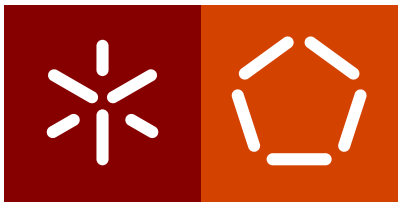
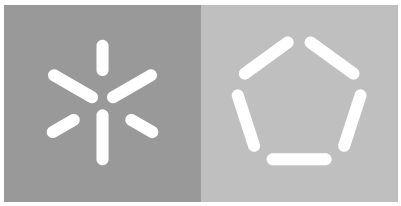


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Sérgio Lucas dos Santos Oliveira

Adaptive Game Content Generation

October 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Sérgio Lucas dos Santos Oliveira

Adaptive Game Content Generation

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
Luís Gonzaga Mendes Magalhães

October 2017

ACKNOWLEDGEMENTS

This work was made possible with the help and contribution of many people in my life. As a token of my gratitude, I would like to address them in these small words.

To Luís Gonzaga Mendes Magalhães:

Thank your for guiding me in this journey. Your insight and availability were unparalleled. Though the guidance was critical to this success, you also gave me the freedom to explore my own ideas in the subject that I find the most joy and for that, I thank you as well. Last, but most certainly not least, thank for providing me with the most exciting challenge I have ever encountered in my academic life. I hope our paths cross again in the future.

To my family:

Thank you for your hard work and sacrifices that made my education - both personal and academic - possible. Thank you for teaching me the meaning and the importance of our blood and roots, for I known that I can always count on those to sustain my life, even in the darkest hours.

To my friends:

Thank you for the countless jokes, moments and games that fuel my life. May our gaming nights last for many years to come.

Finally, as an ending note, I would also like to thank and credit the following artists whose assets I used in the games developed during this dissertation:

- Backgrounds By Zeyu Ren
- UI panels and blocks by "Kenney.nl"
- Item icons by Ravenmore http://dycha.net

ABSTRACT

From a simple entertainment activity to a learning tool, it is undeniable that video games are one of the most active and relevant areas in today's society. Since their inception, video games experienced an evolution unmatched by almost any other area and today's video games go beyond being simple software. They are authentic pieces of art that programmers, designers and artists work together to build.

As such, it presents no surprise that building a state of the art (AAA) video game is a very expensive process. If one breaks down the budget it takes to develop a AAA a video game, one finds that a sizable amount of money goes to developing content for that video game.

Procedural content generation is based on a strict set of rules and it offers an alternative to manual design of video game content, optimizing the process of development and thus reducing its cost.

Because procedural content generation relies on a set of rules, it is possible to take into consideration the specific needs of a given player and dynamically adjust the game content to better suit its play style and increase the value s/he takes out of the game.

This work focuses on the development of a preliminary version of a new Procedural Content Generation (PCG) methodology that generates customized game content that is dependent on the player's context.

The main goal of this methodology is to close the gap between players and developers, offering an easier way for the game developers to access information about each specific player, allowing them to craft generation algorithms that make use of individual player information.

This work also contemplates the development of two games that make use of said methodology and thus generate adaptive content.

RESUMO

Desde o simples entretenimento a uma ferramenta de aprendizagem, é inegável que os videojogos são uma das áreas mais activas e relevantes da sociedade atual. Desde o seu aparecimento, os videojogos têm sofrido uma evolução quase incomparavél às restantes áreas, sendo que os jogos actuais são mais do que um pedaço de software. São autenticas obras de arte que programadores, *designers* e músicos (entre outros) trabalham em conjunto para construir.

Como tal, não é surpresa que desenvolver um videojogo *topo de gama* (AAA) seja um processo muito dispendioso. Ao analisar o orçamento de desenvolvimento de um jogo deste tipo, constata-se que uma grande percentagem do orçamento está atribuída ao desenvolvimento do conteúdo para o jogo.

A geração procedimental de conteúdo é baseada num conjunto rígido de regras e oferece uma alternativa ao desenvolvimento manual de conteúdos, optimizando assim o processo de desenvolvimento e consequentemente baixar o seu custo.

Uma vez que a geração procedimental de conteúdos é baseada num conjunto de regras, é possível que estas tenham em consideração as necessidades de um jogador específico. Deste modo é possível ajustar dinamicamente o conteúdo à medida do jogador, fazendo com que seja tirado o máximo de proveito do jogo.

Este trabalho foca-se no desenvolvimento de uma versão preliminar de uma nova metodologia de geração procedimental de conteúdos para jogos, dependente do contexto do jogador.

O objectivo principal desta metodologia é o de aproximar os jogadores e os criadores de jogos, de modo a que os criadores possam ter um acesso mais fácil à informação de cada jogador específico, permitindo que os criadores desenvolvam algoritmos de geração de conteúdo utilizando a informação individual de cada jogador.

Assim, este trabalho também contempla o desenvolvimento de dois videojogos que fazem uso da metodologia desenvolvida, gerando assim conteúdo adaptativo.

CONTENTS

LIST OF FIGURES

ACRONYMS

AI Artificial Intelligence.

DDA Dynamic Difficulty Adjustment.

EDPCG Experience-Driven Procedural Content Generation.

FPS First Person Shooter.

MMORPG Massively Multiplayer Online Role-Playing Game.

NPC Non Playable Character.
NPCs Non Playable Characters.

PCG Procedural Content Generation.
PEM Player Experience Modeling.

RPG Role Playing Game.

GLOSSARY

AAA Informal classification used for video games with very high development and marketing budgets.

completionist A player that likes to complete everything about a game. This includes side quests, getting all the items, characters or obtaining all the achievements..

1

INTRODUCTION

1.1 CONTEXT

The first video game was created in 1958 by a nuclear scientist named Wally Higginbotham. The scientist was tired of seeing bored people visiting his lab and decided to fix the problem. [HF97]. It had a simple purpose: to entertain. Today, almost 60 years after their birth, video games still maintain that core value. However , nowadays video games go way beyond that. They are a mix of arts and crafts. A teaching tool whose power goes beyond any other invented, but most of all, they are part of modern culture.

As is usual in most industries, video games were not always mainstream. There was a time when the majority of people playing video games were hobbyists and enthusiasts. A time when playing video games was being outside of the mainstream. That time no longer exists and video games invaded most of our society. In fact, the growth of the video game industry was so huge, that it became the segment of the entertainment industry with the fastest growth from 1998 to 2001. [Wil02].

With the popularization of smartphones,video games became even more popular with the mainstream public and today the population that plays video games covers a wide variety of people.

Following the growth of its industry and player base, video games themselves have also grown in size and complexity. A state of the art video game (AAA) is a refined piece of software made by teams of hundreds or in some extreme cases, even thousands of people. Those teams are composed by a wide variety of people such as artists, producers, programmers, game designers, musicians, etc.

As a result of this complexity and work by these multidisciplinary teams, the cost of producing a video game is very high. The development of the content for a video game not only represents a big percentage of the development budget, but also takes a significant amount of time to make. [Ios09]. This means that there is a strong incentive to come up with ways that mitigate the cost of developing content for a video game.

PCG for games can be defined as the process of automatically creating content for a video game, releasing the designers of that task [YT11]. This way, the job of creating video

game content will be assigned to computers, that will follow a strict set of rules. Artist and designers can then focus on controlling the process, adjusting the several parameters of the generation algorithm [HMVDVI13] . On top of simplifying the process of creating reusable video game content, PCG makes possible the development of content in real time, which would not be possible by traditional means.

Since its beginning in the eighties, the main focus of PCG for games has simply been in the creation of game elements, either offline or in real time in order to ease the development process. However, in more recent years, there has been an increasing interest in the development of methodologies that create more than simple content. PCG makes it possible to create custom content for every player, depending on the preferences, play style, expertise and countless other information about the player.

Player context can be defined as everything the game knows about the player. Approaches like Experience-Driven PCG already obtain such information from the players, using subjective, objective and game-play sources [YT11]. The methodology proposed in this work uses those sources of information but it goes beyond that and makes use of data retrieved from the devices in which the player runs the game. Information such as real life location, time of day, weather, battery life, etc, all comply with the proposed definition of ”player context”.

1.2 MOTIVATION

The motivation of this work arises from two different perspectives: optimization of the video game development process and the exploration of new game-design possibilities.

As explained in the previous section, developing content for a video game requires a significant amount of money and therefore it opens the opportunity for optimization. This work aims to offer a preliminary version of a solution for that problem. The reasoning is simple, if the process of creating content for a video game can be partially, or even completely automated, the cost of developing a video game will be lower, both money and time wise.

The other perspective is of opening up game-design possibilities. Traditionally, game content is static. This means that once the content is created, all players will experience the same content in the same situations, no matter how many times they play the game. Because procedural generation takes a set of parameters and generates dynamic content, it opens the exciting possibility of changing the game content every time someone plays the game in a different context. This opens the design space for numerous game ideas that take into consideration the players themselves. Also, it is important to add that, because developers no longer have to spend as much time creating content for the game (assuming

they are using PCG), they have more time to explore this new design space that using this approach creates.

1.3 OBJECTIVES

The main goal of this work is to develop a preliminary version of a new PCG methodology that closes the gap between developers and players, allowing the generation of custom game content based on the player personality and context. To test this, there are going to be created two simple video games with proceduraly generated content that steer the generation process using the newly developed methodology.

1.4 DOCUMENT STRUCTURE

This document is composed by the following chapters:

Chapter 1, Introduction. This chapter, the current one, was dedicated to explain and present the context, motivation and objectives of this work. It introduces the main premises of the dissertation.

Chapter 2, State of The Art. This second chapter explores what has been done in the area of procedural generation for games. First, it presents a taxonomy for procedural content generation and the several approaches used by modern games to generate content. This functions as a general overview of what is PCG and how it is used today. Secondly, it explores the areas of PCG that closely relate to what is proposed in this thesis. Finally, it presents a personal view on the current state of the art and what can be learned and used from that.

Chapter 3, The Methodology. The third chapter is dedicated to the newly developed methodology. It contains a broad but yet detailed overview of the methodology, its models and its attributes as well as some practical examples of a real world applications. The end of this chapter also contains the proposed version of the methodology that is to be implemented in the context of this dissertation.

Chapter 4, Case Study 1: Simple RPG. The forth chapter is divided into four parts. First it is presented the general concept of the game. This explains what are the components of the game and how it is played. The second part defines and details the concrete game model user for this game. Following this, there is a detailed explanation on how the methodology was used to guide the generation of game content in this game. Finally, there are some conclusions on the development of the firs game and what led to the development of the second game.

Chapter 5, Case Study 2: RGBlock. The fifth chapter describes the second game that was developed for this dissertation and its structure is similar to Chapter 4. It contains

the concept of the game, the concrete game model, and a detailed explanation on how the methodology is used in the process of generating game content. In addition to that, this chapter also contains the setup, results and discussion of preliminary tests with users, followed by a conclusion on the obtained data.

Chapter 6, Conclusion and Future Work. The last chapter of this work offers a final overview of the results of the work, as well as possible future paths of research and development.

2

STATE OF THE ART

2.1 RESEARCH METHODOLOGY

Developing new work on a given field, requires an understanding of its current state of the art. Because this work is about building a new PCG methodology for games, it is not only crucial to understand what approaches are being used in today's games but also what new work is being developed in the field as well.

To accomplish that, research was made using some scientific knowledge databases, such as *Google Scholar*, *Scopus*, *Web of Knowledge* and *IEEE Xplore Digital Library*. In order to get the most relevant results for this work, the search terms included the most relevant for the field enhanced by some specific keywords such as "context", "adaptation" and "customization".

The main selection criteria of papers and books were the title, number of citations and publishing year. The reason behind this selection was not only to try and find the most important and relevant papers (using the number of citations) but also the most recent work on the field (publishing year). Although obvious, the title was also a very filter during the research.

With that in mind, the state of the art review was made in two separate steps. The first step was to understand what is PCG, how it works and what are some of its implementation methodologies. This was done by reading surveys and taxonomies published in papers. After having a solid understanding of how "regular" PCG works, the research became more specific and focused on understanding the current approaches closely related to what one pretends to achieve with this work.

2.2 PROCEDURAL CONTENT GENERATION

The previous chapter introduced some concepts such as what is PCG and how it can be beneficial for modern games. In this chapter, those concepts are going to be refined, exploring issues such as advantages, disadvantages and common strategies of employing PCG. As said in the context of this document, PCG for games can be defined as the process of automatically creating content for a game. Taking that definition, it is possible to refine

it a little further and say that PCG is the *algorithmic creation of video game content that contains little or none user input* [TKSY11]. This more precise definition of PCG leads to three important questions:

- What is the taxonomy of PCG
- What is the definition of video game content
- What can and cannot be considered PCG

2.2.1 *Taxonomy of PCG*

Since PCG covers a wide variety of content generation problems, it is useful to provide some structure to the different approaches on PCG. This taxonomy was first provided by Togelius et al. [TYSB11] and then revised by Shaker et al. [STN16].

Online versus offline

Online PCG refers to the content that is generated while the player is playing the game. Technically, this can allow to infinite replay value, although it is difficult to make the content interesting enough so it does not feel repetitive. The Binding of Isaac [EM11] does this very well and manages to create interesting levels in real time. This also allows for one of the best perks of PCG, which is the possibility to adapt the content to the player. However, to succeed at providing the players with a good experience, the algorithms that generate online PCG must comply with two major requirements: they must be fast and they must produce results of a predictable quality.

By contrast, offline content defines the content that is created when the game is in development. When the game is finally release, that content is already part of the game. For example, a game developer could use a software tool such as SpeedRock [DDRT11] to generate the rocks offline, then include that content into the game directly.

Necessary versus optional content

The main distinction between necessary and optional game content is the fact that necessary content must always be correct while optional content does not have that restriction. Because necessary content is needed to advance the game, it is imperative that the content is correct, otherwise the player might get stuck into impossible scenarios, such as an impossible puzzle or walls so high that the character cannot possible jump over them.

On the other hand, optional content does not have that restriction. For example, a game like *Diablo III* [Ent12], uses PCG to generate the loot that the player finds in the game. It

does not matter if a weapon has sub-par stats compared to others, because it can always be discarded or switched to other weapon.

Degree and dimensions of control

Degree and dimensions of control represent how much can the generation of content be controlled by the developer or even the player in order to steer the generation process and how much does that steering costs in terms of computation [vdLLB14].

Games that use random seeds, such as *Minecraft* [Moj11] can always reproduce the same level if the same seed is used (see deterministic generation bellow). It is also possible to specify a list of parameters that the algorithms must respect when generating the solution. For instance, when randomly generating a dungeon, one could specify how many rooms are to be generated, the size and type of rooms, the corridor style, if it should have dead ends or not, etc. The more restrictions one puts on the algorithms, the more its output is controlled.

Generic versus adaptive

In this context, generic content refers to all the content that is generated through procedural means without taking any player behavior into account. Reversely, when generating adaptive content, the player interaction with the game is analyzed and that factors into the generation of the new content. Although most commercial games use generic content, there are some who use adaptive content, like *Left 4 Dead* [Coro8] or *Galactic Arms Race* [Gam10a]. *Left 4 Dead* uses it to adjust the pace of the game, while *Galactic Arms Race* uses it to create new weapons based on previously used weapons and player preference [HGS09]. More detail on adaptive content is provided on 2.3.2.

Stochastic versus deterministic generation

Deterministic PCG methods take the same input values and always generate (or regenerate) the same content, while stochastic methods possess some degree of randomness, which means that it is usually not possible to always obtain the same content. *No Man's Sky* [Gam16] uses a deterministic function to generate its universe, meaning that two players in the same position would see the same planets and stars. On the other hand, *The Binding of Isaac* [EM11] uses a stochastic approach to generate different levels every time the player starts a new game.

Constructive Versus Generate-and-Test

Constructive PCG generates its content in "one go" while Generate-and-test runs its algorithm in a loop, testing the generated content after each iteration of the loop. The loop

keeps running until satisfactory content is produced. For example Yavalath is a game that was invented / discovered by a generate-and-test program developed by Cameron Browne [Bro11].

Automatic generation versus mixed authorship

As discussed in 2.2.1, typical PCG solutions offer some degree of control to the developer that s/he then uses to guide the content generation. The mixed authorship approach makes it possible for the developer or player to cooperate with the algorithm to produce the desired content.

2.2.2 *Video Game Content*

In this context, video game content can be described as most of the things that are inside a game, excluding the game engine, Non Playable Characters (NPCs) and Artificial Intelligence (AI) [STN16]. This includes maps, puzzles, levels, music, textures, weapons, storytelling, etc. The reason to exclude NPCs and AI from this definition is because the utilization of AI for character development is a well researched field, that is independent from the rest of the content generated from PCG techniques.

After applying the methodology described in the beginning of this chapter, there were two surveys that presented the most relevant definitions of what is video game content: *Procedural Content Generation for Games: A Survey* [HMVDVI13] and *Search-Based Procedural Content Generation: A Taxonomy and Survey* [TYSB11].

These definitions offer two slightly different ways of categorizing video game content.

The first one [HMVDVI13], defines five main categories for game content: *Game Bits, Game Space, Game Systems, Game Scenarios, Game Design and Derived Content*. Each one of these major categories holds sub-classes of content, which will be described ahead. For the purposes of the organization of this document, this definition will be denominated *Pyramid*.

The second [TYSB11], structures game content in a different way. It offers a binary division: Necessary and Optional content. Necessary content can then be sub-divided into five categories: *Rules and Mechanics, Puzzles, Tracks and Levels, Terrains and Maps, Narrative and Storytelling* , while Optional content can be divided in two: *Weapons, Buildings, Camera Control, Trees*.This definition is used for search-based PCG and because of that it is denominated *Search-based*.

Pyramid-like structure for game content

In this section are presented the six main classes of game content as defined by Hendrikx et al [HMVDVI13]. In this definition, in addition to the classes of content that are present strictly inside the game, it is considered a sixth class, *Derived Content*, that as the name

suggests represents content that is derived from the game state and has the goal of immersing the player further into the game world. The content presented in this subsection is a summary of the content presented in the above cited article.

GAME BITS Game bits are the most basic type of game content. They are the fundamental pieces on witch the rest of the game is built. Game bits can be divided into two categories: concrete and abstract. Concrete bits (such as trees) can be interacted in the game world, while abstract bits (like textures and sounds) need to be combined to make a concrete bit.

- Textures (Abstract) - Textures are images that add detail to the game. From models to menus, textures often define the art style of the game.
- Sound (Abstract) - Being one of the most important features within a game, sounds help to create an atmosphere for the game and also give feedback on player or game actions.
- Vegetation (Concrete) - Used to create a more realistic and immersive look.
- Buildings (Concrete) - Buildings are essential to represent urban environments in games. They usually have a direct impact with the game-play because players have to interact with them (to save the game, sell items, deposit resources, etc).
- Behavior (Abstract) - Represent the way in which the different games objects interact with each other and the player. This is one of the most used ways to create the idea of complexity and to give player tools to explore.
- Fire,Water,Stone,Clouds (Concrete) - Often used to create a more realistic and believable world. Can often be points of interaction and behavior as well (getting burned by fire, throwing stones to do damage, etc).

GAME SPACE Game space is the environment in which the game takes place and is usually filled with game bits. Game space can also be concrete and abstract. Concrete spaces relate to the human perception of space such has forests, towns or mazes. Abstract spaces are those who only exist in the context of the game, such as a chess board.

- Indoor Maps (Abstract or Concrete) - Represent the structure and position of indoor spaces. Concrete maps usually represent spaces familiar to the human mind, such as buildings and caves) in which the space and position of the rooms and its content is important. Abstract maps usually represent abstract game spaces.
- Outdoor Maps (Abstract or Concrete) - Similarly to indoor maps, outdoor maps offer a depiction of a space (in this case, an outdoor space). These can also be abstract or concrete.

- Bodies of Water(Concrete) - Rivers, lakes, seas and waterfalls, these are often used as obstacles or as part of the game space.

GAME SYSTEMS Game systems are used to represent and simulate the relationships between the several entities of the game. They can be abstract, such as the relationship between animals and vegetation or concrete, such as the simulation of cities and networks of cities. These systems are often very complex and offer an additional layer of immersion into the game.

- Ecosystems (Abstract or Concrete) - Simulate the placement, interaction and evolution of fauna and flora. This includes simulations such as food chains or survival of the fittest algorithms.
- Road Networks (Abstract or Concrete) - Connect the different points of interest in outdoor maps, such as cities of buildings. It is also common to simulate traffic in those roads to add extra realism to the game.
- Urban Environments (Abstract or Concrete) - These are made of a large cluster of buildings, people and their interactions. The simulation can be made to simulate centuries of city growth and evolve the city based on what people lived there and their activities. The more things the simulation algorithm takes into account, the more realistic is the simulation.
- Entity Behavior (Abstract) - To enhance the player experience in the virtual world, it is often useful to simulate the behavior of NPCs to match the behavior of a real person in real life. This could mean that the way the player interacts with the NPCs can change their behavior. Not only that, but the Non Playable Character (NPC) interactions with the simulated world can also be a factor to alter their behavior towards the player.

GAME SCENARIOS Game scenarios are particular set of game events, often handcrafted, that the player has to play through or watch in order to advance the game. As usual, these can also be abstract and concrete. Abstract game scenarios describe how the objects inter-relate. Concrete game scenarios are explicitly presented into the game.

- Puzzles (Abstract) - Puzzles are problems that the player must resolve in order to advance or win the game. When it comes to puzzles, the process of finding the solution is where the player gets the enjoyment. In some cases, the game is mainly about the puzzles (*Thomas Was Alone* [Bit12]) while in other cases, the puzzles are just a small part of a larger game (*Uncharted: Drake's Fortune* [Dog07]).
- Storyboards (Abstract or Concrete) - Usually presented as comics, text or cut-scenes, storyboards contain information about the game. Their utility can range from simply

provide some information to overcome a puzzle or defeat an enemy, but they can also set and tell the entire context of the game world.

- Levels (Abstract or Concrete) - Used in the majority of the games, levels are used to separate different game sequences. They can represent a random dungeon in a rogue-like game or a particular series of runs and jumps in a platform game.

GAME DESIGN Game design can be defined as the act of deciding what is inside a game [Scho8]. This not only means asking the questions of *"what can be done in the game?"* and *"what is the player trying to achieve?"*, but also deciding on what kind of theme, artwork, sounds,etc, the game should have.

- System Design (Abstract) - This represents the set of mathematical rules that govern the game.
- World Design (Concrete) - World design refers to the design of the background and context where the game takes place. This includes the setting, lore and theme of the game.

DERIVED CONTENT As the name suggests, derived content refers to the type of content that is created based on the state of the game world. This offers an increased felling of immersion because the player can see the actions s/he took in the game gaining an extra dimension.

- News and Broadcasts (Concrete) - Similar to a real life news report, these can show the player news of the game world s/he is currently playing. This means it is possible to show the player the impact that his actions had in the game world. Games like Pokemon X/Y [Fre13] implemented such system.
- Leaderboards (Abstract) - These present the results obtained in game to the world outside the game.

Search-based structure for game content

This subsection contains the binary taxonomy proposed by Togelius et al. [TYSB11]. As mentioned in 2.2.1, this article proposes a binary division of necessary and optional content. Those can be further divided into more specific categories. As with the previous subsection, the content presented here is a summary of the content presented in the above cited article.

NECESSARY CONTENT Necessary content is critical to provide the players with the game experience that the developers are trying to achieve.

- Rules and Mechanics: Game rules and their associated mechanics are the core of every game. Even those games that focus more on the storytelling aspect, such as *Heavy Rain* [Dre10] need rules and mechanics to work as a video game. This undeniably places rules and mechanics on the necessary category.
- Puzzles: Puzzles games can be considered a genre by itself, but they are often part of more complex games. However, in both of those cases, puzzles often have to be solved to advance the game and because of that they fall into the necessary category.
- Tracks and Levels: Many games today make the player control a single character that must traverse a series of obstacles in a single region to achieve its goal. This level aspect is often combined with other game aspects such as shooting or racing. Therefore, it is placed on the necessary category.
- Terrains and Maps: A substantial amount of games are built around terrains or maps. These maps often contain certain features that the player might encounter, such as obstacles or resources. Some strategy games like *Civilization V* [Gam10b] or *Starcraft II* [Ent10]) have the map set the pace and strategy of the game for the players. This category or terrains and maps overlaps with the tracks and levels category. For example, a level from a First Person Shooter (FPS) game can be considered a map.
- Narrative and Storytelling: Most games today contain some form of narrative or storytelling, even if it is just to set the flavor of the game. For example *Candy Crush Saga* [Kin12] has the same core "match-three" mechanic as *Bejeweled* [Gam01] but has a different narrative (candy vs precious stones). More serious games like *Heavy Rain* [Dre10] have its entire appeal coming from the storytelling aspect of the game.

OPTIONAL CONTENT Optional content is not critical do the game and sometimes just serves to support creativity from the developers.

- Weapons: Weapons are very common in modern games. While most of the time weapons are needed to make progress in such games, many of those games allow the player to carry more than one weapon at the time, thus making each individual weapon optional.
- Buildings: Similarly to the weapons, buildings are also very common in modern games. Games like *Cities Skylines* [Ord15] revolve around constructing several buildings to achieve a functional simulated city. However, each building by itself can be seen as optional.
- Camera Control: Many games offer some form of virtual camera through which the player sees the game world. Usually the player can manipulate the angle and position of the camera to adjust it to better suit the current situation. This is considered

optional content because the player could typically progress through the game with a fixed camera, although it would make the game substantially harder.

- Trees: Trees are often used to add immersion and realism to the game. It is similar to weapons and buildings in the sense that each individual tree is deemed optional.

2.2.3 *Not PCG*

After looking at the definitions of PCG, its content, implementation techniques and approaches, it is also important to define what is *not* PCG. There are two categories of content that might be perceived as PCG, but in fact, are not [TKSY11].

Offline player-created content

Some games come with content editors that players might use to alter the existing or create new game content. These editors might allow players to edit or create maps, levels, character, items, etc. However, this does not alter the mechanics or logic of the game. It simply alters the content in which that logic operates. *Championship Manager: Season 01/02* [Int01] was a simulation game about taking on the role of a football manager. The content of that editor was very popular because it allowed people to create game scenarios that would not be possible through regular game play. A player might alter the age of a player, or its starting club to create his ultimate dream team, but that team would go through the same logic and algorithms as the regular content that ships with the game.

Little Big Planet [Med08], takes the editor even further and makes it a core aspect of its appeal. Players can create entire levels and upload them to the game servers. Other players can then download those levels and play them, just like if the content was created by the developer.

In the context of this work, this kind of content is not considered to be procedural generated.

Online player-created content

More often that not players can build things inside the game world, even if that is not the core mechanic of the game.

In real strategy games like *Age of Empires II* [Stu99] or *Starcraft 2* [Ent10] players have to build and manage their bases in order to win the game. The main objective of the game is to defeat your opponent by military force, however, the player still has to create its own base by placing buildings and creating working units.

City building games like *Cities Skylines* [Ord15] allow the player to create entire cities and simulates its existence, as if it were a real city (traffic, economy, real-estate, services, etc). The player has no defined goal other than making the city and managing it.

Minecraft [Moj11] pushes this concept even further and allows the player to mix and match several basic elements to create a vast amount of things from simple tools to entire cities.

This kind of content is a result of player strategy and gameplay and it is not considered procedural generated content for the purposes of this work.

2.3 Related PCG work

The previous section contained an overview of what is PCG, its approaches and its building blocks. With that in mind, it is now possible to take a step further into the concepts closest to the objective of this work, which is, in simple terms, adaptive content.

When we, humans, do something long enough, we start to perceive it as second nature. When a person drives a car for the first time, he might get anxious and feel a little bit weird. We might use the wrong turning signal or even let the car shutdown on those nasty climbs. However, as the times goes by, driving starts to feel like second nature. We get to know the car, how fast can we push it and how much space do we need to park.

The exact same thing happens with games. When we first play a game, we might not know anything about it. We do not know the controls or what our abilities are. We most likely fail that long jump and we probably die when facing that scary boss at the end of the level.

Because we are human beings, it comes with no surprise that as time goes by we start to adapt. The controls of the game become printed in our muscle memory and we no longer have to think if we are going to make the jump. We just know that we will do. We adapt to the different situations that the game presents to us in order to overcome them in the best possible way. We know that we have to grind a few extra levels of experience before facing that last boss and we now that we have to run those extra pixels in order to reach the necessary speed to jump the necessary distance.

That is only possible because as we were playing, our brains collected enough data about the game. At this point, we *know* the game.

It is possible to apply the same logic, but in reverse. If it is possible to a player to get to know and adapt to a game, it should be possible for the game to get to know the player and adapt itself according to that information.

The following section presents three approaches of generating adaptive content: Dynamic Difficulty Adjustment (DDA), Experience-Driven PCG and Context-Driven PCG.

2.3.1 *Dynamic Difficulty Adjustment*

The idea behind DDA is to make the game easier or harder depending on how well the player is doing [HC04, Hun05]. This can be seen in most racing games, where the AI controlled cars will drive faster or slower depending on the position of the vehicle controlled by the player. The thinking behind this system is based on a concept called *Flow* invented by the psychologist Mihaly Csikszentmihalyi [Csi91]. The reasoning is if the player is too far ahead, he will feel like the game is not challenging, but if he is too far behind, he will lose hope of ever winning the game.

This simplest implementation of DDA does not know anything about the player and works in three steps [Mis15]:

1. The game starts with any difficulty setting;
2. The player plays the game in said difficulty;
3. The game evaluates the performance of the player and makes changes to the difficulty, if it deems it necessary.

Because DDA is the simplest form of adaptive content and it does not know anything about the player, its effectiveness is maximized when combined with other approaches to PCG that have more information about the player. This way it is possible to generate better content with greater engagement value. The next subsections will present some of those approaches that can incorporate DDA as part of their methodology.

2.3.2 *Experience Driven PCG*

Experience Driven PCG offers a way to generate adaptive content, based primarily on the way the player plays the game. The content of this section is an overview of the work of the paper [YT11] and of the book [STN16].

A important thing to keep in mind when thinking about Experience-driven PCG, is that its focus is on the generation of *subjective* content. What might be difficult for one player, might be considered easy for another. This is valid for several aspects of the game, such as fun, frustration, engagement, etc. For instance, a complex puzzle game might be considered frustrating for many players, but for those who like the process of discovering solutions for puzzles, that is the main source of enjoyment and the biggest selling point of the game.

To further set the idea behind Experience-driven, lets analyze a final example. A player is playing a RPG where he has to explore dungeons that are randomly generated. Those dungeons can be ice or magma caves, with fifty percent chance each. At the same time the player explores those dungeons, the player model evolves according to the actions of the

player. If the player spends more time in the ice dungeons, clears more monsters and gets a higher degree of completion in the ice dungeons, the player model would evolve to reflect those preferences. Then, the generation algorithm would reward the generation of ice caves when compared of magma caves and present ice caves to the player with a higher chance, because that is the kind of content that most engages that particular player. If other player plays the exact same game, with the exact same algorithm but prefers magma caves, then the generated content would predict that to be the most engaging content for him.

The next section is dedicated to explain these concepts of Experience-Driven Procedural Content Generation (EDPCG) with more detail.

Player Experience Modeling

Player experience models can be built using three different approaches of data harvest from the player. They can be directly expressed by the players (subjective), they can be observed, such as body language and facial expressions, from external sources (objective) and they can be obtained through the analysis of the gameplay (gameplay-based).

SUBJECTIVE The simplest way to get data on the player is to simply ask. This can be done with free-response from the player or with forced questionnaires. Free-response typically contains better information about the player, but it is harder to analyze. Usually this is done by narrowing down the players response (verbal or text) to certain keywords and then processing those keywords. By contrast, forced responses lock the player to the questions present on the questionnaire. These questionnaires are usually multiple choice items or binary yes or no questions, albeit it is not always the case.

Even though subjective player experience modeling is able to obtain accurate results, it has to filter all the useless information from the player response. Also, the timing in which these questions are offered to the player is also important. If we present this questionnaires during the gameplay, they might ruin the game experience. However, if we present them at the end of the game session, their responses might not be so rich because of the player's memory limitations.

OBJECTIVE When playing a game, different types of content might elicit different types of player experience patterns. In turn, those experience might trigger changes in the physical body of the player. The player might experience a change in body posture, speech, facial expressions, heartbeat, respiration or even in the skin. Those changes can be monitored, the results interpreted and then routed back into the algorithm so it can generate the appropriate content. For example, if the player starts to slouch, support his/her head with his/her hand and starts to look into other directions, that might indicate the s/he is finding the

content boring. On the other hand, if her/his heartbeat raises and s/he is smiling a lot, that might indicate that the content being generated is of value to that player.

GAMEPLAY-BASED The main premise of gameplay-based Player Experience Modeling (PEM) is that the player actions in the game are linked to the experience s/he is having while playing it. To some degree, it is possible to infer the player's emotional state and preferences through the actions s/he inputs into the game.

For example, in a game like *The Elder Scrolls V: Skyrim* [Stu11], the player's character can yield several weapons like, maces, swords, bows, staves, etc. One possible way would be to record the time that each type of weapon is active and then modify the fitness value of each of those types of weapons according to that, so that the game can generate loot that the player might find more enjoyable.

Gameplay-based PEM is the least intrusive and more computationally efficient way of modeling the player experience, because all the player has to do is simply play the game. However, it usually leads to less accurate results.

A good way to tackle this lack of accuracy is to combine the gameplay-based method with the objective method. This way, it is possible to directly correlate the observed emotions on the player such as player facial expression and body language, with events that happened in the game. For instance, the player could be very focused and engaged on a certain game content and that would be reflected by his body language and facial expressions. It would be possible then to analyze what actions were happening in the game at that point in time and use that information to generate similar content.

Modeling Approaches

There are two approaches to build the models of player experience. *Model -based* and *model-free*.

In the model-based approach, the components of the model and its parameters are constructed in a ad-hoc manner and tested for validity using a trial and error approach. It is simpler and it does not require machine learning or sophisticated tools to implement.

On the other hand, model-free approaches rely on the annotation scheme and consequently in the type of model output available. If the data recorded is scalar, such as ratings, then it is possible to use algorithms such as artificial neural networks, Bayesian networks, decision trees, support vector machines and standard linear regression to build the affective models. If,however, the data is recorded in a ranked format, it is not possible to apply the previous techniques and the problem becomes one of preference learning. In this case, Neuro-evolutionary preference learning and rank-based support vector machines are used.

Content Evaluation Functions

The main purpose of building the player models in EDPCG is to better evaluate the quality of the generated game content. However, just having a model of the player experience is not enough to correctly evaluate the generated content. The purpose of the evaluation function it to evaluate a generated item of game content and assign it a scalar value that reflects its suitability for the game or its capacity to provoke the desired effect on the player. This means that there is no exact formula for the evaluation function. It is the job of the designer to implement the function in such a way that it achieves the intended results. For example, if the designer wants to create a fun and social gaming experience, the evaluation function must evaluate the generated content in such a way that it rewards content that contributes to such emotions on the players. The effectiveness of the content can then be assessed using the data obtained from the PEM. There are three classes of evaluation functions: *direct,simulation-based and interactive*.

DIRECT Direct evaluation functions extract information about the features from the generated content and maps it directly into a content quality value. For example, those features can include the type of a weapon or its fire rate, how many resources there are in a certain area, how many corridors and rooms are in a dungeon, etc. The PEM can influence the results obtained from those mappings.

There are two types of direct evaluation functions: *theory-driven* and *data-driven*. In theory-driven functions, the designer is guided by intuition or some previous data to create the mapping between the experience model and the quality of the content. By contrast, data-driven functions work by analyzing the effects that several types of generated content have on the player (this can be done by the means discussed in 2.3.2) and then automatic processes tune the mapping of the content to the player experience and the evaluation functions.

SIMULATION-BASED A simulation-based evaluation function is a function that is based on an artificial agent playing through a section of the game that contains the generated content that is being evaluated. Features that map to PEM are extracted and analyzed. For instance, the function might test a maze to see things such has how many exits does the maze have or how much did the artificial agent took to complete it. The answers to those questions can than be compared to the data on the PEM to assess the quality of the generated content.

Just like the direct functions, there are two types for simulation-based evaluation functions: *static* and *dynamic*. In the static evaluation functions, the artificial agent playing the game remains the same, while in the dynamic evaluation functions the artificial agent changes, and that is reflected on the quality value.

It is important to note that simulation-based evaluation functions are usually computationally more expensive than direct evaluation functions. That expense becomes even higher when talking about dynamic simulation-based functions. This is usually makes simulation-based functions prohibitive for online content generation.

On top of that, it is also important to note that these simulation-based functions work under the assumption that the artificial agent can play the game in a similar fashion as a human would, and that can be very hard to achieve in some situations.

INTERACTIVE Interactive evaluation functions assess the quality of the generated content based on the interaction with the player, meaning that the fitness is evaluated during the game play. Data can be collected *explicitly* by using, for example, questionnaires or *implicitly*, by checking how often the player interacts with the generated content, how much time he spends with it, his physiological responses, face expressions, etc. This collection of data is used to tune the PEM to the specific player which affects the evaluation function.

Interactive evaluation functions can also be divided into two types: explicit, if the function is coupled with a subjective PEM component or implicit, if it is not.

As explained in 2.3.2, although reliable, explicit data collection can interrupt game play. On the other hand, implicit data collection is typically not as accurate and reliable as explicit data collection.

Optimization of Game Content

Once the PEM and the evaluation function are created, it is common to use evolutionary algorithms to select the best content. When using evolutionary algorithms, the population (i.e., the content to be selected) is evaluated sorted by its fitness. Then, the worst candidates are removed and replaced by better candidates. Those new candidates are copies of the already existent good candidates, with some random changes (i.e., mutations) added. Although not mandatory, it is possible to do a crossover (i.e., mix the features (genes) of two "parents") operation [PLMK08].

Representing The Game Content

A central question in EDPCG (and PCG in general) is how to represent the generated game content. Because most of the content generated by EDPCG can be seen as artificial evolution process, there are two key terms that must be understood to comprehend this problem: *genotypes* and *phenotypes*. Genotypes are the data structures that are handled by the evolutionary algorithm while phenotypes are the data structure or process that is evaluated by the evaluation function) [TYSB11]. In a game content scenario, the genotype could be the instructions to create a level while the phenotype would be the level itself.

There is an important distinction to be made when dealing with representation and that is the distinction between *direct* and *indirect* encoding. In direct encoding, the size of the genotype is linearly proportional to the size of the phenotype and each part of the genotype maps to a specific part of the phenotype. In indirect encoding, the genotype maps non linearly to the phenotype and they do not need to be proportional and thus requires more complex computation.

It is also important that the representation has the right dimensionality in order to avoid very long searches. In addition to that, it is important that the representation possesses a high locality, which means that a small change to the genotype should result in a small change to the phenotype. Finally, it is also important that the chosen representation is capable of representing all the interesting solutions. This can be an issue for indirect encoding, because there could be areas of the phenotype with no mapping from the genotype.

The paper mentioned in the beginning of this section [YT11] provides a useful example to better understand this concept. This example shows the different ways to represent a level on a 2D platformer game:

1. directly as a grid where mutation works on the contents of each cell;
2. more indirectly as a list of the positions, shapes of walls and pieces, list of enemies and items;
3. even more indirectly as a repository of different reusable patterns of walls and free space, and a list of how they are distributed across the grid;
4. very indirectly as a list of desirable properties like the number of gaps, distribution of gaps, number of enemies, etc;
5. most indirectly as a random number seed.

All of these representations offer different search spaces. EDPCG requires a good amount of locality and the more direct the approach is, the bigger the locality is. However, a more direct approach also means a bigger search space, which in turn means that a higher amount of computation is required to run through the entire search-space. Because of the very low locality, the representation number 5 is not usually suitable for EDPCG. On the other hand, solution number one seems very suitable, but the high amount of data might present a challenge for the generation of content of considerable size. The representations 2 to 4, might be suitable for EDPCG, depending on how the genotype-to-phenotype mapping is designed.

Generating Game Content

After the player experience is modeled, the content represented and the evaluation functions designed, the content generator must search the search-space to find the content that

is best suited for the player in question. In a perfect case scenario, the content generator should be able to identify if, how much, and how often content should be generated for a particular player. Because some players like adaptation more than others, it is important for the EDPCG algorithm to be able to identify when such adaptation should be done. This means that, to get the best EDPCG algorithm possible, it should be able to assess the effects of the generated content on the player before it presents said content. This is possible by trying to imitate and predict player behavior, but that is not an ordinary task.

2.3.3 *Context-Driven Procedural Content Generation*

Context-driven PCG is not a well researched area yet, and as such there is not so much information about it. Context-driven PCG can be defined as the generation of content that fits the player explicit or implicit requirements [LHJB13]. These constraints are not gameplay based but rather related to the player itself, like the weather, time of day, location, battery life, time available to play, etc. All this information can be used to steer the generation of content for the game, regardless of it being explicitly or implicitly obtained.

This can have more implications than simply put a constraint in the game (such as limiting the game time). Taking the limited game time as an example, it is possible to make the content difficulty scale with that restriction.

In the example presented by the above mentioned article [LHJB13], the authors created a mobile game with a strong context-based PCG component. The premise for the inclusion of context-based content was simple: convert the player specified duration into the generation of well suited levels. To achieve that, the generator takes basic level fragments from a library and combines them in real time. The obstacles are also dynamically selected from the library and placed in valid locations of the level fragments. This ensures a wide variety of levels. Also, it is important to mention that this approach allows for a decent amount of control over the content that is being generated, because it is possible to select which fragments, obstacles and obstacle locations to combine.

In addiction to the combination between level fragments and obstacles, the implementation is fairly straightforward. There is a non stoppable timer (not even pause or death menus stop it) to impose the limited time restriction set by the player. This means that the context-based restriction is top at the rank of priorities in this implementation. Moreover, the player experience also scales with the time restriction. As the player approaches the imposed time restriction, the game automatically becomes more difficult in order to give a sense of progression to the player.

It is important to refer that this mobile game in question did not exclusively use a context-driven approach. It was a mix of context-driven, experience-driven and DDA approaches.

Those were not mentioned in this explanation because they were already covered in the two previous subsections.

Also important to refer is the fact that this PCG approach and the usage of DDA can compromise the integrity of leader boards or achievements, but it was clearly stated by the authors that this was not the intent of the work, so they were not implemented in the case study.

2.4 Lessons learned from reviewing the state of the art

After reviewing the current state of the art in the field of PCG for games, it is evident that the field has come a long way since its beginning, although there are still many areas to explore and problems to solve. PCG has moved from simple and generic algorithms such as dynamic difficulty adjustment to improve a racing game, to the use of evolutionary algorithms in search-based and experience-driven PCG, which aim to generate the best suited content for a specific context, specially with EDPCG and context-based aproaches. By taking a look at recently released games (by both established companies and independent developers), it is clear that more and more games have been making use of procedural generated content with great success, like for example *Minecraft* [Moj11], *Spelunky* [Moso8] or *Spore* [Maxo8].

It is clear that there are some important properties that implementations of PCG must have in order to be useful. Those desired properties can vary depending on what the end goal for the game is and there are some clear trade-offs between some of those properties [STN16]:

- Speed - If the content is going to be generated in real time (or close), the PCG implementation must be fast, otherwise it will affect the player experience. On the other hand, if the PCG is being used during the development of the game, speed is not so critical.
- Correctness - It is very important for the generated content to be correct, otherwise it might lead to dead ends or frustrating content. It is imperative that a level of a platformer is designed correctly (otherwise impassable obstacles might occur), but it is not so important that every tree in a vast forest is perfectly generated.
- Control - The importance of control can vary from different scenarios. Sometimes the developer might want to tweak some aspects of the generated content when developing the game. Other times the control of the algorithm must be made in real time, as is the case with EDPCG.

As with most things, PCG also has advantages and disadvantages. There are several advantages in using PCG:

- Reduced costs - As explained before, the use of PCG can greatly reduce the cost (time and money) of making a game [vdLLB14].
- Data compression - Because it is possible to generate intricate content in real time by using seeds and/or some guidelines, it is possible to reduce the size of the game or the information transferred during gameplay [TYSB11].
- Opens possibilities - The usage of PCG allows for new game design possibilities, but also for less skilled individuals, both players and developers, to obtain high quality content with very little input [STN16].

Disadvantages

- Complexity - A good PCG implementation is hard to achieve and it requires specialized knowledge and manpower.
- Lack of creativity - PCG follows a strict set of rules, and even though there might be several degrees of randomness, it is still hard to reproduce human creativity and that might lead to stale content after a while.
- Content validation - Because PCG allows for the generation of a great amount of content with a few rules, it is hard to validate the generated content. It is also hard to assess the effects that a new rule might have on the existing rules [YT11]
- Computation power - The generation of content in real time usually requires a good amount of computation power. Consequently, that might lead to substantial loading times. [TYSB11, YT11]

From a technical point of view, there are some things worthy of note.

First, there were several approaches presented in section 2.2.1 on how to generate procedural content. From that it becomes clear that there is no ”one size fits all” solution when it comes to PCG and that is specially true when it comes to PCG for games. The designers of the game must evaluate their needs and choose the approach that best suits their needs and the needs of their game. For example, it is not possible to say ”stochastic generation is better than deterministic generation”. The correct thing to say would be ”there are some games where a stochastic approach is more appropriate than deterministic one”.

In section 2.2.2 there were described two proposed ways of defining what is video game game content. One divided the content in a pyramid-like structure while the other offers a binary division of necessary and optional content first, with each one of those categories holding more specific types of content. Here as well there is no *objectively better* definition, because they serve different purposes. The pyramid-like structure offers a generalized view on what is the video game content. That structure is very useful for introducing new people to the field and it is valid for pretty much every approach on PCG. As indicated by

the name, the search-based definition of game content is better suited for search-based approaches on PCG. Realistically, those two structures have many overlapping concepts with each other, so it is possible to extract good information from both. In a more personal perspective, the pyramid-like definition offers a more structured and pragmatic way of looking at video game content, but is is possible to see the arguments for the search-based structure. However, the evident conclusion from both of those perspectives is that it is necessary for the developers (both the one designing the PCG algorithms and those designing the game) to have a clear vision of what is needed to generate for a game. This serves as a reminder that it is important to choose what content is going to be generated when creating the game to test the new methodology resulting of this work.

The study of EDPCG taught some valuable lessons on the current state of PCG and how to proceed with this work, mainly because this work and EDPCG are closely related. On one hand, EDPCG uses the player preferences and reactions to the game to generate and adapt that content, while this work will focus not only on that but also on the context of the player *outside* the game.

The first lesson is about the importance of the player model when it comes to generating custom content. It is very difficult to generate good content for a specific player if there is no knowledge of said player. This means that when moving forward with this work there must be enough time dedicated on how to model the player. The second lesson is on the importance of the evaluation functions when it comes to generating good content for the player. Evaluation functions can be implemented in different ways, and although it is often complex to implement them, it is imperative to have a good evaluation function when dealing with experience-driven or context-driven generated content, because they act as the judge that decides which content is presented to the player.

Finally, the study of context-based procedural content generation made the path forward a bit clearer, because it showcased some of the potential and practical applications in which the context of the player can be used in games.

In conclusion, the complexity and scale of current video games, as well as a very demanding player base, call for more and better content, and that demand is hard to fulfill with raw manpower alone. This not only translates into a monetary interest in the field, but also to a creative one, due to the numerous game design possibilities that the usage of PCG opens. Complex search-based methods already allow for the generation of high quality content, but there is still much room for improvement. More detailed player models and better evaluation functions are obvious paths for the field. Moreover, there are some types of content (namely the content at the top of the pyramid presented in 2.2.2, which are still very hard to generate. The generation of that type of content still needs more research and improvement to yield satisfactory results.

3

THE METHODOLOGY

3.1 THE BIG PICTURE

The main purpose of this work is to create a preliminary version of a new methodology that game developers can use to increase the players immersion in game, by automatically adapting their content to each player.

To achieve this degree of content adaptation there are two conditions that must be met:

- The game content in question must be procedurally generated
- The developers must know the players so they can adjust the content generation

Right now there is not an easy way (that does not consume many development resources such as time and money) for developers to get player information and to make use of it. As a result of that, this design possibility of content adaptation is often ignored.

Essentially, what this methodology aims to do, is to provide a bridge between the the game, its developers and the players, so that it becomes easier for the developers to know the players and consequently adapt the game content for each one of them.

In order to be able to provide this bridge to the developers, the methodology must know the player, the game and their surroundings. This means that each one of those components must be modeled so that it is possible to know information about the entire context of the player and the game. To tackle with this it is proposed the methodology presented in Figure 1. The role of each model is to contain information about different aspects of the context of the player and the game, and this makes models the center piece of the the methodology, which are described in detail in the next section.

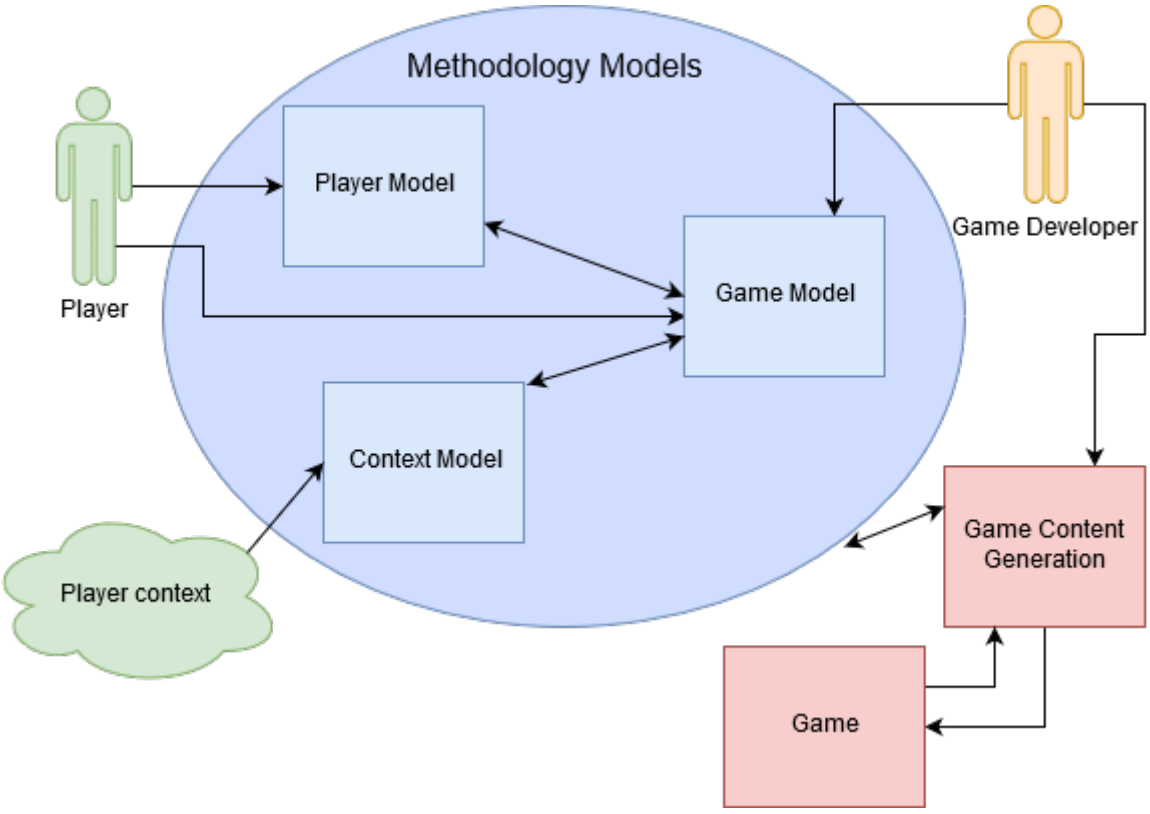


Figure 1.: Diagram overview of the methodology

3.2 MODELS

The developers create a game, that game is played by different players and each player has a different context. This entire chain can be summed up into four major actors:

- Developers
- Game
- Player
- Player context

Developers do not need a model because they are the ones creating the game and therefore, it is not relevant to have their information contemplated in the methodology. The game model exists because each game has its own specifications, and therefore the methodology needs to have information about it. Moreover, because players are the target for games and each player is unique, the player information is critical for the methodology and the player model answers that requirement. Finally, because the playing activity may occur in

different contexts, that information must also be known to the methodology and so there is a context model. To represent this variety of information, there are proposed three models:

- Player Model
- Context Model
- Game Model

Each one of these models is responsible for capturing different types of information that game designers can then use to generate adaptive game content. The rest of this section presents detailed information for each one of those models, as well as some examples on how the information held by those models can directly influence the generation of content in a game.

3.2.1 *Player Model*

The main premise of adaptive PCG is to adapt the content to the specific player, so having a good player model is of the utmost importance. With a good player model, the methodology can provide game designers with relevant user information that they can then use to craft well suited content for the different players that play their game.

Player Characteristics

The player model represents individual information about the player and it is not transferable between players. The first logical step is to gather the most generic information possible such as age and gender. Although those two attributes only provide a very basic first glimpse of the player, it is already possible to make some game design decisions based on that. Considering, for example, a puzzle game designed for players with ages between five and ten years old, it makes sense from a design perspective, to offer completely different puzzle difficulties for players in both ends of that age range. Likewise, it would be possible to change the color scheme of the game or the player character based on the gender of the player.

There is a very soft limit to the detail of this generic type of information. It would be possible to consider other things such as nationality, eye-color, height, weight, etc. The usefulness of that information would depend on the game designer.

Even though this generic information is already useful, it only offers knowledge about a *person*. Because this is a gaming context, it is critical to go beyond the knowledge of the person. It is necessary to know the *player*.

Player Personality

When thinking about video game players, there are numerous features and styles that each player has that are important to consider. A good way to think about it is seeing this as a "player personality". This personality defines the way each player interacts with games, and it is different for every player. This means that a single game causes different experiences to different players, depending on their player personality.

To clarify this concept, it is presented next an analysis of two popular and very different games: *Counter-Strike: Global Offensive* [Cor12] and *World of Warcraft* [Ento4].

COUNTER-STRIKE: GLOBAL OFFENSIVE is a first-person shooting game, where people fight for two different sides (Terrorists and Counter-Terrorists). Each team has a different objective in each map, but the main premise of the game is to eliminate the opposing team using several weapons. When analyzing a game like Counter-Strike, it is possible to identify several types of players. There are the very aggressive players that like to go all in on the enemy team, not caring for the consequences or the team objectives. Their main goal is to get as much kills as possible. This can be represented with an *aggressiveness* attribute. On the other hand, there are the more tactical, cautious players, that like to play around the map and the enemy team, not caring so much about individual kills but the overall team performance. They put more work into helping the team to get objectives than to eliminate the enemy team. This characteristic could be represented with a *teamwork* attribute. To note that these characteristics are not mutual exclusive. It is possible for a player to be aggressive but still care for the team objectives or for a player to be very passive and cautious but still have little desire to work with his/her team.

WORLD OF WARCRAFT is a Massively Multiplayer Online Role-Playing Game (MMORPG) in which each player controls a unique character and plays in a gigantic virtual world. In this game players can wander around its world, doing quests, fighting monsters, getting loot and a lot of other things, while simultaneously interacting with other players. Because this is such a vast and complex game, it holds many game play possibilities and thus, many player personalities. There are the players that focus on the social aspect of the game, where most of the enjoyment comes from interacting with the people from their guilds. At the same time, there are the hardcore players, that put serious time investment into the game and that aim to be the elite. These players want to work hard and optimize their characters to the tiniest detail to be able to conquer hard content such as raids. Their enjoyment comes by get rewarded by the game (with amazing items or achievements, for example) and by being recognized by the community as the strongest players. On the other hand, there are the more casual players that just want some instant gratification from the game. They are the opposite of the hardcore players and get their enjoyment in a completely different way.

These games are just two examples, but the concept that completely different types of players exist within the same game holds to virtually any game. This is a very important issue that the game developers must be aware of when developing their games, if they want to appease a wider audience and it is absolutely crucial to experience-driven and context-based PCG and thus the methodology that this work aims to create.

There are a wide variety of attributes that define the player personality and their relevance to the generation of game content depends on the type of game and the vision of the game designer.

The following itemization contains a series of questions that aim to capture the player personality and that the player model needs to be able to represent:

- How aggressive/passive is the player?
- Does the player want instant gratification or long term rewards?
- Is the player casual or hardcore?
- Is the player a risk taker or does he/she like to play it safe?
- How social is the player?
- Is the player a completionist ?
- How long are the typical gaming sessions of the player?
- Does the player enjoy teamwork?

All of this information alongside the generic player information constitutes the player model. Some players might have different answers to these questions depending on the type of the game being played, and that is addressed ahead in the subsection 3.2.4. The next step is to understand how can this information be used in the context of context-driven PCG.

It is important to note that the only limit to the usefulness of this information is the creativity of the game designers. The methodology merely provides the data for the developers to work with. However, it is clear even at first glance that this methodology opens up exciting new design ideas.

Considering, for example, a game with procedural generated caves. Despite these caves typically having multiple rooms, they usually have a main room or main objective that the player must clear. This means that the smaller rooms are usually useless and filled with very basic and uninteresting content. The way players feel about this type of content is related to their player personality. Players that only care about the main goal can fully enjoy this type of content, because their enjoyment comes from completing the main goal. However, if the player is a completionist, not only does s/he clears the main objective, but is also very likely

that s/he will clear every single room in the cave. If the smaller rooms are empty or do not contain anything interesting, a completionist player might feel disappointed, because all the hard work put into exploring the entire cave was not properly rewarded.

If the game developers had access to the that characteristic of their players, then they could incorporate that information directly into the generation of the caves. A possible way to do that would be to generate more items or monsters in the side rooms if the player was a completionist, or focus all the content in the main path otherwise.

Another example would be a game where the items are procedural generated. A relevant question during the generation process is how powerful should the item be? Should the algorithm generate powerful items right away? What is the power curve of the game? A possible solution would be generate items based on the player level or current in-game location, but that would not take into consideration the player currently playing the game. This new methodology opens the possibility of generating the item based on the player itself. Does the player like instant gratification? If so, generate meaningful loot right away and keep increasing the power level of the item slightly so the player can keep feeling rewarded most of the time. On the other hand, if the player likes long term rewards, make the player work hard for the loot but when s/he completes a big task, generate an immensely powerful item so the player can feel his/hers hard work paid off.

It is very important to keep in mind that these examples are just some possible usages that the player model would have in the development of a new game and that the complexity of the player model is also very flexible with room to go into more or less detail.

3.2.2 *Context Model*

The context model represents information about the context in which the game is being played. Unlike the player model that holds different information depending on the player, the context model suffers little variation for a given gaming session, regardless of the player.

This context does not relate to the game itself, but rather to its surroundings. This means that the context model could hold information such as the environment or the device on which the game is being played. Some examples off this kind of information would be the day of the week, the current season or information about the weather at a specific hour. The location of the player, battery level or type of device running the game could also be valuable information.

This type of information could prove very useful in providing an additional layer of immersion to the game. It is easy to envision a game that uses real information about weather to influence the generation of game content like backgrounds or particle effects to match the real life weather.

Considering a augmented reality game like *Pokémon GO* [Nia16], where the players have to walk in the real world to catch *Pokémons*. The first version of this game spawned the *Pokémons* in a pseudo-random location that is influenced by the density of the population in the given area (amongst other factors). Even though the location where the *Pokémons* spawn is generated by PCG techniques, the individual location of each player is not taken into account to the generation of *Pokémons*. This results in situations where some players might walk for long periods of time without encountering any *Pokémon* (specially in less populated areas).

If the individual context (in this case, location) of each player is taken into account, it would be possible to spawn *Pokémons* based on the location of each player, eliminating the problem of people that do not encounter *Pokémons* for long periods of time.

Note that even though this game uses the location of each player, it is used as an input method, not as a way of influencing the generation of game content.

Currently, the only context taken into account that affects the visual of the game is the day and night cycle. The game changes its appearance to a light sky during the day and dark sky during the night.

Using this methodology it would be possible to take this concept even further by using the weather context of each player. This way, if a player was playing in a city that was raining, the game screen could have some rain effects, while simultaneously, a player playing in a different city with a clear weather would see that reflected in game as well.

The main idea behind using contextual information, is to add something to the game content that makes sense to the player and thus creating a more immersive experience. In the rain example mentioned above, it makes perfect sense to be raining in game when it is raining in real life, because the setting of the game is the real world itself.

Even though the *Pokémong GO* example mentioned above refers to an augmented reality game, the usage of context information is not limited to augmented reality games. It is possible to use similar contextual information in purely abstract games or in game with completely fictional settings. Imagining a mobile side-scrolling game that has its levels procedurally generated, it would be possible to partially base the length of the levels according to the battery level of the device. This way, if the device has a low battery power, the user would get shorter levels, increasing the chances of the player finishing the level before running out of battery.

Again, similarly to the player model, there is a very soft limit to the amount of contextual information that could be gathered. It could go from battery level, to cpu usage to the weather, the location, etc. More importantly, its up to the game developers to decide what information is useful or not, according to their creativity and vision. This means that information that is completely useless for one developer might be incredibly valuable to another.

3.2.3 *Game Model*

The game model is responsible to hold information about the preferences and actions of a specific player in a particular game. Contrary to the player and context models, most of the game model is not generic and cannot be applied to any game, even though some attributes like the average game session duration could apply to any game.

Because developing game is a partially creative process, it is understandable that no two games are the same. There are many different games from many different genres where each game possesses unique elements and mechanics, thus it is impossible to have a one-size-fits-all game model. For example, it makes sense for the game model to have information about a player's favorite fruit in a game about fruits, but that information would be useless in a game about rocks. Likewise, it would not make sense to have a game model for a *sudoku* game where one of the attributes was how many times the player died in-game, because a typical *sudoku* game does not feature player death.

Considering that each game has its own game model, there is a substantial amount of information that could be captured during gameplay. The following questions offer a starting point in the understanding of what might the game model contain.

- How long is the typical gaming session?
- How long does the player spend in each game mode?
- How often does the player die?
- What is the player's favorite class?
- What is the player's favorite weapon?
- How long does each battle take?
- If the player controls multiple characters, does he have a favorite?

The answer to these questions could prove very useful, for example, in a RPG game with procedurally generated loot. If the player has a favorite character it would make sense to generate loot that could be used by that character. Likewise, if the player gives preference to a certain type of weapon, the game could reward the player with powerful versions of that weapon type. Finally, if the player is dying very often or if the battles are taking an exaggerated amount of time, it could make sense to lower the difficulty of the enemies.

It is important to keep in mind that these questions are just a starting point. The full extent of the game model and the applications of its information could only be determined by the game developers depending on the game they envision.

3.2.4 *Relationships Between The Models*

Although each one of these models contains different types of information, it is possible to cross information between them in order to get a more precise profile of the player.

The most primitive way to fill the player model with information, is to ask the player directly. However, as discussed in 2.3.2, this way of collecting information has some issues that might undermine the quality of the information collected. It can be tiresome to answer profile questions when the player is just eager to play the game and have fun. In addition to that, those profile questions are not always intuitive and more often than not the player is not capable of providing an accurate action because the player itself might not know the answer, or the answer might vary from game to game.

Assuming a scenario where a player likes to take extreme risks in FPS games but is very cautious in RPG games. If this player was presented with a generic question about how much of a risk taker s/he is, and the answer was in a scale from 1 (very cautious player) to 10 (takes a lot of risks), what would the answer be?

In a generic sense, the answer would be 5, because the player sometimes is aggressive and sometimes it is not, however, a 5 is not an ideal answer in FPS or RPG games and it would lead to a inappropriate profile for both of those game genres.

One of the most exciting possibilities that crossing information between models opens, is the possibility of a self-correcting player model that uses information from the other two models to adjust the initial profile inputs provided by the player.

The most relevant exchange of information is between the player model and the game model, because that means the player model would be adjusted by reading what is happening in-game. By analyzing the data produced during gameplay, it would be possible to determine with more accuracy what is the type of each player. So, considering the above example where the player answered a 5 in how much of a risk taker s/he is, the attribute of risk taking in the player model could be automatically corrected to a value greater than 5 in case of a FPS game or to a value lesser than 5 in case of a RPG game.

The most immediate way to do this correlation would be to implement an *ad hoc* solution that directly correlates attributes of each model, but the ideal solution would be using something like artificial intelligence. However, that is out of the scope of this work and it would only be possible with further development and investigation.

3.3 Development Using This Methodology

To clarify the usage of the methodology by the game developers, there is going to be presented an hypothetical example of the development of a game using the methodology as a whole.

Considering a development team that is starting the process of developing the game. The team has the main concept of the game and they want to use this methodology to provide their future players with better game content.

The development team would look at the player model and context model to see what information they could use in the content generation process. Then, they would have to decide what kind of information their game model should have and implement that module and any possible relations to the other two models.

Once that process is done, all the developers would have to do is use the information contained in the models and use it as an input in the generation algorithms to adjust the generation of content to each player.

3.4 Methodology Considerations

Previous sections presented the three models that serve as the foundation of this methodology and it showcased some examples on how it can potentially be used by game developers to generate better game content for their players. The scope of the models and their potential reach was presented in an open manner in order to potentiate a better understanding of the full extent of this methodology, however it is not feasible to explore all the possible attributes of the models in the context of this dissertation.

For this dissertation, in order to implement a preliminary version of the methodology, a subset of attributes was chosen for each of the models. The selection of the model attributes was made to be very generic, in order for the methodology to suit different kinds of games. The next section presents the exact attributes used in the methodology for the player and context models while the next chapters of the dissertation contain the description of the developed games, their respective implementation of the game model and how they make use of the methodology.

3.5 Proposed Model Attributes

As explained before, the reach and possibilities of context-based PCG are very vast and as such it is not feasible to implement them all in the context of this work. Because of that, it was necessary to select some attributes for each model that would be useful to procedurally generate content for the proposed game.

3.5.1 *Player Model*

There were a total of five attribute chosen for the player model and those attributes were divided into two categories, player characteristics and gaming personality, as shown in figure 2.

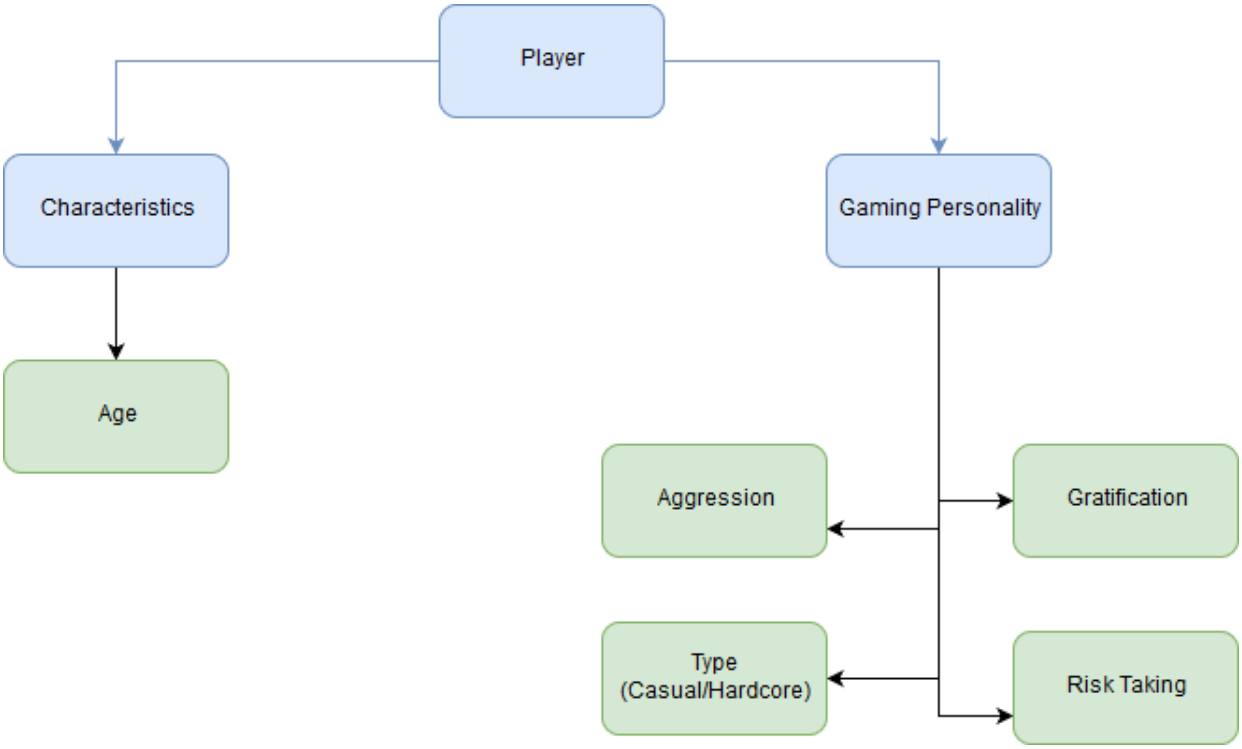


Figure 2.: Player Model Attributes

Age

This attribute represents the age of the player playing the game. In terms of content generation, this could be used, for example, to adjust the overall difficulty of the game or its theme.

Aggresion (1-100)

The Aggression rate ranges between 1 (very passive) and 100 (very aggressive) and defines how passive or aggressive the player is. This could be useful to determine what kind of strategy the player likes to use in games. Does the player like to go all in with his/her characters or does it like to take a defensive approach? This information can then be used to adjust the generation of game content to promote the favorite play style of the player.

Gratification (1-100)

The gratification rate of the player ranges between 1 (very slow gratification) to 100 (instant gratification) and is an indicator on how fast the player likes to be rewarded. A player that likes instant gratification would be rewarded more often, but in smaller power increments. On the other hand, a player that does not enjoy instant gratification would get less rewards overall, but those rewards would be significantly better and would provide a bigger increment in power.

Casual/Hardcore (1-100)

This attribute ranges from 1 to 100 and defines the type of player from very casual (1) to very hardcore (100) . More casual players typically like the game to be forgiving and more accessible while hardcore players like to face harder challenges. This could be used to adjust the difficulty of the game content or to adjusting the amount of grinding required.

Risk Taking (1-100)

This attribute ranges from 1 (very low risk taker) to 100 (very high risk taker) and represents how much risk the player likes to take. This attribute could be used to achieve a risk-reward ratio that better suits the player. This could mean, for example, shorter/longer jumps in a platform game or a smaller/higher chance to hit in a RPG combat game.

3.5.2 *Context Model*

There are only two attributes chosen for the game model, as described by figure 3.

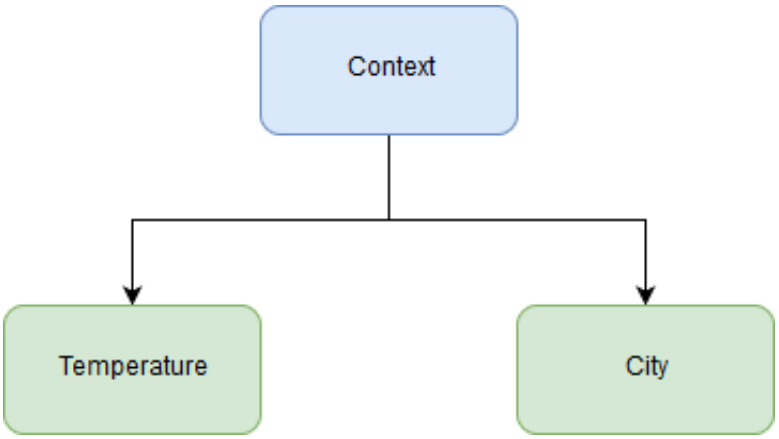


Figure 3.: Context Model Attributes

Location

Name of the location that the player is currently in, like for example Braga, Porto or Lisboa. This location is then used to obtain the current temperature in that location. In a more advanced scenario, this location could be represented by GPS coordinates instead of names of locations.

Temperature

This temperature corresponds to the temperature of the location where the player is. This data is obtained online can be used, for example, to match the weather conditions of the game with real life conditions or to influence the probability of some in-game event.

4

CASE STUDY 1: SIMPLERPG

SimpleRPG is a single player RPG game that was developed to showcase the potential of the PCG methodology. In this section it will be presented the concept of the game as well as a detailed description of its elements.

4.1 GAME CONCEPT

In this game, the player controls a party of multiple heroes and fights against endless waves of monsters. The goal is for the heroes to battle the monsters until the entire party gets wiped out. As the player defeats the waves of monsters, s/he will be rewarded with experience for the heroes as well as several types of items. The items can be equipment such as armors or weapons to directly improve the strength of the heroes as well as runes, which enable special combos. As the heroes get stronger, the player is able to defeat a higher number of waves. The combat of the game is turn based, alternating between the different heroes and monsters that are participating in the combat.

4.1.1 *Heroes*

Heroes are the basic units that the player controls. Each hero has the following main attributes:

- Level
- Experience Points
- Health Points
- Attack Points
- Critical Hit Chance
- Critical Hit Damage

- Defense Points
- Speed

4.1.2 *Equipment*

In order to improve the heroes, the player has the possibility of equipping each hero with several pieces of gear that s/he will acquire throughout the game. This equipment can be divided into two categories: **Weapons** and **Armors**, each with different attributes.

WEAPONS

- Weapon Type
- Attack Points (P.ATK)
- Critical Hit Chance
- Critical Hit Damage

ARMORS

- Defense Points (P.DEF)

4.1.3 *Elemental Runes*

Runes are special items that, when attached to a weapon concedes it elemental proprieties. This opens the possibility of chaining attacks from different heroes in order to activate specific combos.

For example, considering the following combo: fire-water-earth. In a scenario where the player controls three different heroes and each hero has a weapon with a different rune (fire, water, earth), if the player hits the same enemy with a fire-water-earth (in that order), then the combo would be triggered, activating some special effect. On the other hand, if the order was earth-fire-water, then the result would not match any combo and nothing special would happen, other than the regular attack damage.

There are three different types of rune elements in this game:

- Fire (Red)
- Earth (Green)
- Water (Blue)

and they enable three different combos:

- **Fire - Fire - Fire :** Deals two times the damage
- **Earth - Earth - Earth:** Heals the character for 50% of maximum health
- **Blue - Blue - Blue:** Deals two times the damage

4.1.4 *Monsters*

Monsters are the game units that the player must defeat in order to progress from each wave. The monsters work in a similar faction to the heroes, possessing the same attributes.

- Level
- Health Points
- Attack Points
- Defense Points
- Speed

4.1.5 *Monster Waves*

Each wave contains a set of monsters and rewards (equipment and runes). If the player defeats all the monsters in a given wave, it will be rewarded and s/he will move into the next wave. Each wave is progressively more difficult than the last one while simultaneously containing better rewards.

4.1.6 *Gameplay*

The gameplay revolves around two aspects:

- Management of the heroes
- Combat

The player must decide what equipment and runes each one of the heroes is going to use during the combat. These decisions have a direct impact in the probability of success when fighting the monsters.

When the combat starts, the player will be faced with endless waves of monsters. When the entire party of heroes gets wiped out, the waves stop and the player must start again.

The combat is turn-based where each unit (heroes and monsters) has the opportunity to attack the opposing side or activate abilities. The order in which the units attack is determined by their speed. After a unit is selected, the following steps are taken:

1. The player/cpu determines the action to use
2. The player/cpu determines the target of the chosen action

This process is repeated, alternating between units, until either one of the sides is eliminated.

4.2 GAME MODEL

There were a total of two attributes chosen for the game model, as described by figure 4.

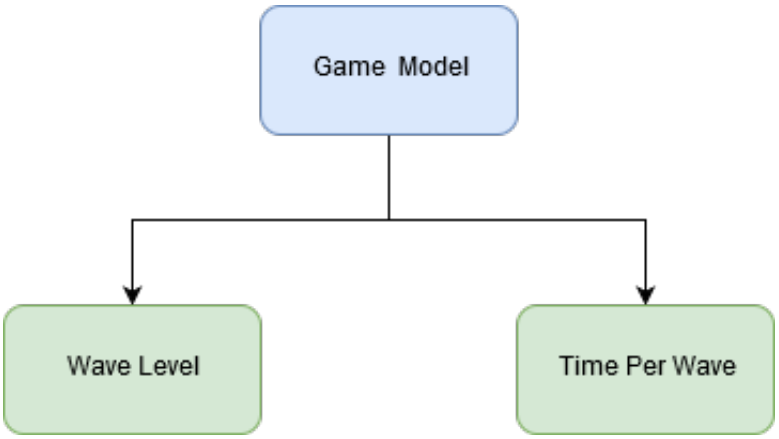


Figure 4.: Game Model Attributes for the RPG game

4.2.1 *Wave Level*

This represents the number of the current wave defeated by the player. This attribute is useful to guide the generation process on how powerful the items should be.

4.2.2 *Time Per Wave Monster*

The average mount of time, in seconds, that the combat takes in each wave. This is useful to determine if the combat is too fast or too slow and adjust the monster generation accordingly. For example, if the combat is taking too long, maybe the monsters have to be tuned down so it does not become boring.

4.3 USAGE OF THE METHODOLOGY IN THIS GAME

This game was developed with the main goal of showcasing the potential of the methodology and therefore it makes use of all the attributes of the three models. Each attribute influenced the **experience** of the characters, the **generation of the items** and/or the **background of the game**. A succinct version of the models used is as follows:

Player Model

- Age
- Aggression
- Gratification
- Player Type
- Risk Taking

It is important to remind that these values (except the age) range from 1 to 100 and the player chooses the right value for each slider at the start of the game.

Context Model

- Location
- Temperature

Game Model

- Time Per Wave
- Wave Level

4.3.1 *Experience*

Experience systems are very common in games. They are a simple mathematical way of representing player progress and they fit the *Game Design* category in the pyramid definition of video game content. When the players defeat a certain wave, they acquire a certain amount of experience points. Those experience points are then added to each character. The amount of experience points required to level up is given according to the formula 1:

$$E = n^3 * a * h \tag{1}$$

where n is the next level for the character, a is the age modifier of the player and h is the player type modifier.

The cubic formula for experience was used because it introduces a sense of progress to the game. The higher the level the harder the player must work to level up their character. Because this is an endless game, it is necessary to make it very hard (but still doable) for players to level up past a certain point, and a cubic function achieves that. This notion of progress is very common in video games, however, using this methodology there are two new factors, the age modifier and the player type. This means that the game becomes easier for new players due to requiring up to less 50% of experience points to level up. At the same time, the player type modifier also changes the experience gain of the characters by adjusting the experience requirement from 50% to very casual players to 150% to very hardcore players.

The age modifier ranges from -0.5 and 1 according to the table 1.

Age Range(Years)	Modifier
< 10	0.5
10-14	0.75
> 14	1

Table 1.: Experience Table

The player type modifier is calculated according to the formula 2:

$$h = 0.5 + playerTypeValue/100 \tag{2}$$

where $playerTypeValue$ ranges from 1 to 100, depending if the player is very casual or very hardcore, respectively.

The range of these modifiers, from $0.5 - 1$ for age and $0.5 - 1.5$ for player type was chosen because it is significant enough to directly impact the game experience of the player and that was the primary concern at this stage of development.

To analyze the impact of the modifiers, 2 shows the difference in experience requirements to reach level 100 with a character, for three players in the same age group (> 14), but with completely different player types, from very casual, to average, to very hardcore.

Age Mod.	Player Type Mod.	Exp. Points Lvl. 100
1	0.5	500000
1	0.75	1000000
1	1.5	1500000

Table 2.: Experience Required for Different Player Types

As shown by the table 2, each type of player would require vastly different amounts of experience (and consequently, time) to level their characters to level 100. This has direct impact in the game experience because a more casual player would require a significantly smaller time investment in the game, which suits the typical play-style of casual players.

The main goal with this experience system is to show that it is generic and modular. The variables were selected to suit this particular game, but other games that make use of experience systems could use the modifiers provided by the in completely different ways.

4.3.2 *Item Generation*

Many games offer something to players when they complete certain tasks in a game. This works both as a reward for the players but in many cases it also functions as a way of enabling further progress into the game. In this game this is done by having a chance of awarding procedurally generated items to the player after s/he defeats a wave. It is important to note that while it is possible to have this PCG occur at the *Game Bits* category, like changing the color of the item, this game implements it mostly at the *Game Design* category by adjusting several formulas to each player. Only one instance of the generation of content in this game is being done at the *Game Bits* level.

The process of item generation begins after the player defeats a wave of monsters. The first thing is to calculate if there was an item drop or not. That is done according to the formula 3:

$$P = 0.2 + \frac{\frac{g}{100}}{1.25} \tag{3}$$

where P is the probability of the player receiving some rewards and g ranges from 1 to 100, depending if the player wants long term rewards of instant gratification, respectively.

With this formula there is a 20% base chance of getting a drop. This means that even the players that enjoy long term rewards and have a value of $g = 0$ have a chance of getting something. At the same time, a player that wants instant gratification and has a value of $g = 100$ will have guaranteed drops after each wave of monsters. This implementation guarantees that the game experience will be vastly different for players in both ends of the Gratification spectrum.

If, in fact, the monsters dropped something, the game has to calculate how many items dropped. The number of drops N is calculated according to formula 4:

$$N = 1 + random(0, 1) + gratificationDrop(g) \tag{4}$$

where the function *random* returns either 0 or 1 with a 50% chance and the function $gratificationDrop$ takes the gratification value (0-100) g of the player and returns 1 with a probability equal to the value of g.

The rationale behind this decision had three main goals, each reflected by each part of the addition. The first was to guarantee at least one drop; The second was to introduce some variance to the game; The third was to reward the player according to its preferences; This means that players the amount of gratification desired for each player will have a direct impact on the amount of items he will receive.

After the amount of item drops are calculated, the game proceeds to generate each individual item, which can either be a *Equipment* or a *Rune*. Table 3 shows the probability of getting each type of item:

Equipment	Rune
95%	5%

Table 3.: Drop Probability of Items

The reason behind equipment having a much higher drop chance than items is because they are more critical to the game. Most of the progression of the characters is done through the equipment they have. On top of that, runes only exist to enable some combos and once the player has a rune, they have little need for a duplicate.

If the drop was a rune, the computer will generate a new rune, with an equal chance of 25% of being of any type (Water, Air, Earth, Fire)

If the drop was an equipment, the next step is to calculate the type of the equipment. In this game there are two possibilities: *Weapons* and *Armors*. The chance of the item being a **weapon** is given by the formula 5:

$$W = 0.5 + aggressionModifier + timeModifier \tag{5}$$

where the $aggressionModifier$ is a value between -0.25 and 0.25 that represents how aggressive is the player and $timeModifier$ is either 0 or 0.25.

The $aggressionModifier$ is calculated using the Aggression Rate (0-100) of the player model in the formula 6:

$$aggressionModifier = \frac{(AggressionRate - 50)}{100} \tag{6}$$

This means that a very passive player with an Aggression Rate of 0 will have a *aggression-Modifier* of -0.25 and a very aggressive player with an Aggression Rate of 100 will have an $aggressionModifier$ of 0.25.

The value for $timeModifier$ is calculated based on the time the player took to defeat the wave. If the player took longer than the expected time (1 minute) to defeat the wave, then the time modifier becomes 0.25, otherwise it is 0. This decision was made to mitigate the scenarios where the battles become too tiresome for the player. If the player is taking too long to defeat the waves, then there is an increased chance of him receiving weapons as rewards, increasing the damage of the characters and thus reducing the time spent on each wave.

The result of this calculations is that the player will have more chance of getting weapons if s/he is aggressive and more armors if s/he is passive, with a shift of 25% towards the weapons if s/he exceeds the expected time per battle.

Finally, after the type of equipment is decided, the game must calculate the stats of the item. If the equipment is a weapon, there are three possible stats:

- P.ATK - Attack points of the weapon
- Crit.Chance - Chance of the attack being a critical hit
- Crit.Damage - Amplification of damage if the attack was a critical hit

The P.ATK of a weapon is calculated according to formula 7:

$$P.ATK = (5 * wave) - \frac{wave}{5 * \log(1+g)} * \frac{g}{100} \tag{7}$$

where $wave$ is the number of the last wave that the player defeated and g ranges from 1 to 100, depending if the player wants long term rewards of instant gratification, respectively.

This is the point of the game where players which prefer long term gratification get rewarded when compared to players that desire instant gratification, because the players that have a desire for instant gratification will receive weapons with lower P.ATK than the players that wanted long term gratification. Table 4 shows the different attack points of a weapon that was generated in a scenario where $wave = 100$.

Wave	Gratification Rate	Weapon P.ATK
100	1	500
100	50	483
100	100	460

Table 4.: Weapon P.ATK According to Gratification

The next two stats, Crit.Chance and Crit.Damage, are both related to the degree of risk that the player likes to take and they are calculated by the formulas 8 and 9:

$$Crit.Chance = \frac{wave}{2.5} * \log(wave) - \frac{riskRate}{5} \tag{8}$$

where *Crit.Chance* is the probability of an attack made with this weapon to critical hit, *wave* is the number of the last wave that the player defeated and *riskRate* ranges from 1 to 100, depending if the player does not like to take risks or if the player is a huge risk taker, respectively. All results of $Crit.Chance < 0$ are treated as 0.

$$Crit.Damage = \frac{1 + \log(wave)}{2} + \frac{riskRate}{100} \tag{9}$$

Formula 9 is similar to formula 8, except that *Crit.Damage* is the amplification of damage of a critical attack made with this weapon and the values are in a different scale because the *Crit.Chance* represents a probability and *Crit.Damage* represents a multiplicative modifier to the damage.

The main idea behind formulas 8 and 9 is to give a risk/reward concept to the generated weapon that ties into the Risk Rate attribute of the player model. If a player likes to play it safe, s/he will have lower Risk Rate in the player model, which will translate into items with higher critical chance (to minimize risk) but that weapon will also do less damage in case of a critical hit. On the other hand, a player with a high Risk Rate in the player model will more frequently obtain weapons with lower critical chance, but the critical damage will be higher (maximize risk), meaning that if the weapon lands a critical hit, then it will do more damage than the weapon of a player who does not take as much risk. Table 5 shows the critical chance and damage values of a weapon where $wave = 100$.

Wave	Risk Rate	Crit.Chance	Crit.Damage
100	1	80%	2x
100	50	70%	2.5x
100	100	60%	3x

Table 5.: Crit.Chance and Crit.Damage According to Risk

If the equipment is an armor, there is only one possible stat, and that is P.DEF, which is calculated according to 10.

$$P.DEF = \frac{(3 * wave)}{10} - \frac{wave}{5 * \log(1 + g)} * \frac{g}{800} \tag{10}$$

where *wave* is the number of the last wave that the player defeated and g ranges from 1 to 100, depending if the player wants long term rewards of instant gratification, respectively. The calculation P.DEF in armors works similarly to the P.ATK in weapons. Players which prefer long term gratification get rewarded when compared to players that desire instant gratification by receiving armors with a higher P.DEF value. Table 6 shows the different defense points of an armor that was generated in a scenario where $wave = 100$.

Wave	Gratification Rate	Armor P.DEF
100	1	29.99
100	50	27.87
100	100	24.99

Table 6.: Armor P.DEF According to Gratification

4.3.3 *Background Generation*

To introduce some visual variance to the game and to make use of the context model, the game procedurally chances its background depending on the temperature of the location where the player is currently playing. There are three possible backgrounds and their usage depends on the temperature ranges described in table 7. Images 5, 6 and 7 show the visual impact of the temperature in the game.

Temperature Range (C°)	Background
< 10	Cold
$10 - 30$	Mild
> 30	Hot

Table 7.: Game Background According to Temperature

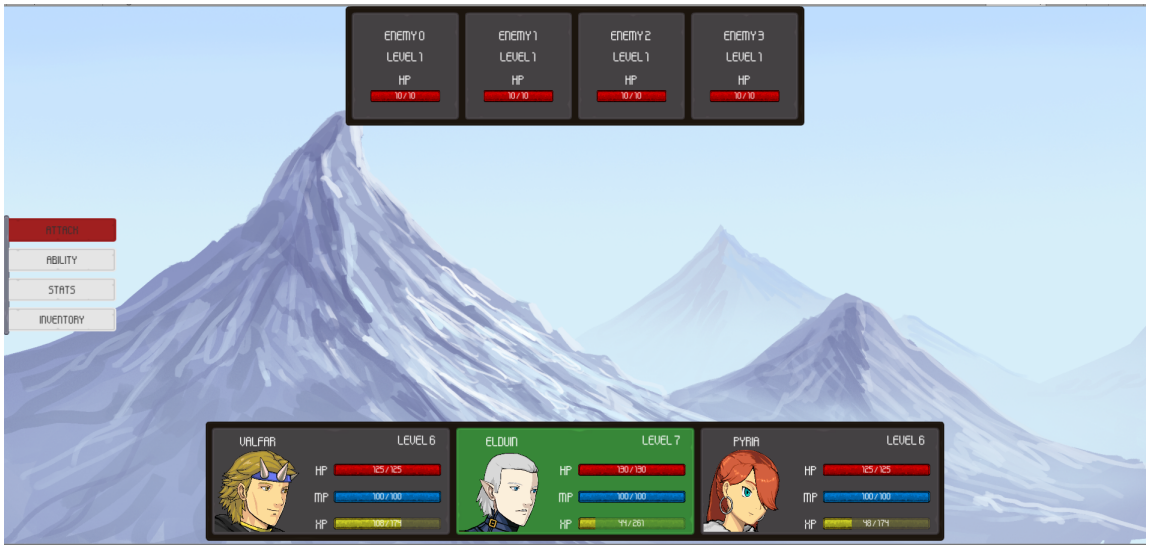


Figure 5.: Background Cold

Figure 6.: Background Mild

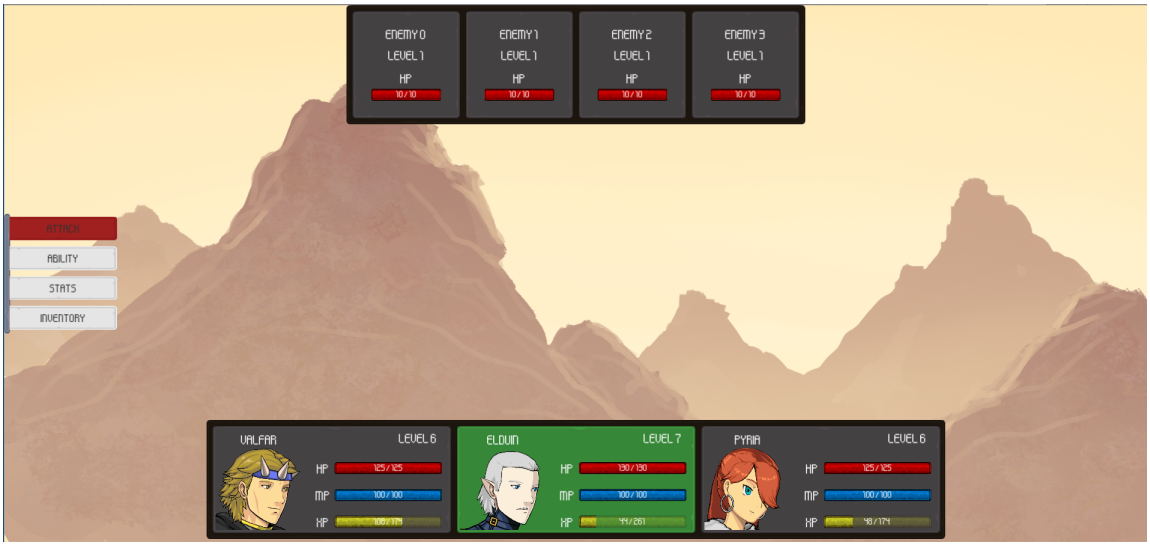


Figure 7.: Background Hot

4.4 CONCLUSION

After the development of the first game was complete, the game was tested with a few users to obtain some data on the effectiveness of the methodology on the game play experience for users. However, it became immediately clear that the developed game was not ideal to obtain such data. To tackle this issue a new game was developed and it is presented next as the second case study.

5

CASE STUDY 2: RGBLOCK

As seen in the conclusion of the previous case study, the first game was not adequate to properly evaluate the impact of the methodology.

The main reason for this is tied to the core nature of the game. In its essence, the first game is focused on optimizing numbers and probabilities and it is very slow by nature, due to being turn based. This translates in a very abstract experience. Moreover, the mechanics that are being affected by the methodology are also abstract. This abstract nature of mechanics and gameplay makes it harder for players to notice the influence of the methodology. Also, the decisions that the player makes in the first game tend to have a **long term** impact, which also makes it harder to assess.

Furthermore, it became apparent that a game solely based on a old school turn-based combat system is not appealing to a great number of players. Typically, turn-based combat systems are part of bigger and more complex games and do not exist by themselves. Due to the nature of this work, the game had to be reduced in scope and complexity and therefore it was not possible to develop a full scale RPG game with more game elements other than just the combat system.

This second game was created with these limitations in mind and the decision making behind its creation aims to provide a solution to them. This translates into several game design decisions that try to overcome the limitations of the first game.

The first is that this second game is considerably faster than the first. In this second game, the player has to constantly make quick decisions that have an immediate impact on the game. This shifts the perspective of the game to the short term, which is a direct contrast to the long term perspective of the first game.

The second decision is a focus on a more visual game. The first lies on the analysis of numbers and probabilities, which are abstract. On the other hand, this second game relies more on the analysis of visual elements of the game. This means that the methodology has a meaningful impact on the visuals of the game.

The third decision was to make the second game independent from any other game context. What this means is that the second game makes sense by itself and does not need to be a part of a larger system to make sense.

Finally, the utilization of the methodology was more focused and translated directly to the mechanics of the game.

Overall, these decisions tried to create a more intuitive and enjoyable game for the players with a simpler and direct application of the methodology.

It is important to note that the creation of this second game was not a setback, but rather a golden opportunity to test the flexibility and adaptability of the methodology. This was also a great opportunity to develop a game more suited to mobile devices, which is important because it opens the door to a future launch on the app store, making it easier to reach more users.

In order to demonstrate the flexibility of the methodology, this second game had to make a different use of some attributes provided by the methodology when compared to the first game. This means that some attributes were not used and some were used in a different way. The details of the usage of the methodology will be provided later in this chapter.

5.1 GAME CONCEPT

In this game, the player is presented with multiple descending rows of two blocks each and the player must select the correct block each time. There are five types of "good" blocks, each with a different effect. If the player selects a "bad" block, s/he will take extra damage. As the player is selecting the blocks, the player is also automatically taking damage, creating a sense of hurry. The aim is to create a fast paced game where the player must quickly select the correct block and try to defeat as many levels as possible, before its own health points reach zero.

5.1.1 *Blocks*

There are five types of *good* blocks and one type of *bad* blocks.

GOOD BLOCKS

- White Block - Does damage to the level
- Red Block - Does extra damage to the level
- Green Block - Heals the player for some amount
- Blue Block - Prevents the level for doing damage for some time
- RGB Block - Combines the effects of the Red, Green and Blue blocks

BAD BLOCKS

- Grey Block - Does damage to the player and resets the combo

5.1.2 *Combos and Abilities*

Each time a player selects a *good* block, it increases a combo counter. If the combo counter reaches certain milestones, the damage done by the player and its abilities is increased. However, each time a player selects a *bad* block, the combo resets and the player must start building the combo from zero.

Alongside this combo, there are four abilities that the player can activate. Each ability is tied to a block color and it requires the player to build specific color combos in order to be able to activate the ability. For example, to activate the Green Ability the player must correctly pick 10 green blocks without picking a Grey (bad) block in the meantime. If the player picks a Grey block by mistake, both the generic and the ability combos are reset.

Each ability has the following attributes:

- Block Effect - How powerful is the effect of the block of the respective color
- Ability Effect - How powerful is the effect of the ability
- Combo Required - How many blocks of the respective color are needed to activate the ability

Red Ability

This ability requires a combo of Red blocks to be activated. Upon activation, this ability does a substantial amount of damage to the level.

Green Ability

This ability requires a combo of Green blocks to be activated. Upon activation, this ability heals the player for a significant amount of health points.

Blue Ability

This ability requires a combo of Blue blocks to be activated. Upon activation, this ability prevents the level from dealing damage for a significant amount of time.

White Ability

This ability requires a combo of White blocks to be activated. Upon activation, this ability transforms **ALL** blocks in the chain into RGB blocks. This means that when this ability is

activated, the player does not encounter any of the Grey blocks. However, this ability has a fixed duration, and upon expiring, the blocks return to normal.

5.1.3 *Loot Boxes and Gems*

When the player loses the game, s/he is awarded a certain amount of loot boxes, depending on how many levels the player was able to defeat. When the player opens the boxes the players can obtain Red, Green and/or Blue gems. These gems are then used to power up the player.

5.1.4 *Upgrades*

The player can spend the gems obtained to upgrade each of its abilities. Upgrading a skill improves some of all aspects of the ability (Block Effect, Ability Effect or Combo Required), as described in 5.1.2.

Each gem color can upgrade the respective ability and the White ability can be upgraded by spending any type of gem, as described in table 8.

	Red Ability	Green Ability	Blue Ability	White Ability
Red Gem	✓	✗	✗	✓
Green Gem	✗	✓	✗	✓
Blue Gem	✗	✗	✓	✓

Table 8.: Gems colors and respective ability upgrades

5.2 GAME MODEL

There are only two attributes chosen for the game model, as described by figure 8.

Even though this model is different from the game model used by the first game (figure 4), this is expected because this methodology requires a concrete implementation of the game model for each game.

5.2.1 *Maximum Level Reached*

How many levels was the player able to defeat before losing the game. This is used to determine the total of rewards given to the player at the end of the play session.

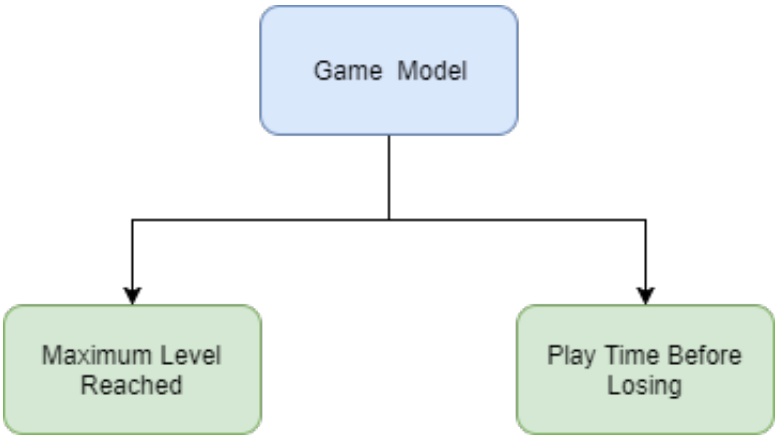


Figure 8.: Game Model Attributes for the RGBlock Game

5.2.2 *Play Time Before Losing*

The amount of time, in seconds, that the player can hold before losing the game. This is used to determine the total of rewards given to the player at the end of the play session.

5.3 USAGE OF THE METHODOLOGY IN THIS GAME

The usage of the methodology in this game is a bit more selective that in the first one. There are two main reasons for this. The first reason, is to showcase the modularity of the methodology. Not every game is equal and therefore it makes sense that different games make use of different aspects of the methodology. The second reason is to avoid redundancy. The first game already showcased the usage of all different models and as such, this game takes a more controlled and accurate approach.

The result of this reasoning was that the context model was not used in this game. A succinct version of the models used is as follows:

Player Model

- Age
- Aggression
- Gratification
- Player Type
- Risk Taking

It is important to remind that these values (except the age) range from 1 to 100 and the player chooses the right value for each slider at the start of the game.

Game Model

- Maximum Level Reached
- Play Time Before Losing

There are a total of four systems within the game that are influenced by these models: The **difficulty system**, the **block system** , the **combo system** and the **reward system**, which will all be detailed next.

5.3.1 *Difficulty System*

The difficulty of this game is tied to two main components: the Hit Points (HP) of the level and the Damage Per Second (DPS) that the player takes in each level. These two components are affected by both the *age* and the *type of the player*, both present in the player model.

The total amount of HP of each level is calculated according to formula 11 while the DPS of each level is calculated according to formula 12.

$$HP = 100 + n^2 * a * h \tag{11}$$

$$DPS = 1 + \frac{n}{5 * \log(1+n)} * a * h \tag{12}$$

where n is the number corresponding to the current level, a is the age modifier ranging from 0.5 to 1 and and h is the player type modifier, ranging from 0.5 and 1.5.

Because this is an endless game, both formula 11 and 12 offer a significant growth, meaning that the levels will get progressively harder both because they have more HP and because they deal more DPS. The actual choice of each formula and its values is tied to the scale of each mechanic. Each formula offers a different starting point at level one (100 to formula 11 and 1 to formula 12). Moreover, the HP of the level grows at a faster rate than the DPS. This is to prevent the player from being overwhelmed with very high damage from the level but at the same time, to provide harder challenges as the levels go up. The exact choice of numbers was found through testing and it subject to further change.

These formulas also scale with both the age and the player type. This is specially important in this game, because this game is about speed and precision when pressing keys, which means that really young and elderly people might have a harder time than the rest

of the players and that must be taken into consideration. This resulted in a new way of calculating the age modifier, when compared to first game. The result of this new calculation is shown in table 9. The calculation of the player type modifier stays unaltered in this game, and thus it is still calculated according to formula 2.

Age Range(Years)	Modifier
< 14	0.5
14-30	1
31-50	0.75
> 50	0.5

Table 9.: New Age Modifier Range

Similarly to the first game, the range of these modifiers was chosen to impact the game experience in a significant way. To evaluate this impact, tables 10 and 11, showcase the variation of the level HP and DPS, for four average players with a player type modifier of 1, according to their age.

Age Group	Age Mod.	Player Type Mod.	HP at level 50
< 14	0.5	1	1350
14-30	1	1	2600
31-50	0.75	1	1975
> 50	0.5	1	1350

Table 10.: Level HP for Different Age Groups

Age Group	Age Mod.	Player Type Mod.	DPS at level 50
< 14	0.5	1	9.5
14-30	1	1	18.0
31-50	0.75	1	13.7
> 50	0.5	1	9.5

Table 11.: Level DPS for Different Age Group

Looking at this values it is clear that there is a significant difference in the values for each age group. The variation according to the player type is not shown on the tables but it carries the same significance.

Because this is a fast paced game, the total amount of HP and DPS of each level has an immediate impact on the gameplay experience. This is a shift from the first game, where the age and player modifiers were used in the experience system, which was much slower and with a almost insignificant impact on the game, due to its scale.

5.3.2 *Block System*

Blocks are the core mechanics of this second game. Until the player loses all of his/hers HP, the game procedurally generates new blocks and adds them to the blockchain. In the generation process, each block is attributed a color which in turn makes the blocks have different effects, as explained in 5.1.1. Because different effects have a different impact on the way the game plays, the probability of generating each block has a direct impact on gameplay. This means that these probabilities can be manipulated to offer different game experiences.

To implement this manipulation, it was used the *Aggression* attribute from the player model. This attribute is a natural choice to impact the block system, because there is a clear division in the block types: *White* and *Red* blocks are damage dealing, offensive blocks, while *Green* and *Blue* blocks offer defensive healing and freezing abilities. Therefore, **in the context of this game**, it makes sense to associate the aggression attribute to those defensive and offensive abilities.

In practice, this means that offensive players have a higher chance of encountering *White* and *Red* blocks, while defensive players have a higher chance of encountering *Green* and *Blue* blocks.

The first step to implement this was to define the boundaries of what is an average, defensive or offensive player. Table 12 shows how those are defined.

Aggression Range	Classification
< 35	Defensive
35 - 65	Neutral
> 65	Offensive

Table 12.: Player Classification According to Aggression

Each one of these classifications corresponds to a different set of probabilities of block generation.

Player Classification	White	Red	Green	Blue
Defensive	50%	5%	30%	15%
Neutral	60%	10%	20%	10%
Offensive	70%	15%	10%	5%

Table 13.: Block Color Probabilities According To Aggression Classification

By analyzing table 13, it is possible to see that defensive players have a higher chance of getting defensive blocks (*Green* and *Blue*) while offensive players have a higher chance of

getting offensive blocks (*White* and *Red*). *White* blocks have a high chance of being generated regardless of the player classification because they are the most basic block of the game and essential to advance levels.

5.3.3 *Combo System*

The combo system is designed to reward players who consecutively select the right blocks, as described in 5.1.2. The combo system can be divided into two separate aspects:

- Combo Threshold
- Multiplier Increase

The *Combo Threshold* represents how many blocks in a row the player must correctly select in order to increase the multiplier and the *Multiplier Increase* represents the growth of said multiplier.

So for example, a combo with a threshold of **50** and a multiplier increase of *0.1* means that for every 50 good blocks the player selects in a row, the effect of the good blocks is amplified by 1.1x.

This system is a good candidate for a risk/reward situation, which means that it makes sense to use the *Risk Taking* attribute from the player model to make this system player-sensitive. This was achieved by using the *Risk Taking* value to calculate both the *Combo Threshold* and the *Multiplier Increase* values, as described by formulas 13 and 14, respectively.

$$ComboThreshold = 50 + RiskRate \tag{13}$$

$$MultiplierIncrease = 0.1 + \frac{RiskRate}{100} \tag{14}$$

where *RiskRate* is a value between 1 (very safe player) and 100 (very risky player). This creates a situation of high risk, high reward, where riskier players are faced with hard, high rewarding combos and safer players encounter easy, less rewarding combos. Table 14 shows some examples of possible values of these functions.

RiskRate	ComboThreshold	MultiplierIncrease
1	51	0.1
50	100	0.6
100	150	1.1

Table 14.: Mininum and Maximum Values For Combos

This means that, for example, a player with a *RiskRate* of 50 would increase his damage multiplier by 0.6 for every 100 good blocks selected in a row. Also, it is important to add that this *MultiplierIncrease* value is always added to the previous multiplier value *before* applying other game calculations. The base multiplier value is 1 and can never be lower than that. This means that the only consequence of failing to combo is the reset of the combo multiplier to the default value of 1.

5.3.4 *Reward System*

The last system influenced by the methodology is the reward system. Every time the player finishes (loses) a game, s/he is rewarded with a certain amount of loot boxes. Each box contains colorful gems that are then used to upgrade player abilities, as described in 5.1.4.

There are two things procedurally generated in this system: the amount of loot boxes that the player receives, influenced by the *Maximum Level Reached* and the *Play Time Before Losing* attributes from the game model, and the contents of each loot box, influenced by the *Gratification* attribute from the player model.

The total amount of loot boxes that the player receives is calculated according to formula 15

$$LootBoxes = \frac{PlayedTime}{120} + \frac{LevelReached}{10} \tag{15}$$

where *PlayedTime* is the total amount of time, in seconds, that the player was able to hold on before losing and *LevelReached* is the maximum level reached by the player, also before losing. This effectively means that the player gains a loot box for every two minutes that s/he plays and a box for every 10 levels that s/he defeats.

The reasoning behind this decision is to make use of characteristics that are specific to this game to enhance the player experience. Due to the nature of this game, where the player must "hold on" and keep playing for as long as possible, it makes sense to reward the player based on those accomplishments. By comparison, it would make no sense to reward the player in the first game based on *Play Time Before Losing*, because it is a turn based game with no sense of hurry or time pressure. In the first game, the players can take 10 seconds or 5 minutes to make their decisions that it is irrelevant to the reward system. The key thing here is to show that different games have different game models and make a different usage of their attributes.

Once the players have the loot boxes, they have to open them, and that is when the *Gratification* attribute from the player model comes in. There are two parts to opening a loot box. First, the game calculates how many sets of gems the player receives and adds them

to their inventory. Then the player opens each set of gems and the game determines how many gems each set contains. This is done according to formulas 16 and 17, respectively.

$$TotalSetsOfGems = 1 + gratDrop(g) + gratDrop(g + 10) + gratDrop(g + 20) \tag{16}$$

$$GemsPerSet = 50 + (100 - GratificationRate) \tag{17}$$

In the formula 16, the function *gratDrop* takes the gratification value (1-100) g of the player plus a certain number and returns 1 with a probability equal to the value of the input, otherwise it returns 0. By using this formula, the players have one gem set guaranteed every time they open a loot box plus up to three extra sets, with probabilities depending on their gratification levels. The maximum of four gems per set is due to the fact that there are three possible colors of gems. By having up to four gem sets, the players have more chance of getting all three colors.

Formula 17 is very simple and it calculates the total amount of gem according to the player's gratification value (1-100). This means that the player will always get between 50 and 150 gems, which, according to initial tests indicates that is a palpable number that is useful to upgrade the skills at a realistic rate.

This means that players with a higher gratification will get more gem sets to open, but get less gems in each set. On the other hand, players with lower gratification values will get less gem sets to open, but they will obtain a higher quantity of gems in each set.

5.4 USER EVALUATION

The goal of any game is to provide the players with an enjoyable experience and this work proposes a methodology that might facilitate the creation of better and more personal game experiences. As such, the aim of this experiment is to get some insight on whether the utilization of the methodology on this second game was effective or not. To obtain such insight, it is necessary to answer the following question:

When playing RGBlock, does the utilization of the methodology improves the game experience for players?

This is the question that this small experiment aims to answer. That said, properly testing games and evaluating player experience takes a significant amount of resources that are not compliant with the scope of this work. This means that this is not an extensive experiment and it does not produce statistically relevant data. This is simply a **preliminary way** to obtain some insight on the impact that this concrete implementation of the methodology had on this second game.

5.4.1 *Setup*

Due to the restrictions previously mentioned, the experiment focused on one simple attribute of the player model, the *Aggression Rate*. As described in subsection 5.3.2, this attribute has a strong influence in the way new blocks are generated, depending on weather the player is *Defensive, Neutral or Offensive*.

There were selected a total of 10 players to test the game, with the following *Aggression Rate* distribution:

- Defensive Players: 5
- Offensive Players: 5

Each one of these players played two versions of the game:

1. Neutral version with the default configuration
2. Adapted version with the configuration dependent on the *Aggression Rate*

This means that defensive players played with the defensive configuration and offensive players played with the offensive configuration and they all played the neutral version.

Note that the players did not know which version they were playing and the order of the versions was randomized, meaning that some players played the neutral version first while others played it last.

After playing both versions, the players were asked the following questions:

- Did you noticed any difference in the player experience between each version?
- If so, which one is the preferred version?

5.4.2 *Results*

Tables 15 and 16 show the answers to the proposed questions by the defensive and offensive players, respectively.

	Player 1	Player 2	Player 3	Player 4	Player 5
Player Classification	Defensive	Defensive	Defensive	Defensive	Defensive
Different Experience	Yes	Yes	Yes	Yes	No
Preferred Version	Defensive	Defensive	Defensive	Neutral	Either

Table 15.: Answers of the defensive players to the proposed questions

	Player 1	Player 2	Player 3	Player 4	Player 5
Player Classification	Offensive	Offensive	Offensive	Offensive	Offensive
Different Experience	Yes	Yes	Yes	Yes	Yes
Preferred Version	Neutral	Neutral	Neutral	Offensive	Offensive

Table 16.: Answers of the offensive players to the proposed questions

From this data it is possible to extract the following:

- **9** out of the 10 players noticed a difference in the game experience between the two versions;
- Of the **9** players that noticed a difference, **5** showed a preference to the adapted version;
- Of the **4** players that preferred the neutral version, **3** players were of the offensive type;.

5.4.3 *Discussion*

By taking a look at the data presented in the previous subsection, there are some interesting aspects that emerge.

The first is that the methodology seems to have a noticeable impact on the game. Of the ten total players, only one was not able to notice any difference between the versions, even though the probabilities of generating each color do not vary more than **10%** between the neutral and each of the adapted versions.

One possible explanation to this is the fact that this second game is very fast and very visual. Even though the probabilities do not vary substantially, because the game is so fast, a great amount of blocks ends up being generated and consequently encountered by the players. On top of that, because the blocks are colorful, they make a visual impact on the player, which can help explain why the difference is so noticeable between the versions.

The second aspect is that roughly **56%** of the players that do noticed a difference preferred the adapted version. This number must be considered with caution, mostly due to the size of the experiment, but it can be an indicator that the methodology has a positive impact on some players.

The third aspect is the fact that of the **4** players that perceived a negative impact from the methodology, **3** were from the offensive type. This might be because of balance of the game and not the methodology itself. Because offensive players tend to have less defensive (**GreenGreen** and **Blue** they could end up losing the game faster, which might lead to frustration and this having a preference towards the neutral version.

5.5 CONCLUSION

As mentioned before, the data obtained through this preliminary experiment is not statistically relevant and as such, it is not possible to draw significant conclusions.

Moreover, the game is not fully balanced yet. This natural unbalance of each block effect might cause some blocks to have a more powerful effect than others. This leads to some blocks being more desirable (or necessary) than others and that might cause the methodology to produce unsatisfying results to the players that end up with the weaker block effects.

Regardless of that, the data seems to point to the usefulness of the methodology. In this small scale experience, more than half of the players preferred the adapted version. By (cautiously) escalating this number to a game with a larger player base, it is possible to end up with thousands of players who get a better game experience, even if it impossible to perfectly adapt the game to everyone.

With some adjustments to the game and with a larger sample size it will be possible to obtain more relevant information that could lead to an improvement of the effectiveness of the methodology.

6

CONCLUSION AND FUTURE WORK

This final chapter discusses the overall work presented in this dissertation and offers some insight on the possible paths of improvement that could be followed in the future.

While traditional PCG techniques are well established in video-game development, the adaptive techniques are still mostly experimental and impractical to use as a generic game development tool. The massive diversity of today's player-base makes it extremely difficult to trace an accurate profile of each individual player without asking for player input. Moreover, reading the emotions of the players by using external sensors is not practical, because it requires specific equipment to get accurate readings.

The main goal of this work was to explore the possibility of creating a methodology that could lower the barrier of adaptive video game content, so that developers could more easily implement such content in their games. The resulting methodology of this work offers a way to obtain the profile and the context of the player. The developers can then use the information obtained by the methodology to creatively adapt the content generation of their games.

To test the methodology, two games were created. As a proof of concept, the first game made use of all attributes of the methodology, but due to the nature of its design, the game turned out to be impractical and not suitable to test with most users.

Learning from the mistakes of the first game, the development of the second game took a more precise approach, offering solutions to the problems presented in the first game. The faster pace coupled with the visual aspect of the game, turned out to be the key aspects that made the game more enjoyable and suitable to test the methodology with users.

Although very preliminary and with many aspects of possible improvement, the tests with the users got a very positive response both on the design of the game but more importantly on its adaptive nature. Most players were able to recognize the impact of the methodology and some were of the opinion that the adaptive version offered a better game experience when compared to the neutral version.

In terms of future work, there are many paths of improvement to take. The methodology needs to be refined to offer a more suitable player profile. One example of this is the simplification of the range of each attribute from a 0 to 100 range to more defined and

specific intervals. The models of the methodology could also be improved to offer more attributes and thus opening more game design possibilities.

Another key aspect is the development of systems capable of identifying the profile of the player without player input. On one hand, this would ease the burden on the player, but it can be argued that the players know themselves better than anything else, so this is a subject that requires additional research.

Finally, the possible release of the second game on the mobile stores can facilitate the gathering of statistically significant data, which is also critical to a precise evaluation on the effects of the methodology.

BIBLIOGRAPHY

[Bit12] Mike Bithell. Thomas Was Alone. Mike Bithell, 2012.

[Bro11] Cameron Browne. *Evolutionary Game Design*. Springer-Verlag London, 2011.

[Cor08] Valve Corporation. Left 4 Dead. Valve Corporation, 2008.

[Cor12] Valve Corporation. Counter-strike: Global offensive. Valve Corporation, 2012.

[Csi91] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York, NY, March 1991.

[DDRT11] Isaac M. Dart, Gabriele De Rossi, and Julian Togelius. Speedrock: Procedural rocks through grammars and evolution. In *Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games*, PCGames '11, pages 8:1–8:4, New York, NY, USA, 2011. ACM.

[Dog07] Naughty Dog. Uncharted: Drake's Fortune. Sony Interactive Entertainment, 2007.

[Dre10] Quantic Dream. Heavy Rain. Sony Computer Entertainment, 2010.

[EM11] Florian Himsl Edmund McMillen. The Binding of Isaac. Edmund McMillen, 2011.

[Ent04] Blizzard Entertainment. World of warcraft. Blizzard Entertainment, 2004.

[Ent10] Blizzard Entertainment. Starcraft II: Wings of Liberty. Blizzard Entertainment, 2010.

[Ent12] Blizzard Entertainment. Diablo III. Blizzard Entertainment, 2012.

[Fre13] Game Freak. Pokemon X/Y. The Pokémon Company, Nintendo, 2013.

[Gam01] PopCap Games. Bejeweled. PopCap Games, 2001.

[Gam10a] Evolutionary Games. Galactic Arms Race. Evolutionary Games, Evolutionary Complexity Research Group, 2010.

[Gam10b] Firaxis Games. Civilization V. 2K Games, 2010.

[Gam16] Hello Games. No Man's Sky. Hello Games, 2016.

[HC04] Robin Hunicke and Vernell Chapman. AI for Dynamic Difficulty Adjustment in Games. 2004.

[HF97] L. Herman and K. Feinstein. *Phoenix: The Fall & Rise of Videogames*. Rolenta Press, 1997.

[HGS09] Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. Automatic Content Generation in the Galactic Arms Race Video Game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, December 2009.

[HMVDVI13] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1):1:1–1:22, February 2013.

[Hun05] Robin Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '05, pages 429–433, New York, NY, USA, 2005. ACM.

[Int01] Sports Interactive. Championship Manager: Season 01/02. Eidos Interactive, 2001.

[Ios09] Alexandru Iosup. Poggi: Puzzle-based online games on grid infrastructures. In Henk J. Sips, Dick H. J. Epema, and Hai-Xiang Lin, editors, *Proc. of the 15th Int'l. Euro-Par Conference on Parallel Processing (Euro-Par)*, volume 5704 of *Lecture Notes in Computer Science*, pages 390–403, Berlin, Germany, 2009. Springer.

[Kin12] King. Candy Crush Saga. King, 2012.

[LHJB13] Ricardo Lopes, Ken Hilf, Luke Jayapalan, and Rafael Bidarra. Mobile adaptive procedural content generation. In *Proceedings of PCG 2013 - Workshop on Procedural Content Generation for Games, co-located with the Eigth International Conference on the Foundations of Digital Games*, Chania, Crete, Greece, may 2013. Society for the Advancement of the Science of Digital Games. ISBN 78-0-9913982-1-8.

[Max08] Maxis. Spore. Electronic Arts, 2008.

[Med08] Media Molecule, SCE Cambridge Studio, Tarsier Studios, Double Eleven, XDev,United Front Games, Sumo Digital. Little Big Planet. Sony Computer Entertainment, 2008.

[Mis15] Olana Missura. *Dynamic Difficulty Adjustment*. PhD thesis, University of Bonn, 2015.

[Moj11] Mojang. Minecraft. Mojang, 2011.

[Mos08] Mossmouth. Spelunky. Mossmouth, 2008.

[Nia16] Niantic. Pokémon go. Niantic, 2016.

[Ord15] Colossal Order. Cities: Skylines. Paradox Interactive, 2015.

[PLMK08] R. Poli, W.B. Langdon, N.F. McPhee, and J.R. Koza. *A Field Guide to Genetic Programming*. Lulu.com, 2008.

[Sch08] Jesse Schell. *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[STN16] Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[Stu99] Ensemble Studios. Age of Empires II: The Age of Kings. Microsoft Game Studios, 1999.

[Stu11] Bethesda Game Studios. The Elder Scrolls V: Skyrim. Bethesda Softworks, 2011.

[TKSY11] Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis. What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games*, PCGames '11, pages 3:1–3:6, New York, NY, USA, 2011. ACM.

[TYSB11] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, Sept 2011.

[vdLLB14] R. van der Linden, R. Lopes, and R. Bidarra. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):78–89, March 2014.

[Wil02] Dmitri Williams. Structure and competition in the u.s. home video game industry. *International Journal on Media Management*, 4(1):41–54, 2002.

[YT11] Georgios Yannakakis and Julian Togelius. Experience-Driven Procedural Content Generation. *Affective Computing, IEEE Transactions on*, 2(3):147–161, July 2011.

A

GAME SCREENSHOTS

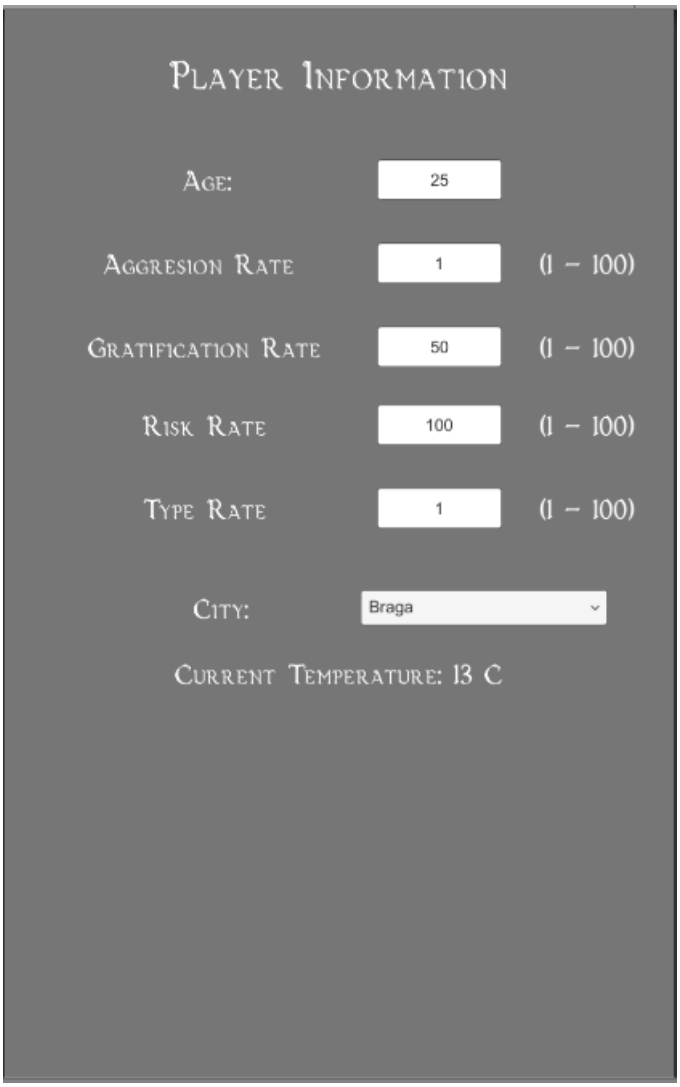


Figure 9.: Methodology Screen - RGBlock

Figure 10.: Upgrade Screen - RGBlock

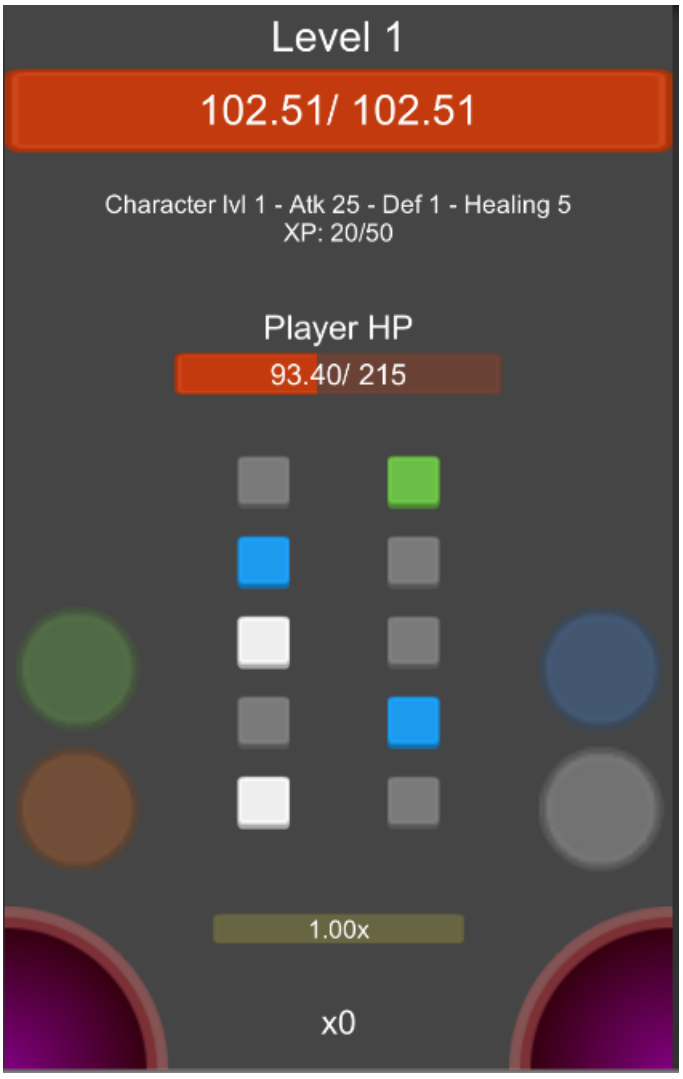


Figure 11.: Game Screen - RGBlock

Figure 12.: Simple RPG