were the UI system and the environment. The UI system provides controls (serves to enhance input) which the players can utilize to structure their answers. The playground contains an educational environment which the players can use to experiment and improve their knowledge. Drag-and-drop blocks are predefined controls (code snippets in the programming course) which were implemented to speed up the

Figure 4.12.: Screenshot from the game showing controls along with other gameplay elements.

process of coding. In other courses, those components can be designed to serve a different purpose. The idea is to drag-and-drop those block onto the content area in order to solve the given challenge. After that, the players can use the virtual keyboard to modify the content generated by drag-and-drop components, as most of the time those components hold a piece of information, which represents the structure and saves time in producing the correct answer, formula, equation, etc. In the practical mode, just like in the theoretical mode, the tasks are collected from the server, as an educator has to define a set of tasks and correct answers for every course, before inviting students to join the courses.

4.2.1. UI System

One of the most attractive features of this project is the custom UI system. There are some complex activities in the practical mode of the game, such as drag and drop blocks, nesting and sorting, which in programming course

allow players to write programs faster and more easily. In other courses that can serve a different purpose. Default UI provided by the Unity3d did not work the same on all mobile platforms. Also, relying only on the default UI is simply not enough to fulfill all requirements. Therefore, it was necessary to implement the custom UI system which forms a mobile IDE, that along with the rest of the game, provides rich programming experience on a mobile device.

In order to classify the content and make it more accessible, it was necessary to implement a tab system. Figure 4.13 shows three different tabs in the practical game part of the programming course. The first tab is an objective or a task introduction tab, which provides information on what players have to do in order to complete the task. The task description is pulled from the server, that is, from the task database of the course which an educator filled in previously. The goal in the programming course, is to program the robot so that it avoids all the obstacles (yellow boxes) and reaches the disk. The next tab is called the action tab or code tab. In this tab, players have to drag-and-drop the predefined components from the left-hand side onto the scene in order to provide the correct answer. In the programming course, there are eight code blocks predefined. Four of them are logic and four are action blocks. The logic blocks are code programming commands which can be used to program the behavior of the robot. Using those four blocks one can easily declare new variables and assign them values, use if statements and for loops. Finally, there is a print command which is useful for debugging and analyzing if the code is behaving as expected. Once the logic is prepared, it is necessary to add the robot movement commands. The robot can move in all directions except diagonals and in order to make it move, it is important to put logic and action commands together.

There is one very important feature which is hard to notice in Figure 4.13. That feature is code manipulation, which was basically constructed out of nesting and sorting. This feature allows players to easily modify the order of the code by tapping onto the desired code block and moving it to any position in the program they like. Another useful functionality is that they can nest the code as well. There are two code elements which can nest more core (or more code elements), and they are the if statement and for loop. As could be noticed, in Figure 4.14, there is a rounded box around for loop, which indicates that the block can nest more elements, while *"robot.Up()"*

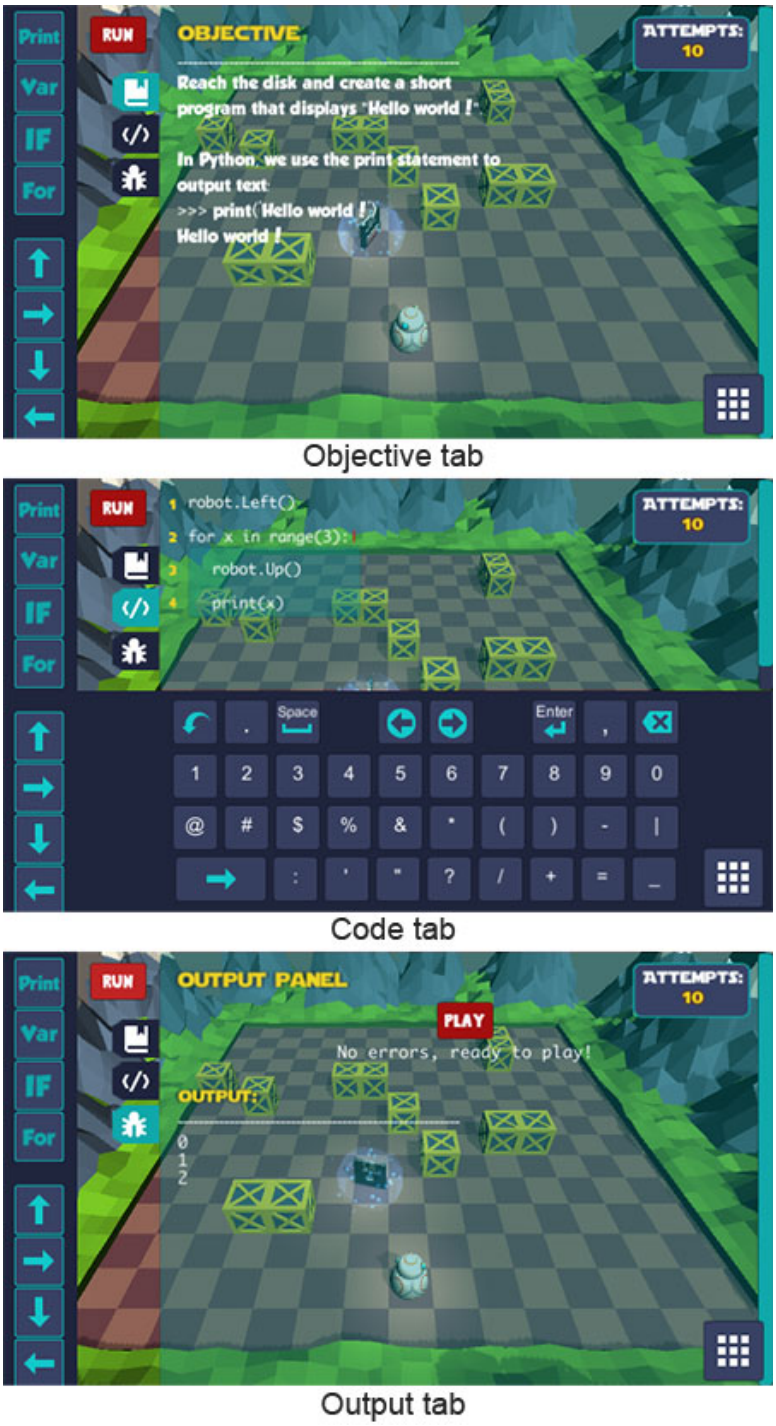


Figure 4.13.: Screenshot from the game showing the practical mode of the programming course.

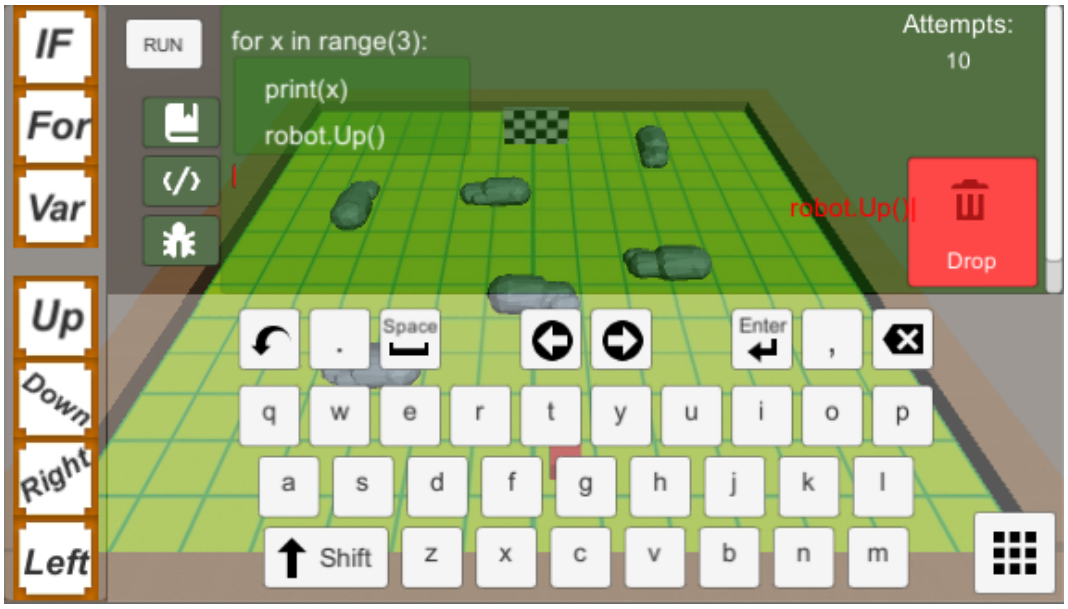


Figure 4.14.: Screenshot taken from the game which shows the early prototype of the practical mode with the delete zone component.

command does not have the same component, as that element is a function call and can not nest more elements. For example, the players can put a print function call inside of a for-loop element, which is parented by a for-loop, in other words, they can nest elements in any way or order they like. Technically, there is no depth limit for nesting, but too much nesting can result in a lot of scrolling, but this scenario is very rare and usually not necessary.

One useful component which helps with the faster code writing is a delete zone (recycle bin). The delete zone removes components so that players do not have to use the backspace key to get rid of a piece of code. They can simply drop a code element they want to remove onto the delete zone which is going to destroy that object. Figure 4.14 shows how the drop functionality looks like, but it also shows the how early-stage version of UI looked like. It is important to note, that the delete zone pops up only when dragging of code element is in the progress, because it is only needed then, while during the writing the code or editing, it disappears.

The third tab in UI system is the output, which can be activated either by tapping on the third icon tab or by tapping on the run button, which is going to activate the interpreter and show the the output in the output tab. If there are no errors, then the ”Play” button will show up, and the player can execute the code on the robot. An important thing to note is that every time a player taps on the ”Play” button, the number of attempts decreases. In case of errors, the interpreter will not be able to process the code and will display the error which caused the program to collapse, as well as, the line number where the error occurred. An example of error reporting is shown in Figure 4.15. When there are no errors, the player is able to see the output which is acquired from the print function calls.

All core UI components are presented and denoted in Figure 4.16. There are two modes, the edit (a) and preview (b). Other components are, a left sidebar containing all the code blocks (1), the introduction, code, output tabs (2). The run button (3) used for processing the Python code and checking for the syntax and interpreter errors. A number of attempts counter (4), which decreases every time the code is run. Finally, the toggle button (5) switches from the edit to preview mode, tapping on the same button in the preview mode (6) will do the reverse action.

The implementation of the virtual keyboard was a very time-consuming process as it was necessary to create a fully functional keyboard, which can work well on mobile devices. First, all buttons had to be responsive, which means that their size and position adapts to any resolution and aspect ratio of the screen. Since there is an implementation of the custom UI system for moving, sorting and nesting code blocks, the default OS’s functionality for positioning cursor in the text could not work here. Therefore, it was needed to implement this functionality as well, and that is why the pair of arrows in the keyboard was needed.

The way the cursor positioning was implement is simple, as every time a new component is added, the pointer moves at the end of that code block. If one wants to change the code at other lines, then it is necessary to tap anywhere on that code block to position cursor at the end of it, so that arrows can be used to place the cursor between desired characters. After that, one can add more characters which will be added to the left side of the cursor or use the backspace button to remove a character which will move

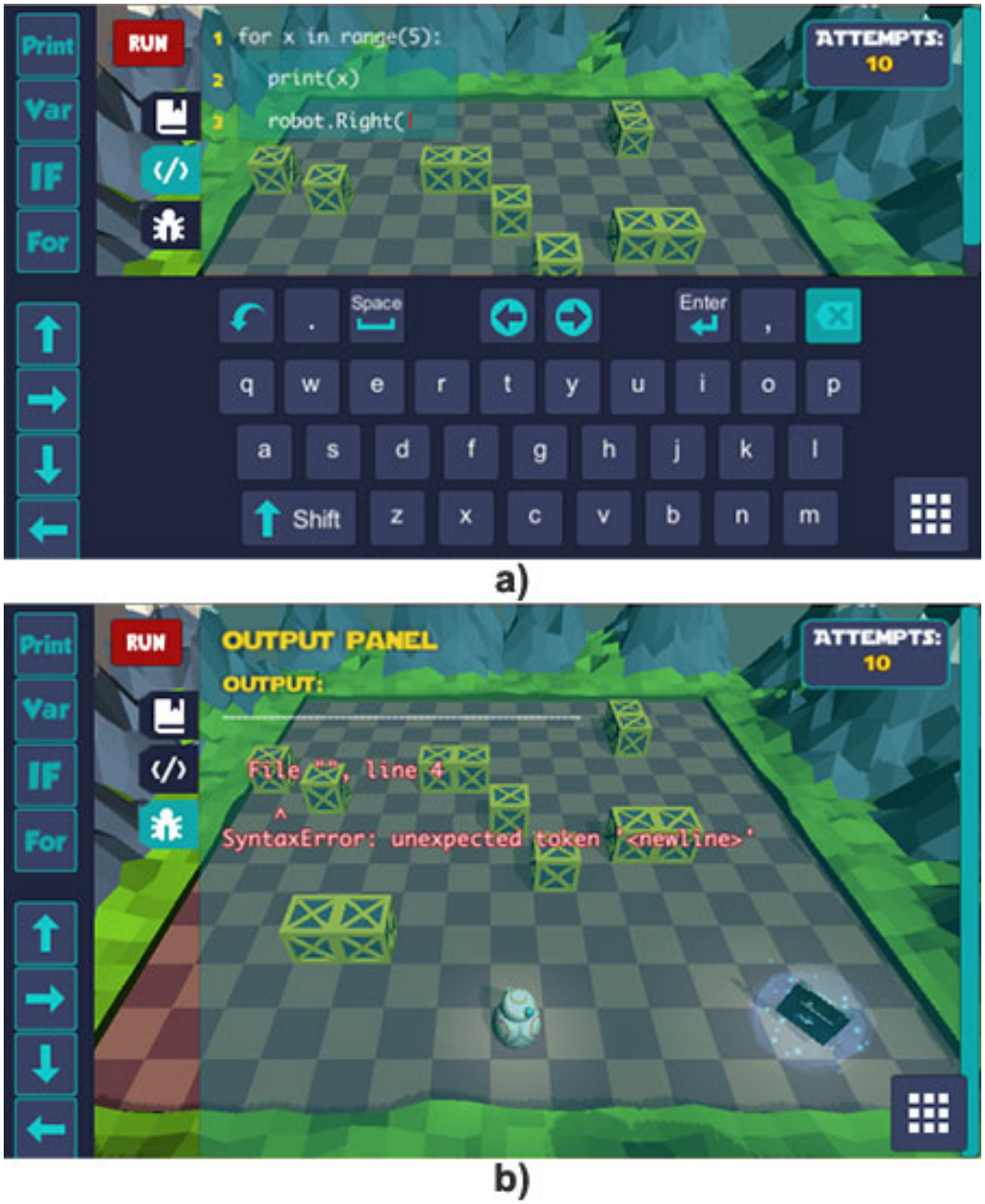


Figure 4.15.: Screenshot taken from the game which demonstrates how error reporting looks like for the syntax and interpreter errors.

Figure 4.16.: Screenshot from the game showing core UI components.

the cursor to the left. Figure 4.17 shows the different keyboard layout panels. First, there is the default layout with lowercase letters only (a). Tapping on the "Shift" button, all letters are converted to uppercase (b) and now when a player is typing, he or she will see the uppercase letters. In order to type a number or symbols, it is necessary to tap on the button in the top left corner of the keyboard (marked in light blue) which will switch from the letter layout to numerical (c). Finally, since not all symbols can fit in the layout from (c) there is the button in (d), located in the bottom left corner of the keyboard (again marked in light blue) which replaces the third row of the layout with the new set of symbols.

4.2.2. The Environment

The environment is the second core component of the practical game mode. It works closely with the UI system, as in the programming course, they depend on each other. The environment represents the scene, that is, everything else which is left when the UI system is removed. Every course should have a different playground system, which is designed and adapted specifically to that course. In the programming course, players have to program a robot and make it move around the world and collect the disk. However, there are some obstacles on the way to the disk (yellow boxes), which have to be avoided, otherwise, a player will lose one attempt when running into one of the boxes. The same happens when the robot touches any of the world boundaries. When a collision with an obstacle occurs, then there is a screen shake animation, which is based on random values acquired from *Random.insideUnitSphere()* function which gives a point inside a sphere with the radius of one. That point is a vector value and was later multiplied by a float value, which determines the amount of shake. This animation emphasizes that the player did something wrong and since he or she is going to lose an attempt which basically represents the robot's health, it is important to bring the player's attention to that.

It is important to note that boxes are randomly positioned every time the practical mode is started. In this way, it was ensured that even when repeating the same level the players will not be faced with the same obstacle layout over and over. Also, the number of boxes is very important, as more

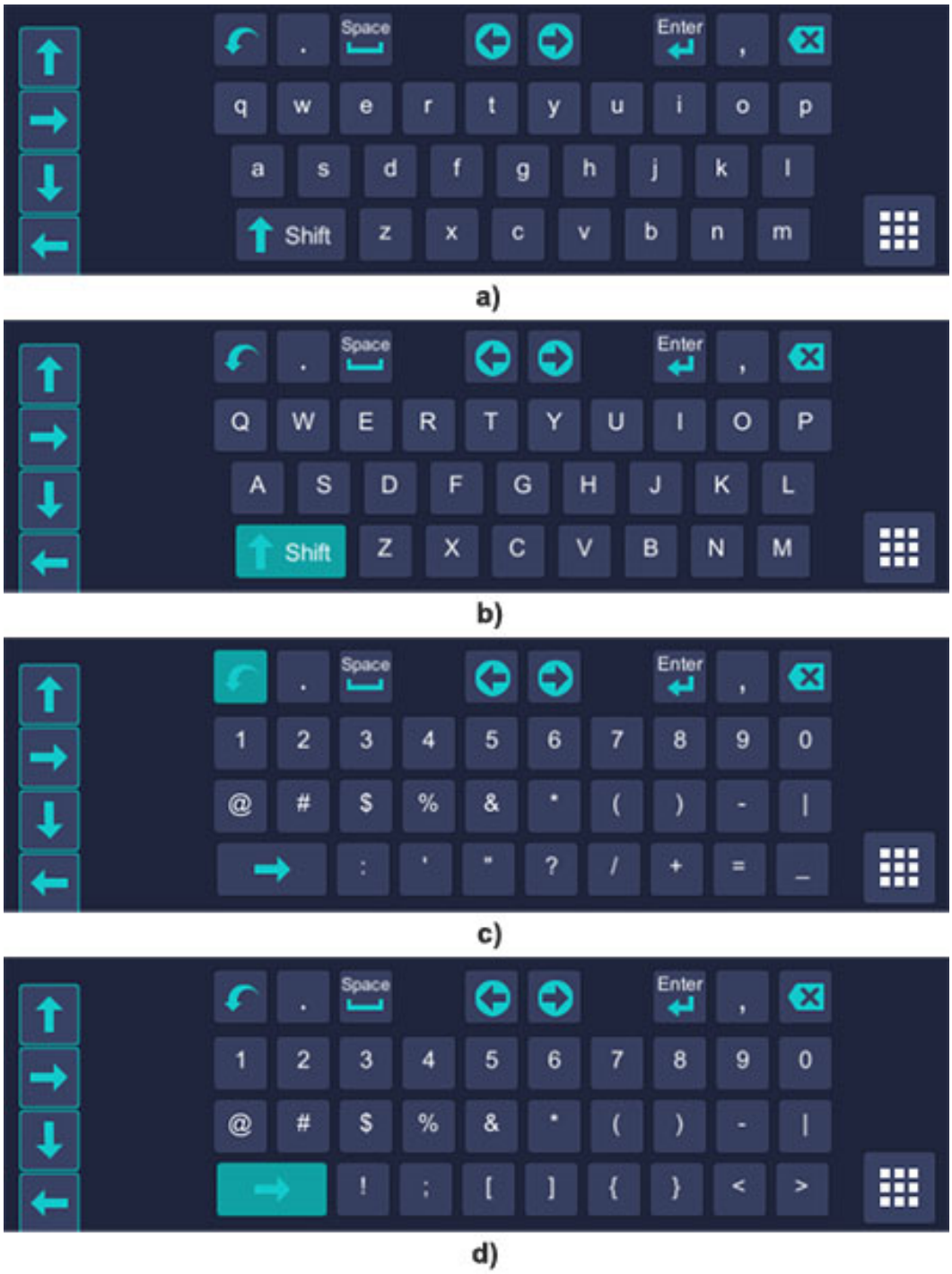


Figure 4.17.: Screenshot from the game the different keyboard layout types.

boxes mean it is harder to reach the disk, which leads to more complex code and more effort from the players. Therefore, the number of boxes depends on the player's behavior which is constantly tracked, so the new value is calculated based on previous attempts and results. The position of the robot stays intact and every time the game starts, the robot is positioned at the same place. However, the disk is also randomly positioned and it will be instantiated on different parts of the map on each run, but compared to the boxes which can be randomly distributed throughout the entire world, the disk is placed to a distance from the robot, which is calculated based on data collected from the server. Every time the code is executed on the robot, the number of attempts decreases and if the player did not manage to collect the disk within the defined number of moves, then the game is over and the player has to repeat that task from the beginning. This time all obstacles and the disk are positioned differently.

4.3. 3D models and UI Imagery

Visuals can help developers convey their story in the desired way, so the selection of the art style mostly depends on the gameplay and game story. Since the decision to go with 3D has been made, the next step is to pick one 3D art style of many. There is no official game art taxonomy to refer to, mainly because of the rapid change of art in video games. Every game has some unique characteristics which depend on the artist's vision and creativity.

In terms of software, instead of using Photoshop[2], Illustrator[3], or GIMP[4], it was necessary to use Blender[5], Maya[6], 3Ds Max[7], etc. However, there were some aspects in 3D which still required the usage of image editors. It was needed to use the image editors to create textures for 3D models. Another need for image editors was in the process of creating color schemes

[2]http://www.adobe.com/products/photoshop.html
[3]http://www.adobe.com/products/illustrator.html
[4]https://www.gimp.org
[5]https://www.blender.org
[6]https://www.autodesk.eu/products/maya
[7]https://www.autodesk.eu/products/3ds-max

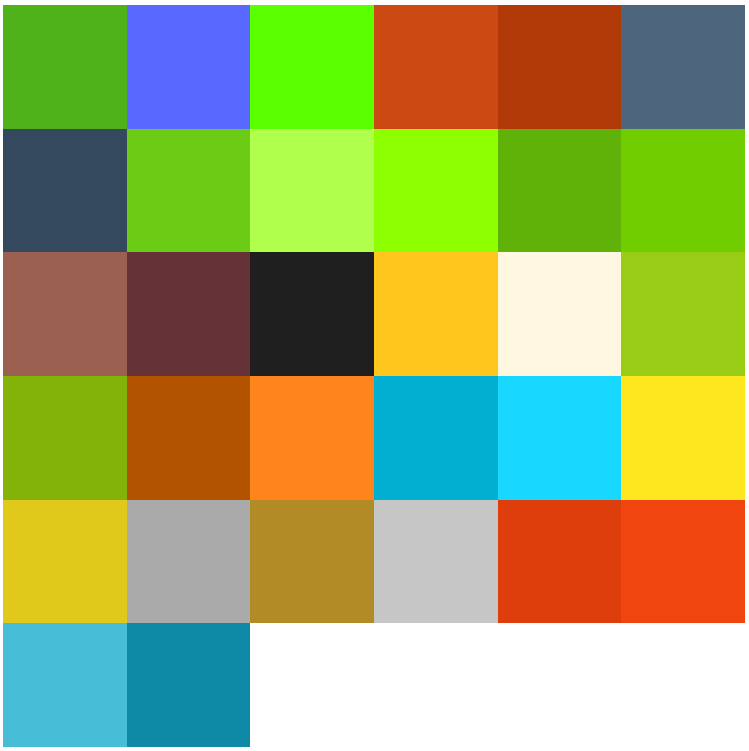


Figure 4.18.: The color palette used to create 3D models in the game.

or palettes for UV unwrapping of 3D models. So even though one is not creating 2D game art, he or she will definitely have to use an image editor, if not in the 3D modeling process, then surely for UI elements. For this project, it was decided to make a 3D game, mostly because the majority of the game which the target audience plays are 3D.

All 3d modeling tools are very similar and picking one is usually a matter of preference. It was decided to use Blender, as it is a free open source software which has a big community, therefore, there were many tutorials and instructions available. All 3D models in the game were created using Blender 3D modeling software.

The color palette for 3D models is shown in Figure 4.18 and the list of all models created with that color palette is shown in Figure 4.20. Figure 4.19 shows color palette used for UI. All models and art components were

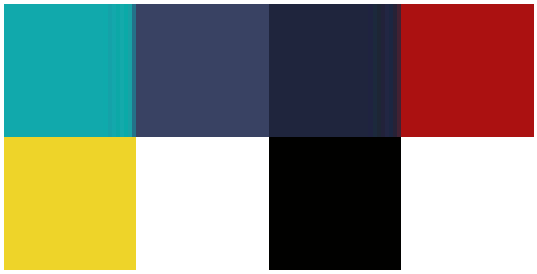


Figure 4.19.: The UI color palette used to create the UI system in the game.

Figure 4.20.: All models created for this project, using the color palette from Figure 4.18.

made by the author of this project, including UI, characters, maps, etc. In other words, nothing has been used from other developers and artists.

4.4. Player Customization

The player customization feature was implemented to improve the engagement of players by allowing them to modify their characters in the game. It is possible to change the character's look by buying a gear which also increases the character's abilities. Figure 4.21 shows how player customization scene looks like. The players can customize the character's skin color, facial expression and change their name. The health, damage, and speed are parameters which improve the character's abilities. The base values of these parameters change as the player progresses and levels up. The gear items like a hat, gun and cloth can improve the characteristics of the main character. Those items can be obtained from the shop, but for buying an item it is necessary to use gold which players can collect by playing both the theoretical and practical modes. When a new item is obtained, then it shows up in the inventory so a player can drag-and-drop it onto one of the gear slots and see how that item affects a specific characteristic.

4.5. Final Game Output

This section is the final part of implementation chapter, which summarizes the entire implementation process. The idea of the project was to build an educational video game which will help students to improve their knowledge in different subjects. In order to prove the concept, there is only one subject available at the moment (programming), used as a prototype. The entire game is built around this subject. The course curriculum is defined by the developer of the project and converted into the skill tree. There is a web application which allows educators to create their own skill trees for their courses and invite students to play the game. That component is implemented by (Kojić, 2017), but it works very closely with the game. The educators are able to track the progress of their students and better

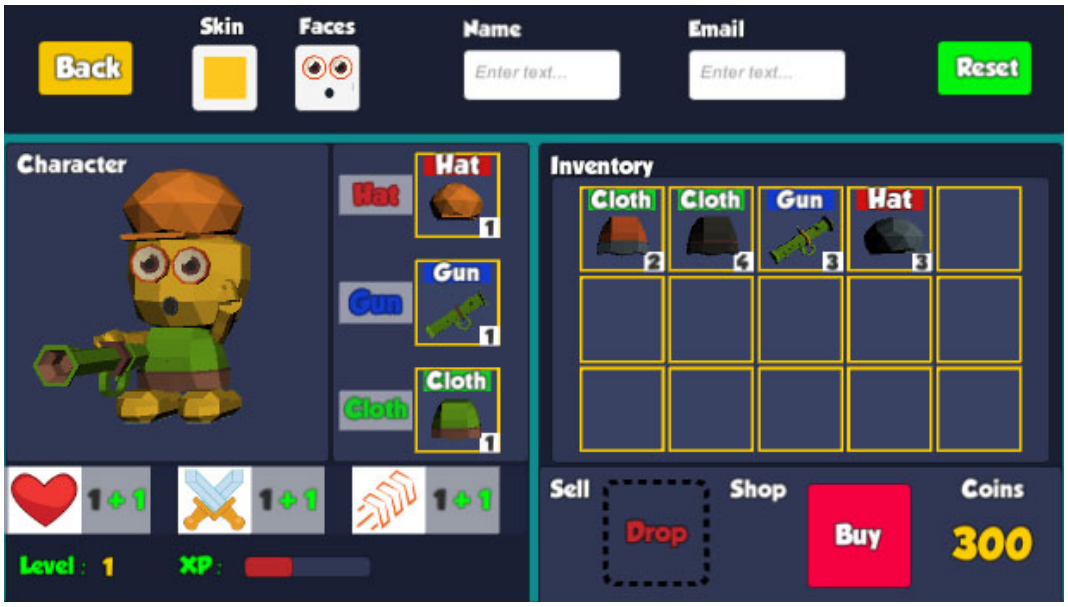


Figure 4.21.: Player customization scene. Sceenshot taken from the game.

understand which topics the students are good at and which of them they have to improve. A topic in the game is denoted as a skill, which can have a number of questions associated with. Since the game is based on procedural generation there are only six scenes, where 4 of them are different menu and level selection scenes, which means that there are only two game scenes, theoretical and practical modes.

In classrooms, educators usually explain theoretical aspects of a topic first and then they move on to the practical aspects. The same approach is used in this project. Exploration is a very important part of learning and students can learn a lot if they explore a topic on their own. That is why in the theoretical mode of the game the players have to explore a map and collect items. At the end of that process, they will get a short description and a question, on which they have to answer correctly in order to move on to the next skill. In the programming course, they will get more information on how the core programming components, like variable declaration, if statements, for loops, and other, work.

Once the students are familiar with the theoretical aspects of a specific topic

they should apply their knowledge in practice. This is where the practical mode comes in. In this part, the players have to solve puzzles, which are designed in a way that they have to use the knowledge from the theoretical mode in order to successfully complete it. Again, in the programming course, the goal is to program the robot which is going to avoid all the obstacles and collect the missing disk.

Since students have to go through, both theoretical and practical aspects of every single topic (skill in the game) within every course, it is necessary to introduce some randomness so that they do not repeat the same action in the game over and over. A proposed solution to that is the usage of procedural content generation. Using PCG algorithms, every time players run the theoretical mode in the game, they will have a completely different map, an environment, a music, enemies, pickups, obstacles. In this way, it is ensured that the game will be totally different on each run and that students will not be bored. Figure 4.22 shows several different game outputs for the theoretical mode.

There are some very similar mechanics in the practical mode as well. It consists of drag-and-drop components, where every component carries a piece of code and an environment. Those components are only one part of the custom UI, as other important parts are the virtual keyboard, tabs, and panels. The code which players have to write is Python code, which once players are done with programming, has to be interpreted and execute on the virtual robot. IronPython is an interpreter for Python code which helps with code execution and error reporting. Finally, just like in the theoretical mode it is important to give players different challenges all the time, so the obstacles, as well as, the disk, are randomly positioned on every play. Figure 4.23 shows several different game outputs for the practical mode.

Figure 4.24 shows different menu scenes, from main menu to course and skill selection scene.

Figure 4.22.: Screenshot from Unity project showing that on every run different map will be generated using PCG algorithms.

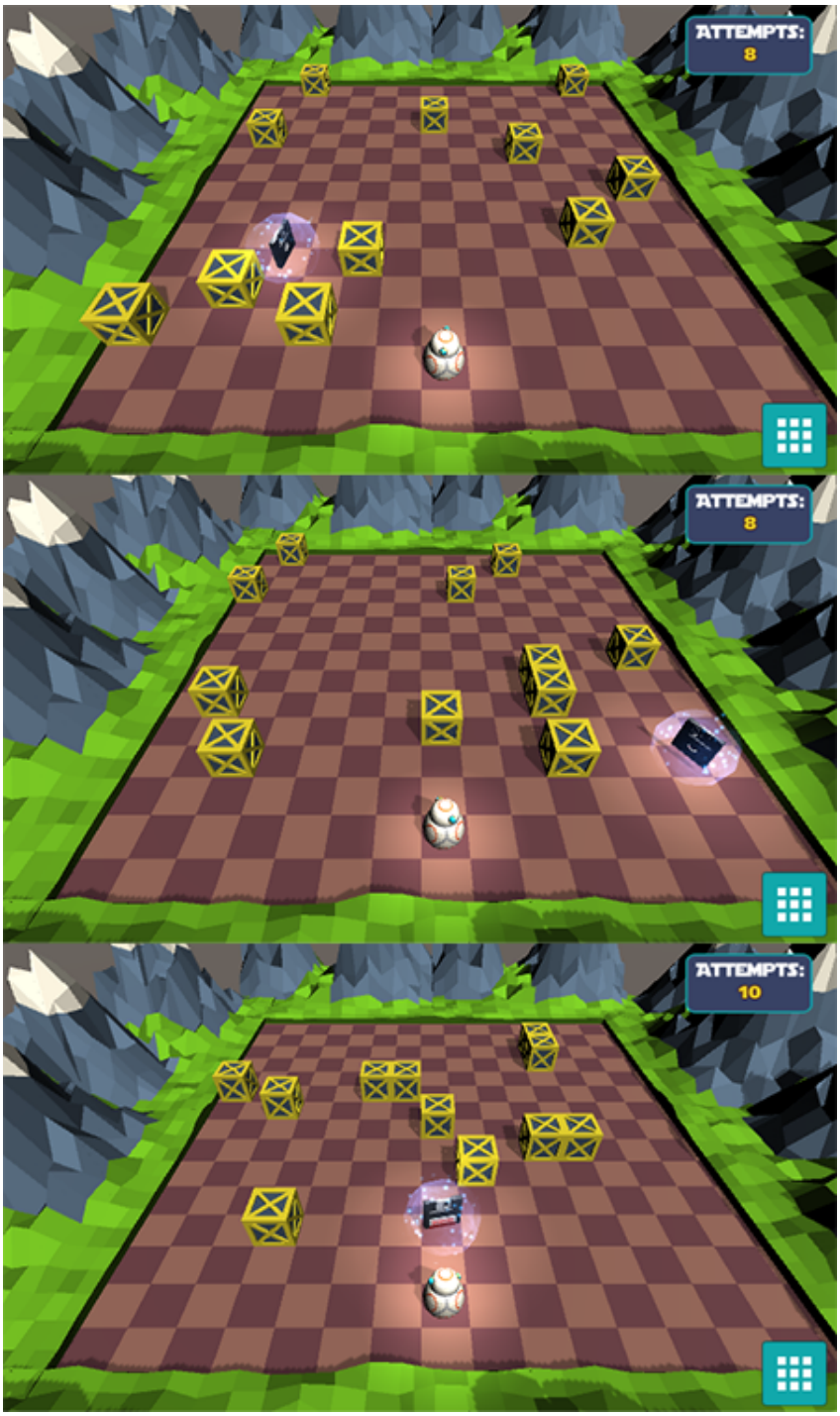


Figure 4.23.: Screenshot from the Unity project showing that on every run of the practical mode the obstacles will be placed at different positions.

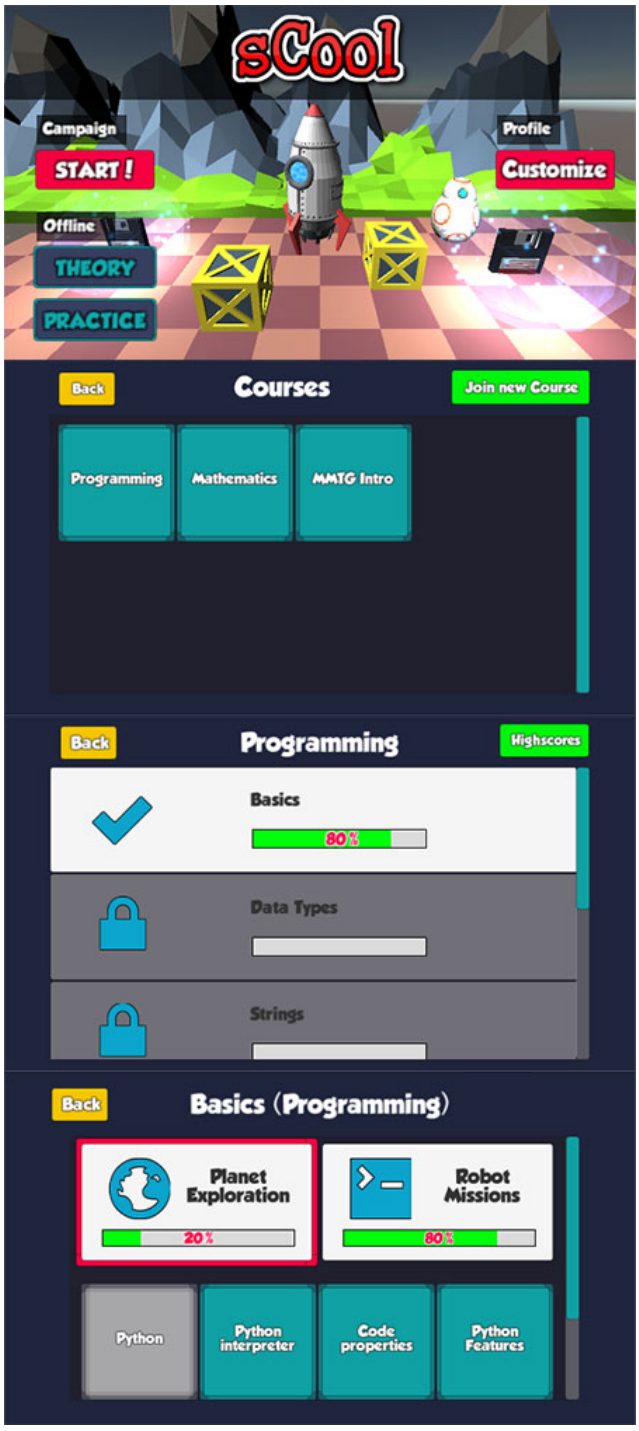


Figure 4.24.: Screenshots from the Unity project showing different menu scenes. It starts with main menu scene and then it shows how scenes for the course and skill selection look like.

4.6. Summary

The theoretical mode is based on exploration and it utilizes PCG algorithms for the maps generation. Cellular automata and Perlin noise are in charge of generating structures and natural looking terrains. The practical mode is designed to serve as a playground where students can acquire the practical knowledge by solving problems from the real environment. The custom IDE is an important feature in the programming course which allows players to program using the code blocks that produce code when dragged and dropped onto predefined drop zones. Another way of producing the code is by using the virtual keyboard. It is important to note that the entire educational content is pulled from the server and loaded into the game. The web application allows educators to create courses and populate them with lessons. The procedural sound generation creates different music based on which scene is loaded. The music in the exploration mode is fast-paced and bit intense as players have to constantly fight enemies, while in the playground part the music is calm and relaxing since players have to be focused. The core game features are designed as modules, so it is easy to add new modules and extend the game's capabilities. All visual assets are created in low poly style using Blender 3D modeling tool, while the entire game is developed with Unity3D engine.

5. Evaluation of the System

The objective of this section is to explain different setups and methodologies which are going to be used in order to measure users' engagement and motivation, as well as, to obtain their general opinion of the game. Also, it provides an overview of all the answers provided by users and the general conclusions acquired from the results.

5.1. Evaluation of the Prototype

The evaluation is performed through two phases, the prototype evaluation and the final evaluation. The first evaluation was scheduled right after the prototype was ready, so that the feedback from users could be taken into account and further improvements made.

5.1.1. Study Setup and Methodology

In order to be sure that the game is engaging, it is important to evaluate the game with users and get their feedback. The feedback should tell if the game is fun, easy to use and helpful. In order to evaluate the gameplay, playability, and flow it is necessary to create a usability assessment questionnaire with system specific questions.

The evaluation consists of:

1. Introduction before using the game (3 minutes)
2. Interaction with the game (10 - 15 minutes)
3. Filling in the questionnaire. (10 minutes)

The User interface with placeholder graphics, basic game mechanics, elementary functionalities and layout were the core characteristics of the initial version of the game. However, even though the game had a number of limitations it was necessary to perform the testing with users so that the feedback can be taken into account and those core characteristics modified and improved while there were still in the early stage. Making changes in further development phases can require a lot of time and effort, therefore, in order to prevent costly modifications, it is a good idea to apply changes as early as possible (Paetsch, Eberlein, & Maurer, 2003).

The idea is to ask questions about the controls, gameplay, intractable and perception, in oder to understand what the users think about it and how those characteristics can be improved.

There are several methods which can be used to convey the evaluation, and collect the feedback and results from questionnaires. Polleverywhere[1] is an online tool which allows users to create polls and distribute them to their target audience. One of the greatest benefits of this tool is that it collects, groups, and structures the results in a way which helps to understand the data and derive conclusions.

5.1.2. Participants

Ten participants took part in the prototype evaluation. The participants were from 19 to 24 year olds and all the them were bachelor computer science students. Two out of 10 were female participants. The average age of the participants was 21.7 years with the standard deviation of 1.49. The native language for the majority of participants is English, as 6 out of 10 participants were from London (UK), while 4 participants were from Graz (Austria) and their native language is German.

[1] https://www.polleverywhere.com

5.1.3. Results

Before the participants started filling in the questionnaire they had to provide their age, gender, country and study programme. There were 17 questions in total, 14 questions were based on the rating from 1 (strongly disagree) to 5 (strongly agree) and 3 were free answer questions. At the very beginning of the questionnaire, the participants were asked what they liked and disliked about the game. One participant stated *"I liked that there are two different sections, one for learning and a second to test you. I like the robot level also, it a good idea to have a simple drag and drop to get people started in coding"*. Concerning, the elements they did not like, one participant wrote *"The shots of the enemies are too hard to avoid, because the delay for moving is really high"*. That was approved by the majority of participants as 5 out of 10 (M = 2.5, SD = 0.97) disagreed that the controls in the exploration part were easy to use. The vast majority (M = 2.4, SD = 0.0.97) also disagreed that the controls were comfy and made sense. Six of 10 participants (M = 2.5, SD = 0.97) disagreed that they would imagine that most people would learn to use the controls very quickly. Almost everyone agreed that the gameplay was easy to understand as 7 of 10 (M = 4.5, SD = 0.97) participants strongly agreed with this statement, 2 agreed and 1 neither agreed nor disagreed. A half of the participants strongly agreed that the enemy AI was not too difficult (M = 4.1, SD = 1.29) and 6 of 10 agreed that the AI worked correctly while the rest of them strongly agreed with that (M = 4.4, SD = 0.52). Four participants agreed that the gameplay mechanics worked correctly, 3 strongly agreed, 1 was undecided and 1 disagreed (M = 3.9, SD = 0.99). Everyone agreed that the game was simple and friendly enough to be played by 12+ year olds, as 8 of 10 participants strongly agreed (M = 4.8, SD = 0.42). When it comes to virtual keyboard, the participants provided different answers. Three of 10 strongly agreed, 2 agreed and 3 participants disagreed (M = 3.5, SD = 1.27). The UI for writing the code was intuitive is something which 4 participants confirmed, 3 strongly agreed while 3 neither agreed nor disagreed (M = 4, SD = 0.82). Finally, the participants suggested the improvements of different game aspects. The majority of the participants want to see better controls and graphics, a tutorial and more features on the UI system. Figure 5.1 shows how the participants rated the controls and the UI for writing the code, while Table 5.1 summarizes the entire feedback from the participants.

	1	2	3	4	5	M	SD
I thought the controls were easy to use.	1	5	2	2	0	2.5	0.97
I felt the controls made sense and were comfy.	1	6	1	2	0	2.4	0.97
I would imagine that most people would learn to use the controls very quickly.	1	5	2	2	0	2.5	0.97
I found the gameplay easy to understand.	0	0	1	2	7	4.5	0.97
I felt the enemy AI was not difficult.	1	0	1	3	5	4.1	1.29
The gameplay mechanics worked correctly.	0	1	2	4	3	3.9	0.99
The animations were nice and smooth.	0	0	2	4	4	4.2	0.79
I liked the different level designs.	0	0	2	3	5	4.3	0.82
The artifacts were easy to pick up.	0	0	1	3	6	4.5	0.70
The enemy AI worked correctly.	0	0	0	6	4	4.4	0.52
I felt this game was not too violent or contained blood or gore.	0	0	0	0	10	5	0
I felt this game is simple and friendly enough to be played by 12+.	0	0	0	2	8	4.8	0.42
The virtual keyboard was intuitive and easy to use.	0	3	2	2	3	3.5	1.27
The UI for writing the code was intuitive.	0	0	3	4	3	4	0.82

Table 5.1.: The summary of the initial evaluation results (1 - strongly disagree, 5 - strongly agree).

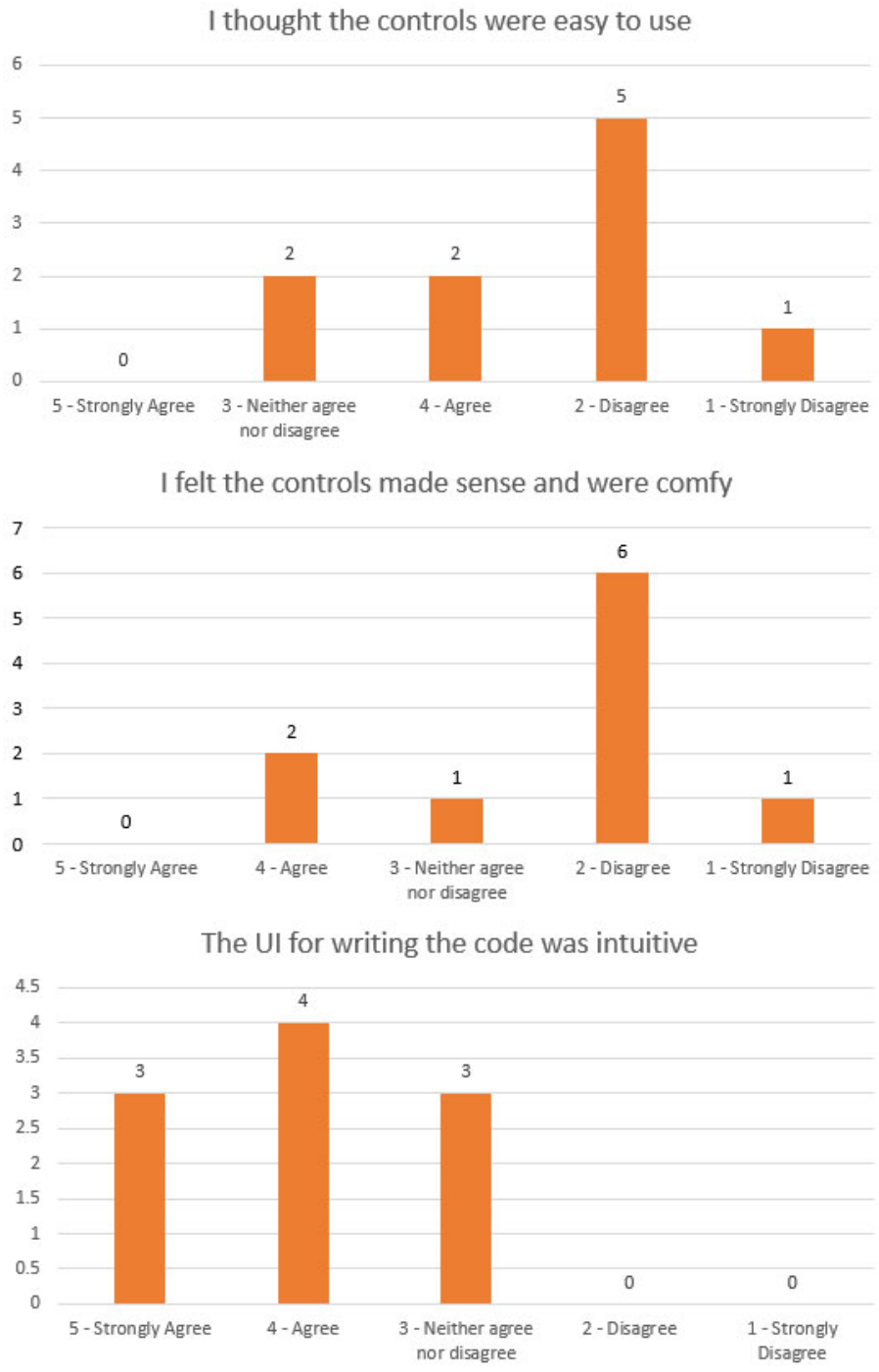


Figure 5.1.: The ratings for the controls and the UI in the prototype evaluation.

5.2. Final Evaluation

Once the project is completed it is necessary to test the system with users and see how they react and whether the game is helping them with improving their knowledge. One of the goals is to test whether all implemented features are working as expected. Also, the final evaluation should justify the modifications made after the prototype evaluation.

5.2.1. Study Setup and Methodology

There are four parts of the questionnaire Pre-Questionnaire, System specific questions, Motivation and Game Engagement Questionnaire (Brockmyer et al., 2009). The pre-questionnaire is used to collect the personal information about users. System specific questions are the second part of the questionnaire and are shown to users as soon as they are done with playing the game. First, it is important to check what players liked and disliked in the game. After that, they have to provide a feedback on the gameplay and usability. The third part is about motivation where users have to rate educational features of the game on the scale from 1 to 7. Finally, Game Engagement Questionnaire (GEQ) serves to test psychological engagement of users playing the game. GEQ measures presence, flow, absorption, and dissociation (Brockmyer et al., 2009). For the purpose of the final testing, it was decided to use LimeSurvey[2] - an open source online survey application, mostly because it offers the statistical and graphical analysis of results, which means that it is not necessary to perform the additional calculations in order to obtain the results.

5.2.2. Participants

Originally, 15 participants took part in the final evaluation, however, 3 of them did not complete the questionnaire, so the results of 12 (80%) of 15 participants were taken into account. All participants were from 12 to 20

[2]https://www.limesurvey.org

years old. Four participants are high school students while 8 (67%) out of 12 were first-year university students. Only 2 (16%) out of 12 students were females, while 10 (84%) of them were males. The average age of the participants was 18.5 years (SD = 1,24). The youngest participant was 16, while the oldest was 20 years old. The majority of participants, 9 (75%) out of 12, were from Graz, Austria, while 3 of them were from London, UK. The native language for the majority of participants is German, while only 3 participants speak English as their first language.

5.2.3. Results

The first part of the final questionnaire obtains users' specific details like age, gender, the amount of time they are spending using their mobile phones, as well as, the average time they spend using their mobile phones for education and their programming proficiency.

When it comes to system-specific questions, the participants stated that they liked the gameplay and that they could learn by playing the game. Seven (58%) out of 12 said they strongly agreed that the gameplay was easy to understand, 3 said they agreed while 2 were undecided. Regarding the controls, 6 participants agreed that the controls are straightforward, 3 strongly agreed and only 1 disagreed, while 2 are undecided. Seven participants (58%) agreed that the gameplay mechanics worked correctly, 4 totally agreed, while only 1 is undecided. Concerning AI, 4 participants disagreed that the AI was not difficult, 3 agreed and 2 totally agreed. Many positive comments were given in regards to different level designs, as none of the participants disliked it. Generally, the participants stated that animations were nice and smooth, as well as that pickups were easy to find and collect. One participant wrote *"I am amazed by animations and the overall design. It's easily understandable and appropriate for the audience. By completing a level you're falling deeper and deeper into the world of programming, 5/5 star ratio for this art work."* Also, 9 (75%) participants said that the game is not too violent while 2 participants agreed with that, and 1 is undecided. Figure 5.2 shows how the participants rated the virtual keyboard and the code writing process.

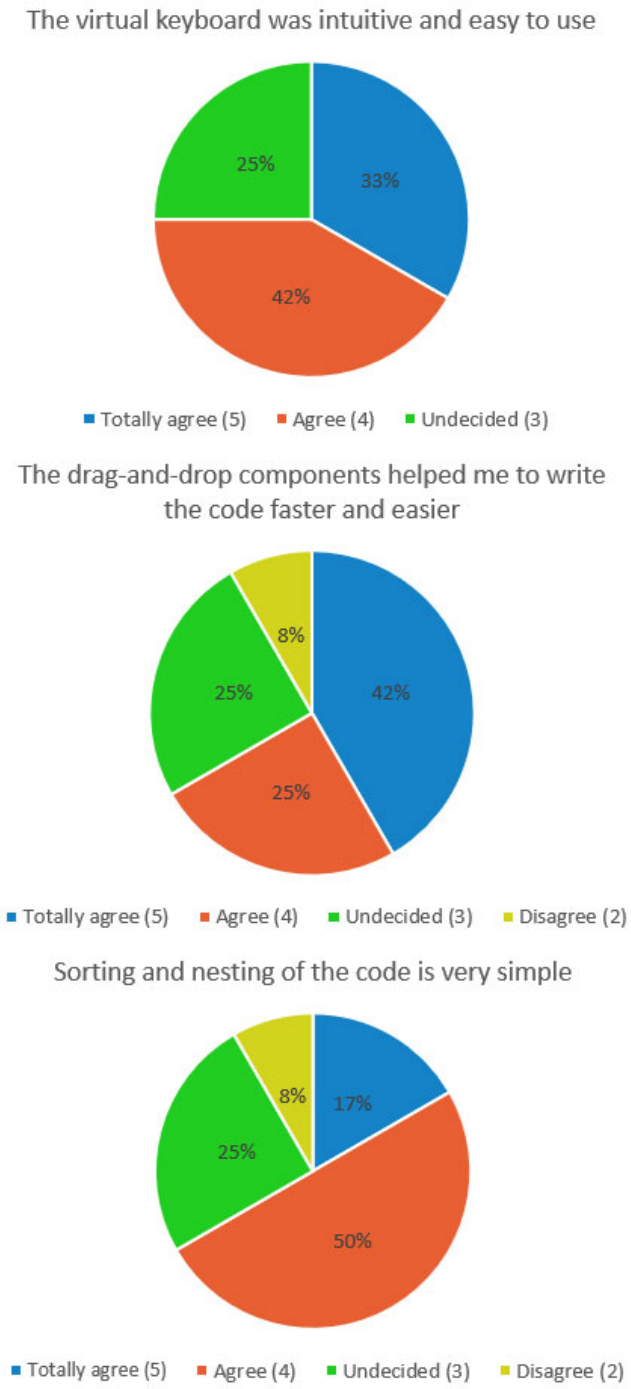


Figure 5.2.: The feedback provided by the participants on the code writing process.

When asked *"What subjects would you like to learn using this game?"* majority of the participants, 10 (83%) out of 12, stated STEM courses like programming, maths, and physics. Only 1 participant did not provide an answer, while 1 participant would like to learn languages using the game. The question *"Who do you think would be a good target group for the game?"* brought pretty much the same answers, as everyone thinks that the most suitable target audience is high school students and first-year university students.

The second part of the questionnaire is designed for evaluating the motivation of participants. The questions are rated on the scale from 1 to 7 (7-point Likert scale), where 7 represents fully agree and 1 fully disagree. The participants confirmed that "sCool" is a good supplement to regular learning (M = 6, SD= 1.13). Also, they verified that they learned something using the game (M = 5.5, SD = 1.38). The vast majority stated that it is a good idea to use "sCool" for learning (M = 6.25, SD = 0.75). When it comes to how "sCool" transforms and presents certain content, the participants agree that it makes content more interesting (M = 6, SD = 1.28) and easier to understand (M = 5.75, SD = 0.87). They also certified that "sCool" makes learning process more engaging (M = 5.58, SD = 1.24) and interesting (M = 5.75, SD = 1.14). Regarding how the game affected their programming skills, the participants confirmed that the game inspired them to learn more about programming (M = 5.58, SD = 0.90) and that learning about programming by playing a video game was entertaining (M = 6.25, SD = 0.86). The questionnaire showed that the participants are undecided when asked if they find regular programming classes interesting (M = 4.17, SD = 1.03) and more prefer to play a game like this on a PC (M = 4.92, SD = 1.44). Finally, they specified that they would like to learn with "sCool" at home (M = 5 75, SD = 0.87) and in classroom (M = 5.84, SD = 1.03).

The final part of the questionnaire is Game Engagement Questionnaire (Brockmyer et al., 2009), which measures the engagement of the participants. There are four main parameters absorption, flow, presence, and immersion. Figure 5.3 shows results for different parameters of GEQ. The absorption (M = 1.87, SD = 0.84) and the flow (M = 2.38, SD = 0.98) are two deepest levels and most difficult to agree with. An average group of participants would most likely provide the undecided answer to presence related questions. In this questionnaire, the values are slightly above the expectations (M = 3.25, SD = 1.22), which means the engagement is high. The immersion is

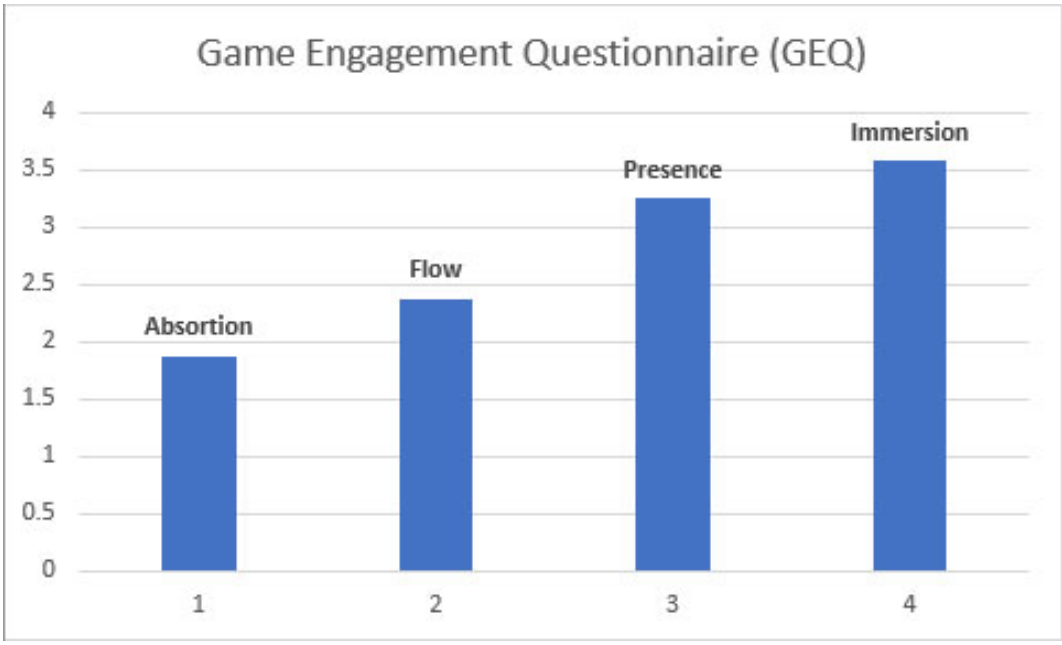


Figure 5.3.: GEQ results for absorption, presence, flow and immersion.

the easiest item to agree with, therefore deviation is usually small, which is confirmed by the results of the questionnaire, as the standard deviation is the smallest of all parameters (M = 3.58, SD = 0.79). Figure 5.2 shows the participants' feedback on the code writing process obtained in the final evaluation. The summary of the system specific questions is shown in Table 5.2. The table 5.3 depicts the participants' opinions of the motivation, while Table 5.4 displays the answers of the GEQ part.

5.3. Discussion and Limitations

The results of the prototype evaluation showed that the participants did not like the controls. The early version of the game had the implementation of A* algorithm which was used for pathfinding. It was concluded that the players have difficulties with interacting with the game in such a way, as in order to move around they had to tap on the desired location, but

	1	2	3	4	5	M	SD
How often do you play mobile games?	1	3	8	2	1	2.93	0.96
How often do you use your smartphone for learning?	1	3	5	6	0	3.07	0.96
How experienced are you with programming?	3	3	1	6	2	3.07	1.44
I would like to use this game in class more often.	0	1	3	6	2	3.75	0.87
I found the gameplay easy to understand.	0	0	2	3	7	4.42	0.79
I though the controls were easy to use.	0	1	2	6	3	3.92	0.9
I felt the enemy AI is not difficult.	0	4	3	3	2	3.25	1.14
The gameplay mechanics worked correctly.	0	0	1	7	4	4.25	0.62
The animations were nice and smooth.	0	1	2	4	5	4.08	1
I liked the different level designs.	0	0	2	5	5	4.25	0.75
The pick ups were easy to find and collect.	0	1	2	5	4	4	0.95
I felt the game was not too violent.	0	0	1	2	9	4.67	0.65
The virtual keyboard was intuitive and easy to use.	0	0	3	5	4	4.08	0.79
Drag-and-drop components helped me to write the code faster and easier.	0	1	3	3	5	4	1.04
Sorting and nesting of code is very simple.	0	1	3	6	2	3.75	0.87

Table 5.2.: The summary of the results on the system specific questions (1 - strongly disagree, 5 - strongly agree).

	1	2	3	4	5	6	7	M	SD
It is a good idea to use "sCool" for learning.	0	0	0	0	2	5	5	6.25	0.75
"sCool" is a good supplement to regular learning.	0	0	0	2	1	4	5	6	1.13
I learned something with "sCool".	0	0	1	2	3	2	4	5.5	1.38
"sCool" makes the content more interesting.	0	0	1	0	3	2	6	6	1.28
"sCool" makes the content easier to understand.	0	0	0	1	3	6	2	5.75	0.87
"sCool" makes learning more engaging.	0	0	1	1	3	4	3	5.58	1.24
"sCool" makes learning more interesting.	0	0	1	0	3	5	3	5.75	1.14
The experience with "sCool" inspired me to learn more about programming.	0	0	0	2	2	7	1	5.58	0.90
Learning with "sCool" was more motivating than ordinary exercises.	0	0	1	0	3	4	4	5.83	1.19
It makes course content more interesting to learn about.	0	0	0	0	3	6	3	6	0.74
I would rather like to learn programming with "sCool" than with traditional methods.	0	0	1	3	3	1	4	5.33	1.43
I find regular programming classes boring.	0	1	1	6	3	1	0	4.2	1.03
Learning programming by playing a video game was interesting.	0	0	0	0	3	3	6	6.25	0.87
Learning programming by playing a video game was more engaging than learning programming in a regular way.	0	0	1	2	1	5	3	5.58	1.31
I would rather use the "sCool" on my PC.	0	1	0	4	3	2	2	4.42	1.68
I would like to learn with "sCool" at home.	0	0	0	1	3	6	2	5.75	0.87
I would like to learn with "sCool" in the classroom.	0	0	0	2	1	6	3	5.83	1.03

Table 5.3.: The summary of the motivation related questions (1 - strongly disagree, 7 - strongly agree).

	1	2	3	4	5	M	SD
I feel scared.	10	2	0	0	0	1.17	0.39
I lose track of where I am.	6	4	1	1	0	1.75	0.97
I feel different.	2	4	2	4	0	2.67	1.15
Time seems to stand still or stop.	3	4	4	1	0	2.25	0.97
I feel spaced out.	6	3	3	0	0	1.75	0.87
I don't answer when someone talks.	9	3	0	0	0	1.25	0.45
I can't tell when I'm getting tired.	4	4	0	3	1	2.42	1.44
If someone talks to me I don't hear.	5	2	4	1	0	2.08	1.08
I feel like I can't stop playing.	2	7	1	1	1	2.33	1.15
The game feels real.	1	4	7	0	0	2.5	0.67
I get wound up.	5	5	2	0	0	1.75	0.75
Playing seems automatic.	2	2	4	3	1	2.92	1.24
I play without thinking how to play.	1	3	3	4	1	3.08	1.16
Playing makes me feel calm.	1	1	5	4	0	3.09	0.94
Things seem to happen automatically.	1	2	3	4	2	3.33	1.23
My thoughts go fast.	1	2	3	5	1	3.25	1.14
I play longer than I meant to.	2	3	2	3	2	3	1.42
I lose track of time.	1	1	3	6	1	3.42	1.08
I really get into the game.	0	1	4	6	1	3.58	0.79

Table 5.4.: The Game Engagement Questionnaire results (1 - strongly disagree, 5 - strongly agree).

that was inconvenient in combat situations as in order to attack an enemy unit it is necessary to tap on it. Sometimes in a situation like that, some players would accidentally tap next to the enemy and instead of attacking, the player would move right next to the enemy, which would result in losing all health points. Thus it was necessary to change the controls and come up with something different. The only logical solution for that was to use controls which will not require to tap on a location in order to move around and evaluate the new controls in the final evaluation.

None of high school students took part in the prototype evaluation, which is an important part of the target audience. Also, all students were master's computer science students, which means that they have been already familiar with programming. The perfect target audience is high school and first-year university students who are getting started with programming, unfortunately, it was not possible to include any high school students in the initial evaluation. A few participants encountered several bugs related to the gameplay, map generation and playground part. Some participants could not reach all the disks on a map as some disks were located in the isolated areas. In the playground part, two participant complained that drag-and-drop blocks did not produce the code.

In the final evaluation the participants did not experience any technical difficulties. In general, they showed interest for the game and they think it helps them to learn more about programming. Regarding the exploration part, they would like the gameplay to be more course specific. In other words, the gameplay should help them better understand theoretical aspects of the course. Also, they showed an interest to see how the playgrounds will be implemented in different courses.

6. Lessons Learned

This chapter discusses different challenges and issues occurred during the literature review, design and implementation of the project, as well as, the evaluation and collecting results. It summarizes critical points of different project phases and lessons learned while working on this thesis.

6.1. Literature

Since one of the most important objectives of this study was to use PCG techniques for generation of maps, assets, and sound, as well as to be able to do all of that on the mobile platform, it was necessary to convey a deep research on which algorithms can be used to fulfill all specified requirements. There is no official taxonomy of PCG methods and other researchers offer different categorization of PCG algorithms. Since there is a number of PCG algorithms, only those which can be used for creating structures of maps and natural-looking terrains, as well as game assets, have been considered in this thesis. Also, PCG methods are not only used for map and asset generation, they are also used for sound production. It was concluded through the literature research, that the vast majority of high school students and first-year university students possess mobile phones and spend a lot of time using them. Another point which contributes to the motivation for building an educational mobile game based on PCG methods, is that these aspects are not common practice but can be very useful for educators, as they do not have to invest their time in creating levels. This approach allows educators to focus on creating educational content only, while PCG techniques create levels and other game content.

6.2. Development

The development process of PCG methods introduced a number of issues which have to be fixed in order to have the effective results. The cellular Automata algorithm which was used for procedural map generation did not provide outputs which could be immediately used in the game. Sometimes the algorithm would generate a map with many isolated sections, that is, the parts of the map which were not reachable. Another issue was that larger maps lose natural-looking cave-like shape. It was necessary to tune up the results a bit in order to create maps which are satisfying the requirements. The map generation process is needed to provide the random maps, however, those maps are only assets and it is still needed to make a playable game based on those maps. For that purpose, it was necessary to come up with the region detection algorithm which will be used to identify potential regions where the game elements can be instantiated. Building a custom UI was very hard and time-consuming process, however, it brought a number of benefits. There are several reasons why it was needed to build a custom UI and one of the most important was to improve the user experience and provide the the rich usability experience. Another reason was that Unity3d default UI did not work as expected on Android devices as it was really hard to type the code. That is why it was necessary to come up with a solution which will simplify that process. One more benefit of the custom UI is that it looks just like an IDE on a desktop platform as it provides all core functionalities which can be found in any desktop IDE. Definitely, the most complex component in the programming playground part was the virtual keyboard, as it was needed to support all core functionalities like typing of letters, number, symbols, as well as switching between lower and upper case letters. Also, it was necessary to add support for special commands like space, enter, backspace, and most importantly, commands for positioning the text cursor.

6.3. Evaluation

The evaluation and testing consisted of two separate questionnaires, conveyed at different stages of the project. The initial testing was performed in the early stage of development, that is, once the prototype was ready. It helped to better understand how players are interacting with the game and improve the controls and gameplay. That improvement reflected in the final testing as the majority of the participants stated that the gameplay was easy to understand and the controls were straightforward. The participants confirmed that they find the game very interesting and engaging. It is a good idea to always have a working prototype so that the game can be constantly tested with users and their feedback applied. Another benefit is that users can easily detect potential issues and bugs, so all of that can be fixed in the early stage of development, which will take much less time and effort then to make those corrections when the game is completed. When organizing an evaluation, it is very useful to use online tools for conveying surveys. In the initial testing Polleverywhere[1] was used to create a questionnaire and collect users' feedback. However, to draw conclusions it was necessary to manually calculate mean, standard deviation, min and max values, percentiles of every group, etc. That process takes some time and is prone to errors, therefore, for final testing, it was decided to use LimeSurvey[2] as that tool is capable of calculating all of that, which saves a lot of time and effort, so that one can only focus on understanding obtained results and feedback.

[1] https://www.polleverywhere.com/
[2] https://www.limesurvey.org/

7. Conclusion and Future Work

This chapter presents the overview of the entire project and gives ideas for future work. Research questions of the thesis are summarized and outlook to future research is presented.

7.1. Conclusion

The primary goal of the project is to use procedural generation to enhance engagement of players and motivate them to improve their knowledge by playing an educational game. Another goal is to examine whether it is possible to create an educational game based on PCG and run it on mobile devices, given that they have very limited processing power, screen size, and input. This work presented the design, implementation, and evaluation of an educational multidisciplinary video game using PCG techniques. Educators usually teach both theoretical and practical aspects of the certain topics. In order to acquire theoretical knowledge of a course, players have to explore maps, defeat enemies and collect all the disks which are spread out the map, and which will reveal a piece of information once they are all collected. Players have to read that information before taking a question which will test how good they understood what they have read previously. If they do not provide the correct answer, they have to read the description again and retake the test before they can proceed to the next task. With procedural map generation, it is ensured that the players will get different maps on each run. Cellular Automata and Perlin noise are two PCG methods which are combined to create natural-looking terrains and separate walkable from the non-walkable area. Custom UI and drag-and-drop blocks are two main components of the playground part which is designed to convey the practical knowledge of a course. The playground in the programming course, helps

players to understand the basic concepts of programming and write their first programs using Python code. The final testing showed that players find the game very interesting and motivating. The majority of the participants (10 out of 12) stated that it is a good idea to use "sCool" as a supplement to learning, as well as, that it is most suitable for teaching STEM courses like programming, math, electrical engineering, and physics.

7.2. Future Work

All core functionalities implemented in the game are working fine, however, the future work can be based on a number of improvements of different aspects of the game. One of the ideas in the exploration part could be to infinitely extend the map size, which can be based on generating maps of fixed size when a player approaches an edge of the screen. This will create a feeling that the environment is endless. Another feature in exploration part which can be improved is the region detection algorithm. Currently, the algorithm works in a way that it splits a map into a number of regions of equal size. The region detection process could be based in future on binary space partitioning in order to generate regions of different sizes.

Playground part of the programming course can be further improved by coloring Python specific keywords like for, if, new, etc. This will create a richer programming experience as it will help distinguish between keywords and other commands. Also, it would be nice to replace the red cursor functionalities with native Android or iOS virtual cursor. It is not possible to use default one at the moment, as it only works with the text fields, therefore, it would be necessary to create a component which works with the code blocks. Currently, there is only one fully implemented course which is programming, however, for some other courses, it will be necessary to come up with different playgrounds and challenges. Also, there is only one game type in exploration part (Shoot 'em up), but it would be nice to have different game types so that educators can specify which type of game their students should play. If the majority of a class are girls, then it would be a good idea to choose a match-three as a game type. This can be further improved by allowing educators to specify different groups and

assign different game types to those groups. In that way, girls could play a match-three game while boys play shoot 'em up. It is important that even though different groups play different genre, they still have the same curriculum and experience the same level of difficulty. It would be nice to have different enemy types as well, which could improve the gameplay and enhance engagement. For example, enemy units could have different abilities and attack styles (melee, rage, AOE - Area of Effect) so that players have to think about different strategies when fighting different enemies.

Finally, the best way to motivate students to learn more about programming and improve their programming skills is to show them how programming in the game could affect the real world. For that purpose, it would be a good idea to build a physical robot and deploy the code which players type in the game to the robot, so they can see their code in action. That would show them that the code they type in the game is working in a real environment and additionally improve their motivation.

Appendix

References

Amory, A., Naicker, K., Vincent, J., & Adams, C. (1999). The use of computer games as an educational tool: identification of appropriate game types and game elements. *British Journal of Educational Technology*, *30*(4), 311–321.

Amory, A., & Seagram, R. (2003). Educational game models: conceptualization and evaluation: the practice of higher education. *South African Journal of Higher Education*, *17*(2), 206–217.

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, *23*(3), 345–405.

Baghdadi, W., Eddin, F. S., Al-Omari, R., Alhalawani, Z., Shaker, M., & Shaker, N. (2015). A procedural method for automatic generation of spelunky levels. In *European conference on the applications of evolutionary computation* (pp. 305–317).

Banks, J. A. (1988). *Multiethnic education: Theory and practice*. ERIC.

Bartle, R. (1996). Hearts, clubs, diamonds, spades: Players who suit muds. *Journal of MUD research*, *1*(1), 19.

Berto, F., & Tagliabue, J. (2017). Cellular automata. In E. N. Zalta (Ed.), *The stanford encyclopedia of philosophy* (Fall 2017 ed.). Metaphysics Research Lab, Stanford University. `https://plato.stanford.edu/archives/fall2017/entries/cellular-automata/`.

Betz, J. A. (1995). Computer games: Increase learning in an interactive multidisciplinary environment. *Journal of Educational Technology Systems*, *24*(2), 195–205.

Bogacheva, A. (2017). *Subway surfers continued success.* Retrieved from `http://www.pocketgamer.biz/news/64936/subway-surfers-continued-success-jan-2017/`

Brockmyer, J. H., Fox, C. M., Curtiss, K. A., McBroom, E., Burkhart, K. M., & Pidruzny, J. N. (2009). The development of the game engagement ques-

tionnaire: A measure of engagement in video game-playing. *Journal of Experimental Social Psychology*, *45*(4), 624–634.

Browne, C. (2017). *Yavalath.* Retrieved from http://www.cameronius.com/games/yavalath/

Chang, V., & Guetl, C. (2010). Generation y learning in the 21st century: integration of virtual worlds and cloud computing services. In *Global learn* (pp. 1888–1897).

Clune, J., & Lipson, H. (2011). Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In *Ecal* (pp. 141–148).

Clune, J. E. (2007). Heuristic evaluation functions for general game playing. In *Aaai* (Vol. 7, pp. 1134–1139).

Collinge, R. (2017). *The importance of visual appeal in web design.* Retrieved from http://blog.usabilla.com/visual-appeal-web-design/

Davies, C., & Eynon, R. (2013). *Teenagers and technology*. Routledge.

De Aguilera, M., & Mendiz, A. (2003). Video games and education:(education in the face of a "parallel school"). *Computers in Entertainment (CIE)*, *1*(1), 1.

De Berg, M., Cheong, O., Van Kreveld, M., & Overmars, M. (2008). *Computational geometry: Introduction*. Springer.

DesignContest. (2017). *Inspiration gallery: Low poly art.* Retrieved from https://www.designcontest.com/blog/inspiration-gallery-low-poly-art

Dewey, J. (1904). The relation of theory to practice in education.

EBGames. (2017). *Tom clancy's the division.* Retrieved from https://ebgames.com.au/xbox-one-162912-Tom-Clancys-The-Division-Xbox-One

Eiben, A. E., Smith, J. E., et al. (2003). *Introduction to evolutionary computing* (Vol. 53). Springer.

Elkner, J. (2000). Using python in a high school computer science program. In *Proceedings of the 8th international python conference* (pp. 2000–01).

Eskerda. (2017). *Dungeon generation using bsp trees.* Retrieved from https://eskerda.com/bsp-dungeon-generation/

Federoff, M. A. (2002). *Heuristics and usability guidelines for the creation and evaluation of fun in video games* (Unpublished doctoral dissertation). Indiana University Bloomington.

Foord, M., & Muirhead, C. (2009). *Ironpython in action*. Manning Publications Co.

Gamasutra. (2017). What game developers are saying about no man's sky. Retrieved from http://www.gamasutra.com/view/news/279284/What_game_developers_are_saying_about_No_Mans_Sky.php

Gamepedia, N. M. S. (2017). *Species.* Retrieved from https://nomanssky.gamepedia.com/Species

Gentile, D. A., Lynch, P. J., Linder, J. R., & Walsh, D. A. (2004). The effects of violent video game habits on adolescent hostility, aggressive behaviors, and school performance. *Journal of adolescence*, *27*(1), 5–22.

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning, 1989. *Reading: Addison-Wesley*.

Grandell, L., Peltomäki, M., Back, R.-J., & Salakoski, T. (2006). Why complicate things?: introducing programming in high school using python. In *Proceedings of the 8th australasian conference on computing education-volume 52* (pp. 71–80).

Hanus, M. D., & Fox, J. (2015). Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. *Computers & Education*, *80*, 152–161.

Hatfield, T. (2017). *Rise of the roguelikes: A genre evolves.* Retrieved from http://pc.gamespy.com/pc/ftl-faster-than-light/1227287p1.html

Hauser, E. (2017). *Why python is a great first language.* Retrieved from http://blog.trinket.io/why-python/

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, *9*(1), 1.

Horn, B., Dahlskog, S., Shaker, N., Smith, G., & Togelius, J. (2014). A comparative evaluation of procedural level generators in the mario ai framework.

Johnson, C. (2017). *Lessons learned: Optimizing a procedural dungeon generator in xna.* Retrieved from http://chrisejohnson.com/wp/lessons-learned-optimizing-a-procedural-dungeon-generator-in-xna/

Kerlow, I. V. (2004). *The art of 3d: computer animation and effects*. John Wiley & Sons.

Klopfer, E. (2008). *Augmented learning: Research and design of mobile educational games*. MIT press.

Kojić, A. (2017). *Design and Implementation of an Adaptive Multidisciplinary Educational Mobile Game* (Unpublished master's thesis). Graz University of Technology, Austria.

Kun, J. (2017). *The cellular automaton method for cave generation.* Retrieved from https://jeremykun.com/2012/07/29/the-cellular-automaton-method-for-cave-generation/

Lague, S. (2017). *Procedural cave generation.* Retrieved from https://github.com/SebLague/Procedural-Cave-Generation

Laja, P. (2017). *First impressions matter: The importance of great visual design.* Retrieved from https://conversionxl.com/blog/first-impressions-matter-the-importance-of-great-visual-design/

Leutner, D. (1993). Guided discovery learning with computer-based simulation games: Effects of adaptive and non-adaptive instructional support. *Learning and Instruction*, *3*(2), 113–132.

Maksym, L. (2017). *Technology as a medium for learning.* Retrieved from http://serendip.brynmawr.edu/exchange/alesnick/technology-medium-learning

Marvie, J.-E., Perret, J., & Bouatouch, K. (2005). The fl-system: a functional l-system for procedural geometric modeling. *The Visual Computer*, *21*(5), 329–339.

MathChief. (2017). *No man's sky - the pathfinder update gameplay.* Retrieved from https://www.youtube.com/watch?v=AYgnMmTNXqs

McPhee, N. F., Poli, R., & Langdon, W. B. (2008). Field guide to genetic programming.

Mentzelopoulos, M., Parrish, J., Kathrani, P., & Economou, D. (2016). Revr-law: An immersive way for teaching criminal law using virtual reality. In *International conference on immersive learning* (pp. 73–84).

Mojang. (2017a). *Minecraft.* Retrieved from https://minecraft.net

Mojang. (2017b). *Minecraft.* Retrieved from https://education.minecraft.net/

Mueller, J. P. (2010). *Professional ironpython*. John Wiley & Sons.

Naylor, B. F. (1998). A tutorial on binary space partitioning trees. In *Computer games developer conference proceedings* (pp. 433–457).

Neal, L. (1990). Implications of computer games for system design. In *Proceedings of the ifip tc13 third interational conference on human-computer interaction* (pp. 93–99).

Nebel, S., Schneider, S., & Rey, G. D. (2016). Mining learning and crafting

scientific experiments: a literature review on the use of minecraft in education and research. *Journal of Educational Technology & Society*, *19*(2), 355.

Nestorgames. (2017). *Yavalath.* Retrieved from http://www.nestorgames.com/#yavalath_detail

Orchestra, P. (2017). *Sound samples.* Retrieved from http://www.philharmonia.co.uk/explore/sound_samples

Oré, J. J. (2017). *Voxel art - variados.* Retrieved from https://www.domestika.org/en/projects/248754-voxel-art-variados

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements engineering and agile software development. In *Enabling technologies: Infrastructure for collaborative enterprises, 2003. wet ice 2003. proceedings. twelfth ieee international workshops on* (pp. 308–313).

Patel, A. (1991). Polygonal map generation for games.

Perlin, K. (2017). *Making noise.* Retrieved from https://web.archive.org/web/20160308022101/http://noisemachine.com:80/talk1/index.html

Pirker, J., Guetl, C., & Astatke, Y. (2015). Enhancing online and mobile experimentations using gamification strategies. In *Experiment@ international conference (exp. at'15), 2015 3rd* (pp. 224–229).

PopCapGames. (2017). *Plants vs. zombies.* Retrieved from https://www.playstation.com/fi-fi/games/plants-vs-zombies-ps3

Prusinkiewicz, P., & Hanan, J. (2013). *Lindenmayer systems, fractals, and plants* (Vol. 79). Springer Science & Business Media.

Prusinkiewicz, P., & Lindenmayer, A. (2012). *The algorithmic beauty of plants*. Springer Science & Business Media.

Ralby, A., Mentzelopoulos, M., & Cook, H. (2017). Learning languages and complex subjects with memory palaces. In *International conference on immersive learning* (pp. 217–228).

RedBlobGames. (2017). Making maps with noise functions.

Relogic. (2017). *Terraria.* Retrieved from https://terraria.org/

Risi, S., Lehman, J., D'Ambrosio, D. B., Hall, R., & Stanley, K. O. (2016). Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games*, *8*(3), 244–255.

Roussou, M. (2004). Learning by doing and learning through play: an exploration of interactivity in virtual environments for children. *Computers*

in Entertainment (CIE), *2*(1), 10–10.

Sabha, M., & Dutré, P. (2007). *Feature-based texture synthesis using voronoi diagrams* (Tech. Rep.). Technical Report CW 492, Department of Computer Science, KU Leuven.

Schank, R. C., Berman, T. R., & Macpherson, K. A. (1999). Learning by doing. *Instructional-design theories and models: A new paradigm of instructional theory*, *2*, 161–181.

Shaffer, C. (2017). *The pr quadtree.* Retrieved from `http://lti.cs.vt.edu/Books/Everything/html/PRquadtree.html`

Shaker, N., Togelius, J., & Nelson, M. J. (2016). *Procedural content generation in games: A textbook and an overview of current research.* Springer.

Šisler, V., & Brom, C. (2008). Designing an educational game: Case study of'europe 2045'. *Transactions on edutainment I*, 1–16.

Smith, G. (2014). The future of procedural content generation in games. In *Proceedings of the experimental ai in games workshop.*

Smith, G. (2015). An analog history of procedural content generation. In *Fdg.*

Spelunky. (2017). *Spelunky.* Retrieved from `http://www.spelunkyworld.com`

Squire, K. (2003). Video games in education. In *International journal of intelligent simulations and gaming.*

Squire, K. (2011). *Video games and learning: Teaching and participatory culture in the digital age. technology, education–connections (the tec series).* ERIC.

Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. (2011). What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2nd international workshop on procedural content generation in games* (p. 3).

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 172–186.

Walton, M. (2012). Minecraft in education: how video games are teaching kids. *GameSpot. CBS Interactive. Retrieved December*, *15*.

Weibell, C. J. (2017). *Principles of learning: 7 principles to guide personalized, student-centered learning in the technology-enhanced, blended learning environment.* Retrieved from `https://principlesoflearning.wordpress.com/dissertation/chapter-4-results/themes-identified/repetition/`

Wen, H. (2017). Interview: Frugal fragging with .kkrieger. Retrieved from http://www.gamasutra.com/view/feature/2277/interview_frugal_fragging_with_.php

Wikidot, P. (2017). *Conway's game of life.* Retrieved from http://pcg.wikidot.com/pcg-games:conway-s-game-of-life

Wikipedia. (2017a). *Minecraft.* Retrieved from https://en.wikipedia.org/wiki/Minecraft

Wikipedia. (2017b). *Spelunky.* Retrieved from https://en.wikipedia.org/wiki/Spelunky

Zelle, J. M. (2004). *Python programming: an introduction to computer science*. Franklin, Beedle & Associates, Inc.

Appendix A.

Questionnaire

A.1. Part 1

Game Engagement Questionnaire (Brockmyer et al., 2009):

	Strongly disagree	Disagree	Undecided	Agree	Strongly agree
I loose track of time.	o	o	o	o	o
Things seem to happen automatically.	o	o	o	o	o
I feel different.	o	o	o	o	o
I feel scared.	o	o	o	o	o
The game feels real.	o	o	o	o	o
If someone talks to me, I don't hear them.	o	o	o	o	o
I get wound up.	o	o	o	o	o
Time seems to kind of stand still or stop.	o	o	o	o	o
I feel spaced out.	o	o	o	o	o
I can't tell when I'm getting tired.	o	o	o	o	o
Playing feels automatic.	o	o	o	o	o
My thoughts go fast.	o	o	o	o	o
I loose track of where I am.	o	o	o	o	o
I play without thinking about how to play.	o	o	o	o	o
Playing makes me feel calm.	o	o	o	o	o
I play longer than I mean to.	o	o	o	o	o
I really get into the game.	o	o	o	o	o
I feel like i just can't stop playing.	o	o	o	o	o
I don't answer when someone talks to me.	o	o	o	o	o

Table A.1.: GEQ

A.2. Part 2

Motivation - Please rate between 1 (fully disagree) and 7 (fully agree):

	1	2	3	4	5	6	7
It is a good idea to use sCool for learning.	o	o	o	o	o	o	o
sCool is a good supplement to regular learning.	o	o	o	o	o	o	o
I learned something with sCool.	o	o	o	o	o	o	o
sCool makes the content more interesting.	o	o	o	o	o	o	o
sCool makes the content easier to understand.	o	o	o	o	o	o	o
[sCool makes learning more engaging.	o	o	o	o	o	o	o
sCool makes learning more interesting.	o	o	o	o	o	o	o
The experience with sCool inspired me to learn more about programming.	o	o	o	o	o	o	o
Learning with sCool was more motivating than ordinary exercises.	o	o	o	o	o	o	o
It makes course content more interesting to learn about.	o	o	o	o	o	o	o
I would rather like to learn programming with sCool than with traditional methods.	o	o	o	o	o	o	o
I find regular programming classes boring.	o	o	o	o	o	o	o
Learning programming by playing a video game was interesting.	o	o	o	o	o	o	o
Learning programming by playing a video game was more engaging than learning programming in a regular way.	o	o	o	o	o	o	o
I would rather use the sCool on my PC.	o	o	o	o	o	o	o
I would like to learn with sCool at home.	o	o	o	o	o	o	o
I would like to learn with sCool in the classroom.	o	o	o	o	o	o	o

Table A.2.: Motivation

A.3. Part 3

Previous experience and system specific questions:

- How often do you play mobile games? [1- never, 5 very often]
- How often do you use your smartphone for learning? [1 never, 5 very often]
- How experienced are you with programming? [1 not at all, 5 Expert]
- What did you like about sCool?
- What did you NOT like about sCool?
- I would like to use this game in class more often. [1 totally disagree, 5 totally agree]
- I found the gameplay easy to understand. [1 totally disagree, 5 totally agree]
- I though the controls were easy to use. [1 totally disagree, 5 totally agree]
- I felt the enemy AI is not difficult. [1 totally disagree, 5 totally agree]
- The gameplay mechanics worked correctly. [1 totally disagree, 5 totally agree]
- The animations were nice and smooth. [1 totally disagree, 5 totally agree]
- I liked the different level designs. [1 totally disagree, 5 totally agree]
- The pick ups were easy to find and collect. [1 totally disagree, 5 totally agree]
- I felt the game was not too violent. [1 totally disagree, 5 totally agree]
- The virtual keyboard was intuitive and easy to use. [1 totally disagree, 5 totally agree]
- Drag-and-drop components helped me to write the code faster and easier. [1 totally disagree, 5 totally agree]
- Sorting and nesting of code is very simple. [1 totally disagree, 5 totally agree]
- What subjects would you like to learn using this game?
- Who do you think would be a good target group for the game?