Master's Degree in Informatics Engineering
Dissertation
Final Report

Generative Game Design: A Case Study

João André Carvalho da Silva
jacds@student.dei.uc.pt

Adviser:
Licínio Gomes Roque
Date: 5th September 2017

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

UNIVERSITY OF COIMBRA

FACULTY OF SCIENCES AND TECHNOLOGY

DEPARTMENT OF INFORMATICS ENGINEERING

MASTER'S DEGREE IN INFORMATICS ENGINEERING

Generative Game Design: A Case Study

Author:
João Silva

Adviser:
Licínio Roque

September 5, 2017

"A game is an opportunity to focus our energy, with relentless optimism, at something we're good at (or getting better at) and enjoy. In other words, gameplay is the direct emotional opposite of depression."

Jane McGonigal, 2011

University of Coimbra

Abstract

Faculty of Sciences and Technology
Department of Informatics Engineering

Master's Degree in Informatics Engineering

Generative Game Design: A Case Study

by João Silva

Correctly defining video game parameters to evoke an intended gameplay experience is not an easy task, and doing it wrong may sometimes lead to a ruined experience. This project aims at employing procedural content generation methods to generate adaptive game environments of an original video game, based on gathered user data and thus allowing to find interesting parameter ranges to elicit the desired gameplay experience. To achieve this, a prototype with a procedural content generation algorithm responsible for creating game worlds was designed and implemented. After an integration with a framework used to analyse game data, several people performed gameplay tests. These tests helped in understanding the parameters to use in the generation method, eliciting the intended gameplay experience. The results allowed us to retrieve values that currently seem the best fit for the desired experience, but that can also be improved with more testing sessions. The framework had its issues but also aided in improving the parameters for the generation algorithm. This dissertation allowed me to learn to design and prototype both a video game and a generation algorithm while integrating them with an external framework.

Keywords: procedural content generation, video games, game design, experience-driven.

Acknowledgements

First, I would like to thank my adviser, Prof. Licínio Roque, for allowing me to work on a theme I am passionate about. Besides that, he helped me succeed during this thesis, providing valuable suggestions and guidance, and allowing me to conclude this dissertation.

Second, I would like to thank my colleagues from LabC62, at the Department of Informatics Engineering of the University of Coimbra, for sharing with me their feedback and enthusiasm during this thesis. More specifically, I have to thank Pedro Oliveira, Elisabete Simões, and Nuno Barreto for all their support. I have to make a special mention to Rui Craveirinha, for continuously helping me during this dissertation (especially during the integration with the server), for all the discussions about video games, and for his fun company.

Third, I would like to thank all my friends who helped and supported me during most of this dissertation: Joana Lopes, Fernando Rocha, Filipe Sequeira, Noé Godinho, João Tiago Fernandes, Tiago Andrade, Bruno Rodrigues, Ricardo Fonseca, Ana Oliveira, Eliana Carvalho, Sandra Silva, Luís Rocha, and many others I probably forgot to mention.

Fourth, I would like to thank my Italian friends from the course of Game Design for their constant support and enthusiasm for my thesis.

Fifth, I would like to thank everyone who tested my game, both during the usability phase and the gameplay phase. Without your help and time, I would not be able to have any data to analyse.

Finally, and probably most importantly, I have to thank with all my heart to my parents and sister. Without their support, I would not be the person I am today, and I would not have been able to come this far.

Contents

List of Figures

List of Tables

List of Abbreviations

AGE	Authorial Game Evolution
EDPCG	Experience-Driven Procedural Content Generation
GA	Genetic Algorithm
HTTP	Hypertext Transfer Protocol
MDA	Mechanics, Dynamics, and Aesthetics
PCG	Procedural Content Generation
PEM	Player Experience Modelling
UI	User Interface
UTC	Coordinated Universal Time

Chapter 1

Introduction

Today, games, specifically video games, have become an ordinary activity in people's daily lives. Ranging from games developed to play in small breaks to long games with complex narratives or challenging gameplay, all of them are played by many people starting with children and going up to older people, leading to a game being better fit for some people, but sometimes being too difficult or uninteresting for a particular group of individuals. Also, with all the advances in hardware available to play games, these have become more and more complex, making them harder and more demanding to develop, resulting in continuously growing teams. This is the main reason why it is important to create new technologies that may help to explore a game design space and its adaptiveness to enable specific forms of gameplay with several target audiences.

The Interaction Design and Game Design areas constitute rich fields of study, with many challenges and open methodological issues. The particular context of generative digital gameplay has been a target of research in previous studies, especially in combination with the goal of giving game designers ways to promote specific flavours of gameplay. A promising area of entertainment computing is the use of generative design techniques and algorithms to dynamically generate game content (e.g. graphics models and textures, music, sound) and the definition of game contexts themselves (game scenarios, interactions, flow, and conditions). These techniques can potentially be used as a basis for building self-adaptive playable media when coupled with monitoring and analysis of player behaviour.

Procedural Content Generation (PCG) is a powerful set of tools, providing a rather big variety of possible utilities in Game Design, being in supporting the developers in content creation, or during gameplay taking into account how the player interacts with the game.

In this thesis, we will study the potential for employing procedural content generation techniques together with gameplay analytics in the context of the design of an original game, to create engaging and playful experiences. This enables the generation of adaptive game environments, based on gathered user gameplay data, in order to better adjust the intended gameplay experience.

Overall we aim to develop a complete case of generative game design, to test the techniques together within a realistic setting of producing a new game prototype for which we do not yet know the details of how to elicit the desired gameplay experience. This case will also work as a practical test for the AGE framework.

The remainder of this report is structured as follows: Section 2 encompasses the State of the Art, where it is presented the research done on the field of Play, Games and Videogames, and Procedural Content Generation. Section 3 refers to the Objectives and Methodology, starting by describing the objectives and planned game concept, the methodology that is going to be used, and the work plan to be followed. Section 4 presents the initial game concept, depicting its characteristics and interaction. Section 5 documents the development process and outcome of the engineering of the game prototype featuring the PCG techniques and mechanisms to collect data and accept parameters from the AGE framework. Section 6 contains the evaluation information and its analysis, divided into usability testing and gameplay testing. Section 7 details future work. Finally, Section 8 concludes the dissertation while revealing lessons learned.

As appendices, this document includes the data required for the server configuration (A), the development log depicting all the tasks done during the prototype's development (B), a list containing the work backlog (C), and all the data gathered during both usability and gameplay testing phases (D and E, respectively).

Chapter 2

State of the Art

2.1 Play and Games

A game is a structured form of play, usually requiring one or multiple players, in which some key components must exist, such as rules, challenge and interaction. Games can be used to many ends and most of the times require some mental or physical aptitudes.

Huizinga, in his book "Homo Ludens", shows the importance of play in culture and society (Huizinga, 1944). Play can be considered older than culture, and not just human culture, but the culture in general, because all animals play and it was not the man that taught them so, leading to the conclusion that humans have not added anything new and essential to the core idea of play. Even on the animal level, play is not just a physiological or psychological aspect, going beyond that and becoming a significant function, and having some sense in it. Play can be seen as having three main characteristics:

1. Play is a voluntary activity, and with this freedom alone it marks itself away from the natural process, being something added to it as if it were an ornament. Children and animals play because they enjoy it, not because someone tells them to, and that is the freedom of play. The need for it comes from the enjoyment taken from it, turning itself away from the labor tasks, since it is done at leisure when someone has free time.

2. Play is not real life. Instead, it is a stepping out of that, into some kind of own world of activities. Every child playing knows she is only pretending, but that does not mean that play cannot have some seriousness in it since one always turns to the other.

3. Play is secluded and limited, since it is always within certain time and space limits, containing its own course and meaning. Once something is played, it will endure in the memory, and possibly be transmitted, and it does not matter how many times it repeats itself because it holds good not only as a whole but also within its structure.

In Play, there is always a tension element, meaning that the player is never certain of how to decide and end the issue because he always wants to accomplish or end something. But, besides his intentions, he is obliged to follow the rules of the game. Those rules are very important in the act of play, and if the player breaks or ignores them, the game loses its illusion. Inside a game, the laws and customs of real life become less important, since players do actions they probably would not do in real life. But in a game, many players end up forming communities with others, and even if real life rules do not apply to the game, some of those communities can carry

on even after the game is over, highlighting the play's social feature.

In its higher forms, play can derive from two aspects, being a contest for or a representation of something, being also possible to merge both. If there is a representation, then it means that there is a displaying of an object or mimicry of some phenomena to someone and, if we leave children's games and focus on sacred performances, an element of play can be found in them. With this in mind, we can say that play can, in fact, have seriousness to it.

Because the sphere of play is contained in the real-life world, at any moment the latter can impact the act of play, interrupting the game, affecting its rules or collapsing the play spirit. This brings one very essential feature, that the player must be aware that he is only pretending, and not in the real life.

2.2 Video Games

A video game has the characteristics of a traditional game, with the difference of being electronic. In this sense, it allows the human interaction to be done with the help of a user interface, providing visual feedback through a video device, such as a TV screen.

The origin of video games takes us back to the early 1950s, but they only became trendy in the 1970s and 1980s, with the introduction of arcade video games, gaming consoles and home computer games to the general public.

The first computer games were merely created for training and instructional ends, research, and demonstration. Since their systems were often dismantled after serving their purposes, they did not influence further developments in the industry. The first video game created only for entertainment was Tennis for Two (Higinbotham and Dvorak, 1958), figure 2.1, but was never considered to adapt into a commercial product, due to the impractical technology of the time. After that, the first widely available and influential computer game was Spacewar! (Russel, 1962), figure 2.2, but could not be turned into a commercial game, since the hardware needed to run it was costly. It was only in 1972 with the release of the arcade video game Pong (Atari, 1972), figure 2.3, that a video game was commercially released with a significant success and popularity. However, others came before it, such as Computer Space (Syzygy Engineering, 1971), figure 2.4, but did not reach any mainstream popularity (Wikipedia, 2017a).

Video games continued to span over the following years, with the console market having several generations, even splitting to the handheld consoles. With each new device, the hardware was always superior to the previously used, allowing games to be even more complex and complete, being it visually or technically.

Today, video games are a popular form of entertainment and a part of the world culture. Not only that, but the video game industry is one of the most profitable, and it keeps growing and gripping the audience without a break (Wikipedia, 2017a). (Wikipedia, 2013b)

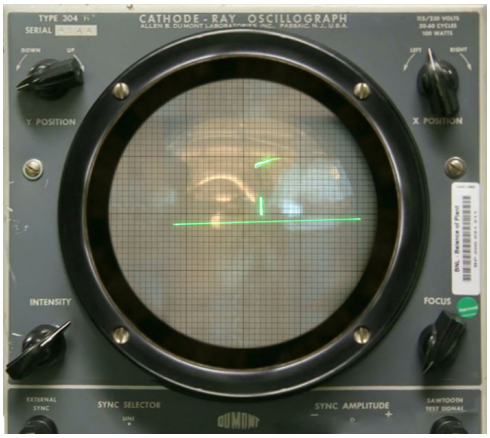

FIGURE 2.1: Tennis for Two being played on a osciloscope. (Wikipedia, 2013b)

FIGURE 2.2: Space-war! being played on a PDP-1 computer. (Wikipedia, 2010)

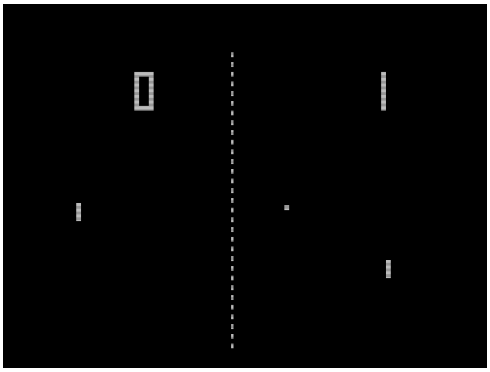

FIGURE 2.3: A screen-shot of the video game Pong. (Wikipedia, 2006)

FIGURE 2.4: Computer Space being played on its cabinet. (Pinterest, n.d.)

2.3 The Making of Video Games

The video game development process starts with an idea, but it must be worked on to become a good game concept. However, developing a game is not only about conception. Game development is a process integrated with the conception while including several other aspects of game production. Knowing this, said development process usually divides into four phases: the concept phase, the pre-production phase, the production phase, and the post-production phase. (Martinho, Santos, and Prada, 2014)

2.3.1 Concept

In this phase, an idea is worked on to define a game concept. It is usually done by a small team or even one person. However, many other people can be heard, and some ideas might come up from group discussions.

The objective is to have a sketch of the game, and convince someone (usually with the decision-making power) that the game is worth it, and should be invested in. To lead other people to believe in our game concept, it is important (and essential) to have a game prototype and a document that briefly describes it. Said document should describe the gameplay and highlight the innovative aspects of the game. The prototype should reinforce and illustrate what is written in the document. If everything goes well, the concept will be approved, going on to the pre-production phase.

2.3.2 Pre-Production

The objective of this phase is to show that it is indeed possible to develop the game. This means proving that it is conceivable to get and manage the needed resources and that there is a good work plan. Besides that, it is in this phase we define every detail in the game, such as the mechanics, challenges, characters, among others, with all of those details being written in a document. The process should be iterative, building various prototypes to test ideas.

During this phase, the team will grow because there will be a need for people from different areas, such as programmers and artists. After the team assembly, there should be work planning, allocating different tasks, planning important dates, and estimating the costs for the project.

One of the most important tasks of the pre-production is to choose the development tools, always dependent on the target platforms for the game. As such, there should be a clear list of platforms where the game should run right from the beginning. The programmers should assess the capabilities of said platforms building some technological prototypes, allowing for a better understanding of what tools are available to use. Artists should also look for tools to create artistic materials, making sure the game's necessities are met. However, usually, the majority of the constraints appear on the programmers' side, since there are usually bigger limitations, and as such these should be identified as soon as possible. Those limitations can include for example the graphic capabilities of the available tools, the algorithms' performance, or even the network performance.

What is expected to come out from this pre-production phase, and that will be the object of evaluation by the investors, is:

- A new document describing in detail the game concept.
- A document with a careful planning of the project, identifying all the team members.
- A document with the budget for the game, which may also include a business plan.
- A prototype of the game, with more quality and more complete than the one presented in the concept phase.

If no problems arise and the project is accepted, it advances to the production phase.

2.3.3 Production

The production phase is the one with the greatest investment from the entire development process. It is in this phase that game is going to be developed. The team should follow an agile development methodology because the development process should be flexible since there are always changes that were not earlier foreseen in the pre-production phase.

Due to the dynamism of the process, it is crucial to identify and list all the intermediate components needed, identify all the dependencies, and define priorities. Dependencies should be reduced to the essential, because the all the work should be done, preferentially, in parallel. For example, a programmer should not have to wait for graphical resources to continue his tasks.

However, even if working in parallel, there should be periodic integration between all the project components, having a playable prototype with all the elements available until that moment. This means that every component should be progressively developed, meaning that it should have some deliveries, with each one being more detailed than the previous. On the other hand, the most difficult parts should be the ones first worked on, not because of the time they take, but because of their uncertainty and risk. Most of the time those tasks are connected with the innovative or complex parts of the game. If they arise problems and are not relevant, the team should not insist, because sometimes there is the need of giving up on some components to prevent further harm to the project.

The production phase has three key goals, each one related to the creation of a game prototype.

1. **Alpha version**, containing all the programming components. This does not mean they will not be updated, but it is not expected any addition of new functionalities. All that is left to do is fix some errors, tune some gameplay components, add more content, and improve the visual part. With this version, the game is ready for an intensive gameplay tests phase, to make sure it complies with the concept.
2. **Beta version**, with no problems regarding programming, and the same goes for the art components. Everything is ready and integrated within the game. All the needed changes are related to refining the gameplay and experience. It

is usually tested in two phases: private and public. The latter usually happens after the former, also including the changes suggested during the private tests.

3. **Final version**, the game is no longer a prototype, and it is ready to release to the market.

The prototype is usually changed during the presented phases, meaning that there are usually many alpha and beta prototypes.

2.3.4 Post-Production

When a game reaches this phase, it means that it is ready to release to the market. However, the game may need to meet some criteria.

- It may have to pass the platform's quality control, and this usually happens with the consoles.
- It must be registered in a video game content rating system.
- It must be localised, preparing the game to be released in many countries. This implies not only translating the game but also adapting some details due to cultural differences.

It is also during this phase that promotional content is created and marketing investments are made. There may also be a release of a demonstration to allow the players to try the game before they decide to buy it. It is also good to communicate with the players, and forums, social media, official site, and others help in that regard. Also, this phase also includes the creation of all the commercial materials for the game's fans, such as manuals, strategy books, concept art books, and limited editions. Sometimes, the production phase does not end before the post-production phase starts.

2.3.5 Summary

As it would be expected, the developed prototype will not follow all the four steps usually required for a commercial game. However, it will go through the concept phase when it is decided what is going to be in the game and some possibilities for it. After that, the pre-production and production phases will more or less merge to develop said prototype. The alpha version will be the one used in the usability tests. The beta version will be first tested in the laboratory, and if everything goes well, it will then be tested by people outside it. Finally, our prototype will not go into a post-production phase, since it is not seen as a commercial game by now.

2.4 Game Design

Because game design is in its core having ideas and making decisions, Jesse Schell said that "Game design is the act of deciding what a game should be" (Schell, 2008). Moreover, those decisions and ideas are written into a Game Design Document, aiding in further discussions that may arouse given the decisions already taken, or even when new ideas appear.

Game design is focused on deciding numerous elements related to the game, such as its story, decisions about rules, look and feel, timing, pacing, risk-taking,

rewards, punishments, and everything else that may impact the player experience with the game.

For video games, even though it is not crucial in the process, it is helpful that game design takes other areas of the development into account. This allows better decisions to be made more quickly. As Schell said, "it is like the relationship between architects and carpenters: an architect does not need to know everything the carpenter knows, but an architect must know everything the carpenter is capable of" (Schell, 2008). Summarily, it is important for the game design process to take any constraints and limitations that may exist during its development into account.

Notice the *during its development* expression, because the game design process occurs during almost the whole time of development, since many decisions or ideas can change during the development stage, or they can be discarded for example for their unfeasibility, or even new ideas may appear. That said, game design is present in almost all the game development, allowing to make decisions on how it should be all along the way (Schell, 2008).

One of most used game design frameworks is MDA, standing for Mechanics, Dynamics, and Aesthetics. It is one of the few academic papers achieving wide exposure within the game industry, probably due to having experienced game designers as authors. (Schreiber and Sohn, 2013)

2.4.1 Mechanics, Dynamics, and Aesthetics

Mechanics, Dynamics, and Aesthetics, or MDA for short, was first presented by Hunicke et al. in “MDA: A Formal Approach to Game Design and Game Research” (Hunicke, LeBlanc, and Zubek, 2004). It is a formal approach to understanding games, trying to connect game design, development, criticism, and technical research. Games are different from other entertainment providers regarding its unpredictability since the results of gameplay are usually unknown when the product is finished. This framework breaks game consumption into three distinct components (Rules, System, Fun) and establishing their design counterparts (Mechanics, Dynamics, Aesthetics), as shown in figure 2.5.

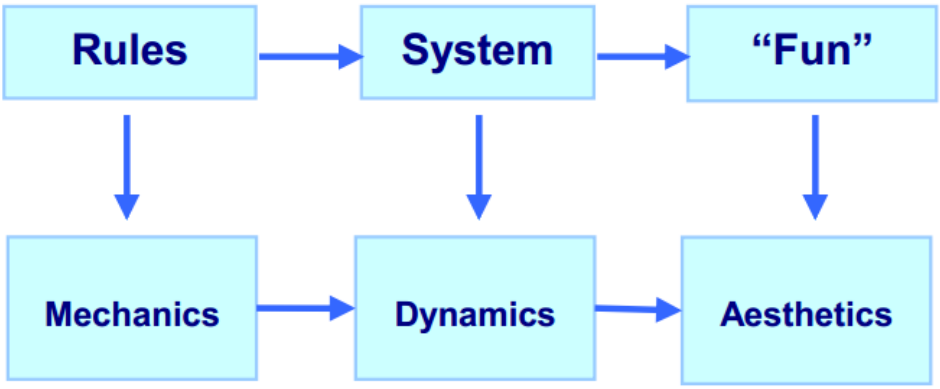

FIGURE 2.5: Components of game consumption and their design counterparts. (Hunicke, LeBlanc, and Zubek, 2004) (Adapted)

Each part of the MDA framework is linked to the others, even if separated from them. But those relations have different meanings to designers and players, as

shown in figure 2.6. Regarding the designer's perspective, mechanics lead to dynamic system behaviour, leading in turn to aesthetic experiences. However, the player has a different view on things, since aesthetics set the tone, leading to dynamics and usually to mechanics. As a result, it is important and helpful to consider both perspectives.

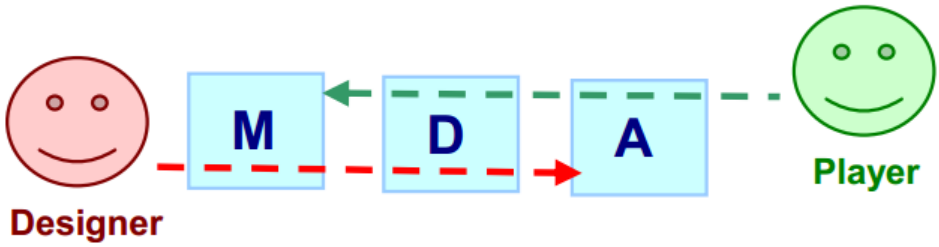

FIGURE 2.6: Designer and Player perspective on games. (Hunicke, LeBlanc, and Zubek, 2004)

Aesthetics can be seen as the desirable emotional responses stimulated in the player during the interaction with the game. When describing this component, a taxonomy can be considered, also helping us to describe games and understanding how and why different games appeal to distinct players. The taxonomy is shown below, with a definition of what a game aims to be with each term. (Hunicke, LeBlanc, and Zubek, 2004)

- **Sensation**, the game as sense-pleasure.
- **Fantasy**, the game as make-believe.
- **Narrative**, the game as drama.
- **Challenged**, the game as an obstacle course.
- **Fellowship**, the game as a social framework.
- **Discovery**, the game as uncharted territory.
- **Expression**, the game as self-discovery.
- **Submission**, the game as a pastime.

Besides helping the aesthetics component, this taxonomy also allows the definition of models for gameplay, which will then help to describe gameplay dynamics and mechanics.

Dynamics describe the runtime behaviour of the player when mechanics act upon him, leading to the creation of aesthetic experiences. Creating models for dynamics prevents common design flaws and allows the identification of feedback systems within gameplay, which will help to determine how some states or changes in it affect the overall state of gameplay.

Mechanics are all the actions, behaviours and control mechanisms available to the player during the game. Together with all the game's content, they will support the gameplay dynamics and, in order to tune the latter, it is needed that mechanics are adjusted.

This framework allows the discussion of the goals related to the aesthetics, how the dynamics will support them, and in the end, it helps figuring out the mechanics needed to satisfy both dynamics and consequently aesthetics.

2.4.2 Summary

Almost from the start of the design process of the game, we wanted to arouse the feeling of exploration and discovery in the player, while also presenting some difficulties to the player, to keep him engaged. In this sense, MDA helped to get there, because we knew what we wanted to arouse in the player, and what were the actions available to the player. However, we did not know how we could make the game respond to the player, while always keeping the desired feeling. MDA helped to connect all the parts of the game design process, leading to a well-thought prototype. Finally, it also helped on designing the procedural content generation algorithm to create game maps, allowing us to decide what should be included in it to arouse the feeling of exploration of uncharted territories while keeping the player aware of dangers that could exist in the map.

2.5 Procedural Content Generation

One of the significant aspects of the developed prototype is the inclusion of procedural content generation, allowing the game to generate a great part of its content. Not only that, but it will also adapt the content using gameplay data gathered from various sessions.

2.5.1 What is Procedural Content Generation?

In order to establish a good idea of what is Procedural Content Generation (PCG), we will start with a definition stating that "PCG is the algorithmic creation of game content with limited or indirect user input" (Togelius et al., 2011). Basically, PCG refers to a computer software that creates game content alone, or together with human designers or players.

Content can be described as almost everything that a game contains, namely: levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles, characters, and others (Togelius, Shaker, and Nelson, 2016). It is also important noting that the game engine and the non-player character artificial intelligence are not considered content.

By games, we are not just talking about video games, but also board games, card games, puzzles, etc. And because of that, a PCG system, in order to generate content for a game, must consider its design, affordances and constraints, because, in the end, that content must be playable.

2.5.2 Why use Procedural Content Generation?

Even though we now know what is Procedural Content Generation, one question remains: why do we need these technologies in Game Design? This question does not have a single answer, as it can depend on different reasons.

In order to get to the first answer, we need to look at some historical data. In the early days of home computers, in the 1980s, those machines had highly limited capabilities, such as storage limitations, and that constrained the space available to store game content. These factors were the main cause leading to an effort on developing PCG techniques. One of the first games to address this was Elite (Braben and Bell, 1984), in figure 2.7, which stored various seed numbers that were used to generate eight galaxies, each one with 256 planets different among them. Another example of that period is Rogue (A.I. Design, 1980), in figure 2.8, a dungeon-crawling game with randomly generated levels, every time a new game was started. Although this enables generating compelling experiences, it comes with the trade-off of most of the time lacking some visual appeal, for example in the video game Dwarf Fortress (Adams and Adams, 2006), in figure 2.9.

Going to the second answer, it addresses the fact that with procedurally generated content, we no longer need to have a human designer or artist to generate variations on that same content, because it can sometimes end up being an expensive and slow process. This is mainly due to the fact that the number of person-month needed to develop AAA games has been increasing constantly, since the debut of computer games, leading to the companies taking fewer risks in creating something different. If one game development company empowered some of its artists and designers with algorithms, it would have an advantage related to the amount of resources needed and, possibly, the quality of the final product. But, of course, people cannot be replaced with algorithms, instead, we can use those programs to help and increase the humans' creativity. In principle, this would make it possible for small teams, with fewer resources than the top companies in the game development business, to produce content-rich games.

Another possible answer is that the use of PCG may allow the creation of new types of games and not just games based on what already exists. In fact, if we have software that can generate game content at the same time it is being played, there is no reason why games need to end. (Togelius, Shaker, and Nelson, 2016) Even more, the generated content can ultimately adapt itself to the tastes and needs of the player. For example, Galactic Arms Race (Evolutionary Games, 2010), in figure 2.10, attempts to create and adapt weapons based on players statistics. These are called player-adaptive games, focusing on maximising the enjoyment of players, but can also focus on maximising other aspects, like the learning or addictive capacities of a game.

One more reason, already mentioned above, is the fact that PCG could enable creators to conceive different and original content since most of us tend to imitate other people's work in our own way. The given example was about empowering artists and designers with PCG methods, that could not just help them to achieve better content, but also new and original content.

The final answer and reason on why to use PCG stands in the fact of enabling us to understand design, since creating a program that could accurately generate content would help in understanding the process humans go through when manually creating content. And this can be seen as a loop, because if we develop PCG methods that better enables us to understand the design process and its results, then it will be possible to create even better PCG algorithms.

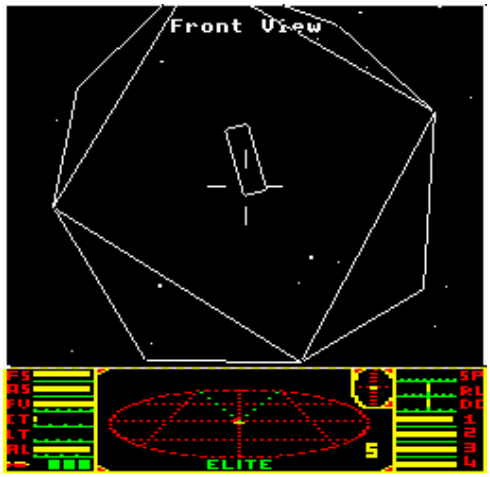

FIGURE 2.7: *Elite*, released in 1984. (Wikipedia, 2007)

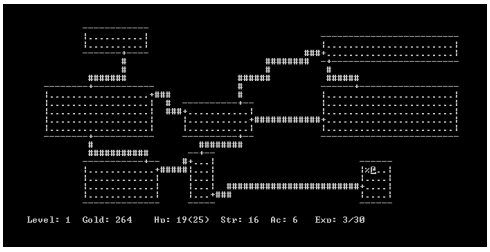

FIGURE 2.8: *Rogue*, originally developed around 1980. (Ritzl, 2016)

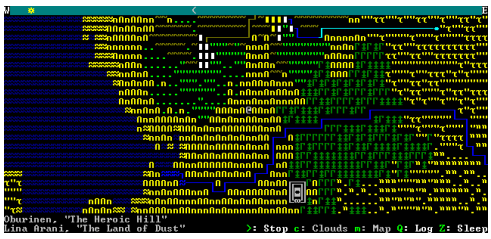

FIGURE 2.9: *Dwarf Fortress*, with its text-based graphics. (Bay 12 Games, n.d.)

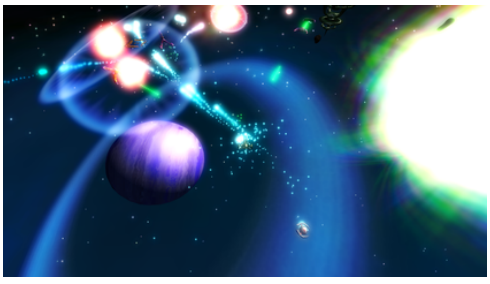

FIGURE 2.10: *Galactic Arms Race*, that adapts the weapons created to the player. (Wikipedia, 2013a)

2.5.3 Desirable properties of a Procedural Content Generation solution

There are many and different implementations of PCG methods, regarding problems in content generation. Those problems may not be the same, since generating a chunk of grass is not the same as generating a whole new idea for a game or helping a designer polish game content, meaning that all take their time and have different purposes. With this in mind, it is important to note that the desirable properties of a solution depend on the situation at hands and, as already mentioned before, there are always trade-offs, since it's hard to generate something fast and with really good quality.

Togelius et al. suggest a list of the most common desirable properties of PCG solutions (Togelius, Shaker, and Nelson, 2016):

- **Speed** - requirements for speed vary very much with the moment we want the content generated, amongst other factors. If we want to generate something during gameplay, the speed has to be in the order of milliseconds, while if we want content to be generated during development time, speed can go to a maximum of months.
- **Reliability** - some generators produce content without ever worrying about its quality, while others generate content always making sure it satisfies some given quality criteria. For example, if a generated level does not have an entrance or an exit, it is a huge failure regarding game design. However, if some generated grass looks weird, this will not automatically break the game.
- **Controllability** - most of the times the content generators need to be controlled in some way, allowing some of the aspects of the content to be generated to be specified by a human user or an algorithm. Most of this control will be based on parameters or fitness functions, which can control, for example, the colour of a vehicle or the number of rooms in a dungeon.
- **Expressivity and diversity** - in order to prevent content that seems like small variations on a theme, it is often needed that there is a diverse set of generated content. At an extreme of non-expressivity, a level generator can always create the same level, changing the colour of some element in it. On the other extreme, the generator creates entirely different levels that do not even make sense and are not playable. However, it is not easy to measure expressivity, and developing level generators that create content taking into account its quality, it is even harder.
- **Creativity and believability** - usually it is important to have PCG solutions that create content which does not look as if it was made by a machine, rather than a human.

2.5.4 Taxonomy by Togelius et al.

Togelius et al. originally presented a taxonomy, containing a number of classes, where a method or solution usually supposedly lies at a point between the ends of that class (Togelius et al., 2010). However, the taxonomy described here is a revised version of the original. (Togelius, Shaker, and Nelson, 2016)

- **Online versus Offline** - this first class compares the different moments where the PCG is applied: if the generation occurs while the player is playing the

game, it is online, otherwise if the content is generated during the development of the game or before a game session, it is offline. An example of online content generation can be found in Left 4 Dead (Valve Corporation, 2008), in figure 2.11, providing a dynamic experience for the player, by analysing his behaviour during gameplay and changing the game state accordingly. On the offline side, most of the games that use some map generation before starting the session apply. Not only those but also video games with content editors, like Spore (Maxis, 2008), in figure 2.12, allowing the creation of game creatures.

- **Necessary versus Optional** - generated content can be necessary, if it is essential to complete a level of the game, or optional if it is auxiliary and can at any time be discarded or exchanged. Necessary content must always be correct, since it can be, for example, the structure of a level, for instance in Super Mario Bros. (Nintendo Creative Department, 1985), in figure 2.13, and, as seen before, a level without entry or exit is a failure. On the other hand, optional content does not have those strict demands, imagine the different types of weapons in a shooter game.

- **Random Seeds versus Parameter Vectors** - PCG can be controlled in many ways, such as using random seeds or setting parameters. The first one helps to gain control over the generation space, and if the same seed is used, most of the previous game world will persist. For example, in The Binding of Isaac: Rebirth (Nicalis, 2014), in figure 2.14, seeds define, amongst other things, the items locations (Gamepedia, 2016). Setting parameters allows the generated content to fit some specifications, for example in Infinite Mario Bros (Shaker, Yannakakis, and Togelius, 2010), which used a vector of content features.

- **Generic versus Adaptive** - When generating content, the player's actions and behaviour may be taken into account, but most of the times, they are not. In the latter situation, we call it generic content generation, mostly used in commercial games. If player interaction with the game is taken into account to generate content, then we say we have adaptive content generation. One example that uses this type of generation is Left 4 Dead (Valve Corporation, 2008), having an algorithm that adjusts the difficulty of the game, keeping the player interested and engaged.

- **Stochastic versus Deterministic** - Most algorithms used in PCG are stochastic, which leads to usually different content with the same input parameters. However, some methods prefer to output the same content if given the same starting point and method parameters. One example of this is the game Elite (Braben and Bell, 1984).

- **Constructive versus Generate-and-test** - When using constructive PCG, all the content will be generated in one pass, as usually done in roguelike games when generating the dungeons (game levels). On the other hand, generate-and-test methods alternate between generating and testing, repeating the process until a suitable solution is found. Yavalath (Browne, 2007a), in figure 2.15, is a board game that was completely generated by a computer program applying generate-and-test techniques.

- **Automatic generation versus Mixed authorship** - Automatic generation allows limited input from game designers, enabling them just to tweak some algorithm parameters, controlling and guiding content generation. However,

mixer authorship focuses on letting the designer or player cooperate with the algorithm, to generate the desired content. One example of this is the game Tanagra (Smith, Whitehead, and Mateas, 2010), in figure 2.16, where a designer draws part of a 2D level, and an algorithm will generate the missing elements while making sure the level is playable.

FIGURE 2.11: *Left 4 Dead*, released in 2008. (Electronic Arts, 2009a)

FIGURE 2.12: *Spore*, released in 2008. (Electronic Arts, 2009b)

FIGURE 2.13: *Super Mario Bros.*, released in 1985. (Wikipedia, 2011)

FIGURE 2.14: *The Binding of Isaac: Rebirth*, released in 2014. (PlayStation, n.d.)

FIGURE 2.15: *Yavalath*, a board game generated by a computer. (Browne, 2007b)

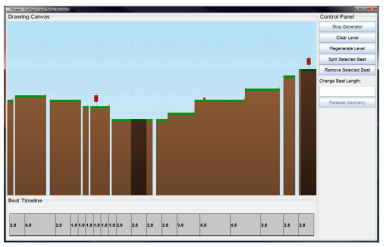

FIGURE 2.16: *Tanagra*, a mixed-authorship tool. (Smith, Whitehead, and Mateas, 2010)

2.5.5 Taxonomy by Gillian Smith

Gillian Smith, in "Understanding Procedural Content Generation: A Design-Centric Analysis of the Role of PCG in Games" shows five main categories that a content generator can fall into (Smith, 2014), shown below.

- **Optimisation** - This type of approaches considers the design process as a search for the combination of elements that best fit some criteria. Those elements can be specified mathematically by the system creator, or validated and curated by a human. Most PCG approaches are usually computationally expensive, which makes them difficult to use in games that require an extremely responsive PCG system. Most of the time, these techniques are used with a human-in-the-loop for the evaluation function and to generate personalised content offline.
- **Constraint Satisfaction** - This approach involves specifying the properties and restrictions of the generated content. The challenge is in determining a correct representation of facts related to the content, and debugging the complex set of constraints. This approach is usually used in tools for designers.
- **Grammars** - Using grammars that the algorithm should expand upon to create content, these methods attempt to reach a balance between designer-specified rules for how content components should fit together and computer exploration of the design space through expanding the grammar. This approach has been used in the offline content generation for games and tools for designers.
- **Content Selection** - There is still some disagreement whether picking content from a library and piece together can be considered complex enough to qualify as a content generation approach. However, Gillian Smith believes that no matter how simple this method is since it is used to create an environment or content for the player procedurally, it can be considered PCG (Smith, 2014). As already said, content selection is a rather simple form of PCG. However, it is also one of the fastest ways to generate content, being usually applied to games where the generator must run during play time, such as "endless runner" games.
- **Constructive** - A constructive generator builds content in an ad hoc way by piecing together customised building blocks. Even though the algorithm may perform some internal search, it does not test the results of the level against any external heuristic to help guide the search process. Roguelike games usually use this kind of methods.

2.5.6 Taxonomy by Craveirinha et al.

Craveirinha et al. in "Towards a Taxonomy for the Clarification of PCG Actors" propose "a classification system that allows for a clarification of the multiple role configurations that human designers, PCG algorithms and players can have in the context of PCG-aided video-game design and production" (Craveirinha, Barreto, and Roque, 2016).

For that, they divided the content-generation into two moments: solution generation and solution evaluation. Also, they divided the actors into designer, player and computer. By designer, they meant every human creator that impacts the design of the game during production and before its release. A player is a human who

plays the game typically after being released. The computer refers to the algorithms applied either during game-production or gameplay that with complete or partial autonomy are able to generate content. All the relations are shown in figure 2.17, being also explained below.

We will first focus on the **designer** and state every aspect on how he can impact evaluation and creative processes in a video game. Regarding **evaluation**, we have three core divisions.

- **None** - no designer input will be considered by the computational agent, regarding the evaluation processes. This also assumes that the designer is not the author of the PCG solution.
- **Implicit** - any aspect the designer might indirectly influence the solutions evaluation by the designer-computer-player complex. This approach divides into Editorial Control, and PCG-Design.
- **Explicit** - all the possible ways in which the designer directly influences how solutions are evaluated. This approach divides into Quality Assessment, and Quality Definition.

Moving on to the **generation** part of the **designer**, we have five main categories.

- **None** - the computational system creates all the aspects of the generated content, with no help from the designer, including no designer-authored content.
- **Configuration** - when the games' creators interact with a pre-programmed PCG method, offering some degree of customisation to its functioning. This approach divides into Method Selection, Method Parametrisation, and Content Parametrisation.
- **Base-Design** - the generation methods need human-authored content to find new solutions. It is divided into Idea, Experiential Chunk, Template, Component Patterns, and Subcomponent.
- **Co-Design** - PCG methods assign both computational and human actors decisive intervention in the creative act.
- **Meta-Design** - designers create their PCG algorithm, that will, in turn, generate game content.

We will now talk about the evaluation and generation processes carried out by the **computational agent**. Starting with the **evaluation**, we have three categories.

- **None** - PCG methods may not have any computational impact on the content evaluation.
- **Implicit** - the algorithm biases the generation process, making it seem that there is some form of computational content evaluation.
- **Explicit** - the computational agent enacts a formal act of content quality judgement. It is divided into the subtypes Content-based, and Player Experience-based. The former also divides into three other sub categories: Heuristics, Simulation-based, and Experience Inference.

Still on the **computational** part, we will focus on the algorithmic processes comprising the **generation**, and it divides into three categories.

- **Content-type** - nature of the generated content. It is divided into Derived, Scenarios, Systems, Space, Decorative, and Design.
- **Phase** - about when the procedural generation happens, and it divides into Design-Time, Pre-play, and Play-Time.
- **Strategy** - understanding how the computational agent proceeds to generate content, and it splits into Optimisation, Constraint Satisfaction, Grammar Derivation, Content Selection, and Constructive.

Finally, it will be seen how the **player** can affect the evaluation of the generated content, and how his/her interaction with the video game impacts the content generation. Starting with **evaluation**, we have three categories.

- **None** - the non-existence of any evaluation by the player, regarding generated content.
- **Implicit** - evaluation based on the player's behaviour, usually done by analysing gameplay metrics. It is divided into Preference Inference, and Experience Model-based.
- **Explicit** - the PCG system uses subject-data, typically in the form of questionnaires, to determine how to assess generated content and generate new content. It divides into Preference, and Experience Self-Evaluation.

Referring to every **player** interaction with a video game in the **generation** process, we have a division into four categories.

- **None** - no player actions are taken into account when generating new solutions.
- **Gameplay** - Play-time PCG considers in-game interactions in order to select content to subsequently generate.
- **Parameterisation** - the player can adjust some input parameters to determine some aspects of the generated content.
- **Co-design** - the player can create or edit content together with the computational agent.

2.5.7 Summary

After a long introduction to procedural content generation, we can connect some of its points to what will exist in the prototype. Even if also being based on the first two taxonomies presented, our approach will focus mostly on the third taxonomy, since we will be using gameplay metrics to adjust the generated content. PCG will be used to generate the game world before each game session, using input parameters in the algorithm. Those parameters will vary to optimise their combinations and therefore find the best ones. Also, because the generation occurs before a game session, it can be seen as a generic, offline and necessary method. The algorithm will always make sure that each world it generates is playable, meaning that a player can go from the start to the end without being blocked along the way. Finally, we will take into account gameplay metrics and the outcome of each session (e.g. if the player successfully finished the level) to optimise the algorithm's input parameters.

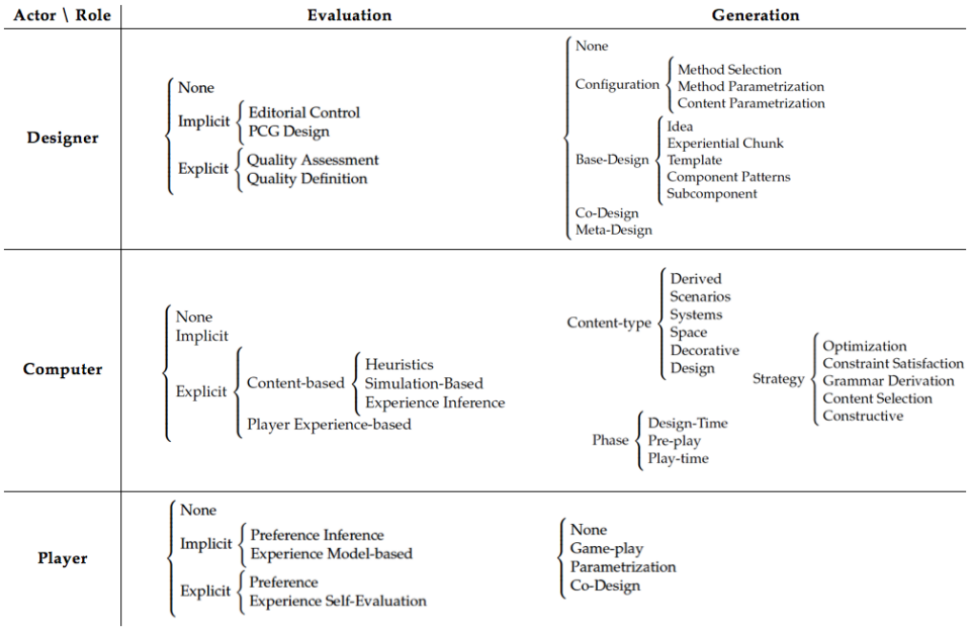

FIGURE 2.17: Taxonomy proposed by Craveirinha et al. (Craveirinha, Barreto, and Roque, 2016)

2.6 Experience-Driven Procedural Content Generation

Experience-Driven Procedural Content Generation (EDPCG) was originally mentioned by Yannakakis and Togelius (Yannakakis and Togelius, 2011), and later revisited by the same authors (Yannakakis and Togelius, 2015).

With the proliferation of video games, players can be found in all parts of the world, meaning that there is a long range of skills, preferences and emotion elicitation to correspond to with one game. Because of that, there is an increasing need for tailoring the game experience to the individual, consequently raising the difficulty in user modelling and affective-based adaptation within games.

Although being able to use PCG in other domains, games are one of the best examples of rich and distinct content creation applications and provide unique user experiences. Based on that, EDPCG is defined as a generic and efficient approach to optimise user experience adapting the content (Yannakakis and Togelius, 2011). However, even if based on game technology and affective computing, EDPCG has managed to integrate other areas of research within human computer interaction.

Games are designed thinking in what affective experiences can be influenced by player's feedback, making them go through a range of emotions to heighten engagement, offering the best and most meaningful realisation of the affective loop. With this in mind, EDPCG offers complete realisation of affective interaction: eliciting emotion as a result of variant game content types, integrating game content to computational models of user affect and using game content to adapt the experience.

Since the quality of the generated content needs to be assessed, there are some key components of EDPCG:

- **Player experience modelling** - model player experience as a function of game content and player.
- **Content quality** - assess the quality of the generated content and link it to the modelled experience.
- **Content representation** - represent content accordingly, maximising search efficacy and robustness.
- **Content generator** - let the content search through the content space for content that optimises the experience for the player, taking the acquired model into account.

The key components shown will be described in more detail next, surveying them, presenting a taxonomy of approaches for each and, finally, outlining the main research challenges faced.

2.6.1 Player Experience Modelling

There are many approaches to Player Experience Modelling (PEM), based on the type of data retrieved from the players (Yannakakis and Togelius, 2011). Three main classes allow us to model player experience in games: subjective PEM, objective PEM and gameplay-based PEM, which can also combine into more capable hybrid methods (Yannakakis and Togelius, 2011). On a final point, PEM methods can only be used when the retrieved data includes scalar representation of experience, or classes and annotated labels of user states since it allows the use of some machine learning algorithms to build the affective models. Otherwise, if the experience is given in a pairwise preference format or ranking, we are facing a preference learning problem, where decision trees, artificial neural networks and support vector machines can be applied.

With **Subjective PEM**, the player is directly asked about his playing experience, building a model from the obtained data, and it may depend on either player's free-response during play or on forced data retrieved using questionnaires. Players can be asked to rate their experience or to compare and rank their experience in two or more different sessions of the game. Even if Subjective PEM provides accurate models, it has some limitations. Player's responses may be subjectively biased depending on his experience and self-deception effects, among others Also, self-reports can be intrusive, depending on when they appear, or sensitive to memory effects if a player is asked to express his experience after a long game session.

Objective PEM incorporates access to multiple modalities of player input, to model the affective state of the player during play. Most emotions reflect on the player's physiological and bodily behaviour, and monitoring it can assist in recognising and synthesising the emotional responses of the player. Implementations of this approach can be model-based, top-down with models deriving from emotion theories, or model-free, bottom-up with the construction of unknown mapping between modalities of player input and an emotional state representation via annotated data (e.g. facial expressions). However, these approaches are within a continuum space, meaning that all the approaches can be seen as hybrids between model-based and model-free. The main limitations of Objective PEM include its high intrusiveness, low practicality (combined with high complexity), and questionable feasibility since many modalities of player input are not plausible within the commercial

game development.

In **Gameplay-based PEM** it is assumed that player actions and real-time preferences link to player experience accepting that games may affect the player's cognitive processing patterns and focus, which may in turn influence emotions. The basis of this approach is formed by the elements derived from the player-game interaction, including parameters from player's behaviour derived from responses to system elements. This method classifies as model-based, model-free or a hybrid between the two. Model-based are inspired by a general theoretical framework of behavioural analysis or cognitive modelling, but there are game-specific theories about user affect as well. Model-free approaches normally comprise the processing and mining of the extensive sets of player data that modern games usually collect. Most features extracted are mapped to levels of cognitive states such as attention, challenge and engagement. Many measures have been used, such as performance and time spent on a task, weapons selected, the number of times the player died, but the chosen measures are usually game and genre related. Gameplay-based PEM is the most computationally efficient and least intrusive of the three presented approaches, but that also results in low-resolution models of playing experience and its affective component, since those models are usually based on assumptions relating the player experience to gameplay actions and preferences.

2.6.2 Content Quality

The fundamental meaning of the acquired player models is to assess the quality of game content items, being necessary for the generation phase, evaluating and using candidate content to generate a new one. The evaluation function has the task of evaluating the item and assigning it a scalar or vector of a real number, which will show the suitability for use in the game and promote the desired experience, using for that the PEM in some part (Yannakakis and Togelius, 2011).

Designing the evaluation function is not an easy task because we first need to decide what should be optimised and how to formalise it, since it may be desired to provide an entertaining, frustrating or challenging content. We can define direct, simulation-based, and interactive functions as the three core classes regarding evaluation functions.

With **direct evaluation functions**, some features are extracted from the generated content, being directly mapped to a content quality value. This mapping can be linear or nonlinear, usually not involving significant amounts of computation and being specifically tailored for the game and content type. One important distinction in this type of approaches is between theory-driven and data-driven functions. In the first ones, the designer is guided by intuition or some qualitative theory of emotion or player experience to derive a mapping between an experience model and the quality of content. On the other hand, data-driven functions are based on collecting data on the effect of various examples of content and the using automated means to adjust the mapping from content to player experience and ultimately to evaluation functions (Yannakakis and Togelius, 2011).

Simulation-based evaluation functions rely on an artificial agent playing through some part of the game involving the content being evaluated. Features, mapped to player experience models, are afterwards extracted from the observed gameplay

and used to determine the quality value of the content. There is a worth mentioning difference between static and dynamic simulation-based functions, since while in the former the agent does not change while playing the game, in the latter the agent changes and the quality value somehow incorporates this change. Even though simulations can be typically executed faster than real-time, simulation-based evaluation functions are usually more computationally expensive than direct ones (Yannakakis and Togelius, 2011).

In **interactive evaluation functions**, the fitness is evaluated during the actual gameplay, assigning a score to content based on interaction with a player. Data can be retrieved from the player explicitly (e.g. questionnaires) or implicitly (e.g. how often does the player interact with a certain piece of content, the intensity of button presses). This data allows tailoring the player experience models of that player, affecting the evaluation function of the content presented to him. As said before, the issue with explicit data is that it can interrupt gameplay, while implicit data can be noisy, inaccurate, delayed and of low-resolution (Yannakakis and Togelius, 2011).

2.6.3 Content Representation

Content can be represented symbolically within a tree or a graph data structure, and it is the common practice in the computational narrative community. While humans can easily read this type of representation, sometimes non-symbolic representations (e.g. artificial neural networks) might allow for more efficient search in many domains.

EDPCG primary focus is bottom-up, search-based approaches for generating content, and since most search-based PCG methods use evolutionary algorithms, an important matter is the mapping of genotypes into phenotypes. An important distinction is between direct and indirect encodings. In the first, the size of the genotype is linearly proportional to the scale of the phenotype, with each part of the genome mapping a specific part of the phenotype. However, in indirect encodings, the genotype maps nonlinearly to the phenotype, without the need of the former being proportional to the latter.

As an example, a 2D platform game level can be represented:

- Directly as a matrix, with the contents of each cell being specified separately, where mutation works by changing those cells directly.
- More indirectly as a list of positions and shapes of walls and pieces of ground, occupying more than a single cell in the matrix, and another list of the enemies' and items' positions.
- Even more indirectly as a set of various reusable patterns of walls and free space, and a list with the way those are scattered through the level.
- Very indirectly as a list of desirable properties (e.g. number of gaps).
- Most indirectly as a random number seed.

2.6.4 Content Generator

After the player experience is captured, content is adequately represented, and evaluation functions are designed, the content generator needs to find the content that

maximises particular aspects of the player experience. In general, the more direct the representation is, the larger the content search space becomes. In theory, the content generator should know if, how much, and how often it should generate content for a player.

2.6.5 Summary

In the developed prototype, we will analyse gameplay data retrieved during the playing sessions, which leads us to a Gameplay-based PEM. However, there will be no attempt to map players' cognitive levels to any states, but rather evaluate the input parameters that allow the players to successfully end the level. For that, it will be used a direct evaluation function, attributing a quality value (fitness) to the successful game sessions, and thus the used input parameters. Those parameters will be scalar values, with different intervals, where the objective is to find the best values for each one of those parameters.

2.7 Author-Centric Approach to Procedural Content Generation

Craveirinha et al. in "An Author-Centric Approach to Procedural Content Generation" state that game designers and producers are running the risk of thinking of user satisfaction instead of user experience (Craveirinha, Santos, and Roque, 2013). But, on the other hand, users can be superficial and incapable of expressing their own thoughts and emotions.

Taking this into account, the work presented by the authors proposed an Author-centric approach to EDPCG for games. In this way, it is possible to evaluate the content quality taking into account the designer's expectations of what gameplay behaviours should be elicited.

2.7.1 Purpose

The authors aim to provide a procedural generation architecture that gives designers a better way to get their game-artefact to match the desired gameplay experience. To do that, they enable the designer choices to lead the end-result for gameplay, meaning that only the elicitation of gameplay behaviour is meant to be improved by this approach.

The data source for the experience model of architecture is based on Gameplay Metrics, providing the "how" of gameplay, while also being quantitative, objective and making possible large scale automatic data collection. With this approach, instead of just pre-selecting what the player should take from the artefact, and even if it is influenced by it, it still retains voluntary translation of its meaning.

This approach focuses on improving elements of an already existing game prototype (e.g. physics or structural level components), meaning that it is used to obtain and validate improved versions of that prototype. However, there is no reason why it cannot be generalised to generate other aspects of game's content.

2.7.2 Process

Initially, the design team has to provide three items: a game Archetype (the base prototype), a set of Target Indicators and their target values, and finally a definition of Parameters to generate new variations of the Archetype. Content quality is measured as the inverse of the difference between the Target Gameplay Indicator values chosen by designers, and the values for those Gameplay Indicators that generated artefacts mediate to real players. All the elements require a configuration process carried through a platform user-interface and direct integration using computational interfaces.

Once all the elements have been gathered and integrated into a platform, it is possible to start the actual process, being the A-cycle in figure 2.18.

1. In the Candidate Generation phase, we find new Parameter set values using a search algorithm, more precisely a Genetic Algorithm (GA) with selection, recombination and mutation procedures.

2. By integrating new Parameters into to Archetype, new candidate artefacts become ready for playing, being play tested by players, which will, in turn, result in extracted metrics, characterising their gameplay accurately.

3. Metrics are compiled into Gameplay Indicators defined by the designers, and comparing to their target values, using a distance metric.

4. If the best candidate's quality is not sufficiently close to the target values, then this micro-cycle repeats with a new GA iteration, giving each chromosome a fitness proportional to the candidate's quality values.

Once the A-cycle ends, a candidate matching the target indicators is given as output to designers, that must choose if the end-result is satisfactory. If it is not, designers can fine-tune any of the elements provided, forwarding a new design specification to the platform, in a new iteration of the B-cycle.

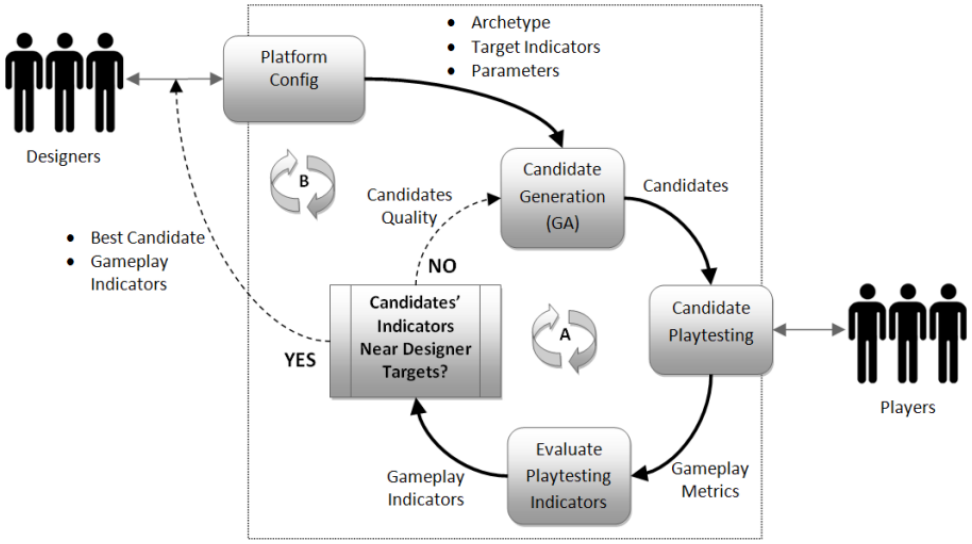

FIGURE 2.18: Architecture for an Author-Centric approach to PCG. (Craveirinha, Santos, and Roque, 2013)

2.7.3 Summary

We will be using this approach to evaluate and evolve the developed prototype. Input parameters can have a great impact on the performance of any algorithm and, because of that, we need to understand how those parameters can be altered while keeping the intended gameplay-experienced. Because of that, we will make an ecologically realistic case study, that is making and tuning a new game. In the end, we will be evaluating the players' performance against some criteria defined by us, allowing the platform to proceed through its process of gathering the gameplay data and attribute fitness values to the candidates that were previously provided to the procedural generation algorithm.

2.8 Procedural Content Generation Cases

In this section, we will analyse two different games, both containing some sort of procedural content generation in them, for the purpose of gaining a better grip on the concrete challenges involved. Both cases utilise PCG techniques to generate their worlds, but while doing it in different ways.

2.8.1 Spelunky

Derek Yu created Spelunky (Mossmouth, 2008, 2012), in figure 2.19, and originally released it as freeware for Microsoft Windows, in 2008, and re-released it in 2012 as an enhanced version with new content (e.g. graphics, music, monsters, items). The game draws inspiration from many games, such as La-Mulana (GR3 Project, 2005), in figure 2.20, and the Super Mario series (Nintendo Creative Department, 1985). On its basis, it is a dungeon crawl with elements from the roguelike genre, including randomly generated levels, no save points, frequent and easy death, and discovery mechanics. Also, it gets inspiration from 2D platformers in the way of real-time interactions with enemies (Wikipedia, 2016). Even though it generates random levels, it also has a level designer, allowing levels to be shared (Wikidot, 2013).

The game uses PCG when generating its levels before the play starts, and the first part of the algorithm, the solution path, is described in the website "Spelunky Generator Lessons" (Kazemi, 2009). A level is composed by 16 rooms in a 4x4 grid, having four different basic room types (0, 1, 2, 3) influencing the number of guaranteed exits it has and if it is on the solution path. It starts by placing a room in the top row, usually of type 1 or 2, and every time a new room is placed, it is always a type 1 room. In order to decide which direction to take next, the algorithm picks a uniform distribution random number between 1 and 5. If it is 1 or 2, it moves left, if 3 or 4 it moves right, if it gets 5 it goes down, and if reaching the edge of the screen, it drops down and changes its left/right direction. If we move left or right, there is no need to do anything since the room is guaranteed to have left/right exits, but if we moved down, the generator overrides the room type to 2, which always has a bottom drop. After moving to the next room, it asks whether the last room was type 2, and if it was this room has to be another type 2 bottom drop, or a type 3 upside-down T-shape, and since both types have left/right exits, we can restart the algorithm. When we are in the bottom row, and try to drop, instead of dropping we place the exit room. Now that we have the solution path generated, we add some random type 0 rooms to every grid space that is not a solution path, which may or may not have exits on any side. If 3 or 4 type 0 rooms are making a vertical line, there is a chance they will

become a snake pit. On a top-down manner, we can place a sequence of room type 7-8-9, or 7-8-8-9, depending on the depth wanted. Snakes and jewels are manually placed at this point since they are part of the landscape.

Each room type has between 8 and 16 templates that it chooses from, consisting of basic layouts, which include a mixture of static and probabilistic tiles, as well as obstacle blocks. Static tiles are tiles that no matter what will always happen in that place, while probabilistic tiles have probabilities of being different kinds of tiles. Finally, obstacle blocks are usually some structure the player has to manoeuvre around (Kazemi, 2009).

FIGURE 2.19: *Spelunky* in its 2012 enhanced version. (Spelunky World, n.d.)

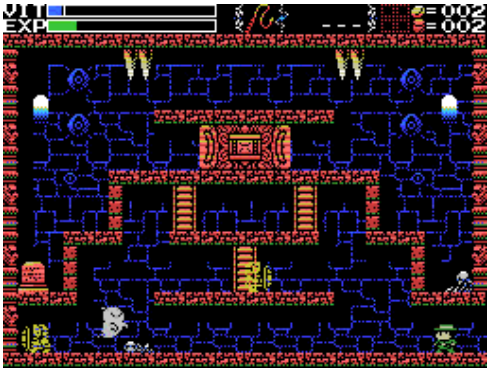

FIGURE 2.20: *La-Mulana*, originally released in 2005. (Wikipedia, 2009)

2.8.2 No Man's Sky

No Man's Sky (Hello Games, 2016), in figure 2.21, is an action-adventure survival game. The gameplay is built on four pillars: exploration, survival, combat, and trading, leaving the players free to perform within the entirety of a procedurally generated deterministic open universe, including (as said by the developers) "18 quintillion planets, many with their own sets of flora and fauna" (Wikipedia, 2017b).

In No Man's Sky, gameplay systems build on one another, existing an interdependence between them, instead of each system building itself without taking the other into account as it happens in most games. The results of one system become parameters of another, which will in turn output more results that will be used as parameters of yet another system, providing a wider range of gameplay-changing possibilities. For example, the algorithm determines the planet distance to the nearest star, if it has water, which in turn allows for life to exist, among others. The second concept of the game is that everything is deterministic, making that when the player flies near a planet, all its details are immediately generated, and even if the player interacts with the world's entities and leaves it, the planet will disappear without being saved on disk. This determinism is possible since the world's state and behaviour of its inhabitants are determined by mathematical functions, which with any point in space and time will always output the same results (Lee, 2015).

FIGURE 2.21: *No Man's Sky*, released in 2016. (No Man's Sky, n.d.)

2.8.3 Summary

After analysing both games, and their procedural content generation methods, there are some conclusions to be made. Technically, No Man's Sky systems seem way more complex than the ones used in Spelunky, since every system builds upon another while making a significant number of calculations for each planet. However, Spelunky received many positive feedbacks from critics and players[1], while No Man's Sky did not [2]. Putting marketing and commercial points aside, it probably happened because even if the systems of one game were way more complex than the other, the latter had a more captivating and interesting gameplay.

In the end, one of the things that make a good game is its gameplay. As such, our produced prototype will have a PCG algorithm to generate its world, hopefully making the players interested in exploring new and different worlds. However, its gameplay will be something that will make players want to continue playing it, not just because of the exploration, but for the combination of discovery and challenge to keep playing it.

[1] Metascore: 90/100, User score: 7.2/10 (Metacritic, 2017b)
[2] Metascore: 61/100, User score: 2.6/10 (Metacritic, 2017a)

Chapter 3

Objectives and Methodology

This section presents the objectives, methodology and work plan taken for the realisation of the empiric work underlying this dissertation. It will begin by showing the objectives defined at the beginning of the thesis, followed by the methodological approach used to fulfil them, and the work plan applied during the dissertation.

3.1 Objectives

The primary objective of this dissertation is the production of a game prototype whose purpose is to be played by several people and during many sessions. The data from the game experience will be used to adapt the generated levels using procedural content generation techniques. In this way, it will be possible to tailor the experience to the players, even though it is based on the same game concept. This will help in optimising the parameters used to generate the levels, instead of having some predefined values, which sometimes may be wrongly attributed. Getting some data from the players will help to adapt those values, allowing the game to feel more captivating and interesting for them. Summarily, the primary objective is to show in which way PCG may enable a form of Generative Game Design, where a scenario generation algorithm will be used to foster some level of perpetual novelty while working within parameters boundaries that generate a playable and interesting experience.

The developed prototype will have inspirations in the period of Portuguese Discoveries when several navigators set sail towards the unknown. This theme seemed like a good idea since we had an exploration game in mind, and thus we can do that while creating an experience that models the kind of challenges felt in mapping the unknown, managing risks, and keeping the resources to advance further. With this in mind, each game level will have a map, with a starting and an arriving point, represented by a square matrix with a not yet defined size, where the player will ride with his/her ship. Each cell will consist of a single type of terrain having the following variations:

- **Water cells**, subdivided in **flat** (with no obstacles to the movement), **wavy** (with some obstacles to the movement), **stormy** (with several obstacles to the movement, and damage to the ship), and **windy** (changing the course of the movement).
- **Land cells**, subdivided in **cliffs** (the ship cannot dock, and receives damage), **plains** and **hills** (the ship can dock safely), and **mountains** (not appearing on the coast, so there will be no interaction with the ship). Each one of the safe docking cells (plains/beach and hills) may also contain wells (providing fresh water), forests (providing wood), or animals (providing food).

The player will be able to move in any of the water cells, to any direction in a Moore neighbourhood. However, he will be limited by the number of available resources, these being fresh water, wood, food, and crew. The first three are obtained either by making contact with cells containing those resources or by returning to the starting point. The crew is only restored when the player returns to the starting point.

Procedural Content Generation will be used during loading time, creating scenarios between game iterations. This way, it will be possible to take into account player events from previous sessions, which may lead to adjustments to the parameters given to the level generation algorithm.

3.2 Design Science Research

Design Science Research is a methodology aiming at the creation of new knowledge, as a consequence of research made through design, which will help in solving a certain problem (Vaishnavi and Kuechler, 2008).

The Design Science Research, as shown in Figure 7, contains five steps: Awareness of Problem, Suggestion, Development, Evaluation, and Conclusion, each one producing its artefacts.

1. **Awareness of Problem** - this first step includes the definition and identification of a problem. At this point, and to clarify the question at hands, State of the Art research is conducted.

2. **Suggestion** - in this step, a proposal to solve the problem is made, based on state of the art previously created. This part will output the game concept to be used.

3. **Development** - the third step is the development itself, where the prototyping is going to take place, given the game concept established before. This stage can be seen as a Proof of Concept, showing that the proposed solution is indeed possible.

4. **Evaluation** - this step aims at evaluating the prototypes created in the previous step, where players will be gathered to do so, providing the feedback (circumscription) needed to adjust the prototype, making it an iterative and incremental development process.

5. **Conclusion** - this final step refers to the close of the project when the knowledge can be produced.

Before addressing the workplan, we should thus clarify the questions for which we will be researching the answers, justifying the adoption of a Design Science Research approach:

- What should be the design of a game that uses PCG methods to generate its game scenarios?

- How can such a game be built?

- What would be useful parameters for generating interesting and playable game scenarios?

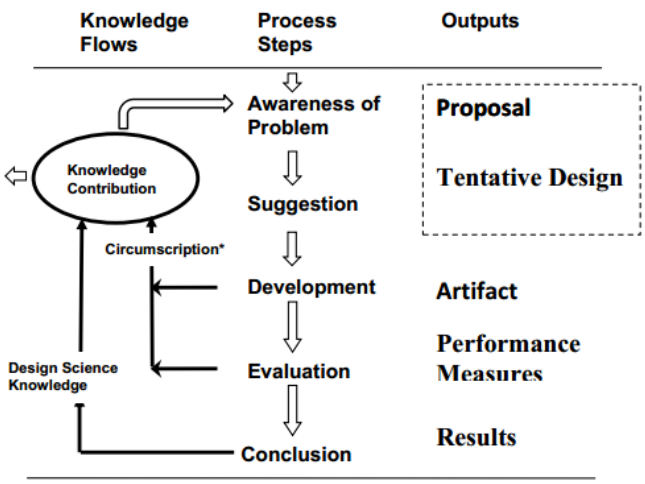

FIGURE 3.1: Cognition in the Design Science Research Cycle. (Vaishnavi and Kuechler, 2008)

3.3 Work Plan

The work plan devised for this dissertation was initially divided into 9 steps, or activities, spread across both semesters, and it is presented in figure 3.2. As it is possible to see, the diagram follows the steps in the Design Science Research methodology, with an evaluation phase always following the development parts, depicted as Prototype #1 and #2 in the diagram. In the end, it is then possible to take conclusions regarding the research done.

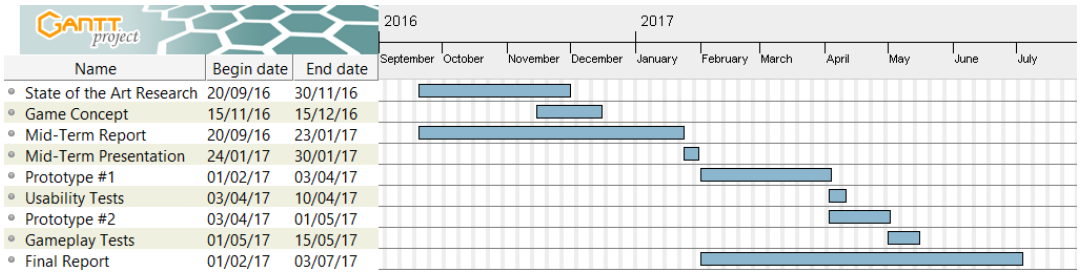

FIGURE 3.2: Initial Gantt diagram for the dissertation.

However, when starting the second semester, the initial planning was revised to add the required time for the integration with the server that would help the prototype in sending and receiving the data required in each game session. The revised diagram is shown on figure 3.3, now divided into 10 steps.

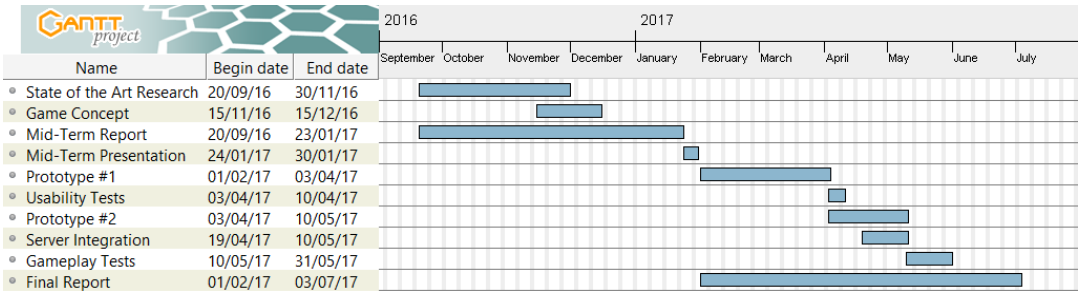

FIGURE 3.3: Revised Gantt diagram for the dissertation.

In the end, the planned schedule was not completely fulfilled, since there were some issues during the final stages of development, regarding the integration with the server. This delayed the start of the gameplay testing phase and, as a consequence, the dissertation was extended until September. On this period, only the gameplay testing phase and the final report writing occurred.

Chapter 4

DSR Initial Design Proposal: Game Concept

4.1 Characteristics

The developed prototype gets inspirations from the Portuguese Discoveries era, when many sailors left their country to the unknown, often without knowing what awaited them along the way. The game world consists of cells, divided into water and land, and each one of them contains subtypes. The land cell can also be a start point and an end point, the latter being the objective the player must reach with his ship. So, the prototype aims to be an exploration game, where it is possible for the player to travel through the world, with this being previously generated by an algorithm, containing several obstacles and help for the player.

The player can control his/her ship, and drive through the ocean of the game world. The ship will be subject to different interactions with the ocean terrain, due to the existence of winds and currents that can push the player into undesirable directions, sometimes containing obstacles.

The obstacles along the way may be of several natures, for example, the cell where the player is currently on can push him into a cliff, or on the other hand, it can be a dead end that will force the player to return a long way on the path. The help can be seen as all the cells that contain necessary resources for the player to gather, or a path that does not present adversities to him.

The prototype comprises many components, as described above, each one with unique characteristics, rightly described. As already mentioned, the **player** will sail through the map using a ship, with the primary objective to depart from the **start point** and reach the **end point**, whose location is always unknown to him. The **water cells** have a probability formula that will influence the player's movement, taking into account the player's intention, the flow's (ocean current) direction, and the wind's direction, resulting in **(intention, flow, wind)**, with the sum of all the weights being always five. It is important to note that this value could easily be changed to something else, all it would need was a readjustment of the weights accordingly. These cells are divided into:

- **Flat** – with calm waters and movement without obstacles, having the calculation values (5, 0, 0).
- **Wavy** – with slightly wavy waters and presenting some obstacles to the player's movement, having the calculation values (3, 1, 1).

- **Stormy** – with stormy waters and presenting many obstacles to the player's movement while also damaging the ship, having the calculation values (1, 3, 1).
- **Windy** – with strong winds which may take the player to a completely different direction than that was intended, having the calculation values (1, 0, 4).

The **land cells** can be interacted by the player, allowing it to dock or not (damaging the ship), and to get resources from them, if they have any. These cells divide into:

- **Cliffs** – the player cannot dock in these cells, suffering damage if he attempts to.
- **Plains** and **Hills** – the player can dock safely in these cells and there may resources in them.
- **Mountains** – the player cannot dock since these cells cannot be accessed by him since they are in the middle of land agglomerates.

The **resources** are used when the player moves or repairs his ship. In the first case, **fresh water** and **food** are consumed, which can be obtained from plains and hills cells, and if the player does not contain any of those two resources, the number of **crew** elements is reduced, since there are no resources to maintain it. Those resources decrease in a certain number of movements ratio. In the second case, if the player repairs the ship, **wood** will be consumed, which can be obtained from plains and hills cells. If the player decides to return to the start point, then he will replenish all the maximum amount of every resource, as well as repairing any damage taken by the ship. Finally, the **map** will have a predefined size, where there will be all the cells and resources, as well as the player's ship.

There was the idea of including two more components to the game but were under-prioritised in this simplified model due to time constraints. The first was marine obstacles, which included enemy ships and sea monsters, both damaging the player's ship and sometimes reducing its crew. The second component was treasures, which were thought to increase the interest of the player into exploring the game world.

4.2 Interaction

Since the beginning, the game was intended to be played on both on desktop and mobile or touch devices, leading its interactions to be done through a touch screen. So, to move the player will have to click or touch on adjacent water cells to the one he is on, and to pick resources he will have to click on adjacent land cells that contain them. To move the game's camera, he will have to swipe the finger to the left or right, and any other interaction available with the user interface will also be done using touches on the respective buttons.

In the game's screen, the player will be able to see part of the map and his ship, since he can only perceive the cells he already went through while the others are under fog circumstances, preventing him from knowing what cells are under it. Besides that, the player can understand every cell's subtype, and in case of land cells, he can verify if they have resources and of what type. Since the ship is influenced by

the flow and the wind, the player should be able to understand those two elements through the direction the ship is facing, and the direction the sail is facing, respectively. For example, if the flow is running north the ship will turn its front to north, and if the wind is blowing east the sail will turn its tip to east.

Regarding the user interface, there are two buttons which the player can use, one of them being to open/close the minimap (showing the map's portion discovered by the player), and the other to repair the ship. Regarding information given to the player, he knows the quantity of all the resources he has in every moment, and there is a compass that tells him the direction the camera is facing since both rotate at the same time.

Chapter 5

Development

5.1 Game Engine: Unity

Unity is a general-purpose engine, comprising several visual tools and three different custom scripting languages, while also having a proprietary online store called asset store, where many plugins, either third party or official, can be downloaded or bought. Adding to this comes the fact that the engine has a nice support for developers via tutorials, community forums where Unity's employees answer to any problems, and directly contacting Unity's support team. The engine has four license options including a free personal one, which lets the developer access all engine features and platforms while also applying some restrictions to the product. However, these limitations do not interfere with the objective of the application we are aiming to develop.

Another important aspect when selecting this engine was related to the previous experience. Previously having developed some games and applications with it, allowed me to better adapt to the current project, and to focus on the development rather than the learning curve that I would face if the chosen engine was something I have never worked with. Since the aim of the project was making a mobile application, Unity seemed the right choice to take because it would be possible to easily deploy to mobile, optimising the build for the selected platform, and when talking about mobile applications this is a significant factor.

With all that was said above, Unity seemed from the beginning the right engine to choose, and because of that we never had any discussion regarding other possibilities.

5.2 Game Architecture

In this section, we will verify how the game concept can be built into an actual game. Based on the description and core idea presented for the game, it ended up being divided into several components that helped both having a better overall idea of how the prototype would look and what would be the best way to divide tasks in its development. Those components will now be described and can also be seen, along with their different relations, in the diagram in figure 5.1.

We can reduce our diagram to the four core components of the game, since most of the remaining components are children of those, with them being the **player**, the **map**, the different map **cells**, and the **resources**. The **player** is the object that will be controlled by the user itself, allowing him to do all the actions that he is allowed to,

such as moving through the game world. This game world, depicted as the **map**, is composed of several **cells**, each one of these having different characteristics, types of interaction by the player, and influence on the player. Besides that, and as already seen above, these cells are also divided into **water** and **land** cells. The player can only move through the first ones, having different weight on his movement, while the latter cells can contain resources, and the interactions with the player will always differ based on their cell subtype as described in 1.1. Finally, the **resources** can be picked by the player if he is facing the cell containing them, and every different type of resource will help the player in various ways, as already described in 1.1.

Summing it all up, the **map** can only have one **player** (and vice-versa), but it is allowed to have multiple cells given its size limit. Those **cells** are divided into water and land cells, with the latter being able to contain just one **resource**, with this being also divided into fresh water, food, and wood.

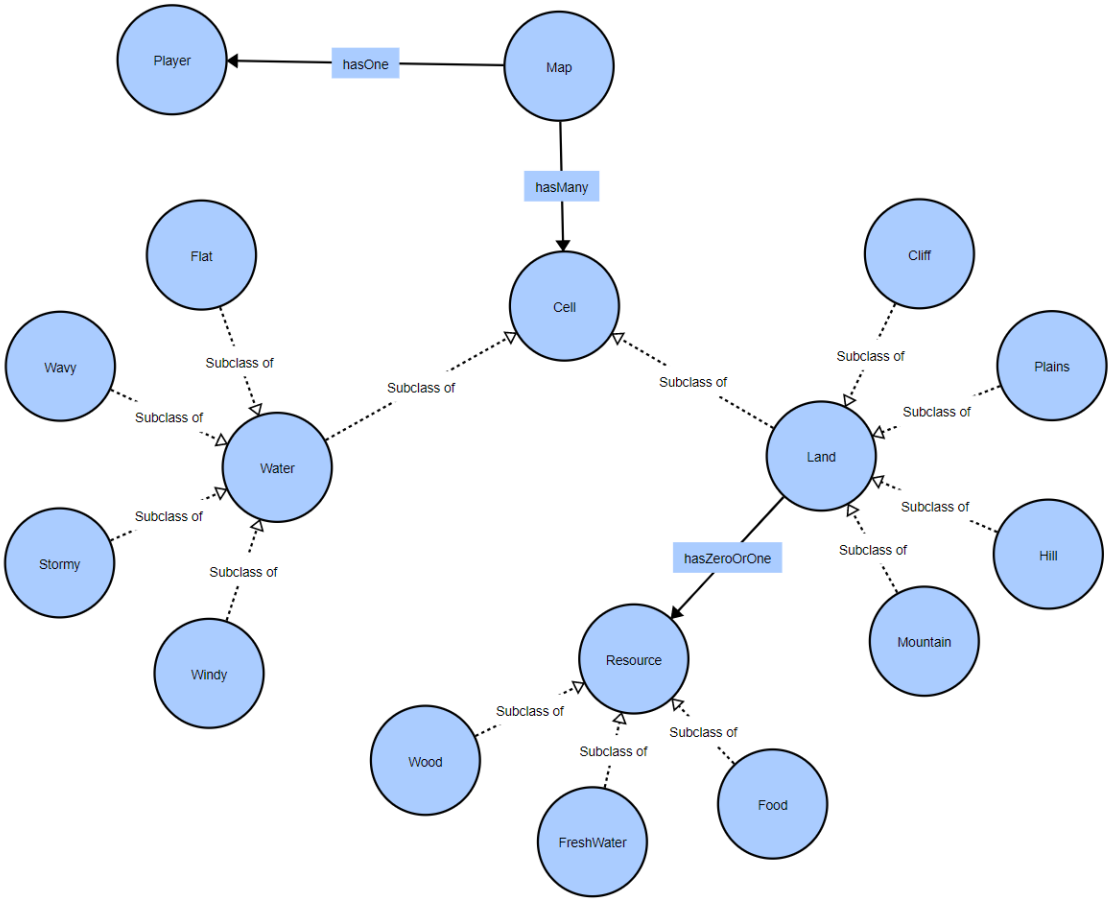

FIGURE 5.1: Ontology diagram containing the relations between each game component.

Moving to the Unity side, the prototype comprises many C# scripts with each of them having a different objective. This scripts include **HTTPManager**, **Algorithm**, **gameMaster**, **playerMovement**, **FlashingCell**, **followPlayer**, and **Epoch**, and a data flowchart showing their connections is shown in figure 5.2.

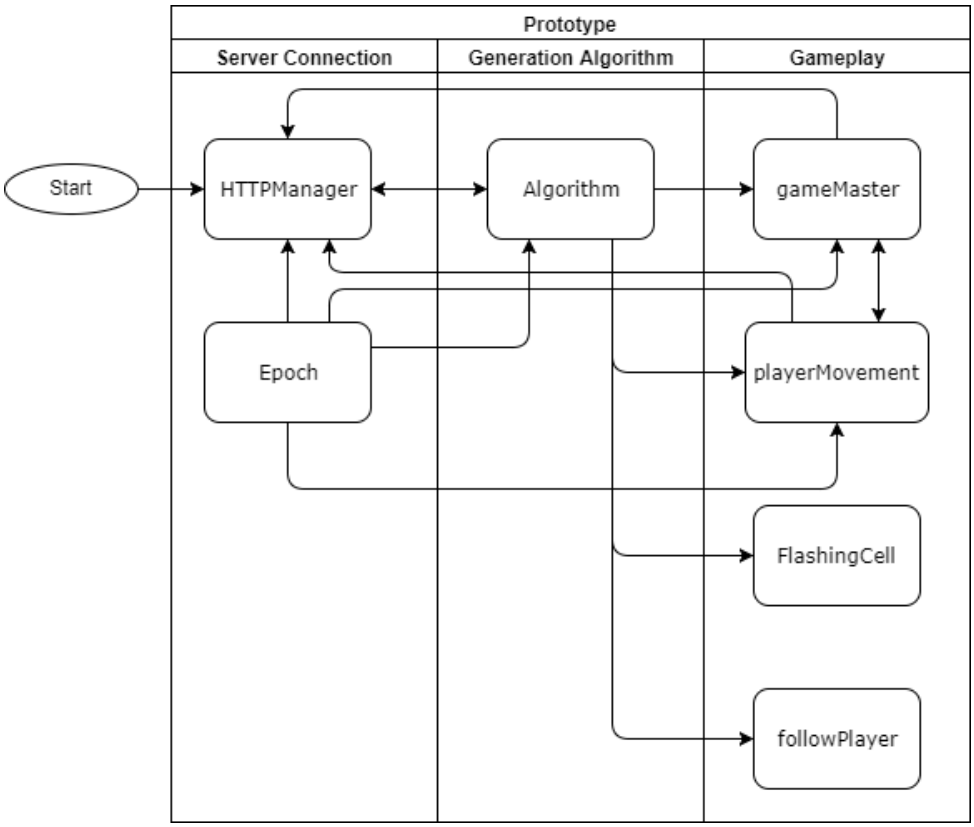

FIGURE 5.2: Flowchart depicting the data flow on the prototype.

- **HTTPManager** – this is the first script to run in the application since it will try to retrieve initial parameters (also named as feature set) from the server to start the session, and in case it is not able to retrieve them, default parameters will be kept. Besides that, it manages all the connections to the server, as well as all the information needed to achieve them. Also, instead of sending every single line of the log file to the server, it keeps a local copy of the log file during the session and sends it at the end of the game session. The script has public functions that can be used to send and retrieve data from the server. After its main process has ended, it will send the initial parameters to the Algorithm script.

- **Algorithm** – when this script runs for the first time, it will get the initial parameters stored in the HTTPManager script, in order to execute the map generation algorithm and generate a new map for the game session. After this map is created, the script will log it so that it can later be sent to the server. Also, after those actions occurred, the script will spawn all the clouds and map limiters, the gameMaster object, and the player object (containing the playerMovement script and the flashing cells). The script has public functions to get a cell's information, get the matrix's side size (and thus the size of all the sides since it is a square matrix), check if a cell exists in the matrix, and get the direction from the start point to the end. It needs to retrieve the current time from the Epoch script, in order to send logs to the HTTPManager script.

- **gameMaster** – during the game, this script will check for swipes to move the camera. Also, it has functions to start, end, reset, and exit the game, as well as a function to open and close the minimap (and refresh it if needed). It retrieves

data from both the Algorithm and playerMovement scripts, and gets the current time from the Epoch script to send logs to the HTTPManager script.

- **playerMovement** – this script is responsible for the player's movement, for example checking which cells the player can move to. It manages the player's resources, reducing them by a given decaying rate, and increasing them when the player picks up a resource. Besides that, it is also responsible for showing the amount of resources in possession of the player, and the respective animations anytime there is a change in their quantity. Finally, it is also responsible for animating all the player's movement, for example, the ship rotation. It retrieves data from both the Algorithm and gameMaster scripts, and gets the current time from the Epoch script to send logs to the HTTPManager script.
- **FlashingCell** – this script is always attached to various indicators above the cells the player can directly move to, and will make them flash from fully opaque white to fully transparent white.
- **followPlayer** – whatever object has this script attached will follow the GameObject depicted as the player.
- **Epoch** – this script is used to calculate the Epoch time, defined as the number of seconds that have elapsed since 00:00:00 UTC, Thursday, 1 January 1970 minus the number of leap seconds that have taken place since then (Wikipedia, 2017c).

5.3 Map Generation Algorithm

Since one of the main goals of this work was to have some kind of level generation, as it is implied in the title as generative game design, I needed to think in what approaches I could follow to create an algorithm that would fit the game concept and my own visions about the game. Initially, there was some research regarding some types of map generation algorithms, and as it turned out there are some approaches that are usually used, such as:

- **Perlin Noise**, where it generates a grid of values ranging from 0 to 1, making it possible to adapt it into a height map, where on a 0-10 scale we could have for example below 3 water, 3-4 beaches, 4-6 forests, 6-9 hills, and 9-10 mountains (Red Blob Games, 2015). One example of this approach can be seen in figure 5.3.
- **Cellular Automata**, which is a grid of cells with their own state, but with the same ruleset that determine state transitions. All these transitions happen at the same time, and every cell looks at its neighbourhood and decides what state it will transition to. This method is often used for dungeon generation, mostly caves, since it creates organic looking patterns (Wikidot, 2016). One example of this approach can be seen in figure 5.4.
- **Fractals**, describing a broad set of shapes, which characteristics like non-integer dimension, and detail at all scales, which leads fractals to be used due to their self-similarity simulating natural processes, like erosion and plant growth (Wikidot, 2010). One exapme of this approach can be seen in figure 5.5.

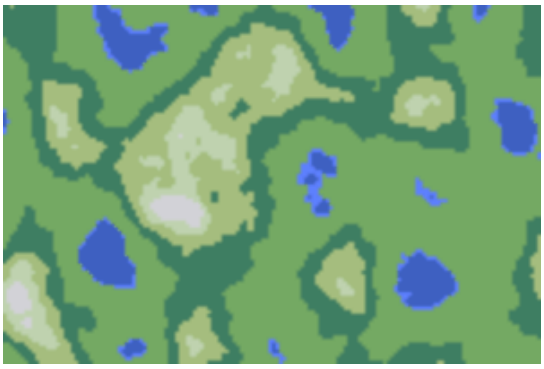

FIGURE 5.3: Map generated using Perlin Noise. (Red Blob Games, 2015)

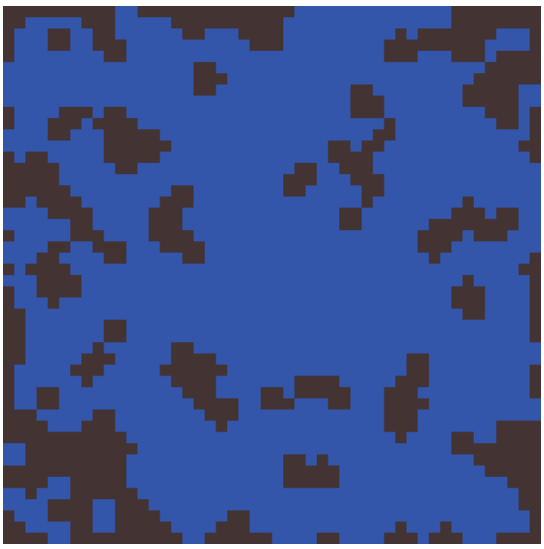

FIGURE 5.4: Map generated using a Cellular Automata. (Envato Tuts+, 2013) (Generated)

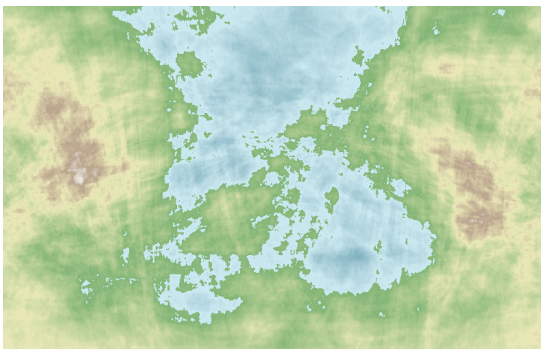

FIGURE 5.5: Map generated using a Fractal. (Donjon, 2011) (Generated)

However, besides these types of approaches being commonly used in map and terrain generation, I ended up not using them. That happened because they did not allow the customisation and singularity that I had in mind for the game, and sometimes being too complex for the game as well. I wanted for example to be able to limit the amount of existing land right from the beginning, to limit the size of the land agglomerates, without having to apply some post-processing method on the map matrix and replace cells to fit into the wanted values. Furthermore, the Perlin Noise and Fractal methods seemed to be too complex for the purpose, creating worlds with more detail than it was intended for the developed prototype. In the end, those characteristics led me into creating my own algorithm to generate the maps.

Before going into detail, we will first show an overview on how the algorithm is supposed to work, showing its pseudocode, and after that, explaining the entirety of the algorithm, since some parts of it require more functions or steps that do not appear in the pseudocode algorithm. It is important to note that this is simply our algorithm design proposal, and we plan to find out through research if this algorithm is adequate and produces interesting and controllable results.

```
1  initialise map matrix with water cells;
2  while there are not enough land cells do
3  |   add land cell into available cell;
4  |   while there are not enough land cells and the size of the land
   |    agglomerate is less or equal than the allowed do
5  |   |   if adding cell to the agglomerate then
6  |   |   |   add land cell;
7  |   |   else
8  |   |   |   end agglomerate cycle;
9  |   |   end
10 |   end
11 end
12 define start and end points;
13 define land cells subtypes;
14 define water cells subtypes, flow, and wind;
15 define resource distribution;
```

Now that we have a general understanding of how the algorithm works, we can move on to a deep explanation on all its different parts, including the type of data it works with and how it changes this data to create a map in the end.

The algorithm needs some input parameters to work, which will be explained below:

- **matrixSide** – the side length of the square matrix used, meaning the number of columns or rows it has.
- **totalSize** – the total number of cells in the matrix, which is given by

$$totalSize = matrixSide^2$$

- **percentageLand** – the percentage of land available in the map, on a scale from 0 to 100.

- **amountLand** - the number of land cells that will exist in the map, given by

$$amountLand = totalSize * \frac{percentageLand}{100}$$

- **maxLandSize** – the maximum size allowed for a single land agglomerate, which in our case will be thirty percent of the maximum number of cells allowed in the map, and is given by

$$maxLandSize = amountLand * \frac{30}{100}$$

- **minDistance** – the minimum distance between the start and end points.
- **maxDistance** – the maximum distance between the start and end points.
- **pickableFreshWater** – the number of cells that will contain the resource fresh water.
- **pickableFood** – the number of cells that will contain the resource food.
- **pickableWood** – the number of cells that will contain the resource wood.

As stated before, the algorithm uses a matrix that stores everything about each cell in it. This is accomplished by using integers with a predefined format, allowing the algorithm to know every information it needs from a cell. This format consists of six numbers (which can be seen as categories) with the following order: **type**, **subtype**, **resource**, **enemy**, **flow**, and **wind**. Before showing all the possible values for each number, it is important to explain why there is a value for enemies: at the start of the development there were thoughts of having water cells that could contain enemies to difficult the player's passage, but those were not implemented. Still, the number for it was kept in case there was the possibility to include said enemies. Moving on, in table 5.1 it is possible to check every value for each number, and what they mean. So, if we had a plains cell with fresh water its value would be 221000, and if we had a stormy cell with flow's direction to the north and wind's direction to the south its value would be 130015.

Type	Subtype	Resource	Enemy	Flow	Wind
1 - Water	1 - Flat 2 - Wavy 3 - Stormy 4 - Windy	0 - Non-existent	0 - Non-existent	1 - North 2 - Northeast 3 - East 4 - Southeast 5 - South 6 - Southwest 7 - West 8 - Northwest	1 - North 2 - Northeast 3 - East 4 - Southeast 5 - South 6 - Southwest 7 - West 8 - Northwest
2 - Land	2 - Plains 3 - Hills	1 - Fresh Water 2 - Food 3 - Wood	0 - Non-existent	0 - Non-existent	0 - Non-existent
	1 - Cliff 4 - Mountain 5 - Start 6 - End	0 - Non-existent	0 - Non-existent	0 - Non-existent	0 - Non-existent

TABLE 5.1: Possible values for each number of a matrix cell.

Having now all the required data information, we will explain the algorithm in more detail, including everything needed in each step shown in the pseudocode.

Initialise Matrix

Starting with **line 1**, the algorithm will initialise the world matrix, considering the defined side length of the matrix. However, it will not initialise the matrix with zeros. Instead, it will fill each cell the value for a standard water cell, meaning 100000. This decision was made because most of the map will be water cells and only a small percentage filled with land cells, leading me to change all the values to the base water cell value.

Generate Land Mass

Moving onto **line 2**, the algorithm will start the process of placing land cells, which is divided into two different phases. First, in **line 3** it will try to pick an available spot for a land cell and, when it finds one, will rightly place said land cell there. In **line 4** the algorithm will try to create a land agglomerate until it reaches the maximum size for the agglomerate or the maximum number of land cells, or until it is forced to break the cycle, with a decreasing probability of adding another cell as it gets bigger. To calculate this probability, it uses the formula

$$probability = \frac{numerator}{maxLandSize} * 100$$

with *numerator* having an initial value of *maxLandSize-1* and decreasing by one every time a new land cell is added to the agglomerate and while *numerator* is greater than 1. The algorithm will then generate a value between 0 and 100 and check it is below the probability value. If it is, it will then try to place a land cell in the Moore neighbourhood (figure 5.6) of the land cell that is now being checked. Otherwise, if the value is greater than the probability, it will cease the expansion of the land agglomerate. It is important to note that all the land cells placed are added to a List, so they can be easily accessed later, instead of searching the full matrix for land cells.

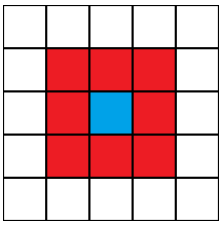

FIGURE 5.6: Moore neighbourhood with the player in the middle. (Wikipedia, 2012a)

Define Start and End points

In line 12, the algorithm defines the start and end points and begins by picking two different land cells from the list previously created. It will then check if both cells are coast cells, and the algorithm does this by seeing if their Von Neumann neighbourhood (figure 5.7) has at least one water cell. If they both pass, then the algorithm will check if the Euclidean distance, in this case given by

$$distance = \sqrt{(start.x - end.x)^2 + (start.y - end.y)^2}$$

with *start* and *end* being the points, is contained in the interval between the minimum and maximum distances received as an input. If they are within those values,

then all that is left is checking if we can get from the start to the end, preventing any of those points from being in a lake created by the algorithm. It achieves it by first creating an adjacency matrix initialised with zeros, with the size of the world matrix and with each cell containing the Euclidean distance from that cell to the end, and a List that will contain all the cells left to check. In the list, the algorithm uses a Vector3, since the x and y values are equal to the cell's but the z value is the Euclidean distance to the end. The first cell to appear in both the matrix and the list is the start point. The algorithm will then begin checking the cells in the list until there is none, always by retrieving the first cell of the list. It then starts by verifying if it is possible to reach the end point through the current cell, returning *true* in case it happens. If not, it will add all the water cells contained in the current cell's Von Neumann neighbourhood (figure 5.7) to the matrix and list. However, since in our game the player can move in all the Moore neighbourhood space we need to take more cells into account, and thus the algorithm will check if the diagonal cells, in relation to the current one, are both water cells and accessible. We cannot simply add them without verifying their accessibility because the player cannot move if the cell is blocked by two other land cells, for example if in figure 5.7 the top and right red cells are land cells and the player wishes to go through them. After adding all the available cells, and before starting another iteration of the cycle, the algorithm sorts the cells by their distance to the end point, thus almost discarding those that are far away from it. If the algorithm exits the cycle, it is because there is no path from the start to the end, and will then return false. If any of the previous checking phases failed to pass, the algorithm would need to pick two new points.

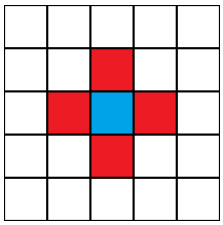

FIGURE 5.7: Von Neumann neighbourhood with the player in the middle. (Wikipedia, 2012b) (Adapted)

Define Land Cells Subtypes

Regarding the land cells, what is left to do in line 13 is to define their subtypes. To do this, the algorithm will go through the list containing the land cells, and if a cell does not have a subtype defined, it will then start the process. It first verifies if we have a coast cell, using the function described before. If it is a coast cell, then the algorithm generates a value between 0 and 100 and checks if it is within the interval for each of the cliff, hills, and plains subtypes, meaning that each of those subtypes has approximately a $\frac{1}{3} * 100$ percent chance of happening. In the case it is not a coast cell, then it will directly become a mountain.

Define Water Cells Subtypes, Flow, and Wind

In line 14, the algorithm will start by defining the subtype for all the water cells, and there are some criteria for doing it, only being applied to cells that do not have a defined subtype. First, it will verify if the current cell belongs in a lake, meaning that it is in a water agglomerate surrounded by land. To do this, the algorithm will

first set a maximum value maxWater for the size of the lake, using the formula

$$maxWater = \frac{matrixSide^2 - amountLand}{5}$$

The algorithm then creates a list where it will store every water cell contained in said lake, and two integer variables: one to count the number of water cells in the list and another to save the list's index being checked, since it will constantly be adding new elements to it. The verification starts in a cycle with the algorithm filling the list with water cells contained in the Von Neumann neighbourhood of the current cell. The cycle will stop if the number of water cells reached the maximum allowed or the index being checked reached the number of water cells it currently has. After ending the cycle, if the number of cells in the list is less than the maximum allowed, then it means there is a lake, and all the cells in it will have the subtype flat. Continuing line 14, if the cell is not in a lake, then the algorithm will make verifications whether it is near a land coast cell, and depending on this land cell's subtype different probabilities for the water cell's subtype will occur, as seen in table 5.2.

Land Subtype	**Flat (%)**	**Wavy (%)**	**Stormy (%)**	**Windy (%)**
Plains	60	40	0	0
Cliffs	10	60	0	30
Hills	40	50	0	10

TABLE 5.2: Probabilities for each water cell subtype, depending on the neighbour land cell.

If the water cell being checked is not in a lake or a coast cell, then it means it is in high seas, being able to be any of the four possible subtypes, but the subtype attribution will be based on the Moore neighbourhood of the current cell. To do this, the algorithm first creates two integer arrays: quantityArray and neighborArray. The first is used to tell the probability of each subtype occurring, starting with 25% to every subtype. The second array will store the number of neighbour water cells for each of the four subtypes. After neighborArray has been filled with data, the algorithm will sum its values to each subtype in quantityArray. For example, if neighborArray is 3, 0, 1, 0, quantityArray will then be 1+3, 2+3, 3+3+1, 4+3+1 = 4, 5, 7, 8. The algorithm will then pick a value between 0 and the maximum value in quantityArray, which will be compared to the values in the array to verify what interval (meaning subtype) contains it. Still in line 14, the algorithm will now define both flow and wind of the current water cell, also considering its Moore Neighbourhood, similarly to the previously explained method for defining high seas water cells' subtypes.

Distribute Resources

Nearly finishing the algorithm, what is left to do in line 15 is to define the resources distribution. The process is equal to all the three resources, and what the algorithm does is place one resource until the amount reaches its maximum, moving on to the next resource (if there is one).

5.4 Integration with the Crowdplay Server

During the final moments of development, there was a need to integrate the game with a server, named CrowdPlay, which hosts a tool that works on improving an

existing base-game prototype, with the help of a procedural generation algorithm, until it meets an author's (designer) agenda for player's experience. And as such, this tool would fit in the developed prototype, since we needed to evolve our input parameters for our algorithm to try and find the most suitable values for each parameter. The tool is named the Authorial Game Evolution (Craveirinha and Roque, 2016) tool (AGE for short), and three main elements must be provided: a **Base-Game**, a set of **Game Variations** intended to evolve the base game, and a set of **Design Goal Tests** being the designer's player-experience objectives.

The **Base-Game** is the prototype integrated with AGE, leaving some aspects open to evolution. The **Game Variations** are essentially the range of possible values for each game variable defined by the designer. The **Design Goal Tests** are attributions of quality scores to the candidate solutions whose player experience indicators pass a given test condition. Those indicators can be calculated based on gameplay metrics, automated subject questionnaires or bio-metrics signals. The data is then processed by the AGE, and the score is attributed based on formulae defined by the designer. These design goal tests are the basis for helping the procedural algorithm's evaluation of prototypes, measuring their closeness to the intended player experience.

The procedural algorithm starts by generating a new set of game variations with a simple genetic algorithm. Then, when a player plays the game, AGE will send one candidate to the prototype in order to generate game-content for the experience. After that, the data is sent to the AGE tool and processed based on the defined indicators. The scores for each candidate will then be attributed based on the design goal tests. This cycle will repeat until a given end condition is achieved. The process can be seen in figure 5.8.

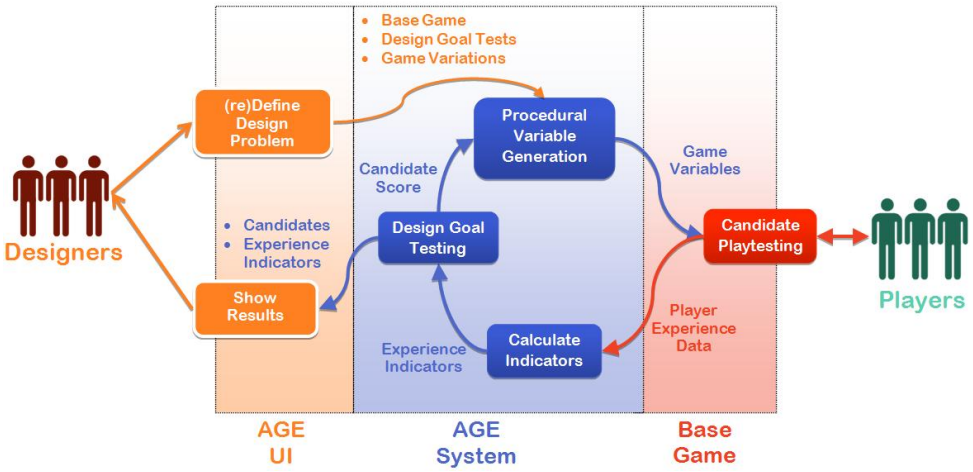

FIGURE 5.8: Diagram of the AGE-powered design process. (Craveirinha and Roque, 2016)

The developed prototype will use the AGE tool to retrieve some of the initial parameters needed for the map generation algorithm, to send the game session's data. This allows the tool to process the data and attribute scores to the different candidates, as seen above. The input parameters and design goals defined in the tool will be later discussed in the document, in section 6.

In order to have the platform prepared for the developed prototype, we had to register a project and a considerable number of objects that will be used as data to describe the different game sessions. The data is sent to the server in a subject-event-predicate format, but allowing more information such as time, position, and values for the subject and predicate. This data is sent as text to the server, and each line connects to an action in the game, and the line format is **"session_id, date-time, time_engine, x, y, z, id_subject, subject_value, id_event, id_predicate, predicate_value"**. As we can see in this format, every object needs its own ID, so we needed to make a list of what objects the game needs in order to register them in the server and get their identification (as seen in Appendix A).

5.5 Game Interface

The game interface and overall functioning will be now explained while showing some screenshots of its final version, and explaining a bit of both the user and software sides.

On Start

When the game is started, the first thing it will do is to retrieve the initial parameters (feature set) from the server via an HTTP connection, since those are required for the map generation algorithm. In the case the user does not have an internet connection, those parameters will keep the default values defined in the application code. After the parameters have been defined, they will be passed to the generation algorithm, and so the world map will be created.

When the generation algorithm finishes, the user will be presented one screen with instructions and, sometimes, the direction he should follow to get to the end point of the game (figures 5.9 and 5.10). Notice the *sometimes* term, because the decision to whether the user gets information on the initial directions or not is an input parameter, and as such may not be available for some sessions. When the user has finished reading all the provided information, he can continue by pressing the button Play. It is important to note that every action that the user will make from now on will be recorded in the data format required to send to the server.

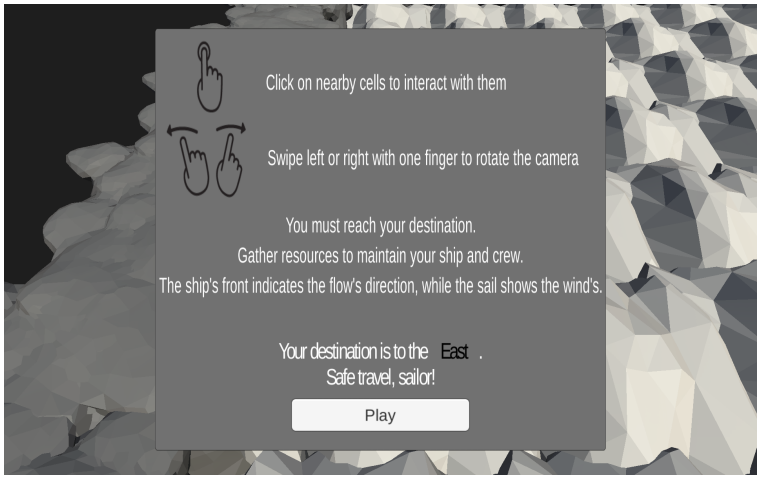

FIGURE 5.9: Prototype's screenshot of the initial screen, with directions.

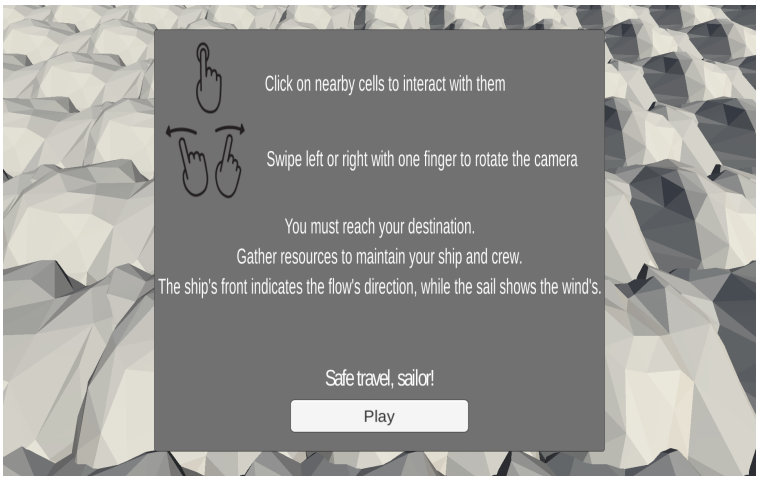

FIGURE 5.10: Prototype's screenshot of the initial screen, without directions.

Interface Design

The user interface was thought to accomplish some objectives that allowed the player to be aware of the state of the game.

- The interface should present to the player all the details about his/her resources while being easily relatable to them.
- The interface should allow the player to detect when a resource's quantity has been changed.
- Since this is an exploration game, the interface should include a component (for example, a compass) that ensures the player can perceive his/her direction in the game.
- Since the game has to be played on devices with a touchscreen, there should exist buttons with the objective of repairing the ship, and opening and closing the minimap.
- The user interface should not occupy a significant portion of the game screen. For example, it should not overlap the interactable cells.

Playing the Game

When the user starts playing, the first thing he will see is the overworld map, with his ship slightly in the middle of the screen and near the start point (figure 5.11). Besides that, he can also see all his resources in the top right corner just below the compass indicating the camera's direction, and two buttons in the bottom left corner for opening/closing the minimap and for repairing the ship. Finally, he can also notice the fog around him, preventing him from knowing what is further in his path until he discovers it.

The user is allowed to click in any cell in its Moore neighbourhood, but can only move to the water cells that are highlighted with a blinking white rectangle. When he attempts to move into a water cell, the game will make some calculations to verify how the player's movement will go. These calculations are made using the current cell's calculation values, stated in section 4. After he has wandered around the

FIGURE 5.11: Prototype's screenshot of the overworld map, with the player in the middle.

world, the player will most likely need to check his surroundings and the discovered area so far, and thus he can open his minimap with the respective button, displaying a screen showing the portion of the map discovered by the user (figure 5.12). He is able to exit said display by pressing the mini-map button again.

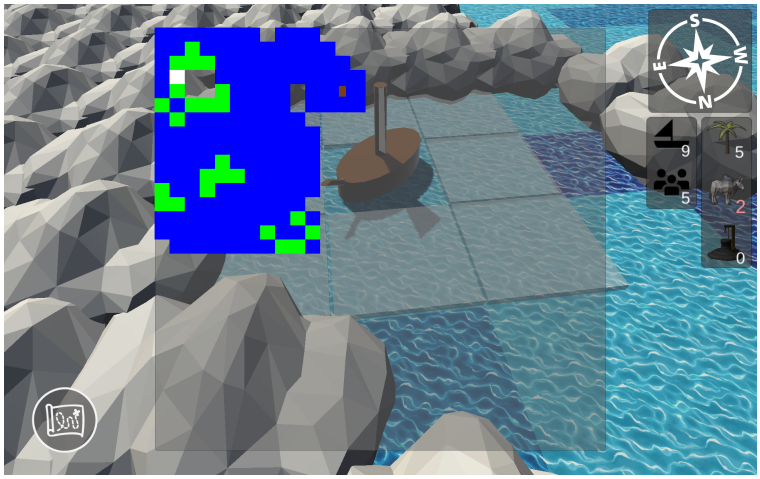

FIGURE 5.12: Prototype's screenshot of the minimap.

Managing Resources

Since getting to the end can prove to be a hard task, the user will need to gather resources whenever he can. If he does not, he may run out of food and fresh water (figure 5.13) to maintain his crew or can end up with a damaged ship and no wood (figure 5.14) to repair it. So, it is important to the user to be aware of his surroundings and to remember where he last found a certain type of resource, as he may need them urgently. Of course, sometimes the generation algorithm can create maps with resources concentrated in one place, or so far apart that it becomes a hard task to gather them.

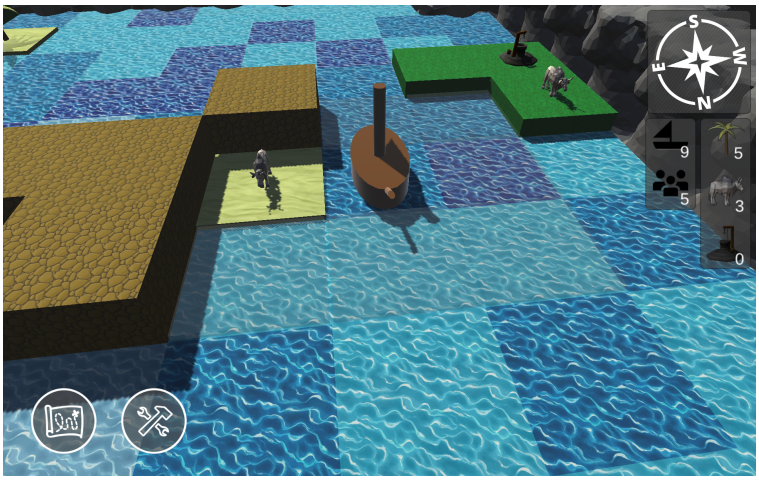

FIGURE 5.13: Prototype's screenshot showing cells containing food (animal) and fresh water (well).

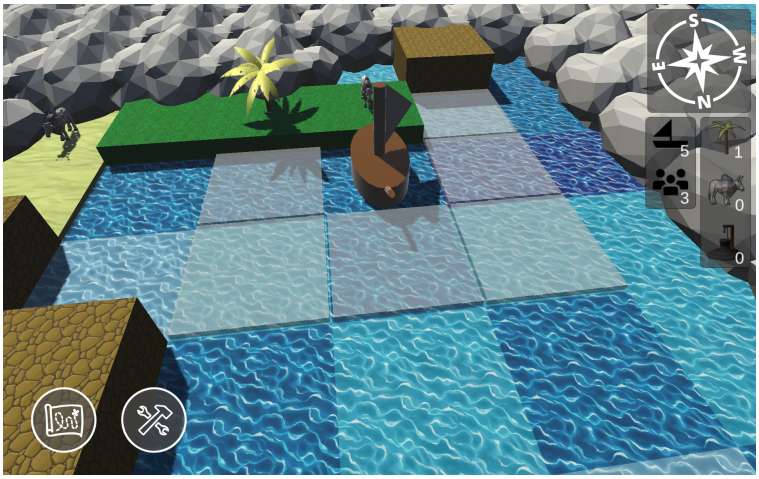

FIGURE 5.14: Prototype's screenshot showing cell containing wood (tree).

Winning or Losing

Eventually, after a possibly difficult path, the user will find out the location of the end point (figure 5.15), and thus will be able to finish the game successfully (figure 5.16). After he does, he can choose to play a new session with an entirely different map, or to exit the game if he does not want to continue playing.

Of course, things may not go as expected, for many reasons, and the user may end up losing all its resources and crew members or getting his ship destroyed. If this happens, a game over screen appears, and the user can try to play the game again in a new map, or exit the game (figure 5.17).

In both the winning and losing situations, the game will send the play session data to the server, which will then be used in the AGE genetic algorithm to attribute scores to the candidate (feature set) used in this session.

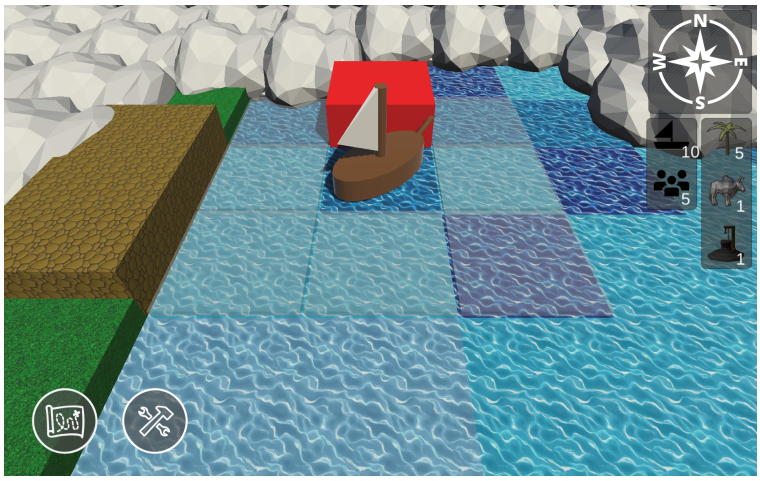

FIGURE 5.15: Prototype's screenshot showing the end point.

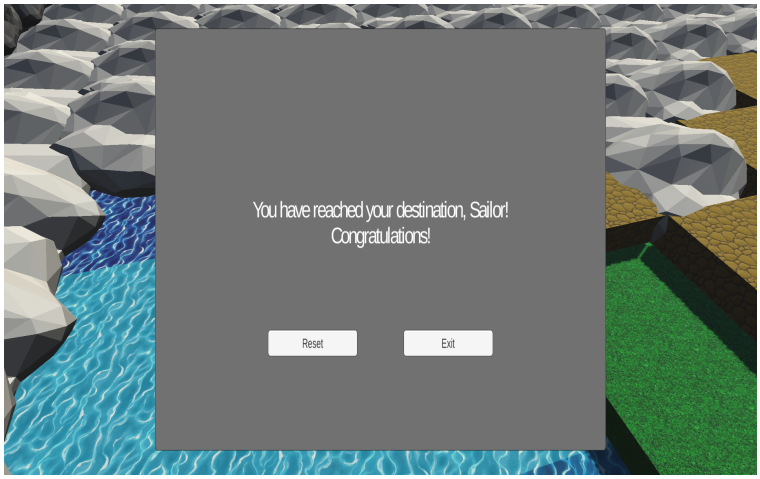

FIGURE 5.16: Prototype's screenshot showing the winning screen.

FIGURE 5.17: Prototype's screenshot showing the losing screen.

5.6 Development Activities

The following list states the activities that took place in each iteration of the development process. A list depicting all the performed tasks can be seen in Appendix B.

- Iteration 1 (8/2 – 14/2) – Definition and documentation of the Game Design Document, stating all the key characteristics of the game.
- Iteration 2 (15/2 – 21/2) – Definition of the system's architecture. Definition of the initial parameters of the generation algorithm, its steps, and the data structure to be used.
- Iteration 3 (22/2 – 31/3) – Development of the first version of the prototype, including the implementation of the generation algorithm, the visual translation of the world map, and the gameplay elements of the game.
- Iteration 4 (1/4 – 11/4) – Usability testing. Analysis and conclusions about the results.
- Iteration 5 (12/4 – 31/5) – Prototype revision based on the results taken from the usability tests. Integration with the Crowdplay server.
- Iteration 6 (1/6 - 15/8) – Pilot gameplay tests of the integrated prototype. Definition of the gameplay tests. Gameplay testing with AGE to understand how what is generated with different PCG parameters influences the way players explore the gameplay.

5.7 Work Management and Prioritisation

In order to manage how the development process took place, I adopted a backlog list that contained what activities remained to be done. At the beginning of each week, I would choose a subset of activities and divide them into tasks. During the week, I would develop and complete said tasks and, if any of them were not completed during that time, it would transition over to the next week. Otherwise, if everything went as planned, I would select another subset from the list, based on the priority of having a working and playable prototype. After the completion of each activity, some tests were performed to access if it was working as intended with no defects. Those tests were performed firstly by me, at the time of development, and later on with colleagues from the laboratory. Also, as the development went on, some elements were added or removed from the worklist. This list is included in Appendix C.

Prioritisation was given based on the previously described generation algorithm steps and with the intention of experimenting the gameplay in order to test its concept. Based on those criteria, the selecting order would be tasks that remained from earlier weeks, tasks that could either be developed independently of others or served as support for other tasks, dependent tasks. All these while making advances into a playable prototype, ready to be tested by players.

Chapter 6

Evaluation

In this section, I will be talking about the testing done on the developed prototype. It was divided into usability and gameplay testing, with the former happening first to make sure the game was understandable and playable.

6.1 Usability Testing

These evaluations focused on understanding how well players understood how to play the game, what was the purpose of each game's component, the information that was provided to them, among others relevant for control and interpretation of the interface and the game.

At the moment of these tests, the prototype was not yet playable on a mobile platform, but instead, it was played on a computer. That lead to game actions being mapped to keyboard keys. Besides that, in this version of the prototype there were not any animations in the game, and for example, the ship's movement was simply a "quick-jump" to the destination. The prototype, regarding feedback to the player, was still in a raw stage, and that probably influenced the testing outcomes.

6.1.1 Test Setup

These tests consisted of individual sessions where each voluntary tester was asked to play a build of the game prototype, with no previous information besides the one provided by the prototype itself. During those sessions, testers were accompanied by an evaluator, with the task of taking notes on any behaviour or events occurred during the tests and forbidden to answer any questions regarding the game.

Each player was asked to play the game at least once, but after that, they were allowed to play more times while still counting for any notes we could take. Each time they started the game, there would be a tutorial message explaining a portion of the game, as seen in figure 6.1. After that, the players could start the actual game. Also, there was not any predefined script for the game session. Since it is a game, we opted for letting the testers to play the game as they wanted, because restraining them to a script would go against the wanted feeling of freedom and exploration, and could influence the results. So, all the players were asked was to reach the destination point, thus the end of the level.

After the playing session, each player was asked to fill a questionnaire about their experience with the game, containing twelve questions with the first ten being multiple choice, following a Likert scale with the values 1 to 5, respectively ranging from strongly disagree to strongly agree. The questions are as follows:

1. I understood what I had to do.
2. I understood how I could do my actions.
3. I understood what each game's component meant.
4. I understood the outcome of each one of my actions.
5. I understood that flow and wind could influence my movement.
6. I understood the flow's and wind's directions.
7. I understood that different cells produced different results.
8. I was able to locate myself in world through the mini-map.
9. I perceived the world's limits.
10. I understood I could pick up resources from land cells.
11. What visual aspects would you change, if any?
12. Any comment or suggestion?

Tests were performed with six subjects, and data was collected through the notes taken during the game sessions and the questionnaires. From those six testers, only five were able to reach the destination point, although none were able to do it on the first try. The subject who was not able to reach the end gave up at the start of the second try, as it was hard for the subject to understand the overall objective and experience of the game. However, the results were still taken into account, including the questionnaire's answers.

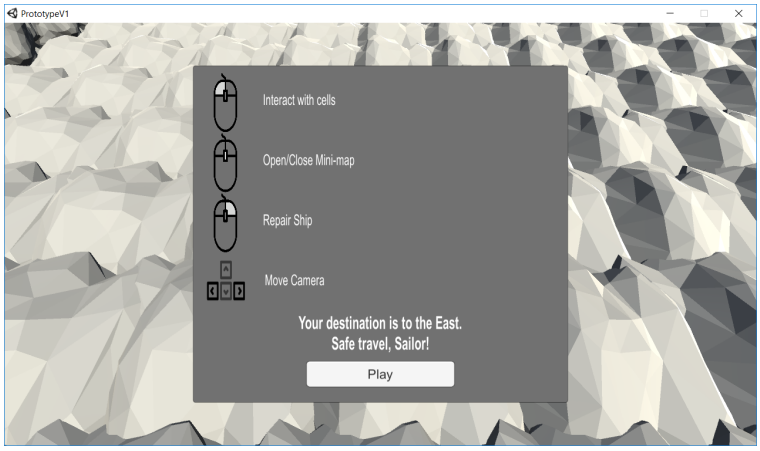

FIGURE 6.1: Prototype's screenshot showing the first version of the tutorial.

6.1.2 Results and Analysis

Since data was collected through questionnaires and my own notes, I will be analysing the questionnaire results and, in the end, take conclusions based on results, suggestions, and observations. Also, I ended up mixing testers' suggestions and my own observations in a document, which can be found in Appendix D, as well as the tables containing every value depicted in this section.

Before going into the analysis, it is important to note that the objective is not to make a statistic analysis, nor is it representative. This analysis together with the observations will only allow identifying problem areas that must be solved in order to be able to advance to the gameplay testing phase.

The questionnaire's objective was primarily to understand if the tester was able to comprehend the intended gameplay experience, and how well that happened. Regarding the multiple-choice questions, the results can be found in figure 6.2, with each colour identifying a different tester. As it is possible to see, the results are too dispersed, regarding each question. However, it is also possible to verify that some players understood the prototype better than others. For example, based on the answers, we can assume that tester 1 had a better understanding of the game when compared to tester 5, which was the one who was unable to reach the end of the game.

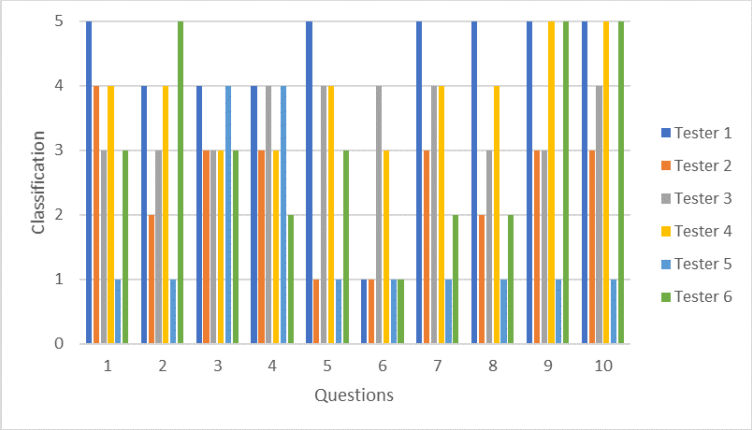

FIGURE 6.2: Questions classifications by each tester.

In order to help better understanding each question result, I calculated both the average and the standard deviation of the classification per question, as shown in figure 6.3. One of the first things that gets our attention in the graph is the fact that question number six had the lowest average classification of all the questions, by a considerable margin. This question refers to a critical point in the game, the understanding of the directions of both the flow and the wind, which sometimes dictates the player's movement.

It is possible to verify that the majority of questions (eight, to be precise) are within the classification values 3 and 4, proven by the calculated average of 3.15 ± 0.55. This allows us to understand that even if the game is not completely understandable by now, it is also not impossible to play. Some design choices that maybe caused some problems while playing the game were:

- The lack of feedback whenever a resource changed, preventing the players from noticing when their ship's health was decreasing, leading to their loss, for example.
- The absence of an explanation about the influence of the wind and flow, making players wonder why the ship seemed to be spinning whenever they moved.
- The absence of any movement animations made it even harder to understand what was happening to the ship while it moved.

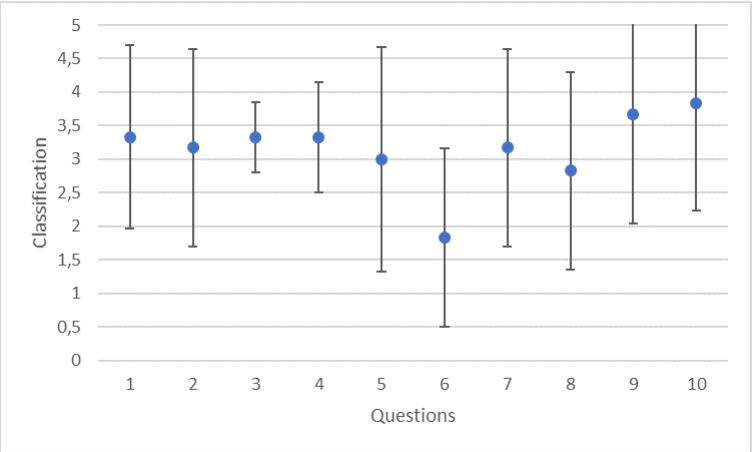

FIGURE 6.3: Average and standard deviation of the classification per question.

Finally, the graph in figure 6.4 contains the average and the standard deviation for the classifications given by each tester. Based on those results, I was able to assess that the answers to the questionnaire somewhat match the seen performance by each player, and their understanding of the game, in the end of the test session. Again, based on this graph we can verify that tester 1 was the one who better understood the game, while tester 5 had the most difficulty to play and understand the game.

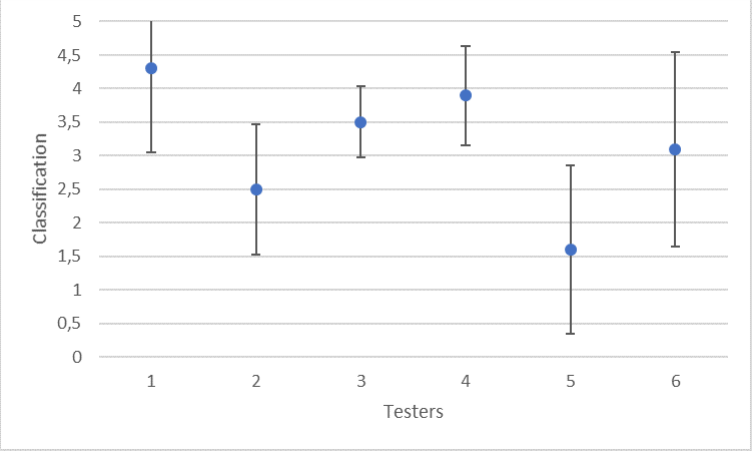

FIGURE 6.4: Average and standard deviation of the classifications per tester.

With all the tests completed and several problems noted, some things came up immediately and, therefore, were extremely urgent to fix, while other issues did not impact the experience that much, making them not so urgent. Because of that, these

problems that arose were divided into **critical problems** and **noncritical problems**, specified below.

There were five **critical problems** identified, and besides just identifying them, we also thought immediately of possible ways of fixing the issues, based on the notes and suggestions.

1. All the testers were confused about the ship movement, due to the lack of animations related to:

   a. Movement between cells. Fixing this would probably also help on the issue regarding the fluidity of the camera.

   b. Ship rotation, influenced by the flow.

   c. Mast rotation, influenced by the wind.

   The last two are extremely important since most testers were not able to understand the influence of both flow and wind.

2. Some testers did not notice the initial directions. There are two ways to solve this issue:

   a. Highlight the direction in the initial menu with a more appealing way.

   b. Allow the player to access the same initial menu, during gameplay. This was also suggested by some testers.

3. Most of the testers tried to click on non-adjacent cells to move the ship. This can be solved by highlighting the adjacent cells while considering restrictions imposed by the gameplay rules.

4. Many testers did not notice the resources changing, either by losing or picking them. This can be solved by highlighting when something related to the resources happens.

5. The need for a brief description of the game, giving a goal to the player. This can be solved by adding information to the initial menu. This information should include an explanation of the influence of the flow and the wind on the ship and mast, respectively.

Regarding the **non-critical problems**, there were three issues identified. Again, instead of just identifying these issues, I also wrote down some notes of my own.

1. Many testers tried different keys to do some actions. One possible way of solving this was to change the keys related to some game actions. However, this will probably be discarded because the game is going to be touch-based.

2. Some testers suggested changing the water cells colours. Specifically, the windy and flat water cells should be coloured a bit more different from one another.

3. Some testers tried to check the resources on the mini-map while playing the game.

6.1.3 Proposed Design Corrections

With all the problems identified, it was time to start fixing them. To help keep track of each issue progress, I created a table containing all the problems, including critical and non-critical. I would write on the said table all that has been done in order to fix each problem, and if it was corrected or not. This information can be seen in table 6.1.

Problem	Corrected	How was it corrected
Critical #1	Yes	All the identified animations were added. Besides those, I also implemented an animation when the player is pushed into a place he cannot go, for example, a land cell or a blocked diagonal water cell.
Critical #2	Yes	The directions text was changed to different colour from the rest of the text (black), as well as bolded.
Critical #3	Yes	Implemented flashing white cells, that would be placed above the cells the player was able to move to.
Critical #4	Yes	Implemented a colouring and zoom each time a resource has changed. For example, if the player picked any resource its number would blink green and zoom out. If the player lost any resource, it would blink red and zoom out.
Critical #5	Yes	A more in-depth description was added to the initial menu, explaining the controls and the interactions between player-game.
Non-critical #1	Yes	Since the game was aimed at mobile touch-based devices, this issue was not considered. However, it is important to note that in the transition to touch-based controls, two buttons were added to open and close the minimap and to repair the ship. Also, the camera rotation, originally on the right and left keys, was implemented into swiping right and left.
Non-critical #2	Yes	The windy cell colour was slightly softened.
Non-critical #3	No	In my idea, it was more interesting to make the player remember his whereabouts, and thus knowing where the nearest resources were.

TABLE 6.1: Usability testing phase problems and their corrections.

Besides the changes related to problems identified, I ended up changing the user interface, in order to make it more relatable to the gameplay experience and easier to understand. Those changes are explained below, and the differences can be verified in figures 6.5 and 6.6.

- Replaced the resources names with pictures depicting each one of them.
- Repositioned the resources display and compass, moving them to the top right corner of the screen.
- Added some transparency to the panels containing the resources and compass information.

Besides those user interface changes and disregarding the different scenarios on the figures, it is also possible to notice other differences:

- The start point had its texture changed.
- It is possible to notice the flashing cells on the top and bottom cells regarding the player, in figure 6.6.
- The buttons to open and close the map, and to repair the ship, can be seen in the bottom left corner of figure 6.6.

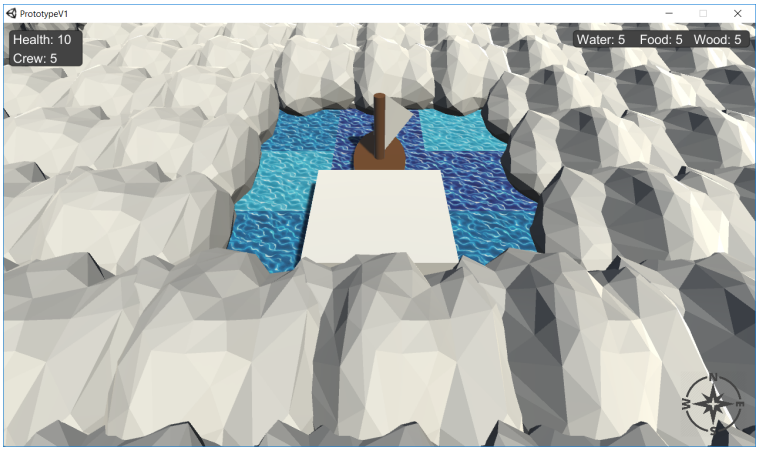

FIGURE 6.5: First version of the prototype's user interface.

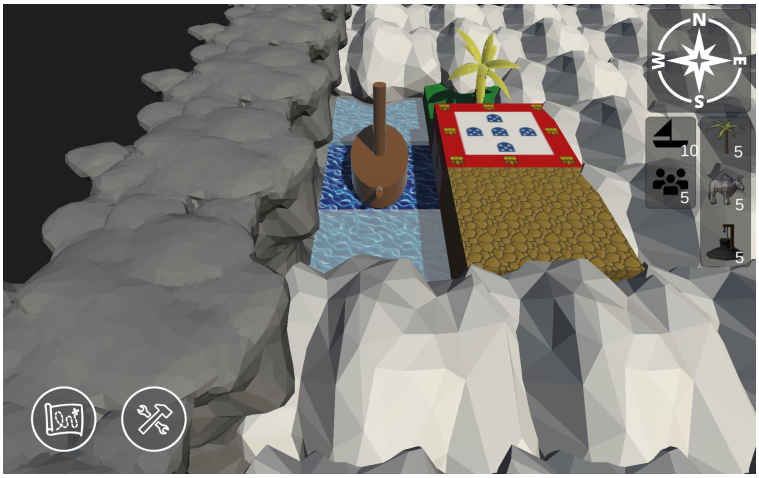

FIGURE 6.6: Final version of the prototype's user interface. Note how the interface has new icons, new placements, new colors, and two previously missing buttons.

6.2 Gameplay Testing

These tests focused on trying to get intervals containing the best values or value ranges for each of the algorithm input parameters. The current prototype was already playable on a mobile platform, with all the corrections to problems that arose in the usability testing phase.

6.2.1 Test Setup

In order to prepare the tests, I had first to decide which input parameters for the generation algorithm were going to be targeted during this testing phase, to register them in the server. The chosen **input parameters** were as follows, indicating each interval. The intervals were defined based on the default values used by the prototype, which are applied if no internet connection exists when trying to reach the server. These default values were initially based on the map side size value of 30, proving to be reasonable values for the parameters.

- **Minimum Distance** – the minimum distance between the start and end points. Its interval is [5, 15], and its default value is 10, which is $\frac{1}{3}$ of the map side size. The interval limits were obtained by decreasing and increasing the default value by 50%.
- **Maximum Distance** – the maximum distance between the start and end points. Its interval is [16, 30], and its default value is 20, which is $\frac{2}{3}$ of the map side size. The interval limits were obtained by decreasing and increasing the default value by 50%, but to avoid an overlap with the minimum distance interval, the lower limit was increased to 16.
- **Fresh Water** – the number of cells containing fresh water. Its interval is [7, 23], and its default value is 15, which is $\frac{1}{2}$ of the map side size. The interval limits were obtained by decreasing and increasing the default value by 50% while making sure the values remained integers.
- **Food** – the number of cells containing food. Its interval is [7, 23], and its default value is 15, which is $\frac{1}{2}$ of the map side size. The interval limits were obtained by decreasing and increasing the default value by 50% while making sure the values remained integers.
- **Wood** – the number of cells containing wood. Its interval is [7, 23], and its default value is 15, which is $\frac{1}{2}$ of the map side size. The interval limits were obtained by decreasing and increasing the default value by 50% while making sure the values remained integers.
- **Directions** – variable depicting if the player would receive the initial directions or not. Its interval is [0, 1], and its default value is 1. From now on, I will be using [No, Yes] instead of [0, 1].

To use AGE, we also needed to set a **design goal** required to be met for a game session to be considered successful. Moreover, because we targeted a casual gameplay setting, we intended for the players to **finish each scenario within five to ten minutes**.

With the parameters and design goals defined, the simulation in the server was ready to be started, so that players could connect to it and start playing. After the

simulation started, the platform generated ten candidates, which were the feature sets containing the input parameters for the algorithm that met the range criteria. All the candidates are stated in table E.1 in Appendix E, containing their feature set values, their generation, and their score based on the design goal.

These tests consisted of individual sessions where each voluntary tester was asked to play a build of the game prototype, usually in their mobile platform, with no information besides the one provided by the prototype itself. The testers got access to the prototype when we sent them the application so that they could install it on their device.

Each player was asked to play the game at least once and connected to the internet, but since the game was on their personal device, they were allowed to play as many times as they wanted. Each time they started the game, the server would pick a candidate and send it back to the game session requesting it. After the game session ended, the game would send the log containing all the actions taken, including the elapsed time and travelled distance, to the server, and then the candidate would be attributed a success or not, depending on its time and winning condition. Every candidate had to be evaluated ten times until it was not selected anymore during that generation.

6.2.2 Problems

During this testing phase, many problems arose either from issues regarding the server, because sometimes data was not correctly sent to or received by the server, or because the number of necessary evaluations was far too high for the current needs.

Regarding the server, sometimes it would crash the evaluation system because of issues regarding the database connection. During the testing phase, this led to some evaluations not being accounted for in the candidate, allowing for some candidates to require more actual game sessions than they should. This can be verified in table E.1 in Appendix E, wherein generation one the maximum evaluations were ten and some candidates have greater numbers than this. Another issue that appeared during the second generation of candidates was the server sending the best candidate from the first generation for game sessions, even though it was not in the current population. One of the successful sessions from Generation 2 had this candidate and, because of that, was not accounted into the total successful sessions.

About data not being correctly sent to or received by the server, it was because some game sessions did not have the information they should at the end. This probably happened because one of three factors: either the game was unable to send said data to the server, the player closed the game during the session, preventing the data from being then sent to the server, or the server was not able to process the data and then rejecting it. However, we still could not yet assess clearly why this happened.

Finally, regarding the number of evaluations, during the first generation of candidates, we realised that ten evaluations per candidate required more availability from players than we could easily get at the time. This lead us to change the required number of evaluations to five, which did not influence any value in the first generation since all the candidates were above five evaluations by that time.

6.2.3 Results and Analysis

Since the candidates are generated through a genetic algorithm, the initial population plays a big role when trying to find the best candidates, sometimes helping the algorithm to converge faster to them. However, in this case, there was not just a computational agent doing some calculations on the candidates. We needed people to play the game and, in that sense, it made things go at a slower pace and increasing the importance of the initial population. Besides that, since we are working with people, they usually need some time to get used to a game and, of course, that fact can influence the results. Finally, some players may think more than others, taking their time to act upon the game, influencing the elapsed time. That is the reason why we initially set 10 gameplays per instance (rule of thumb and statistical reasons).

By the end of the testing phase, **162 game sessions** were **evaluated** out of **198 played sessions**. Moreover, out of those, **59 sessions** were **won**, with only **15** of those being **successful regarding the design goal**. As it is possible to verify, the number of successful sessions was not too high, and that lead us to decide to gather data related to all the sessions in which the player won. Since all the winning sessions and candidates are stated on tables E.2 and E.3, in Appendix E, I chose to show just the candidates with a fitness value different than zero in table 6.2, being identified by their server identification number.

Generation	Candidate	Score	Times Played	Successful Sessions	Minimum Distance	Maximum Distance	Fresh Water	Food	Wood	Directions
1	15468	8.33	12	1	10	17	19	12	15	No
	15469	13.33	15	1	14	29	8	11	17	Yes
	15470	10	10	1	12	19	17	20	11	Yes
	15472	10	10	1	14	25	19	11	16	No
	15476	9.09	11	1	13	22	13	21	13	Yes
2	15486	20	5	1	14	29	8	20	11	Yes
	15488	28.57	7	2	14	29	19	11	16	Yes
	15489	20	5	1	14	25	8	11	17	No
	15490	14.29	7	1	10	21	15	13	15	No
	15492	20	5	1	14	25	19	20	11	No
	15494	60	5	3	12	19	17	20	11	Yes
	15495	16.67	6	1	12	19	17	20	11	Yes

TABLE 6.2: Successful candidates, across two generations.

Looking at table 6.2, we can verify that out of the twenty candidates, only twelve were attributed some fitness value different from zero, meaning that they were successful at least one time. However, relating the number of successes and their times played, we can verify that there is a rather significant disparity between those values. The candidates score is calculated by summing all their achieved scores and dividing them by the number of times each candidate was used in a play session.

In Generation 1, even though there were theoretically ten evaluations needed for each candidate, the number of successes per candidate was no more than one. Even though it is too early to get concrete conclusions, it is possible to get some ideas through the presented results from this generation:

- Some parameters could have their intervals changed, to better fit the evaluations results. For example, even though the lowest value available for Minimum Distance was 7, the best candidates did not have a value lower than 10.
- Showing the initial directions seems to be the most favourable option in the successful sessions, appearing in 60% of them.

In Generation 2, even though the required number of evaluations was reduced to five, some candidates were played more times again due to server problems. However, even with only 57 evaluated sessions against 105 of the first generation, the second generation was able to get more successful generations than the first, with ten versus five. Also, two candidates from this generation were successful more than once, with two times for candidate 15488 and three times for candidate 15494, allowing them to reach the highest fitness values (score) of this population, respectively 28.57 and 60. It is possible to go further on the previous conclusions from Generation 1.

- As suspected in the first generation, some parameters could indeed have their intervals changed. For example, taking the same parameter as before, Minimum Distance still seems to perform better with values above or equal to ten.
- Showing the initial directions got a slight increase in its presence in the successful sessions, increasing its quantity to 70%. On the other hand, this also signals that it is possible to solve several scenarios without this information.

In the following analysis, I will be using Generation 2 as my basis for results, since being the last completed generation, it can be considered to be the one having the most favourable candidates.

Since the defined design goal had a time constraint, it is also important to analyse how each candidate performed in that regard and see if there are any conclusions to take from it (table 6.3).

Candidate	**Session Time (s)**	**Average Time (s)**	**Start-End Distance**	**Average Distance**
15486	343	343	16.55	16.55
15488	373	456	24.21	25.35
	539		26.48	
15489	318	318	20.22	20.22
15490	492	492	10.44	10.44
15492	455	455	16.97	16.97
15494	382	359.67	14.56	16.79
	383		18.6	
	314		17.2	
15495	368	368	12.17	12.17

TABLE 6.3: Session times and distances from start to end, for each successful candidate.

Analysing table 6.3, we can point off some important aspects regarding session times and distance from the start to the end.

- 70% of the successful sessions took between 300 and 420 seconds to finish, meaning that it seems candidates are getting closer to the lowest time allowed (300), rather than having a similar dispersion within the design goal interval [300, 600]. In comparison, only approximately 28% of all sessions from Generation 2 had a time between 300 and 420 seconds, which is not a bad value since almost 59% of the times are below 300 seconds.
- 70% of the successful sessions had a distance from the start to the end between 10 and 20, and 30% was above 20. In comparison, approximately 83% of all

sessions from Generation 2 had a distance between 10 and 20, showing that the successful sessions distances are not far from the global values.

- Notably, 50% of the sessions had both a time between 300 and 420 seconds and a distance between 10 and 20. Even though I cannot create a definitive connection between those time and distance intervals, they seem to be somehow related.
- Out of all sessions, 20% had a time greater than 420 seconds while the player did not know the initial directions to the end point. This means that hiding those directions from the player may increase the time they spend to finish the level successfully.

Considering the results from the second generation, I gathered all the parameters in table 6.4 and accounted how many times each value occurred in a certain parameter. Based on the table values, if we were to pick the best combination of values by choosing the most frequent ones, then it would be {14, 19, 17, 20, 11, Yes}. We can notice that this combination is almost the candidate 15494 who got the highest score on this generation, having only a different value for Minimum Distance. However, even if it seems that both the best candidate and the combination of the most frequent values are heading in the same direction, it is important to note that some parameters have other values that also appear frequently, usually with a difference of one from the most frequent value.

Parameter	**Value**	**Times**
Minimum Distance	10	1
	12	4
	14	5
Maximum Distance	19	4
	21	1
	25	2
	29	3
Fresh Water	8	2
	15	1
	17	4
	19	3
Food	11	3
	13	1
	20	6
Wood	11	6
	15	1
	16	2
	17	1
Directions	Yes	7
	No	3

TABLE 6.4: Number of times each parameter's value was successful.

So, taking into account that more value possibilities may exist, I will pick the two most frequent values of each parameter from table 6.4, and see where that leads us.

- For Minimum Distance, the values are 14 and 12.
- For Maximum Distance, the values are 19 and 29.

- For Fresh Water, the values are 17 and 19.
- For Food, the values are 20 and 11.
- For Wood, the values are 11 and 16.
- For Directions, the values are, of course, Yes and No.

So, from these values, it would be possible to make combinations of them and see if that would give any significant results. However, since I would not want to bias the generation algorithm and its results into accepting these values as the best, we should let it go as it is now. Talking of which, the algorithm ended up generating a third generation with nine candidates, stated in table E.1 in Appendix E. Even if none of those candidates was tested, I will be stating their most common values for each parameter, and compare those to the previous values taken from generation 2.

- Minimum Distance – 12 and 14.
- Maximum Distance – 19 and a draw between 25 and 29.
- Fresh Water – 17 and 19.
- Food – 20 and 11.
- Wood – 11 and a draw between 15, 16, and 17.
- Directions – Yes and No.

As expected, those values match almost perfectly with the most frequent values from generation 2, and that may indicate that these values are the ones most likely to pass on through future generations.

In conclusion, through two generations I found some frequent values that seem to be performing well among the players. However, this cannot be seen as a final statement, since more evaluation is needed. Also, most of the players that achieved the design goal finished the game between 300 and 420 seconds, meaning that even if the mentioned values are performing well, they are not covering all the intended time range between five and ten minutes.

Quoting this thesis adviser, "we can look at this as a search for the parameters for the best candidate, or, we can be searching for the intervals that are likely to elicit gameplay within the boundaries defined, and still generate diverse play scenarios to keep the player interested. Should the ranges be narrower, or larger, to allow for the circumstantial difficulty loss? That can be a more interesting design consideration as we learn to deal with PCG in the dark art of Game Design" (oral quote from Licínio Roque, 2017). As pointed before, even with 162 evaluated sessions, only 15 were effectively successful, which can indicate that there is some difficulty inherent to the game. Although this difficulty can be addressed varying the intervals that are currently being used, it is not possible for us to determine if we should decrease or increase said intervals, since that would require even more considerations through more testing.

Chapter 7

Further Work

7.1 Critical Aspects to Correct

There were some aspects regarding this project that should be fixed as future work. Firstly, as it was mentioned in Gameplay Tests, there were several problems regarding the server containing the AGE platform. If the prototype would continue to use this platform, it is almost needed that those issues need to be fixed to keep all the aspects of the testing phase running smoothly and with minor annoyances. Secondly, since the game was targeted at mobile devices which sometimes are not that powerful and capable of running demanding applications, the prototype should have to go through a process of performance optimisations. Because even if Unity does make some optimisations by itself, there are certainly some aspects that could be improved by the developer, allowing the game to run in a more stable condition even on low-end devices. Finally, although the game contains almost every visual aspect needed, it was not developed as much as it should. It would be interesting, and probably needed, to make the game look and feel more related to its concept, while keeping it visually interesting to its audience.

7.2 Future Developments

After the corrections based on the usability testing phase results, there was no formal verification that those changes did indeed improve the overall experience of the game, mostly due to time constraints. For future development, it would be interesting to verify how the interface changes impacted the current gameplay experience.

Regarding the generation algorithm, some aspects ended up not being implemented, such as the addition of enemies, treasures, and resources distribution based on land cells. Enemies would occupy certain water cells, having the player fight them allowing him to reach those cells. Treasures were mentioned when there were conversations of having different levels, with each one of those having a different amount of treasures, and making the player replay the levels to gather all the treasures. They would be found on certain land cells, being considered as achievements for the player. Finally, there was also the idea of different land cells having different probabilities of containing certain resources, since plains and hills acted essentially the same way. These elements were not implemented because of their complexity and the lack of available time.

Finally, and depending on what we want the PCG for, the gameplay testing phase should continue to tune the current candidates further, and find the best values for each parameter given the current design goal since two generations are not enough to do it.

Chapter 8

Conclusions

In this thesis, we studied the employment of procedural content generation methods in the design of an original game, allowing for the adaptation of parameter values used in the creation of game environments. For this, we developed a game prototype focused on exploration, where the players must manage their resources and ship to reach an end point. The prototype creates its environments using a map generation algorithm, developed during this thesis.

To ensure that the prototype was easy to understand, usability tests were made with six subjects. The results from these tests helped on improving the prototype, with most changes being made to its user interface, controls, and animations.

During this thesis, the prototype was integrated with a server that contained a tool that would help the evolution of the candidates containing the parameters, by using a genetic algorithm. At the beginning of each game session, the server would send a candidate set of parameters to the prototype for the map generation algorithm. At the end of a session, the prototype would send a log file to the server, which would then analyse it and attribute scores to the candidates based on predefined design goals. At the end of each generation, the tool kept the best candidates and generated new ones.

After the prototype-server integration was complete, several gameplay tests were performed, helping the server tool to evolve the candidates with its genetic algorithm. At the end of this testing phase, and with two completed generations, it was possible to verify that some values performed better than others. For the defined parameters, if we were to select the most successful values for each parameter, we would end up with the set {14, 19, 17, 20, 11, Yes}. However, since playing a video game is dependent on each person experience, more tests would be needed to arrive at more conclusive values on the best value ranges for each parameter.

The AGE tool proved to be of good use because even with its flaws and instability, it helped us on having a range of values to test the generation algorithm with while allowing us to define the desired gameplay experience. Besides that, its genetic algorithm was important on achieving and evolving those values by attributing scores to each candidate set. Finally, it also gave us tools to store gameplay data that could later be analysed, enabling us to retrieve data that could prove crucial when analysing the game sessions.

The integration of PCG methods within a game right from the start proved to create interesting outcomes, and we only scratched the surface of what can be accomplished using PCG methods. Even if we only used six initial parameters for the

algorithm, it already created interesting and explorable scenarios. The addition of more parameters, or even the inclusion of a PCG method during the game sessions could prove to improve the experience even further.

At the end of this dissertation, it is possible to conclude that we successfully designed a game that incorporates a PCG algorithm that generates its scenarios. Moreover, the developed video game works as intended, allowing the players to explore a vast world generated using the said algorithm, which contains parameters that were defined with a desired gameplay experience in mind. The generation algorithm proved to create adequate scenarios that seem fit for the wanted experience, but more elements could be added to it, such as the components that were discarded due to time constraints, and verify if those elements improved the desired experience.

During this thesis' research and development, I had the opportunity to deepen my knowledge base regarding several computer engineering topics. Most notably, even though I had heard about procedural content generation before, it was only with this thesis that I implemented an algorithm of this type, allowing me to learn the design process and better predict the outcomes that can result from a map generation algorithm. Besides that, with some research in the game design field, I was able to understand better how to concept a game and implement it given the concept. The integration with the server also helped me realise that many problems can occur when a software depends on other thirt-party systems, and that we must be prepared for such issues. Finally, I improved my capability of devising and conducting usability tests, which can be rather difficult to perform. In conclusion, this thesis helped me to grow my skills, hopefully leading me to become a better professional.

References

Adams, Tarn and Zach Adams (2006). *Dwarf Fortress*. Bay 12 Games.

A.I. Design (1980). *Rogue*. Epyx.

Atari (1972). *Pong*. Atari.

Bay 12 Games (n.d.). *Dwarf Fortress*. URL: http://www.bay12games.com/dwarves/screens/adv44.png (visited on Aug. 22, 2017).

Braben, David and Ian Bell (1984). *Elite*. Acornsoft.

Browne, Cameron (2007a). *Yavalath*. URL: http://www.cameronius.com/games/yavalath/.

— (2007b). *Yavalath*. URL: http://www.cameronius.com/games/yavalath/yavalath-deluxe-1.jpg (visited on Aug. 22, 2017).

Craveirinha, Rui, Nuno Barreto, and Licínio Roque (2016). "Towards a Taxonomy for the Clarification of PCG Actors' Roles". In: *CHI PLAY '16 Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*.

Craveirinha, Rui and Licínio Roque (2016). "Exploring the Design-Space: The Authorial Game Evolution Tool Case-Study". In: *Proceedings of the 13th International Conference on Advances in Computer Entertainment Technology*. ACE2016. Osaka, Japan: ACM.

Craveirinha, Rui, Lucas Santos, and Licínio Roque (2013). "An Author-Centric Approach to Procedural Content Generation". In: *10th International Conference on Advances in Computer Entertainment*.

Donjon (2011). *Fractal World Generator*. URL: https://donjon.bin.sh/world/ (visited on Aug. 22, 2017).

Electronic Arts (2009a). *Left 4 Dead*. URL: https://web-vassets.ea.com/Assets/Richmedia/Image/Screenshots/l4d08.jpg (visited on Aug. 22, 2017).

— (2009b). *Spore*. URL: https://web-vassets.ea.com/Assets/Richmedia/Image/Screenshots/spor-pc-creatureretai2-screenshot.png (visited on Aug. 22, 2017).

Envato Tuts+ (2013). *Generate Random Cave Levels Using Cellular Automata*. URL: https://gamedevelopment.tutsplus.com/tutorials/generate-random-

cave-levels-using-cellular-automata--gamedev-9664 (visited on Aug. 22, 2017).

Evolutionary Games (2010). *Galactic Arms Race*.

Gamepedia (2016). *Seeds*. URL: http://bindingofisaacrebirth.gamepedia.com/Seeds (visited on Jan. 7, 2017).

GR3 Project (2005). *La-Mulana*. GR3 Project.

Hello Games (2016). *No Man's Sky*. Hello Games.

Higinbotham, William and Robert Dvorak (1958). *Tennis for Two*.

Huizinga, Johan (1944). *Homo Ludens: A Study of the Play-Element in Culture*. Routledge & Kegan Paul Ltd.

Hunicke, Robin, Marc LeBlanc, and Robert Zubek (2004). "MDA: A Formal Approach to Game Design and Game Research". In: *Game Design and Tuning Workshop*.

Kazemi, Darius (2009). *Spelunky Generator Lessons*. URL: http://tinysubversions.com/spelunkyGen/ (visited on Jan. 17, 2017).

Lee, Joel (2015). *No Man's Sky and the Future of Procedural Games*. URL: http://www.makeuseof.com/tag/no-mans-sky-future-procedural-games/ (visited on Jan. 18, 2017).

Martinho, Carlos, Pedro Santos, and Rui Prada (2014). *Design e Desenvolvimento de Jogos*. FCA - Editora de Informática, Lda.

Maxis (2008). *Spore*. Electronic Arts.

Metacritic (2017a). *No Man's Sky*. Metascore: 61/100, User score 2.6/10. URL: http://www.metacritic.com/game/pc/no-mans-sky (visited on Aug. 21, 2017).

— (2017b). *Spelunky*. Metascore: 90/100, User score 7.2/10. URL: http://www.metacritic.com/game/pc/spelunky (visited on Aug. 21, 2017).

Mossmouth (2008, 2012). *Spelunky*. Mossmouth.

Nicalis (2014). *The Binding of Isaac: Rebirth*. Nicalis.

Nintendo Creative Department (1985). *Super Mario Bros.* Nintendo.

No Man's Sky (n.d.). *No Man's Sky*. URL: https://nmswp.azureedge.net/wp-content/uploads/2017/02/NewEridu.png (visited on Aug. 22, 2017).

Pinterest (n.d.). *Computer Space*. URL: https://s-media-cache-ak0.pinimg.com/originals/a7/d1/70/a7d170ea621d836047b73e42988cf1d5.jpg (visited on Aug. 22, 2017).

PlayStation (n.d.). *The Binding of Isaac: Rebirth*. URL: https://media.playstation.com/is/image/SCEA/the-binding-of-isaac-rebirth-screenshot-01-ps4-psvita-us-15oct14?$MediaCarousel_Original$ (visited on Aug. 22, 2017).

Red Blob Games (2015). *Making maps with noise functions*. URL: http://www.redblobgames.com/maps/terrain-from-noise/ (visited on July 20, 2017).

Ritzl, Björn (2016). *The Rogue Archive*. URL: https://britzl.github.io/roguearchive/images/rogue.png (visited on Aug. 22, 2017).

Russel, Steve et al. (1962). *Spacewar!*

Schell, Jesse (2008). *The Art of Game Design: A Book of Lenses*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 0-12-369496-5.

Schreiber, Ian and Sebastian Sohn (2013). *Process of Design & MDA Framework*. URL: https://learn.canvas.net/courses/3/pages/level-4-dot-1-process-of-design-and-mda-framework (visited on Aug. 12, 2017).

Shaker, Noor, Georgios Yannakakis, and Julian Togelius (2010). "Towards automatic personalized content generation for platform games". In: *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Smith, Gillian (2014). "Understanding Procedural Content Generation: A Design-Centric Analysis of the Role of PCG in Games". In: *CHI '14 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

Smith, Gillian, Jim Whitehead, and Michael Mateas (2010). "Tanagra: a mixed-initiative level design tool". In: *Proceedings of the Fifth International Conference on the Foundations of Digital Games*.

Spelunky World (n.d.). *Spelunky*. URL: http://www.spelunkyworld.com/images/spelunky-ss05.jpg (visited on Aug. 22, 2017).

Syzygy Engineering (1971). *Computer Space*.

Togelius, Julian, Noor Shaker, and Mark J. Nelson (2016). "Introduction". In: *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Ed. by Noor Shaker, Julian Togelius, and Mark J. Nelson. Springer, pp. 1–15.

Togelius, Julian, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne (2010). *Search-based Procedural Content Generation*.

Togelius, Julian, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis (2011). "What is Procedural Content Generation?: Mario on the borderline". In: *Proceedings of the 2nd Workshop on Procedural Content Generation in Games*.

Vaishnavi, Vijay and Bill Kuechler (2008). *Design Science Research in Information Systems*. URL: http://www.desrist.org/design-research-in-information-systems/ (visited on Jan. 20, 2017).

Valve Corporation (2008). *Left 4 Dead*. Valve Corporation.

Wikidot (2010). *Fractal*. URL: http://pcg.wikidot.com/pcg-algorithm:fractal (visited on July 20, 2017).

— (2013). *Spelunky*. URL: http://pcg.wikidot.com/pcg-games:spelunky (visited on Jan. 17, 2017).

— (2016). *Cellular Automata*. URL: http://pcg.wikidot.com/pcg-algorithm:cellular-automata (visited on July 20, 2017).

Wikipedia (2006). *Pong*. URL: https://en.wikipedia.org/wiki/Pong#/media/File:Pong.png (visited on Aug. 22, 2017).

— (2007). *Elite*. URL: https://en.wikipedia.org/wiki/File:BBC_Micro_Elite_screenshot.png (visited on Aug. 22, 2017).

— (2009). *La-Mulana*. URL: https://en.wikipedia.org/wiki/La-Mulana#/media/File:La-Mulana_gameplay.png (visited on Aug. 22, 2017).

— (2010). *Spacewar!* URL: https://en.wikipedia.org/wiki/Spacewar!#/media/File:Spacewar!-PDP-1-20070512.jpg (visited on Aug. 22, 2017).

— (2011). *Super Mario Bros.* URL: https://en.wikipedia.org/wiki/File:NES_Super_Mario_Bros.png (visited on Aug. 22, 2017).

— (2012a). *Moore Neighbourhood*. URL: https://en.wikipedia.org/wiki/File:CA-Moore.png (visited on Aug. 22, 2017).

— (2012b). *Von Neumann Neighbourhood*. URL: https://en.wikipedia.org/wiki/File:CA-von-Neumann.png (visited on Aug. 22, 2017).

— (2013a). *Galactic Arms Race*. URL: https://en.wikipedia.org/wiki/Galactic_Arms_Race#/media/File:Galactic_Arms_Race_Screenshot.png (visited on Aug. 22, 2017).

— (2013b). *Tennis for Two*. URL: https://en.wikipedia.org/wiki/Tennis_for_Two#/media/File:Tennis_For_Two_on_a_DuMont_Lab_Oscilloscope_Type_304-A.jpg (visited on Aug. 22, 2017).

— (2016). *Spelunky*. URL: https://en.wikipedia.org/wiki/Spelunky (visited on Jan. 17, 2017).

— (2017a). *History of video games*. URL: https://en.wikipedia.org/wiki/History_of_video_games (visited on Jan. 17, 2017).

— (2017b). *No Man's Sky*. URL: https://en.wikipedia.org/wiki/No_Man's_Sky (visited on Jan. 18, 2017).

— (2017c). *Unix time*. URL: https://en.wikipedia.org/wiki/Unix_time (visited on July 18, 2017).

Yannakakis, Georgios N. and Julian Togelius (2011). "Experience-Driven Procedural Content Generation". In: *IEEE Transactions on Affective Computing* 2.3, pp. 147–161.

— (2015). "Experience-Driven Procedural Content Generation (Extended Abstract)". In: *International Conference on Affective Computing and Intelligent Interaction (ACII)*.

Appendix A

Server Configuration

Object	Type	ID
Player	Game Object	193
FreshWater	Game Object	191
Food	Game Object	198
Wood	Game Object	192
Health	Game Object	190
Crew	Game Object	199
WaterCell	Game Object	188
LandCell	Game Object	189
Minimap	Game Object	196
World	Game Object	194
Ship	Game Object	197
Directions	Game Object	195
TotalTime	Game Object	200
DistanceTraveled	Game Object	201
attemptedMoveTo	Event	166
movedTo	Event	167
picked	Event	160
clicked	Event	162
opened	Event	163
closed	Event	164
has	Event	159
won	Event	158
lost	Event	161
repaired	Event	165

TABLE A.1: Game objects and events required for the server.

Subject	Event	Predicate	Other
World	has	WaterCell	Predicate value: Matrix value (x,y,z): Position in the map
		LandCell	Predicate value: Matrix value (x,y,z): Position in the map
		Player	(x,y,z): Position in the map
		FreshWater	Predicate value: Quantity
		Food	Predicate value: Quantity
		Wood	Predicate value: Quantity
Player	has	FreshWater	Predicate value: Quantity
		Food	Predicate value: Quantity
		Wood	Predicate value: Quantity
		Health	Predicate value: Quantity
		Crew	Predicate value: Quantity
		Directions	Predicate value: 0 or 1
		TotalTime	Predicate value: total time in the end, in seconds
		DistanceTraveled	Predicate value: distance travelled by the player, in cell units
	clicked	WaterCell	(x,y,z): Position in the map
		LandCell	(x,y,z): Position in the map
	attemptedMoveTo	WaterCell	(x,y,z): Position in the map
	movedTo	WaterCell	(x,y,z): Position in the map Predicate value: who had influence on ship's movement (1-Player, 2-Flow, 3-Wind)
	picked	FreshWater	(x,y,z): Position in the map Predicate value: amount picked
		Food	(x,y,z): Position in the map Predicate value: amount picked
		Wood	(x,y,z): Position in the map Predicate value: amount picked
	opened	Minimap	-
	closed	Minimap	-
	won	-	-
	lost	-	-
	repaired	Ship	-

TABLE A.2: Game objects required for the server, events connecting them, and additional information.

Appendix B

Development Log

8th February 2017 to 14th February 2017

Definition and documentation of the Game Design Document, specifying:

- Game concept;
- Game components:
  - Player;
  - Start point;
  - End point;
  - Water cells;
  - Land cells;
  - Resources;
  - Map;
  - Marine obstacles;
  - Treasures
- Mechanics

15th February 2017 to 21st February 2017

Definition of the system architecture, describing the interactions between its different entities.
Definition of the initial parameters for the procedural content generation algorithm.
Definition of the existent steps in the procedural content generation algorithm.
Definition of the data structure to use in the procedural content generation algorithm.

22nd February 2017 to 28th February 2017

Implementation of the water and cells procedural generation (without subtypes).
Implementation of the land agglomerates.
Implementation of the land cells subtypes, given these criteria:

- Definition of the start and end points, making sure that it is possible to get from one to the other.

- Land cells without water cells in their Von Neumann neighbourhood become mountain, not containing resources.
- The remaining cells (coast cells) will be defined taking into account that they have 1/3 chance of becoming any of the remaining subtypes (plains, hills, cliffs).

Implementation of the resources distribution for the different land cells, being that not all land cells contain resources.

1st March 2017 to 7th March 2017

Implementation of the water cells subtypes, given these criteria:

- The subtype for the water cells near land is influenced based on the land cells around them.
- The subtype for the water cells in the high seas is defined influenced by the water cells around them.

Implementation of the lake detection algorithm, making sure that water cells in lakes always have the "flat" subtype.
Beginning of the visual implementation of the game, starting to convert the matrix values into a 3D map with cubes.

8th March 2017 to 14th March 2017

Implementation of the flow and wind for water cells, given these criteria:

- The flow/wind of one cell is influenced by the cells around it, allowing to existence of small flow or wind paths. Implementation of the player's ship:
- Initial movement mechanic, based on clicks.
- Camera rotation, currently using the right/left keys.

Research about ways of limiting the player's vision of the terrain, followed by its implementation.

15th March 2017 to 21st March 2017

Implementation of the ship's health system, which is modified given these criteria:

- Reduced if player is in a stormy cell.
- Reduced if player is pushed against a cliff cell.
- Increased if the player uses wood to repair ship.
- Increased if player returns to start point.

Implementation of the interactions with each water cell, based on its subtype, flow, and wind.
Implementation of the interactions with the ship's health system.

22nd March 2017 to 28th March 2017

Implementation of the interactions with the land cells:

- Plains and hills to gather resources.
- Cliffs to reduce ship's health.

Research and implementation of the User Interface (UI) for the game:

- Information about each resource's quantity.
- Compass allowing the player to know the camera's direction.

Preparation of the usability testing phase, to be discussed in the next meeting.
Implementation of a mini-map, showing only the cells where the player has passed, allowing him to know the amount of map discovered, since it has a limited field of view.

29th March 2017 to 4th April 2017

Changes to the method of limiting the player's vision of the terrain, turning areas not yet discovered to be under fog, only disappearing when the player passes near them.
Implementation of both ship and mast rotation, based on flow and wind respectively.
Implementation of textures and 3D models in the world map:

- Textures for each water and land cells subtype.
- Models for each resource.

Creation of an initial tutorial for the player, taking into account that the usability testing prototype will be played in a computer.

- Interaction using the mouse and right/left arrow keys.
- Initial directions the player should follow to get to the end.

Usability tests done with six people.

5th April 2017 to 11th April 2017

Analysis of the usability testing results and conclusions about them.

12th April 2017 to 18th April 2017

Modification of the User Interface (UI) positioning.
Implementation of 2D textures for the resources fresh water, food, and wood, based on their 3D models. (UI)
Implementation of textures for the ship's health and crew. (UI)
Modification of the ship's movement, making that it slides to a cell instead of "jumping" to it.
Implementation of highlighters indicating which cells the player is currently allowed to move to.

19th April 2017 to 25th April 2017

Modification of the compass, changing its colour to white on a grey background. (UI)
Modification of the colours of the limiter clouds in the game world.
Modification of the start point texture for the Portugal's flag.
Discussion about the utilisation of the Crowdplay server:

- How will it be used in the prototype.
- What will be sent and received by the server.

26th April 2017 to 2nd May 2017

Study about the functioning of the server requests, as well as all the required data.
Implementation of some functions for the communication with the server.
Incorporation of a framework used for reading JSON files, which will be used to read the information obtained through the server communications.

3rd May 2017 to 9th May 2017

Implementation of buttons to open/close the mini-map, and repair ship.
Implementation of the remaining functions for the communication with the server.
Incorporation of external code to calculate the Epoch time for the dates, being required for the server.
Implementation of loggings required for the server.
Registration of objects and input parameters (feature set) in the server.

10th May 2017 to 16th May 2017

Implementation of the right and left swipes, used to move the game's camera on mobile devices.
Addition of sounds:

- Game's ambiance.
- Repair ship.
- Open/close mini-map.

17th May 2017 to 23rd May 2017

Tests related to the functioning of the prototype with the server, allowing to find some issues in the server.

24th May 2017 to 30th May 2017

Addition of two more objects to the server, related to travelled distance and elapsed time.
Server was down for some days of this period.

31st May 2017 to 6th June 2017

Server remained down for some days of this period.
Preliminary tests with laboratory colleagues, to make sure everything was working as expected.

7th June 2017 to 13th June 2017

Some issues with the server which resulted on the impossibility of making tests.

14th June 2017 to 20th June 2017

Definition of the gameplay testing experience:

- Values for the input parameters.
- Success factor for the tests.

21st June 2017 to 22nd August 2017

Gameplay Testing
Data gathering and analysis.

Appendix C

Work Backlog

- Implement water cells
  - Implement water cells subtypes
    - Implement subtype based on surrounding land cells
    - Implement subtype based on surrounding water cells
    - Implement algorithm for lake detection to have flat water cells
  - Implement water cells flow
    - Implement flow based on surrounding water cells flow
  - Implement water cells wind
    -
    - Implement wind based on surrounding water cells wind
  - Implement enemies that appear on water cells (discarded)
- Implement land cells
  - Implement land agglomerates
  - Implement start and end points
    -
    - Implement algorithm to verify if there is a path between start and end points
  - Implement land cells subtypes
    - Implement mountain land cells in the middle of land agglomerates, cannot contain resources
    - Implement coast cells: 1/3 chance of being any of the remaining three subtypes
  - Implement resources distribution
    - Implement resources distribution based on the land cell
  - Implement treasures distribution (discarded)
- Implement visual representation of the matrix, using coloured cubes.
  - Implement textures for each cell subtype
  - Implement 3D models for each resource (fresh water, food, wood)
  - Add map limiters
    - Change map limiters' colour
  - Change start point texture

- Implement player's ship
  - Implement movement mechanics, through clicks/touches
    - Implement ship's animations for movement
  - Implement camera rotation using directional keys (usability prototype)
  - Implement camera rotation using swipes (final prototype)
  - Implement player's view limiter using a shader
  - Implement player's health system and resource management system
    - Implement water cells' interactions
    - Implement land cells' interactions
  - Implement clouds to limit player view and that disperse on player's passing (replacing previous view limiter method that used shader)
  - Implement boat rotation, influenced by water cell's flow
    - Implement ship's animations for boat rotation
  - Implement mast rotation, influenced by water cell's wind
    - Implement ship's animations for mast rotation
  - Implement accessible cells' highlighters
- Implement UI
  - Implement resources information
  - Implement compass to show direction
    - Change compass colour
  - Implement minimap
    - Implement button to open minimap
  - Implement initial tutorial for the player
  - Replace UI resources text for 2D models of them
  - Implement button to repair ship
- Implement functions to connect to the Crowdplay server
  - Integrate JSON framework to parse server text
    - Implement loggings to be sent to server
  - Integrate Epoch time calculation to send times to server
  - Register objects in the server
- Implement some sounds into the game

Appendix D

Usability Testing Data

Questionnaire Structure

Name:
Age:

	Totally Disagree 1	2	3	4	Totally Agree 5
1. I understood what I had to do.					
2. I understood how I could do my actions.					
3. I understood what each game's component meant.					
4. I understood the outcome of each one of my actions.					
5. I understood that flow and wind could influence my movement.					
6. I understood the flow's and wind's directions.					
7. I understood that different cells produced different results.					
8. I was able to locate myself in world through the mini-map.					
9. I perceived the world's limits.					
10. I understood I could pick up resources from land cells.					

11. What visual aspects would you change, if any?

12. Any comment or suggestion?

Questionnaire Results

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**
T1	5	4	4	4	5	1	5	5	5	5
T2	4	2	3	3	1	1	3	2	3	3
T3	3	3	3	4	4	4	4	3	3	4
T4	4	4	3	3	4	3	4	4	5	5
T5	1	1	4	4	1	1	1	1	1	1
T6	3	5	3	2	3	1	2	2	5	5

TABLE D.1: Testers (T) classifications to each question (Q).

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**
Average	3.33	3.17	3.33	3.33	3	1.83	3.17	2.83	3.67	3.83
Standard Deviation	1.37	1.47	0.52	0.82	1.67	1.33	1.47	1.47	1.63	1.6

TABLE D.2: Average and standard deviation values for each question (Q).

	T1	**T2**	**T3**	**T4**	**T5**	**T6**
Average	4.3	2.5	3.5	3.9	1.6	3.1
Standard Deviation	1.25	0.97	0.53	0.74	1.26	1.45

TABLE D.3: Average and standard deviation values for each tester (T).

Suggestions

	Importance
Tester 1	
Implement movement animations, specially related to the flow and wind directions.	Critical
Give the option to restart the game.	Noncritical
Initial information should appear in-game.	Noncritical
Tester 2	
None	-
Tester 3	
None	-
Tester 4	
Differentiate a bit more the navigable cells colors.	Noncritical
Tester 5	
Would change the mini-map, although did not explain how.	Noncritical
A brief description of the game should exist.	Critical
There should be feedback from hitting obstacles.	Critical
Tester 6	
Camera should be farther, and the boat shouldn't be so close to a border of the screen.	Noncritical
The camera's movement has to be more fluid.	Critical
The flow and wind direction and their influence on the boat and sail should be well explained, since not everyone knows how it works.	Critical
Change the middle-button symbol from the initial menu.	Noncritical

Observations

	Importance
Tester 1	
Tried to look for resources on the mini-map.	Noncritical
Due to experience in games, tried to click on the 'Tab' key to open the mini-map.	Noncritical
Tester 2	
Unsure about the objective of the game.	Critical
Couldn't figure out (initially) what was the boat's front.	Critical
Couldn't figure out the boat's rotations.	Critical
Felt a little confused about the different game components.	Critical
Tester 3	
Tried to close the initial menu with the 'Enter' key.	Noncritical
Tried to click non-adjacent cells.	Critical
Not sure if the tester understood there were resources.	Critical
Tester 4	
Couldn't perceive the destination point.	Critical
Didn't notice the initial directions.	Critical
Tried to click on the "Escape" key to open the mini-map.	Noncritical
Tester 5	
Didn't notice the direction to be followed.	Critical
Tester 6	
Couldn't perceive the boat's direction.	Critical
Couldn't perceive the wind's influence.	Critical

Appendix E

Gameplay Testing Data

Candidate	Generation	Score	Times Played	Minimum Distance	Maximum Distance	Fresh Water	Food	Wood	Directions
15467	1	0	11	7	23	19	12	15	No
15468	1	8.33	12	10	17	15	18	9	No
15469	1	13.33	15	14	29	8	11	17	Yes
15470	1	10	10	12	19	17	20	11	Yes
15471	1	0	12	10	20	17	17	20	Yes
15472	1	10	10	14	25	19	11	16	No
15473	1	0	7	9	20	16	16	12	No
15474	1	0	8	10	21	15	13	18	Yes
15475	1	0	9	14	20	13	10	19	Yes
15476	1	9.09	11	13	22	13	21	13	Yes
15486	2	20	5	14	29	8	20	11	Yes
15487	2	0	5	12	19	17	11	17	Yes
15488	2	28.57	7	14	29	19	11	16	Yes
15489	2	20	5	14	25	8	11	17	No
15490	2	14.29	7	10	21	15	13	15	No
15491	2	0	8	7	23	19	12	18	Yes
15492	2	20	5	14	25	19	20	11	No
15493	2	0	4	12	19	17	11	16	Yes
15494	2	60	5	12	19	17	20	11	Yes
15495	2	16.67	6	12	19	17	20	11	Yes
15496	3	0	0	12	19	17	20	11	Yes
15497	3	0	0	12	19	17	20	11	Yes
15498	3	0	0	12	19	17	20	17	No
15499	3	0	0	14	25	8	11	11	Yes
15500	3	0	0	10	25	19	20	11	No
15501	3	0	0	14	21	15	13	15	No
15502	3	0	0	14	29	19	11	16	Yes
15503	3	0	0	12	19	17	20	11	Yes
15004	3	0	0	14	29	14	20	11	Yes

TABLE E.1: Candidates with their values, score, and times played.

Session	Candidate	Success	Time (s)	Distance
1051	15472	No	601	15.03
1053	15474	No	63	11.4
1057	15471	No	643	11.31
1059	15467	No	172	19.85
1060	15476	Yes	394	17.72
1071	15472	Yes	553	20
1072	15473	No	999	9.22
1075	15469	No	107	15.52
1076	15468	Yes	355	10.2
1078	15470	No	96	15.03
1083	15474	No	118	12.04
1085	15469	No	73	15.13
1086	15468	No	269	13.6
1093	15474	No	78	13.34
1095	15469	Yes	311	27.46
1096	15468	No	112	12.65
1098	15470	No	212	14.14
1099	15467	No	1025	20.62
1102	15469	No	60	18.25
1106	15467	No	113	7
1112	15469	No	98	19.24
1123	15468	No	102	11.05
1127	15476	No	82	14.76
1131	15475	No	130	17.09
1132	15469	No	108	22.47
1134	15471	No	656	19.03
1153	15468	No	127	10.05
1155	15470	Yes	588	12.65
1161	15475	No	56	14.32
1165	15470	No	40	13

TABLE E.2: Winning sessions from Generation 1.

Session	Candidate	Success	Time (s)	Distance
1170	15488	Yes	373	24.21
1173	15487	No	171	14.04
1174	15494	Yes	382	14.56
1175	15495	Yes	368	12.17
1178	15469	Yes	366	14.87
1184	15487	No	221	15.3
1185	15494	Yes	383	18.6
1186	15495	No	210	18.87
1188	15493	No	126	17.26
1191	15491	No	140	15.03
1192	15488	No	115	18.25
1195	15487	No	281	12.04
1198	15492	No	675	18.44
1201	15490	Yes	492	10.44
1209	15492	No	99	22.2
1214	15488	No	252	19.7
1228	15469	No	199	16.76
1229	15490	No	204	19.8
1230	15491	No	104	10.44
1231	15488	No	226	21.59
1232	15486	Yes	343	16.55
1233	15489	Yes	318	20.22
1234	15487	No	85	14.56
1235	15494	Yes	314	17.2
1236	15495	No	118	17.89
1237	15492	Yes	455	16.97
1240	15490	No	126	14.56
1242	15488	Yes	539	26.48
1243	15486	No	87	14.21

TABLE E.3: Winning sessions from Generation 2.

TABLE E.4: All sessions, with the successful ones highlighted.

Session	Candidate	Success	Time (s)	Distance
1046	15468	No	151	14.21
1047	15471	No	233	17.46
1048	15470	No	109	15.3
1049	15467	No	120	18.44
1050	15476	No	243	21.38
1051	15472	No	601	15.03
1052	15473	No	17	14.21
1053	15474	No	63	11.4
1054	15475	-	-	-
1055	15469	No	288	16.49
1056	15468	No	66	13.93
1057	15471	No	643	11.31
1058	15470	No	155	15.03
1059	15467	No	172	19.85
1060	**15476**	**Yes**	**394**	**17.72**
1061	15472	No	182	17.26
1062	15473	-	-	-
1063	15474	No	324	17.46
1064	15475	No	188	14.56
1065	15469	No	141	18
1066	15468	No	799	13.6
1067	15471	No	197	15
1068	15470	No	632	17.72
1069	15467	No	764	14.87
1070	15476	No	309	21.26
1071	**15472**	**Yes**	**553**	**20**
1072	15473	No	999	9.22
1073	15474	No	138	15.13
1074	15475	No	69	17.46
1075	15469	No	107	15.52
1076	**15468**	**Yes**	**355**	**10.2**
1077	15471	No	423	15.81
1078	15470	No	96	15.03
1079	15467	No	147	10.3
1080	15476	No	118	15.13
1081	15472	No	49	21.59
1082	15473	No	432	15.81
1083	15474	No	118	12.04
1084	15475	No	214	14.32
1085	15469	No	73	15.13
1086	15468	No	269	13.6
1087	15471	No	438	12.73
1088	15470	No	116	12
1089	15467	No	94	12.08
1090	15476	No	97	16.12

Continued on next page

Table E.4 – continued from previous page

Session	Candidate	Success	Time (s)	Distance
1091	15472	No	337	17.12
1092	15473	-	-	-
1093	15474	No	78	13.34
1094	15475	No	357	17.09
1095	**15469**	**Yes**	**311**	**27.46**
1096	15468	No	112	12.65
1097	15471	No	206	15.65
1098	15470	No	212	14.14
1099	15467	No	1025	20.62
1100	15476	No	211	18.44
1101	15472	-	-	-
1102	15469	No	60	18.25
1103	15468	-	-	-
1104	15471	-	-	-
1105	15470	-	-	-
1106	15467	No	113	7
1107	15476	No	126	16.28
1108	15472	No	188	14.32
1109	15473	No	95	16.28
1110	15474	-	-	-
1111	15475	No	229	17
1112	15469	No	98	19.24
1113	15468	No	146	15.03
1114	15471	No	408	16
1115	15470	No	230	18.97
1116	15467	No	164	11.18
1117	15476	-	-	-
1118	15472	No	257	14.87
1119	15473	No	150	11.4
1120	15474	-	-	-
1121	15475	No	77	17.09
1122	15469	No	106	14.42
1123	15468	No	102	11.05
1124	15471	No	225	10.63
1125	15470	-	-	-
1126	15467	No	301	13.45
1127	15476	No	82	14.76
1128	15472	No	539	24.7
1129	15473	-	-	-
1130	15474	No	270	12.04
1131	15475	No	130	17.09
1132	15469	No	108	22.47
1133	15468	No	160	15.56
1134	15471	No	656	19.03
1135	15470	No	97	13
1136	15467	-	-	-

Continued on next page

Table E.4 – continued from previous page

Session	Candidate	Success	Time (s)	Distance
1137	15476	No	151	20
1138	15472	No	146	17
1139	15473	No	192	17
1140	15474	No	123	12.17
1141	15475	No	159	17.12
1142	15469	No	72	18.36
1143	15468	No	236	11.05
1144	15471	No	183	17
1145	15470	-	-	-
1146	15467	-	-	-
1147	15476	No	239	14
1148	15472	-	-	-
1149	15473	-	-	-
1150	15474	-	-	-
1151	15475	-	-	-
1152	15469	No	484	24.84
1153	15468	No	127	10.05
1154	15471	No	246	13.04
1155	**15470**	**Yes**	**588**	**12.65**
1156	15467	No	163	8
1157	15476	No	55	13.42
1158	15472	No	462	15.3
1159	15473	No	277	11.18
1160	15474	No	188	16.12
1161	15475	No	56	14.32
1162	15469	No	152	14.32
1163	15468	No	229	12.04
1164	15471	No	67	14.87
1165	15470	No	40	13
1166	15467	No	104	10.63
1167	15469	-	-	-
1168	15490	No	129	18.97
1169	15491	No	308	20.25
1170	**15488**	**Yes**	**373**	**24.21**
1171	15486	No	285	25.06
1172	15489	No	255	18.87
1173	15487	No	171	14.04
1174	**15494**	**Yes**	**382**	**14.56**
1175	**15495**	**Yes**	**368**	**12.17**
1176	15492	-	-	-
1177	15493	No	358	18.44
1178	**15469**	**Yes**	**366**	**14.87**
1179	15490	-	-	-
1180	15491	No	104	8.6
1181	15488	No	293	28.18
1182	15486	-	-	-

Continued on next page

Table E.4 – continued from previous page

Session	Candidate	Success	Time (s)	Distance
1183	15489	No	307	23.02
1184	15487	No	221	15.3
1185	**15494**	**Yes**	**383**	**18.6**
1186	15495	No	210	18.87
1187	15492	No	898	16.55
1188	15493	No	126	17.26
1189	15469	No	404	16.76
1190	15490	No	208	14.04
1191	15491	No	140	15.03
1192	15488	No	115	18.25
1193	15486	No	122	28.32
1194	15489	No	299	14.14
1195	15487	No	281	12.04
1196	15494	No	191	18.97
1197	15495	No	62	17
1198	15492	No	675	18.44
1199	15493	No	357	18.87
1200	15469	No	494	18
1201	**15490**	**Yes**	**492**	**10.44**
1202	15491	No	339	18.68
1203	15488	-	-	-
1204	15486	-	-	-
1205	15489	-	-	-
1206	15487	No	217	18.38
1207	15494	No	105	13
1208	15495	No	53	12.37
1209	15492	No	99	22.2
1210	15493	No	94	18.87
1211	15469	No	170	23.02
1212	15490	No	158	10.05
1213	15491	No	201	21.38
1214	15488	No	252	19.7
1215	15486	No	151	22.8
1216	15489	No	87	20
1217	15487	-	-	-
1218	15494	-	-	-
1219	15495	No	258	13.42
1220	15492	No	288	18.03
1221	15493	No	301	17.72
1222	15469	No	182	17.46
1223	15490	No	617	14
1224	15491	No	177	8.25
1225	15488	No	364	19.1
1226	15486	-	-	-
1227	15469	-	-	-
1228	15469	No	199	16.76

Continued on next page

Table E.4 – continued from previous page

Session	Candidate	Success	Time (s)	Distance
1229	15490	No	204	19.8
1230	15491	No	104	10.44
1231	15488	No	226	21.59
1232	**15486**	**Yes**	**343**	**16.55**
1233	**15489**	**Yes**	**318**	**20.22**
1234	15487	No	85	14.56
1235	**15494**	**Yes**	**314**	**17.2**
1236	15495	No	118	17.89
1237	**15492**	**Yes**	**455**	**16.97**
1238	15493	No	170	16.28
1239	15469	No	98	17.69
1240	15490	No	126	14.56
1241	15491	No	152	15.3
1242	**15488**	**Yes**	**539**	**26.48**
1243	15486	No	87	14.21