TÉCNICO LISBOA

Procedural Content generation for Cooperative Games

Pedro Costa Borges

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Carlos António Roque Martinho
Prof. Rui Filipe Fernandes Prada

Examination Committee

Chairperson: Prof. Mário Jorge Costa Gaspar da Silva
Supervisor: Prof. Carlos António Roque Martinho
Members of the Committee: Prof. João Miguel De Sousa de Assis Dias

May 2018

Acknowledgments

First of all I would like to thank to my wonderful supervisors, the professors Rui Prada and Carlos Martinho for all the help, guidance and knowledge they provided in this long journey. I would also like to thank my dearest friends for their support, patience and friendship in all these years and all the battles we faced together. And very special thanks to my father, mother and sister as without their unconditional love and support I would not be the man I am today.

To each and every one of you – Thank you.

Abstract

Procedural content generation is a big part of the gaming industry nowadays, from generating levels, to weapons and even characters, this method of content creation has multiple uses and covers a variety of areas. In this work, we tackle the existing gap in procedural content generation, specifically for cooperative games by presenting a solution of our own. Our work is comprised by a level editor with the capability of generating cooperative levels for the game Geometry Friends. The level generator was based of the algorithm Monte Carlo Tree Search and a flood field based evaluation algorithm. We conducted experiments to see if we were successful in developing such tool and if the levels generated were interesting, fun and were in fact cooperative. Our results suggest that we achieved our goals in creating a tool that produced cooperative levels that are engaging.

Keywords

Procedural Content Generation, Cooperation in Videogames, Level editor, Monte Carlo Tree Search.

Resumo

A geração de conteúdo procedimental é uma grande parte da indústria de jogos hoje em dia, são gerados níveis, armas e até personagens, abrangendo varias áreas na industria. Neste trabalho, abordamos a lacuna existente na geração de conteúdo procedimental, especificamente para os jogos cooperativos, apresentando uma solução. Nosso trabalho é composto por um editor de níveis com a capacidade de gerar níveis cooperativos para o jogo Geometry Friends. O gerador de nível foi baseado no algoritmo Monte Carlo Tree Search e num algoritmo de avaliação de "flood field". Diversas experiências foram realizadas para averiguar e validar a ferramenta criada e se os níveis gerados são interessantes, divertidos e de fato cooperativos. Os nossos resultados sugerem que alcançamos nossos objectivos na criação de um algoritmo que gera níveis cooperativos.

Palavras Chave

Geração Procedimental de Conteúdos, Cooperação em Videojogos, Editor de niveis, Monte Carlo Tree Search.

Contents

1 Introduction **1**

1.1 Motivation . . . 3

1.2 The problem and objectives . . . 4

1.3 The Contributions . . . 4

1.4 Outline of the Document . . . 4

2 State of the art and related works **7**

2.1 Geometry Friends . . . 9

2.2 Cooperation in videogames . . . 9

2.2.1 Evaluating Cooperative games . . . 11

2.2.2 Evaluating Cooperation . . . 15

2.3 Procedural Content Generation . . . 16

2.4 Procedural Content Generation for Cooperative Games . . . 21

2.5 Monte Carlo Tree Search . . . 23

2.5.1 The simulation . . . 26

3 Implementation **29**

3.1 Technologies used . . . 31

3.2 Solution overview . . . 31

3.3 Level editor . . . 33

3.4 Level generator algorithm implementation . . . 34

4 Experiment and Results **37**

4.1 The experiment preparation . . . 39

4.2 The experiment . . . 40

4.3 Experimental results . . . 40

4.4 Result significance . . . 51

5 Conclusion **53**

5.1 Conclusions . . . 55

5.2 Future work . . . 55

A MOJO's questionnaire **61**

B MOJO's metric table **65**

C Levels presented at MOJO **66**

C.1 World 1 67

C.2 World 2 69

C.3 World 3 70

List of Figures

2.1 Example of a level in Geometry Friends, where the circle uses the rectangle to reach a platform. . . . 10
2.2 Mercy's abilities that can only be used on another player. . . . 12
2.3 Rocket league's [1] 2 players split screen[1]. . . . 13
2.4 GTA: San Andreas's [2] one character focus camera[2]. . . . 13
2.5 Magicka's [3] all characters are in focus[3]. . . . 13
2.6 Cho ability set. Cho and Gal are the same character, Cho'Gal. Source: Heroes of the storm [4] . . . 14
2.7 Gal ability set. Cho and Gal are the same character, Cho'Gal. Source: Heroes of the storm [4] . . . 14
2.8 A single iteration from the algorithm. Figure taken from [5] . . . 25

3.1 The level editor. The numbers in this figure represent the various components of the editor. 33

4.1 Players evaluation of the Cooperation and Balance for World 1. . . . 41
4.2 Players evaluation of the Cooperation and Balance for World 2. . . . 42
4.3 Players evaluation of the Cooperation and Balance for World 3. . . . 42
4.4 World 1 - Comparing the nº of events of *Laugh and excitement together* between levels. . . 44
4.5 World 1 - Comparing the nº of events of *Work out strategies* between levels. . . . 45
4.6 World 1 - Comparing the nº of events of *Global strategies* between levels. . . . 46
4.7 World 1 - Comparing all events between levels. . . . 46
4.8 World 2 - Comparing all events between levels. . . . 47
4.9 World 3 - Comparing the nº of events of *Laugh and excitement together* between levels. . . 48
4.10 World 3 - Comparing the nº of events of *Work out strategies* between levels. . . . 49
4.11 World 3 - Comparing the nº of events of *Global strategies* between levels. . . . 50
4.12 World 3 - Comparing all events between levels. . . . 50

A.1 Questionnaire presented at *Montra de Jogos do IST* (MOJO). . . . 63

B.1 Table used to record the number of events during a play session. 65

C.1 Tutorial level. This was the first level the played in a game session to help players learn the game and the controls. 67
C.2 Level 1 from World 1. This level was generated by our tool. 67
C.3 Level 2 from World 1. This level was picked from the list of levels of Geometry Friends. . . 68
C.4 Level 3 from World 1. This level was generated by our tool. 68
C.5 Level 1 from World 2. This level was generated by our tool. 69
C.6 Level 2 from World 2. This level was picked from the list of levels of Geometry Friends. . . 69
C.7 Level 3 from World 2. This level was generated by our tool. 70
C.8 Level 1 from World 3. This level was generated by our tool. 70
C.9 Level 2 from World 3. This level was picked from the list of levels of Geometry Friends. . . 71
C.10 Level 3 from World 3. This level was generated by our tool. 71

List of Tables

4.1 World classifications in terms of cooperation and balance. 40
4.2 World 1 - *Laugh and excitement together* averages and Standard Deviation. 43
4.3 World 1 - *Work out strategies* averages and Standard Deviation. 44
4.4 World 1 - *Global strategies* averages and Standard Deviation. 45
4.5 World 3 - *Laugh and excitement together* averages and Standard Deviation. 47
4.6 World 3 - *Work out strategies* averages and Standard Deviation. 48
4.7 World 3 - *Global strategies* averages and Standard Deviation. 49

List of Algorithms

2.1 Main loop 23
2.2 Selection step 24
2.3 Expansion step 24
2.4 Simulation step 25
2.5 Backpropagate step 25

Acronyms

GF	Geometry Friends
PCG	Procedural Content Generation
MOJO	*Montra de Jogos do IST*
CPMs	Cooperative Performance Metrics
IST	*Instituto Superior Técnico*
NPCs	Non-Playable Characters
MCTS	Monte Carlo Tree Search
MMORPG	Massively Multiplayer Online Role-playing Game
MOBA	Multiplayer Online Battle Arena
FPS	First Person Shooters
AI	Artificial Intelligence
WFA	Windows Form Application
UCT	Upper Confidence Bound 1 applied to trees
IST	Instituto Superior Tecnico

1

Introduction

Contents

1.1 Motivation 3
1.2 The problem and objectives 4
1.3 The Contributions 4
1.4 Outline of the Document 4

This chapter contains the introduction to our work, what motivated us to realize this work, what are our objectives, contributions and the outline of this document.

1.1 Motivation

The game industry has come a long way since its beginning, today being one of the largest in the world. Nowadays there is a high demand for games, but creating a good game takes a lot of effort and time. Gamers consume content faster than developers can create, this creates a content drought. This phenomenon is especially evident in cooperation driven games, because every level must be designed, created and tested by hand by a level designer. Level designers need better tools to help them accelerate the creation process of levels or tools to automatically generate the levels. Nowadays, developers attempt to save time by trying to automatize everything they can, by using Procedural Content Generation (PCG), they use it to generate Non-Playable Characters (NPCs), terrain, foliage, skies and everything that can be randomized. This way they can create entire levels in an instant. This type of techniques are used in a variety of games such Elite: Dangerous [6], where an entire universe is created at a realistic scale of 1:1. This is a clear example where, if a level designer were to create everything by hand, it would be impossible to create something with the same magnitude. That's why we believe that developers and level designers need more tools to create more levels and games, mainly cooperative driven games. Most topics covering PCG do not take cooperation into consideration. Typically cooperative video games which utilize PCG do not have the PCG affect cooperation between players. For example, the game The Binding of Isaac: Rebirth [7], uses PCG to generate the levels of the game. You can play alone (Single-player) or Co-op (cooperation, in this case share screen with a friend), the level generation does not take the fact you are playing alone or Co-op. Even developers do not use PCG to generate cooperative driven levels and simply add a second player to a single-players game, this effects the cooperative experience, since one player can simply pass the level alone and ignore the other player. Video games like Portal 2 [8] the players are forced to cooperate with one another to complete the levels, but the levels are all made by hand and the game does not contain any kind of PCG in the levels.

As mentioned above, there is a lack of tools to help developers create cooperative driven levels. In our opinion, games like The Binding of Isaac: Rebirth [7] would highly benefit from a more cooperative approach as it would make the co-op gameplay more interesting and fun. Games like Portal 2 would also benefit, because they would have a tool to generate new levels, creating more diversity and keeping the players engaged in the game for more long periods of time. PCG seems a great solution to this problem, although it creates some problems of its own. To create cooperative driven challenges, there has to be some sort of evaluation of the challenges and levels.

1.2 The problem and objectives

The main problem that stated in this document is procedural generation of cooperative levels. We developed a tool that generate levels automatically and test them based on cooperation. Our main focus was the development of a complete level editor that, given some inputs, can generate a cooperative level. We set the following objectives to reach at the end of this work:

- Developed a level editor, where anyone can create levels for the game Geometry Friends;
- Created a solution to generate cooperative levels;
  - A solution based on the algorithm Monte Carlo Tree Search and in conjunction with a flood field evaluation method;
  - This solution will have as inputs the platforms that will make the level and the level objectives (as it will be explained in the following chapter the level objectives are gems);
- Developed a solution that besides generating cooperative levels, can also generate balanced and interesting levels;
- Test and prove that our generator works.

1.3 The Contributions

The main contributions of this paper are the proposal of a solution for procedural level generation for cooperative games. Other contributions are, a collection of papers relevant to the state of the art of procedural content generation, cooperation in games and some decision making algorithms, all of which have helped us arrive at the current solution.

We also provide an algorithm to produce the solution of creating cooperative levels, in this case it was developed for the game Geometry Friends (GF) [9]. A level editor and an experiment is also provided to where we test if the solution is reliable.

1.4 Outline of the Document

This document consists of several sections. We begin by examining the background on our two main topics, PCG and cooperation in videogames and their definitions. We shall also discuss the state of the art for PCG, analyzing the current state and important works. To devise our solution, we analyze several works that we can use as contributions to our approach, this includes: approaches on PCG in cooperative games; on how to evaluate cooperative games and cooperation; a section on the decision

making algorithm Monte Carlo Tree Search that was used has base for our tool; and an approach that used Monte Carlo Tree Search (MCTS) to run simulations. Finally we present our approach in a complete breakdown of the solution, how it works and the other several approaches that were tested. We finalize the document with the experiment that was done to test the proposed solution, a collection of data from said experiment and some conclusions on the matter/subject.

2

State of the art and related works

Contents

2.1 Geometry Friends 9
2.2 Cooperation in videogames 9
2.3 Procedural Content Generation 16
2.4 Procedural Content Generation for Cooperative Games 21
2.5 Monte Carlo Tree Search 23

In this chapter we will present all the related work for of both cooperation in video games and PCG. We will start be presenting the research for cooperation, followed by PCG and then presenting an important work for the conjunction of both subjects, Procedural content generation for cooperative games.

2.1 Geometry Friends

The game that was chosen to develop and test our level generator was Geometry Friends. It is a game that can be cooperative driven or single-player driven, this way we can tests if the levels are made for cooperation or not. It is developer friendly, it already have an array of tools at our disposal and even the source code. And for testing proposes, it has a built in level creation tool.

GF is a physics base platform game where the player tries to catch gems spread across a level. The games ends when all gems are caught. There are two different characters that can chose by a player, a green rectangle or a yellow circle. Each character possesses different abilities that are used to solve the levels. The green rectangle can stretch and shrink to reach high places or pass through small spaces. The rectangle cannot jump. On the other hand, circle has the ability to jump. The levels can be composed by black solid platforms, where no character can pass through. Green platforms where only the rectangle character can pass and yellow platforms where the circle can pass. The game can be played solo, choosing a character to play or it can be played with a friend where which player controls one of the characters and they cooperate to finish the level. When two players play together they have to combine their characters abilities to solve the most difficult challenges. For example, the rectangle can stretch and the circle jumps on top of the rectangle and uses the extra height to reach a place previously unreachable. In Fig 2.1 we have an example of this interaction and what the game looks like, showing the two characters (green and yellow), the platforms in black colors and the purple gems.

2.2 Cooperation in videogames

Cooperation has always existed outside videogames, in sports, board games and even in our daily lives. The act of working together to achieve an objective is what cooperation is all about, so it was a logic step to be translated to videogames. As Rafael et al. [10] stated, cooperation in videogames is where players work together to achieve their goals, it could be an implicit game objective or something personal to the players, where they create their own goals. Cooperative videogames date as early as the 70's decade. One of the first recorded cooperative driven games was an arcade game Fire Truck [11], where a player controls the truck and another player uses the hose to put out the fires, games like this helped shape the cooperative driven genre in arcade games and further more the videogames in general. More and more arcade games follow the rule of incorporating cooperation and even competition (the opposite

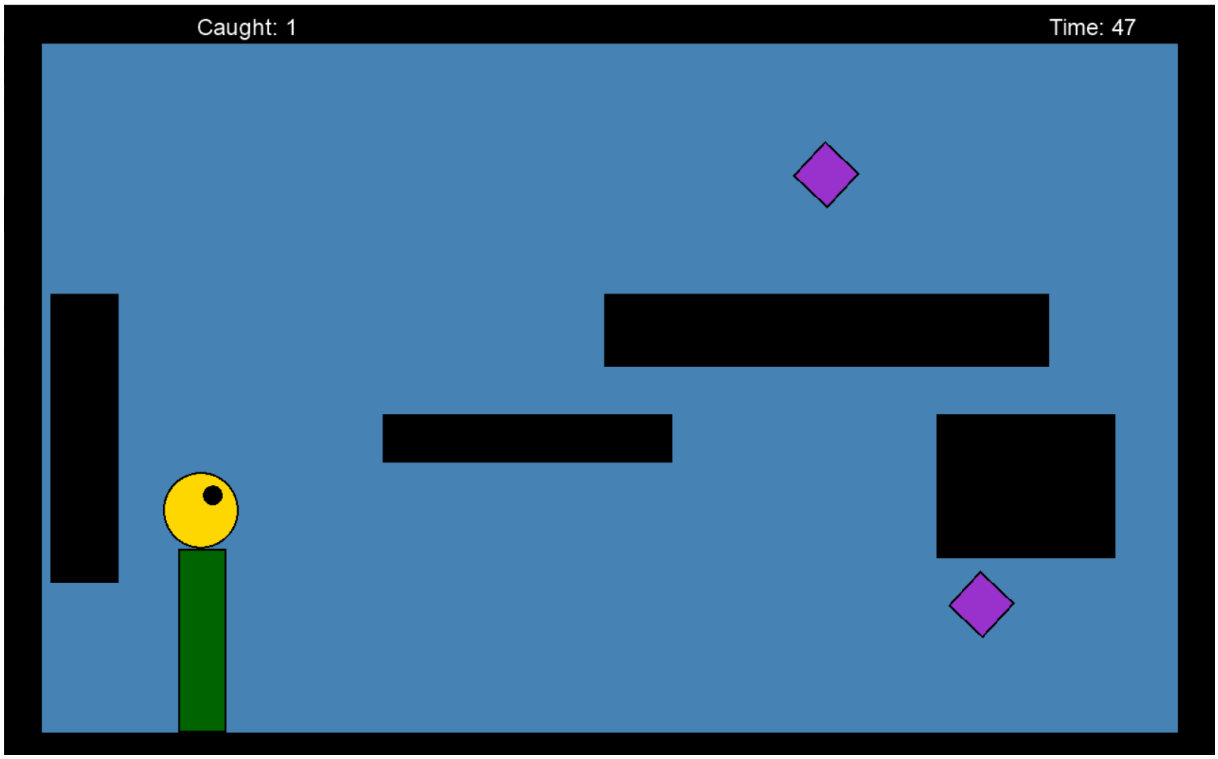


Figure 2.1: Example of a level in Geometry Friends, where the circle uses the rectangle to reach a platform.

of cooperation) in their games. Games like Time Crisis [12] and House of the Dead [13] are perfect examples of cooperation in arcade games, where two arcade machines are connected and two players help each other passing the levels.

After the big boom of arcade games, games started to settle down on consoles and with it cooperative games accompanied this change. Games started to be played on a television and instead of using two big arcade machines, players used two controllers. Games like contra and Double dragon This open the possibility of extending cooperative games to more than two players. Another big change factor for cooperative games was the introduction of the Internet. Now players do not even need to be in the same room to play together, this allowed for more opportunities and possibilities when developing cooperative games. Massively Multiplayer Online Role-playing Game (MMORPG) and Multiplayer Online Battle Arena (MOBA) are examples of a genres born from the integration of Internet to cooperative games. Some of the most popular games today have some sort of cooperation, or a mixture of cooperation and competition at the center of their gameplay. take the example of League of Legends [14] one of the most played games today. Where five players work as a team to defeat other five players. The best coordinated team wins the game.

We already defined what is cooperation, but we did not mention how can we translate that to games. In this case, the various types of cooperation are defined as designs patterns. As described in [15]

by Rocha et al.. A design pattern for cooperation is a guideline to game developers to help in the construction of cooperative games and challenges. This work was further extended by El-Nasr et al. in [16] where they also defined patterns to evaluate cooperative games. We will use this Patterns to: first helps us choose a game to create our solution and second to help us evaluate the cooperative challenges.

2.2.1 Evaluating Cooperative games

As was said above we will focus on the work done by Rocha et al.. Their work will show what type of cooperative challenges that exist and help identify what type of cooperative that we will encounter in our work. The Design Pattern are the following:

Complementary - as stated in [15], this is the most common Design Pattern used. Players control different characters with different abilities but they complement each other. In Geometry Friends, the rectangle can not jump, but the circle can, so the circle jumps to places the rectangle can not reach. On the other hand, there are some tight places the circle can not pass through, but the rectangle can. This way they complement each other and they need to cooperate to end the level.

Synergies between abilities - when the abilities of a character affect the abilities of an other character. This can be seen in League of Legends [14] where the character Cassiopeia has an abilities called *Twin Fangs* where it gains a bonus damage on poison characters, if a character like Singed or Teemo inflict poison on an enemy, Cassiopeia receives the bonus damage. In Geometry Friends if the circle jumps as the rectangle makes itself taller, the circle will be propelled higher than it normally would.

Abilities that can only be used on another player - an example can be seen in Ovewatch [17] where Mercy can only heal or give a damage boost to her teammates, as we can see in the figures 2.2(a) and 2.2(b) .

(a) Merci healing an ally. Source: Overwatch [17]

(b) Merci Giving a damage boost to a ally. source: Overwatch [17]

Figure 2.2: Mercy's abilities that can only be used on another player.

Shared goals - everyone has the same goals and they can be achieved by working together. These types of goals are designed to be done in group and trying to complete them alone is either impossible, incredibly difficult or time consuming. Players in World of Warcraft can do quests alone or join a group to save time. In the cooperative portion of Portal 2 [8], two players have to work together to reach the end of the level, the same goal.

Synergies between goals (Interlaced/Intertwined goals) - in this pattern the players have different goals but they are connected somehow forcing them to cooperate.

Special Rules for Players of the same Team - this type of pattern often appears in competitive games, where we have an ability we can use it offensively against the adversary team or defensively to help our team. An example is the character Ana from the game Overwatch where she has two abilities with the Design Pattern: The sniper shot - if it hits an teammate it will heal him and if hits an enemy it will deal damage; and her Biotic Grenade - if used on an ally he will be healed, if used on an enemy, he cannot be healed for a specific amount of time.

These patterns were later expanded by El-nasr et al. in [16], where they analyzed 14 cooperative driven PC games. Their initial research included 215 games, although it was narrowed down to the 14 games because the majority of games included some sort of competition and for a more precise

analyses they decided to only play cooperative games. In the end, they came defined the following additional pattern, which will be described and given examples.

Camera Settings - they identified three design choices for camera settings in the shared screen co-operative games:

- Split screen horizontally or vertically - commonly used in living rooms, where the screen is split in two, three or four parts, depending on the number of players. We can see in games like Resident Evil 5 [18] and The Lego Movie Videogame [19]. An example can be seen in figure 2.3;
- One character in focus - where the camera focus on one character, as seen in figure 2.4;
- All characters are in focus - where we have numerous characters on screen and the camera only moves unless everyone moves together and are close. Games like Magicka (figure 2.5) and Metal Slug [20] are good examples of games that use this type of camera configuration.

Figure 2.3: Rocket league's [1] 2 players split screen[1].

Figure 2.4: GTA: San Andreas's [2] one character focus camera[2].

Figure 2.5: Magicka's [3] all characters are in focus[3].

[1] Taken form: https://i.ytimg.com/vi/qrdSyiPeHKo/maxresdefault.jpg - Accessed at 2/12/2016
[2] Taken form: https://i.ytimg.com/vi/3HEY1svFQIA/maxresdefault.jpg - Accessed at 2/12/2016
[3] Taken form: https://content.paradoxplaza.com/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/m/a/magicka2_16_3.png - Accessed at 2/12/2016

Interacting with the same object - players can be provided with objects that can be manipulated by their abilities. Good examples of this type of mechanics can be seen in World of Warcraft. To enter the raid Zul'Aman a group of players must ring a gong in almost harmony, or in Beautiful Katamari [21] where players have to share a ball.

Shared puzzles - this pattern is similar to Shared goals, however it focuses on cooperative design puzzles. Geometry Friends and Portal 2 are good examples of this pattern.

Shared characters - the players are provided with a shared character with special abilities that players can control, this way players are confronted with a dilemma of which player controls the character and how it will be shared among them. The tanks and other machinery in Metal Slug give a significant power boost to the player who uses it, although only one player can control the tanks. While doing our research we stumble upon a curious example, in the game Heroes of the Storm [4] there is a character named Cho'Gall. Cho'Gall must be controlled by two players, one player controls Cho and the other Gall. They are given two sets of different abilities, the player who controls Cho controls the movement and the melee aspect of the character and the other player controls the ranged aspect, both "characters" can be seen in the following figures 2.6 and 2.7. This example is the truest form of shared characters.

Figure 2.6: Cho ability set. Cho and Gal are the same character, Cho'Gal. Source: Heroes of the storm [4]

Figure 2.7: Gal ability set. Cho and Gal are the same character, Cho'Gal. Source: Heroes of the storm [4]

Special characters targeting the lone wolf - this pattern was design to target players who refuse to work as a team. It encourage players to stick and work together to minimize the effects of this type of NPCs. The Hunter, Smoker, Jockey and Charger of Left 4 Dead 2 [22] are prefect examples of NPCs that are extremely deadly to a lone wolf player but are easily dealt as a team.

Vocalization - this pattern utilizes automatic vocal expressions to give all sort of information to the other players on their team. This is very common in First Person Shooters (FPS) and other tactical

games. Games like Killing Floor 2 and Overwatch have an array of pre-recorded messages to display tactical information or simply express emotion.

Limited resources - when the number of resources are scarce, players tend to share and exchange them. This increases their chance of success of completing the objective. This pattern is regularly used in survival games, where players can share some kind of resources, like in Killing Floor 2, players can share money or weapons with their teammates.

2.2.2 Evaluating Cooperation

Beside expanding the cooperative design patterns, El-Nasr et al. developed a set of Cooperative Performance Metrics (CPMs). These metrics are used to evaluate the cooperation in games, if the game, is in fact, promoting cooperation and if it is fun and rewarding.

They started by defining the metric *Laugh and excitement together*, this happens when the participants laugh or express happiness and excitement at the same time, in a particular situation. To use this metric there is a rule imposed by El-Nasr et al., it can only be used once per cause in events that are happening in the same space. Next they presented the *Worked out strategies* metric. It was identified when players shared solution and make decisions together on how to solve a shared challenge.

Helping is a metric used when there is a significant gap between the skill of the players. It happens when a more experience player teaches the novice, it could be about the controllers, or the veteran player leading the others players through the game. It is important to note that *Helping* and *Worked out strategies* are two distinct metrics. *Helping* is when a player helps another player and *Worked out strategies* is when they help each other. To complement the *Helping* metric they created the *Waited for each other*. Once again, when there is a gap between skills, the experienced players waits for the novice to catch up. *Global Strategies* is a metric where they refer to events when players assume different roles and positions during gameplay, this way they complement each other. The last metric is *Got in each others' way*. This happens when the players' opinions differs from each other and they want to take different approach or actions, ultimately interfering with each others goals.

These metrics are important for evaluating cooperative challenges and enable the evaluation of the player's reactions. CPMs are made for evaluating cooperation based on human observation, because of this we cannot use them directly in out generator, however we will use them in our experiments to evaluate the generated levels, because they are an important tool to evaluate the reaction of players and draw crucial conclusions.

2.3 Procedural Content Generation

Procedural content generation has come a long way. A lot as changed over the years, on the other hand the definition stayed almost the same. We will use the definition provided by Shaker et al. in the book [23] in the first chapter [24], thus quoting Shaker et al.: "PCG is the algorithmic creation of game content with limited or indirect user input. In other words, PCG refers to computer software that can create game content on its own, or together with one or many human players or designers." The first use of PCG in games can be traced to the games like Elite [25] and Rogue [26]. Rogue provides us with a excellent example of classic PCG. Every time we start a new game, the levels are randomly generated, normally creating dungeons like levels. These types of games came to be known as *roguelike* games, thus spawning a whole new genre. Games like Binding of Isacc [7] and Rogue Legacy [27] are clear cases of games that took Rogue as inspiration and falling into the category of *roguelike*. On the other hand, Elite took a different approach, it used a group of seeds numbers to later procedurally generate the galaxies and planets. It generate exactly eight galaxies each with 256 planets. All these games (Elite, Rogue and other games from 1980 era) needed PCG, not only with the propose of creating new content and keeping the game fresh, but because of storage limitations from computers and disks alike. This was the predominant reason for the creation and usage of PCG in games. As computers and storage continued to grow, the focus of PCG shifted mainly to the generation of content with the intent of keeping games fresh and increasing their replayability.

As was said before, our work is focused on creating a PCG for cooperative games. Before we started developing our solution, a thorough research was conducted to help identify the most important aspect of PCG, what problems we could face and what solutions exist. Taking all these into consideration, in this chapter we will present the state of the art for PCG and the research done prior to the development of the solution.

Every year there is a clear progression in this area, being it academic or commercial (in games). One of the most resent works is the book by Shaker et al. [23] on Procedural Content Generation in games, that describes in detail the current state of PCG. This book is almost a compilation of their previous works and a lot more.

Nevertheless, we need to define some proprieties of a PCG solution, to better help us conclude if our PCG is suitable or not. In chapter 1 of [23] they list the desirable properties of a PCG solution. These proprieties allow us to evaluate a generator and what kind of solutions can be built. As we will see, there are some tradeoffs in the properties we need to take into account.

- *Speed*- this represents the time that it takes to generate a solution. It can vary from a couple of seconds to entire months depending on what are we generating. In our case the generation will be pre-computed, this means it will be during the level design, so the speed of generation can go

from a few seconds to two or three minutes;

- *Reliability*- if the generator satisfies our needs and quality criteria. Depending on the content the generator is creating, this can be important or not. For example generating a dungeon. If the dungeon does not have an exit, this ruin the gameplay experience. In our tool this will be verified in conjunction with the cooperation metrics and design pattern, if our generator fails to create a level with our requirements, it is considered a failure.
- *Controllability*- if the content to be generated can be controlled some how or some aspects can be modified. Some generator do not need this propriety, other have some specifications that vary from generation to generation.
- *Expressivity and diversity*- when we need to generate an array of content, we want that content to be diverse and different from one another, so the generator must produce diverse content. If our generator fails to meet this criteria it will be considered a failure, since we want players to be interested and invested on our levels.
- *Creativity and believability*- this represents if the produced content can be distinguished between procedural generated content and man-made content. In most cases, the content should look like it was created by a human rather than a procedural content generator. We do not intend to explore this propriety, our generator will not have any kind of believability tests.

These sets of proprieties are important to clarify, due to the fact that they help define what we want in a PCG and keeping it true to the original solution.

On the other hand, there are various methods of creation that exist and problems that occur when devising a PCG. A taxonomy was developed to guide the thought process behind the creation of PCG. Was design by Togelius et al. [28] that was later revised in the book [23] in the first chapter [24]. Now, we will go through all giving a brief description.

Online versus offline - the generation can occur during gameplay (online) or be pre-computed (offline). The generation during gameplay can be used to create endless variations of the game or level, making the game, almost, infinitely replayable or generate player-adapted content (the content that will be generated will be adapted to the current playthrough. For example as the game is progressing it will be more difficult). As for offline, this can be done during game development or before a game session. Concerning our solution, the PCG will be offline. The level generation will be done during level design or prior to a gaming session were players want a cooperation driven experience.

Necessary versus optional - this is referring to the content that is being produced. Content can be necessary, where the content is vital to the level completion. Or optional content, where the gener-

ated content can be discard or exchanged for other content. For example, games like Borderlands [29], use PCG to generate guns. This is optional content, since the player can always change, sell or trade weapons, making one gun generated not vital to complete the game, because it can always be exchanged by another. For our solution, the content produced it is necessary, because we will be producing levels, and of course they are essential to the game.

Degree and dimensions of control - if a PCG has some sort of controlability and at what degree (see *Controllability* page: 17).

Generic versus adaptive - a generic content generation happens when the content is produced without taking the players actions and behavior into consideration. In adaptive content generation, it is the opposite. Content is created based on player's previous behavior, creating a player-centered experience and the level can be adapted to the player. We are focused on a generic approach, as our solution will not take into consideration the player behavior. An adaptive solution could be done in future work.

Stochastic versus deterministic - in deterministic solutions, given the same start point and parameters, it will generate the same content every time. This can be seen in the original Elite [25], where the galaxies are always generated in the same place and future regeneration will have the same result. In stochastic PCG the same content can only be generated once, if we try to use the same starting point and parameters it will result in a different content. Our PCG will be stochastic, in each generation we want different results and new levels.

Constructive versus generate-and-test - generate-and-test is done in three stages, the first one is generate the content; the second is test the content, if the test gives good results we stop the generation; and the third is repeating the loop until a suitable solution is produced. In constructive PCG, the content that is produced is the one that is used. On a side note, generate-and-test is the type of solution that would be best suited for the generation of cooperative levels, because the content must always be tested. We must insure that the level has cooperation in it and therefore is classified as cooperative. And the focus of our document are PCGs that are generate-and-test, because in our generator every solution that is generated will be tested by a simulator.

Automatic generation versus mixed authorship - the main difference between these two paradigms is cooperation between humans and PCG. In Automatic generation, the computer generates the content alone, where in mixed authorship, while the content is being generated, the designer or player is incorporating his ideas into to the content and modifying the final result. This way the product will be something that was created by machine and human alike. Our work fits in mixed authorship, because

the user needs to create the platforms and the generator places them. It is a cooperative work between human and machine.

These works are important to help define and design a solution for a PCG problem and should be used as first steps to design and define any PCG tool. Furthermore, they are important to us since we will use these properties to help us develop a solution fitting for a PCG. As for PCG algorithms and solutions there is a vast list and it is almost impossible to cover them all, nevertheless we will present some recent works and studies that have been made.

Togelius et al. in [30] defined what PCG is, what it is not and what it might be. This is one of his works that led to the creation of the previous mention book [23]. In this article Togelius et al. explained that offline and online player-created content it is not considered PCG. When they talk about offline player-created content they are talking map, level, characters and other kind of editors. Online player-created content refers to strategy games and ”God” games. The example provided are SimCity games [31] and The Sims Series games [32] where the main feature of the game is creating content. There are no clear objectives only creating families and houses, in The Sims case, or building and maintaining a city, as it happens in SimCity. They also said that PCG can be random or not (already mentioned in *Stochastic versus deterministic* section). The last subject they talked about was if it is adaptive or not, once again this topic was already discussed previously in this document (see *Generic versus adaptive*). However the main focus of this paper is the approach they take on creating a level generator for Infinite Mario Bros, a open-source clone of Super Mario Bros. [33]. Togelius et al. decided to developed two versions to attack this problem, an online and offline version. As we said above the difference is, if it is producing content during or prior to gameplay. The offline version started by letting the player play a simple level that was randomly generated using the level generator of Infinite Mario Bros. The level was almost flat and with some gaps to create some diversity in the gameplay. Then the player played the level and his actions are recorded. The level generation starts with a copy of the level played and then it will go through the recorded actions and modifies the level according to those action. For each action there is a simple set of rules that help balance and determinate how to modify the level, actions like Jump button pressed and release determines the ground modification and the creation of gaps and platforms. The speed/fire button pressed balance the enemy variety. The enemy that is placed is chosen at random. All modification are tested and corrected if necessary to control the level consistency. Summing up the generation process: a player plays the level, every action is recorded and a new level is generated; this cycle can continue as long as the player wishes, generating progressively more complex levels.

The second version, the online version is very similar to the offline one. The main difference is that the level generation is happening during gameplay, the level is being created as the player is progressing. This version works the same as the last one, but the generation is shorter, this means that instead of

generating a complete level it only generates the next section. To help accommodate the change two more rules were added; Coins collected, when coins are collected an enemy is spawned; and enemy stomped, when enemies are stomped, coins spawn. Both version were tested using volunteers, and they came to the conclusion that players preferred the offline version, as they concluded that the online version was harder to play, the unpredictability and events happening on screen during gameplay in real time contribute for this decision. This type of works are important to give a wider view of the Procedural Content Generation world and what creative approaches can be developed.

One of the most recent works done in PCG is "Monte Carlo Tree Search to Guide Platformer Level Generation" done by Summerville et al. [34]. In this work Summerville et al. combined Markov Chain Generation and Monte Carlo Tree Search, resulting in a new algorithm that they named Markov Chain Monte Carlo Tree Search (MCMCTS). Markov Chain [35] is an algorithm that given the present state, one can make predictions for a future state independent of the past. This way a level generator was created. The Markov Chain generation of a platformer has seen two major approaches in the representation of the states, the tile-to-tile transitions and the vertical slices of levels. For their work, they used levels from the game Super Mario Bros. [33] and chose to use the vertical slices method due to the fact that this had a higher change to produce playable levels. This is the main problem when using a Markov Chain generator, there is a high probability that the level generated is unplayable, this means that the level cannot be finished by the player. To solve this problem they used MCTS. Using MCTS guarantees that the level produced is playable, this happens, because MCTS has a step that gives a score to every solution, based on heuristics that we can define. The choice of MCTS was not something random, they came to the realization that a step in MCTS (rollout step, MCTS will be fully explained later in this document) is equal to a standard Markov Chain generation. Two more things were done, the creating of *score function* and a solvability simulation. The score function was used to help chose the best level generated and it is represented by the subsequent equation:

$$\textit{score} = S + g + e + r \tag{2.1}$$

And consisted of the following features:

- S - it he level is solvable, the value for S is 0, otherwise the value is set to a large negative number(-100,000);
- g - the desirability of the level in relation of the number of gaps;
- e - the desirability of the level in relation of the number of enemies;
- r - the desirability of the level in relation of the number of rewards, such as coins or power-ups;

The last thing that was done was the solvability simulation. In order to calculate the S in the *score*

function, they had to create a simulator to solve the level, this means reaching the end of the level and proving that it is solvable. It consisted of a A* Mario controller that tries to solve the level. In this simulation the game physics are tuned down to accelerate the sampling process in MCTS. Running a full physics simulation of the game it is computationally heavy and slows the evaluation process and they simplified the process by simulating the running and jumping physics well enough to reach a good result. In correlation with our solution, we used also used a simple score system to evaluate each simulation and our solvability simulation is done trough a flood field algorithm. This work showed us that having a proper evaluating tool is important for the success of a PCG. Generating content is important, but there is always a need to maintain some source of quality control over the content that is generated. We also used this work as a guideline for our solution on developing a PCG using Monte Carlo Tree Search.

2.4 Procedural Content Generation for Cooperative Games

As right now, there is little to no work done in this area, in contrast, PCG and cooperation in games are areas quite developed, as we saw. The inspiration for this paper is the thesis written by Rafael Ramos on Procedural Content Generation for Cooperative Games [10]. This is the only work in the area since it was the only one that tried to tackle the problem of using to PCG to create cooperative driven challenges. A lot of cooperative games like Biding of Isaac: Rebirth [7] use PCG but it does not affect the cooperative gameplay.

Rafael et al. created a tool that generates cooperative driven levels to the game Geometry Friends [9]. The most important aspect of his work is that he uses PCG to define the levels/challenges and that they are in fact focused on cooperation, it was borderline impossible to find a game or tool that met this criteria. He created a tool that receives a level (or a level can be created on the spot) and it generates the gems according to the specifications done by the user. To achieve this, he divided his level generator program into various phases: the cell evaluation phase; three reachability phases(rectangle reachability, circle reachability and cooperation reachability); the gem generation phase; and the heuristic evaluation phase. Now we will follow with a quick description of each phase.

Cell evaluation phase - for this phase, the level was divided into a grid of cells. Each cells contains various information about the level (like what cells are occupied) and where the circle and rectangle can go. The first iteration is to identify where the circle and rectangle can fit. He used a method called Moore's Neighborhood [36] to check if a given cell's neighbors cells are occupied. If they are not occupied then the cell is marked as fitsRectangle or fitsCircle, depending on which one is being iterated.

Rectangle reachability phase - this phase verifies what cells can be reached by the rectangle player. The iteration begins at the rectangle's starting position, which is given a direction flag of 0. This flag can be negative or positive depending if it left (-1) or right (1). Every cells that was visited by the algorithm is flagged as reachesRectangle, this way he knows what cells can be reached by the rectangle and what cells were already visited by the algorithm. To ensure that the algorithm does not end in a infinite loop, where the rectangle falls and climbs back to a platform, two more flags were added, the traversedRectangleLeft and traversedRectangleRight. These flags represent the direction that the cells analysis is taking. For each cells, a verification is made to check if a rectangle fits there. The fitsRectangle flag is used for this. If its true then the cells can be marked as reachable (reachesRectangle flag). The next step is to see if the rectangle is falling or can fall. This is done by checking if the cell below our current cell is marked as fitsRectangle. If it is true he knows the rectangle is falling and adjacent cells we be tested to account for momentum while falling.If it is not falling, he checks the left and right cells if it has a fitsRectangle flag, if not he checks the 3 cells above that cell, this way he can account the rectangle's ability to overtake small object, using momentum. Finally he proceeds to pick the next cell, using the traversedRectangleLeft and traversedRectangleRight flags.

Circle reachability phase - the circle reachability phase is very similar to the rectangle's. The main difference it is the fact that when the circle is on the ground, he can jump. So he first checks if the circle can fit (fitsCircle flag), if it is true then the reachesCircle flag is marked. After, he verifies if the cell below the current cell has true on the fitsCircle flag. If not it means the circle is on the ground, so it can jump. To represent the jump reachability, Rafael created a variable, that represents the number of cells that can be reached by jumping in the current cell. To verify the jump arc, each subsequent cell is tested for collisions and a jump arc is created.

Cooperation (rectangle + circle) reachability phase - this phase is identical to the circle reachability phase, the only difference is when he sets the jumpStrength. Instead of setting it to the normal jump height, he set it to in a way that uses the max height of the rectangle plus the jump strength of the circle, this way he accounts for the assisted jump with the rectangle. This only occurs when the cells is reachable by the circle and the rectangle simultaneous(reachesCircle = true and reachesRectangle).

Gem generation phase - This is the final phase of the Procedural level generator. In this phase he creates three lists consisting of the zones that were determined before: the coopExclusiveCells list - containing the cells that can only be reached using cooperation between the circle and rectangle; the rectangleExclusiveCells list - containing cells that can only be reached by the rectangle character; and the circleExclusiveCells list - containing cells that can only be reached by the circle character. Prior to the level generation, the user must set the numbers of gems and if it wants a more or less cooperative

level and if it is balanced or not (the balance is determined by the number of gems that are exclusive to the rectangle or circle, so it can go either way). The cooperation and balance are the two heuristics that will impact the gem distribution. Depending on the preference of the user, the gems distribution will set the level to be more or less cooperative driven and to be more focused on the rectangle or the circle.

This concludes the review of Rafael's et al. work and the state of the art about procedural content generation for cooperative games, as can be seen, the work done on the topic is almost nonexistent, so we will take upon ourselves to extend Rafael's et al. work and expand this particular area.

2.5 Monte Carlo Tree Search

Monte Carlo Tree Search is a decision making algorithm created to find the optimal solution to a problem, by taking random samples and building a search tree [5]. We chose to talk about the algorithm, because it will be the back bone of our simulator. As it was mention MCTS is a decision making algorithm, so we will use it to make the decisions on our simulator and run the simulations. This way we can evaluate the level based on the decision made in the simulation.

One of the strengths behind MCTS is its simplicity, the algorithm builds a search tree according to the outcome of simulated playouts. Each MCTS iteration can be divided into four steps: selection, expansion, rollout and backpropagation, and they can be defined in the following way:

We present to pseudocode for the the main loop in Algorithm 2.1.

Algorithm 2.1: Main loop

```
Output: Best Sequence
while currentIterration < maxIterrations or no more combinations do
    selectedNode ← Selection(initialnode);
    reward ← Simulation(selectedNode );
    Backpropagate(selectedNode, reward );
bestNode ← get the node with the best score;
bestSequence ← get the best sequence from the bestNode;
Return bestSequence;
```

As mentioned previously, an iteration of MCTS as 4 steps and accordingly to our implementation they are defined as followed.

Selection

This step is where the next node to expand, is chosen. First, we try to select a random action, if one is selected a new node is created based on that action. Else, we will search the node with the best score. This search is used with the equation 4.1. With this search method we can define our search to

be based on exploration and exploitation. A terminal node, a node that represents a game over situation, can not be expanded. And a node must be fully expanded (all the actions done related to that node), only then his children are elected to be selected. The pseudocode for this step can be seen in Algorithm 2.2

Algorithm 2.2: Selection step

```
Output: Next node to expand
while node is not terminal do
    if node not fully expanded then
        Return Expand(node, nextAction );
    else
        node ← BestChild;
Return node;
```

Expansion

When a node is selected to be expanded, a copy of the current node will be performed. Besides that copy, the action that was chosen to in the selection step will be done, in this new newly formed node, also known as childNode. This step always returns a new node child, as seen in Algorithm 2.3.

Algorithm 2.3: Expansion step

```
Input  : parentNode, actionToBeApplied
Output: childNode
/* actionToBeApplied is an untried action from parentNode                    */
Add a new child childNode to parentNode;
State of childNode ← Result(GetState(parentNode), actionToBeApplied );
actionToBeApplied applied to childNode ← actionToBeApplied;
Return childNode;
```

Simulation

After a node is selected, either being by expansion or chosen as the best node, a new simulation will be done. This simulation is performed by doing random actions until a game over situation occurs, as seen in Algorithm 2.4 After the simulation is done, the result of that simulation will be evaluated by our evaluation algorithm and a score is given to that node.

Backpropagate

The last step of an iteration, is the Backpropagate, Algorithm 2.5. After, a node is given a reward all his parent nodes must be updated with this information. This step simply updates all the information of those nodes.

Algorithm 2.4: Simulation step

Input : Node to evaluate
Output: Reward
while $CurrentState$ is not terminal **do**
    action ← `GetRandomAction`;
    CurrentState ← `ActionResult(`CurrentState,action `)`;
Return `Reward(`CurrentState `)`;

Algorithm 2.5: Backpropagate step

Input: Node to update
while Node is not null **do**
    `N(`Node `)` ← `N(`Node `)` + 1;
    `Q(`Node `)` ← `Q(`Node `)` + `Reward(`Node, Parent of Node `)`;
    Node ← Parent of Node;

As shown in the main loop, Algorithm 2.1, these for steps will be repeated until there is no possible actions left to be done or a number of max iterations is reached. After one of these conditions is met the main loop end and a search for the best node is done and is drawn in the the level editor.

These steps are the main loop in MCTS (figure 2.8), they help to build, expand the tree and decide the best course of action to solve the problem.

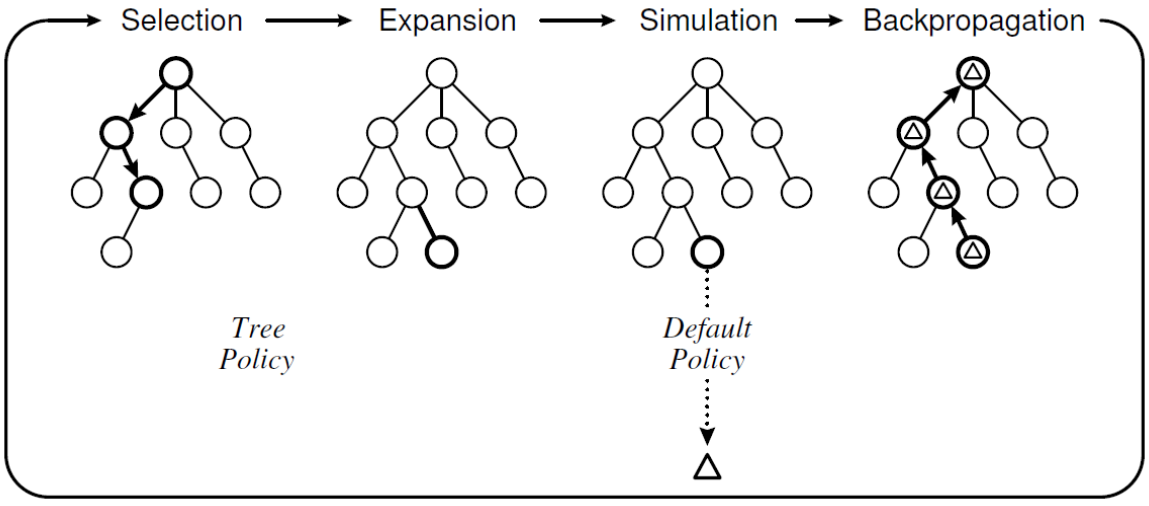


Figure 2.8: A single iteration from the algorithm. Figure taken from [5]

Another important factor in the MCTS algorithm is the evaluating function in the Selection step. The Selection step determines the growth of the tree and if we find a good or bad solution. To help balance the selection process the concept of Upper Confidence Bound 1 applied to trees (UCT) was introduced [37]. As stated by Magnuson in [38], UCT as the propose of balancing the concepts of

exploration and exploitation. A algorithm heavy on exploration, will choose nodes that are less visited, this way we ensure that we do not ignore areas of the tree that can possibly give good results. If the algorithm is heavy on exploitation, only the nodes that give the best value will be picked, avoiding a greedy approach. UCT tries to balance out these two concepts by giving unexplored nodes a boost. The evaluation function is represented by the following equation:

$$UCT(n) = u_i + C\sqrt{\frac{\ln N}{n_i}} \tag{2.2}$$

Where:

- u_i = estimated value for a node i, which is represented by the number of simulation that resulted in a win state divided by the number of simulation done;
- C = exploration factor, usually $\sqrt{2}$;
- N = total number of simulations done by the parent of node i;
- n_i = total number of simulations done in the node i.

MCTS it is a great algorithm to use for our work, because it can be used without knowledge of the game except its rules and restrictions. It does not need any type of strategic or logic background to create a solution to a problem. Therefore, with slight tuning our simulator can be used for various games. Another enticing advantage, stated in [5] the decision making process is, in some ways, identical to the process used by humans. It will always focus on better sequence of actions, but occasionally will check the weaker ones in search of better results.

Another decision making algorithm was considered, Goal-Oriented Action Planning (GOAP), where sequence of actions are created to meet the goals that need to be reached and a discontentment value is associated to indicate if the goal is reached. As stated in the book [39] one of the major problems with GOAP is the scalability. The algorithm scales almost exponentially by the number of actions that exists. This means that if a game is slightly more complex the algorithm will scale uncontrollably. And eventually it will need to consider all combinations of actions to reach the best one, this brute-force approach would ruin any PCG, if the evaluation phase takes to much time and does not scale properly it will create a bottle neck and delay all other actions.

2.5.1 The simulation

Another work that we want to acknowledge is a simulator done by Fisher et al. [40] where they use MCTS to simulate a car in a racing game [41]. This article is important to us because it will helps devise our simulator and avoid mistakes that were already detected. The objective of this article is to create

an artificial intelligent agent to control the car, using MCTS. The TORCS [41] competition only allows the access to some data during the race, one can only know the car's position at any given time, as they stated, this is not sufficient to create a prediction of the actions. Fortunately, each participant has a chance to do an warm-up lap and gather as much data possible. They use the warm-up to record sensor data and construct a model of the track, which will be used for MCTS. To make the simulation work, MCTS needs two things: a world model that is capable of predicting the actions result; and an evaluation function to decide the desirability of a given game state. They used three simple actions for the MCTS, accelerating, braking and steering. Each action is conditioned to a interval, to avert sudden changes in the action and reduce the tree branching as much as possible. Instead of making a drastic 90 degree turn, he will make soft steering changes. The method in question is also applied to the acceleration and breaking actions, to avoid drastically changes of speed. The world model is partially solved in the warm-up phase as explained above. This model is mapped out in a Euclidean space [42] to better calculate the predictions based on the actions and basic physics. They summarize the main loop in the following way:

1. Observe the sensor space and create a state *s*;
2. Transform the sensor space state *s* into an Euclidean space σ;
3. Run MCTS with the new model σ;
4. Transform σ back to sensor space, into a new state *s'*;
5. Use the new state *s'* with the evaluation function.

As far as the evaluating function work, they had to devise an end condition, since the goal of the game is to drive as fast as possible with unlimited number of laps. Alternatively, they use a fail terminal states to evaluate the desirability of a given state. A unsuccessful state happens when the car exits the track.

$$k = \frac{d - d_0}{\frac{1}{2}a_{max}t^2 + v_0 t} \tag{2.3}$$

$$k_\perp = k^2 * P \tag{2.4}$$

To calculate the function they take into consideration the distance that was traveled *d*, the distance along the track at the root of the search d_0. $\frac{1}{2}a_{max}t^2 + v_0 t$ represents the equation of motion an *k* the value of a non-terminal state. $k_\perp$ is the value of a terminal state and *P* represents a penalty factor, between 0 and 1. They concluded that the MCTS algorithm is a viable approach to racing simulators and handle the track well. They also used a different approach, instead of using the tree to search for good terminal states they search undesirable stats and remove them those actions.

We can conclude that to devise a MCTS simulation we will need a world model to represent the action's predictions. And that MCTS is a good choice the run short or longer simulation. As it was referenced in [5] [38] [40] the longer the algorithm runs the better will be the result.

3

Implementation

Contents

3.1 Technologies used 31
3.2 Solution overview 31
3.3 Level editor 33
3.4 Level generator algorithm implementation 34

In this chapter, we will approach the solution to the problem presented in the Introduction chapter. We will start by presenting the technologies that were used to develop the tool.Then, we will introduce an overview of our solution followed by our final approach to our problem.

3.1 Technologies used

Several technologies and programming languages were considered to help us develop our solution. The main decision that was pondered was the programming language, there were two choices, based on our knowledge of them and the best suitable to develop a tool of this kind. The choices were JAVA and C#. We chose the latter. JAVA and C# are programming languages that tackle the same paradigms and are both object oriented. The winning factor in favor of C# language was the compiler Visual Studio and its built in functionality, Windows Form Application (WFA). Windows Form Application is a GUI (Graphical user interface) that uses C# and gives programmer a rich client to create applications. Using WFA, creating the design for our level editor was much simpler than creating an user interface using JAVA. This way we could focus on the core of the development. We also used XML to be able to import levels to the level editor and export them. The choice of XML was due to the Geometry Friend levels being in XML and this way we could test and play all levels.

3.2 Solution overview

To better explain our thought process and solution, we will explain everything with an analogy. MCTS is normally used to create Artificial Intelligence (AI) agents to solve and play games. Thus, this is what we did, we created an agent that played a game. In this case, the game is to create a level using all the platforms that we give to him. To win the game, the level must be cooperative and playable.

To integrate this agent, we develop a program, a level editor where we could simply create a level by our selfs or with the help of our agent. With this editor we can import and export levels to XML and acts as an evaluation tool.

To develop a MCTS we need to define 4 properties:

1. The actions - what actions does the algorithm do to construct the level. When an action is executed a new node/state is created;

2. A deterministic playing field - knowing the exact result of doing an action and the exact state of the playing world;

3. The game over condition - when does the game ends. It could be a win or a loss;

4. The winning condition - when a game ends we also need to distinguish if it ended on a win or a loss.

    (a) Evaluation method - encapsulated in the winning condition, we need a method to evaluate the result of all actions that were done to reach the game over. This will evaluate the winning condition and determine its score.

In relation to our solution, these properties were defined in the following manner. We define our actions as a simple action that could be used in different manners. Meaning that the action was a simple ”put platform in X and Y” and its variations was the X and Y coordinates in the level. Each action can be done once per platform.

Next we had to defined a deterministic world. Because, of the nature of GF on not having circular platforms and shapes, besides the circle player, we divided the playing field into squares. Each square represents a coordinate, and the top left square is (0,0). Having a coordinate based world, all actions can be predicted and tracked.

The next decision that we had to make was the winning condition. The principal question to answer was, when does the the game ends? In our case, when does the level creation ends? We decide that we can only use the platforms once per algorithm iteration, meaning that an iteration ends when there are no more platforms to place in the level. Thus, the ”game” ends when all platforms are placed in the playing field, independently if the result level is playable or not.

Now we know when the game ends, however we need the distinguish if our agent won or lost the game. As any game, there are several winning/loss conditions, and inside the winning condition we also have a score, meaning that our agent can win with a better or worse score.

To evaluate the levels and give a score to them, we need an evaluation method. We used the evaluation method developed by Rafael et al. in [10]. We used the base structure and implementation of his work, however we changed the final score system to contemplate our needs. The base idea remains the same, however, we added a new level of evaluation, if it is balanced. We decided that the base of a balanced level is a level that has at least a gem in each evaluation area. Meaning, it needs a gem in the circle exclusive area, a gem in the rectangle exclusive area and one in the cooperation exclusive area. This way we can guarantee that the players need to cooperate and have objectives of their own, creating an equilibrium between both players.

To complement our evaluation method developed a score system, composed by 4 points:

- A level is scored with 0 if it is not playable. This happens when the level that was produced does not have an end. This is our loss condition.
- A level with 1 point, means that it is at least playable. This is first of our winning conditions. A level with this score can be played, however it does not required cooperation and coordination to

be finished. Consequently it is not balanced.

- Having 2 points, means that it is playable and requires cooperation to be finished. This type of levels cannot be beaten without the use of cooperation. However, it is not guarantied that the level is balanced.
- The last score, is 3 points. Levels evaluated with 3 points are playable, have cooperation and are balanced. This is the perfect score for a level and the most difficult to obtain. Having all these requisites requires the perfect combination of gems placement, platforms placement and shape. Not all inputs can result in cooperative levels or balanced levels.

With all these points defined we were ready to develop our tool. However, we need a program to support our level generator, with this in mind we concluded that we need a level editor.

3.3 Level editor

The first implementation of our project was the level editor. Having a visual representation of the levels and a graphical interface helps the process of testing the levels creation, evaluation and generation.

The level editor, which can be seen in Fig. 3.1, is composed of various components that can be used to test the generation algorithm or simply create new levels to Geometry Friends.

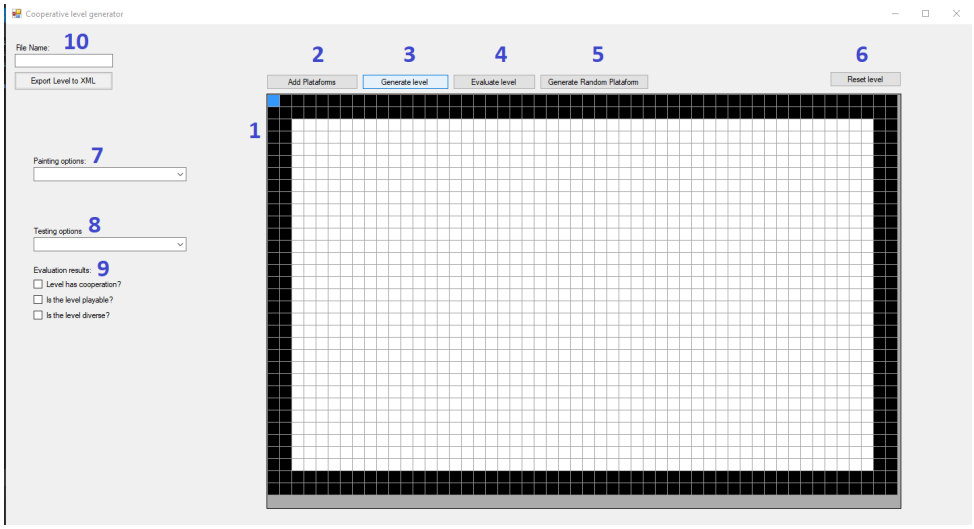


Figure 3.1: The level editor. The numbers in this figure represent the various components of the editor.

The first big component is the grid that represents the game field, marked as '1' in the Fig. 3.1. Each cell of this grid can be painted black that represents a platform, or we can chose to place a gem or one of the playable characters. There is no limit to the number of gems in the field, however only one of each characters can be placed. To chose what we want to "paint", there is a dropdown box where we can pick

these options, marked as ’7’. The grid is also surrounded by platforms that illustrates the limits of the level.

As mentioned before, we can use the editor to create levels from scratch or use the PCG component of the tool to generate a new level. Regardless of these options we can always evaluate the levels, by pressing the button ’4’ and we can chose the type of information we want to see in the grid, by selecting a option in the field ’8’. These option as similar as the ones presented in Rafael’s work: show exclusive areas, show cooperation areas and show where the characters fit. The last field and information about the evaluation of the levels is in field ’9’ that represents the evaluation result. As mention in the section about the solution overview, a level can be cooperative or not, playable or balanced/diverse, when the evaluation is complete the checkboxs beside these question can be checked or not. If it is check it means the level has that property.

To generate a level, we first need to chose what platforms we want to have in our final level. To do that we click in ”Add Platforms” button (marked as ’2’), a new window will appear where we can create a new platform. After, we are satisfied with our platforms we can click in button number ’3’ (Generate level) and after 1 minute the generated level will appear in the grid.

Subsequently, we can export the level to XML format, in the field number ’10’, to be read by the GF game. If we do not like the level or simply just want to restart the creation process, we can press button ’6’ to reset and clean the grid.

The last button, button number ’5’, is a random platform generator, for when the level designer is out of ideas. It simply generate a random platforms without any logic.

This concludes the section about the level editor. As explained, this tool can be used as a simple level editor/creator, it can be used to evaluate levels and generate completely new levels. Furthermore, all these tools can be used at the same time. One can create a level, add new platforms to be integrated to the level by the generation tool and in the end it can be evaluated.

3.4 Level generator algorithm implementation

The idea of the first implementation was to develop the base structure of the generating tool that could be used as a testing ground and use that as a starting point to expand, improve and polish the whole algorithm.

As mentioned previously, the back bone of our procedural generation tool is the algorithm Monte Carlo Tree Search, having this in mind and the properties explain in the Section 3.2, we devised some base structures to represent the game objects and information holders.

- Cell - it represents a cell in the grid. It holds information about the position of the cell, what is in the cell at that moment, if it is a platform, a starting position of one of the characters or a gem. It also

has the information about its own evaluation, if the circle and/or square can fit, if it is a reachable cell, the information about the exclusiveness areas and the circle jump information, all that was described in Rafael's et al. work.

- Grid - it is the group of cells that represents the game world. It is a table of 33 per 52 cells (33 lines and 52 columns). It also has the information about the gems, circle and square location.
- Map Division - A Map Division, is a 5 cells by 5 cells square. This way we can divide the whole grid into Map Divisions. Thus, we changed the meaning of action. Instead of selecting a random coordinate (cell), now we select a random Map Division. With this change we reduced the ranged and number of actions drastically, from 1595 possible actions to 63. And only one Map Division can be used per platform. With this change, we improved the algorithm's performance, meaning in the same run time we could had better results.
- Action - it holds the information of the coordinates a certain platform will have preforming that action.
- World Model - it represents a state of the world. It holds a grid with all the actions performed, until the state. And all the actions that can be done in the future.
- Reward - represents the reward level of a action. After the world model evaluation a Reward (score) is given.
- MCTSNode - represents a node in the MCTS algorithm. It holds all the information about that node, including:
  - The current world model;
  - The parent node;
  - The selected action to preform;
  - All the children nodes;
  - The Monte Carlo values, such as the N (total number of simulations done in the node) and Q (total of accumulative rewards in that node).

With these structures and classes we built the core of the algorithm. The algorithm starts with a set set of actions for each platform and each row of the tree represents all the actions for that platform. The generator will pass through all steps of MCTS (the ones explained in Chapter 2.5), going through the loop several times until a max number of iterations are made and returning a level.

Having the generated level, one can modify it, evaluate it and export to XML and play it.

4

Experiment and Results

Contents

4.1 The experiment preparation 39
4.2 The experiment 40
4.3 Experimental results 40
4.4 Result significance 51

In this chapter, we will describe how we evaluated our tool, how we prepared the experiments, how the experiments were conducted and what results we gathered.

As mention previously in the document, the main focus of our work is to develop a tool that generate cooperative levels, to test this we generated several levels, mixed them with levels that were not cooperative and asked random people to test the levels and to try to differentiate the cooperative levels from the non-cooperative.

4.1 The experiment preparation

For this experiment, nine levels were used, six of them were generated. Three of them by us and the other three were created by fellow colleagues (all using the tool we developed). This way we had a series of diverse levels, created by different people and, in a simple way, we tested that our tool can be used by anyone. We add three more levels to be tested, however these levels were created by a person, instead of being procedural generated. The levels not procedural generated were chosen from the list of already created levels in GF.

All the levels were tested and ranked according to difficulty and distributed by three ”Worlds”. A World is a combination of two levels generated by our program and a non generated level. For testing proposes we used three Worlds, World number one was the easiest and number three the hardest. Since the majority of the play testers never played GF before and because it is a physics based game, it was important to have some sort of difficulty progression. If they started playing the harder levels in the beginning they would become frustrated and this could adulterate the final results.

To determine their difficulty, every level was played and then ranked. This ranking consisted in: the number of cooperative challenges and the skill necessary to complete each challenge and the entire level. Levels where the jump precision and coordination was greater were assigned to World 3. On the contrary, levels where the challenges were more forgiven and did not require so much personal skill were assigned to World 1. Players were not obliged to play the all the worlds and could chose to play only World 1. However some players, after they played the first world, liked the game and the challenge so much that they opted to play all three worlds or jump to World three in search for a greater challenge.

To complete the experiment preparations, a questionnaire was created. The purpose of this questionnaire is to know if the players could distinguish if a level was cooperative or not, balanced or unbalanced and identify what levels were created by a human or by the tool. The latter does not have any scientific purpose, because we did not include any heuristics to verify if the levels could pass as human made. This questionnaire can be consulted in Appendix A. The Table 4.1 shows the results of the level evaluation and how our tool classified when their were generated. All the levels that were used in this experiment can be seen in Appendix C.

Table 4.1: World classifications in terms of cooperation and balance.

	World 1			World 2			World 3		
Levels	1	2	3	1	2	3	1	2	3
Has cooperation?	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Is Balanced?	No	Yes	No	Yes	No	No	Yes	No	No

4.2 The experiment

After all the preparations were made, we were ready to test the levels. We had the opportunity to make our tests at MOJO.

MOJO is a gaming event that happens every year at Instituto Superior Tecnico (IST) Tagus Park where the students can showcase and test the games that they developed. This year, students that were doing their master thesis could seize this opportunity to conduct and playtest their games/work. It was the perfect moment to gather data and have at our disposal different types of people to help us test our solution.

The experiment can be divided into the following steps:

1. First we explained what were we doing, what was the focus of the experiment and what would they be doing.

2. They would play a tutorial level to get familiar with the controllers and the game. This tutorial level was simple, with three gems and no platforms and did not count throward our experiment.

3. After, they would play the first 3 levels (World one).

4. Finally, they responded to the questionnaire.

5. If they wanted, they could play the other worlds and they needed to respond to the questionnaire for each playthrough.

This was the normal cycle of a playtest. This cycle had an average duration of 10 minutes, all depending on the individual skill of the players, their coordination, cooperation and the fact that none of the players had previously played the game.

4.3 Experimental results

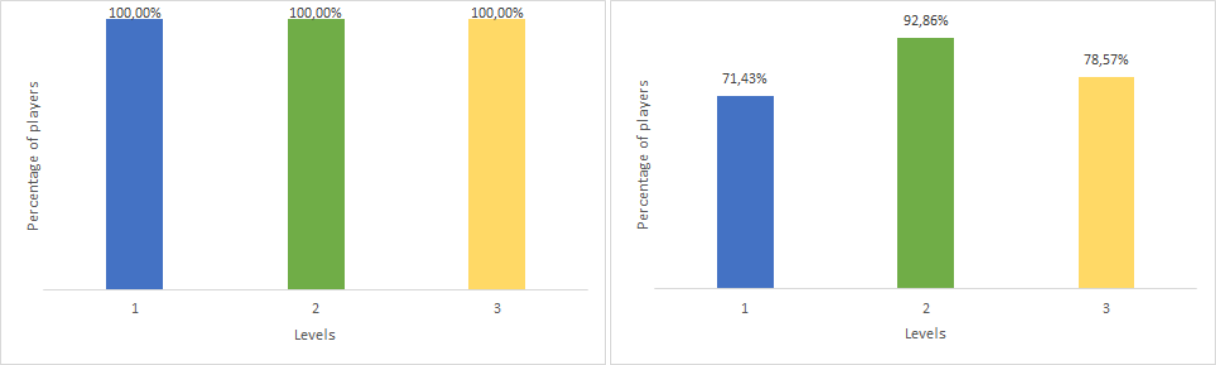
In this session we will present all the data that was gathered in the experiment explained earlier, to help us validate and improve our solution. First we will present our sample population, followed by the level evaluation done by our playtesters and finalizing with the data from the CPMs.

As mentioned before, we had the opportunity to playtest our solution at MOJO, meaning that our sample population was composed primarily of IST students and personnel, yet being at MOJO we had some game developers and other video game enthusiasts playing our levels.

In total, we had 26 forms entries meaning we had 13 level runs. The World 1 was the most played with 14 people playing it, followed by World 3 with 8 people playing it and World 2 with only 4 people. The number of plays per level discrepancy it is the result of people wanting to be challenged and jump right to the most difficult levels (World 3) and skipping the intermediate levels.

We divided our level evaluation in two categories: cooperation and balance. As previously mentioned, every player based on their playthrough of the level classify them if they had cooperation or not and if their were balanced or not.

For World 1, the Cooperation results are presented in Fig. 4.1(a) and Balance results are in Fig. 4.1(b).

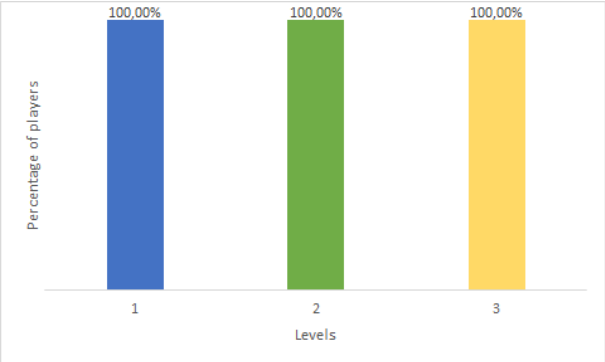


(a) Cooperation - percentage of players that evaluated each levels as cooperative.

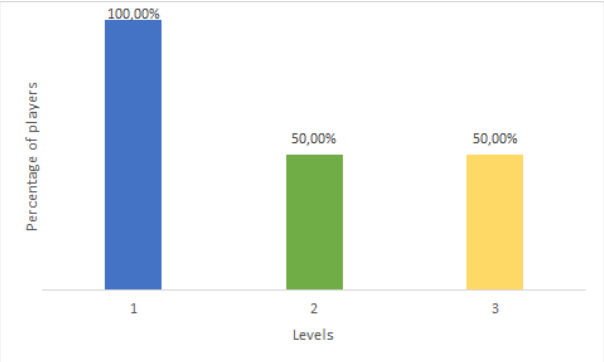
(b) Balance - percentage of players that evaluated each levels as balanced.

Figure 4.1: Players evaluation of the Cooperation and Balance for World 1.

As shown in both figures, 100% of the players assumed that all the levels had some source of cooperative challenges. Has stated in Table 4.1, level 1 could be completed without any use of cooperation, this assumption is caused by the fact that players automatically presumed that they needed to cooperate to finish the levels and created their own strategies and challenges. In terms of balance, the only one was level 2, 92.86% of the players agreed. For the other two levels more than 70% of the players said that they were balanced, the cause of these high values are because most of the strategies that were developed included some sort of division in catching the gems, as result of this they assumed that there was some level balance. For world 2, there was a consensus for level 1 with 100%, however for the other two levels the players were divided in their strategies and in consequence, if the levels were balanced or not.



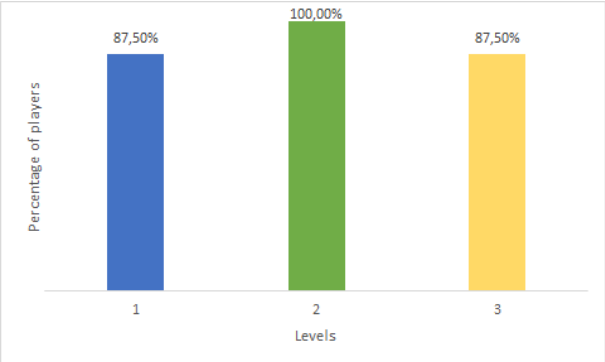
(a) Cooperation - percentage of players that evaluated each levels as cooperative.



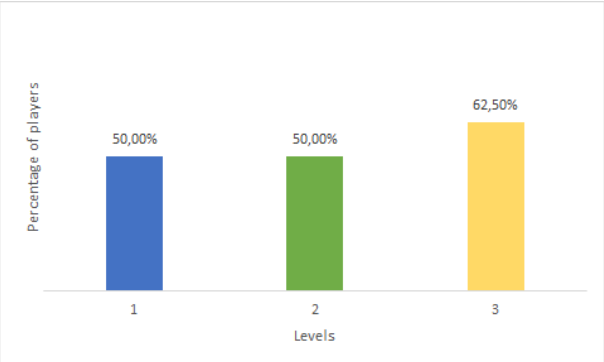
(b) Balance - percentage of players that evaluated each levels as balanced.

Figure 4.2: Players evaluation of the Cooperation and Balance for World 2.

In World 3 (Fig. 4.3), all three levels were difficult and required a considerable amount of skill to complete. The majority of players confirmed that there was cooperation in all the levels with only a small minority that disagreed. Regarding balance, it was a tie. 50% of the players believed that the levels were balanced, this division in opinions is the result of the way players approach and conceive cooperation. This conclusion was drawn through the observation of the teams and players. Some players consider that giving instructions and suggestions while not directly playing (also known as *backseat coaching*) a form of cooperation and coordination. For the balance of the levels, there was an



(a) Cooperation - percentage of players that evaluated each levels as cooperative.



(b) Balance - percentage of players that evaluated each levels as balanced.

Figure 4.3: Players evaluation of the Cooperation and Balance for World 3.

As mentioned before, we used another evaluation method to complement our research, we used the CPMs. During the experiment event, every CPMs was observed, however some of them occurred with more frequency. In this document we will focus primarily on *Laugh and excitement together*, *Work*

out strategies and *Global strategies*. Everyone of the CPMs are important, yet do to the nature and circumstances of the experiment or simply the nature of the game, the other metrics do not have the same weight.

The *Helping* metric was hard to detect, because every player was playing it for the first time. Some of the occurrences that were detected happened because a player had a must higher progression and quickly helped is fellow teammate, nonetheless this was something rare and most of the players had a similar progression.

A similar case of *Helping*, was *Waited for each other*. As result of these metrics complementing each other, the number of occurrences of *Waited for each other* were low because of the same reasons of his counter part.

The last low occurrence metric was *Got in each others' way*. This was the result of the players mentality on not disturb each other and ruin all the work that was done to complete the levels. Nevertheless, we had some events as result of some players breaking this logic or trying to messing around with his teammate resulting in some comical moments.

Next, we will present the three most important metrics. They will be presented in sessions that corresponds to their respective worlds. Each world session will conclude with an overview of all their levels.

World 1

Starting with World 1 and the metric *Laugh and excitement together*, in Table 4.2 and Fig. 4.4 we can see the number of events, averages and Standard Deviation that occurred all across World 1. The averages and Standard Deviation are per gaming session. We can conclude, that in World 1, Level 3 had the best results with an average of 4,428 events per game session and a total of 31 events, followed by Level 2, with an average of 4,333 and 26 events and finishing with Level 1 with an average of 3,571 events and 25 events.

Table 4.2: World 1 - *Laugh and excitement together* averages and Standard Deviation.

	Average	**Std. Deviation**
Level 1	3,571	1,133
Level 2	4,333	1,751
Level 3	4,428	1,902

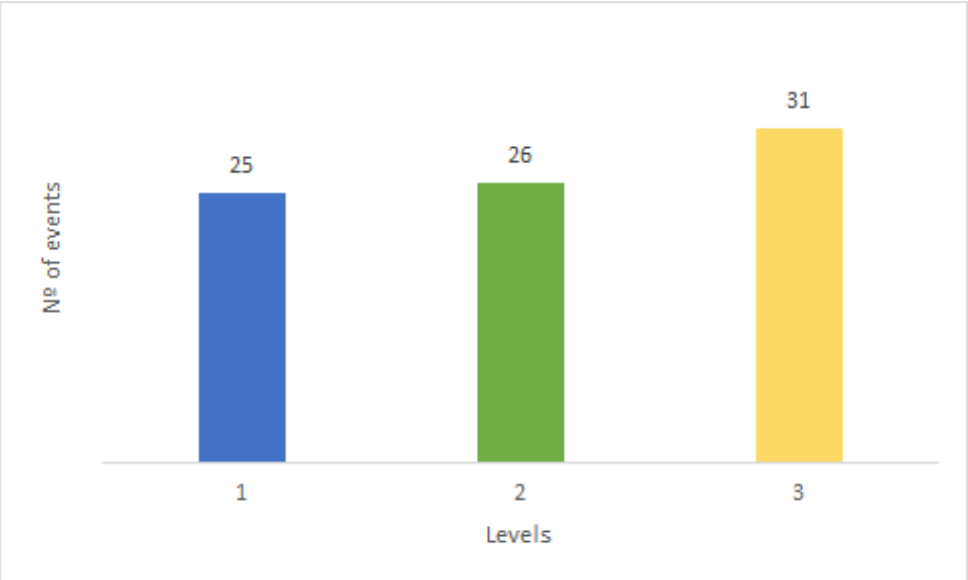


Figure 4.4: World 1 - Comparing the nº of events of *Laugh and excitement together* between levels.

Our next metric is *Work out strategies*. Fig. 4.5 shows the totals for all the levels and Table 4.3 shows the averages and standard deviations. As it can be seen, Level 3 was the level that required the most strategies to be completed, followed by Levels 2 and 1. An expected result because of the level progression created for World 1 that was explained in Session 6.1. World 1 was the only world that had an internal ranking of the levels in terms of difficulty, the reason is because it was the first World to be played and this way we could give players a sense of progression.

Table 4.3: World 1 - *Work out strategies* averages and Standard Deviation.

	Average	**Std. Deviation**
Level 1	3,285	1,380
Level 2	4,142	2,035
Level 3	5,571	1,988

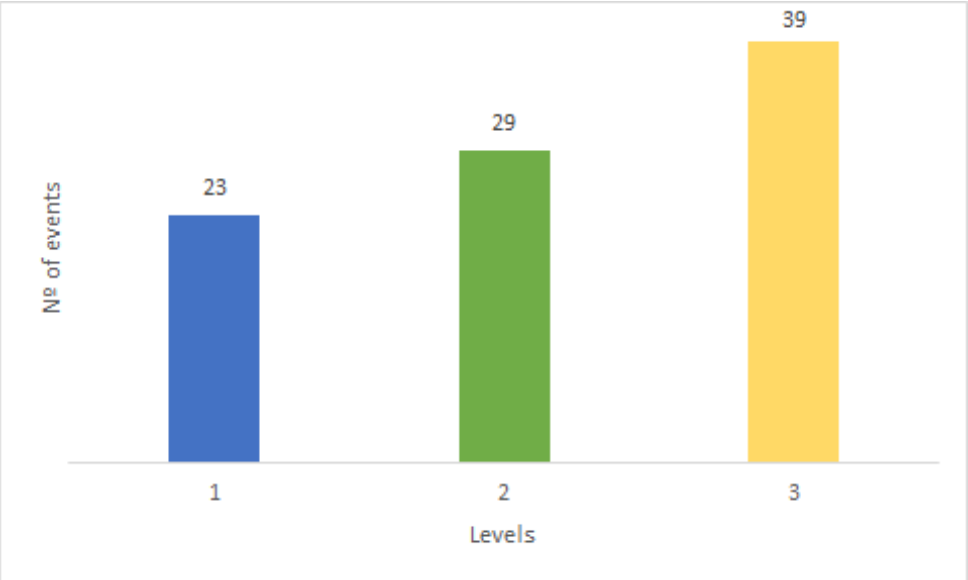


Figure 4.5: World 1 - Comparing the nº of events of *Work out strategies* between levels.

The last metric is *Global strategies*. This metric did not have as much events has the other two, the reason for this is the fact that most groups would discuss strategies during the gameplay, however some teams would tackle the strategies for the level before playing the game or would even pause the game to do it so. This shows the difference of mentality of the groups, some were more cautious then the others. We can see all these results in Table 4.4 and Fig. 4.6.

Table 4.4: World 1 - *Global strategies* averages and Standard Deviation.

	Average	**Std. Deviation**
Level 1	0,714	0,755
Level 2	0,285	0,487
Level 3	1,142	1,463

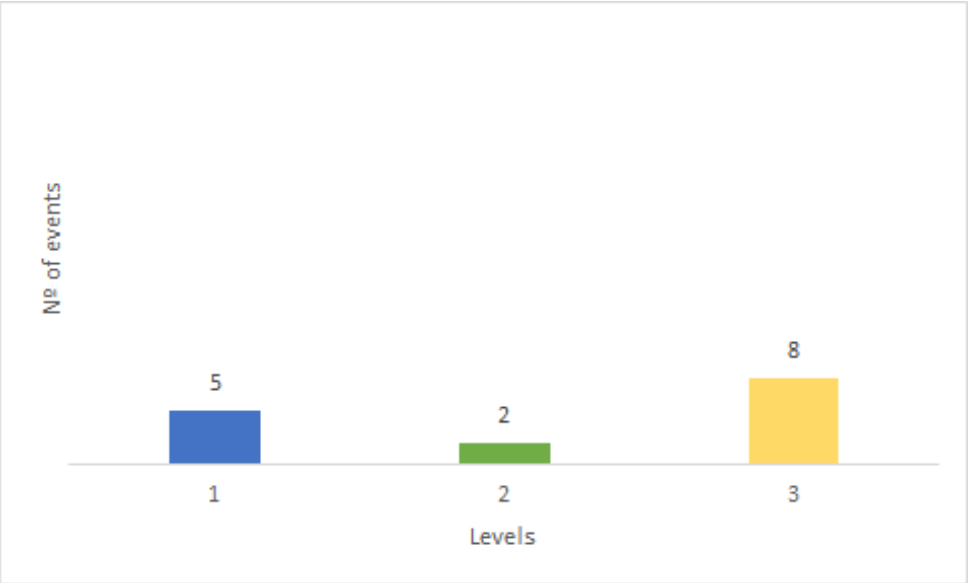


Figure 4.6: World 1 - Comparing the nº of events of *Global strategies* between levels.

To conclude the World 1 discussion, in Fig 4.7 we present a overview of all the levels and all the events. As expected, the number of events of the principal metrics increase with the difficulty of the the levels, meaning the more difficult the level is, the players have to discuss more and more tactics. However, it does not mean that the levels are better or more engaging.

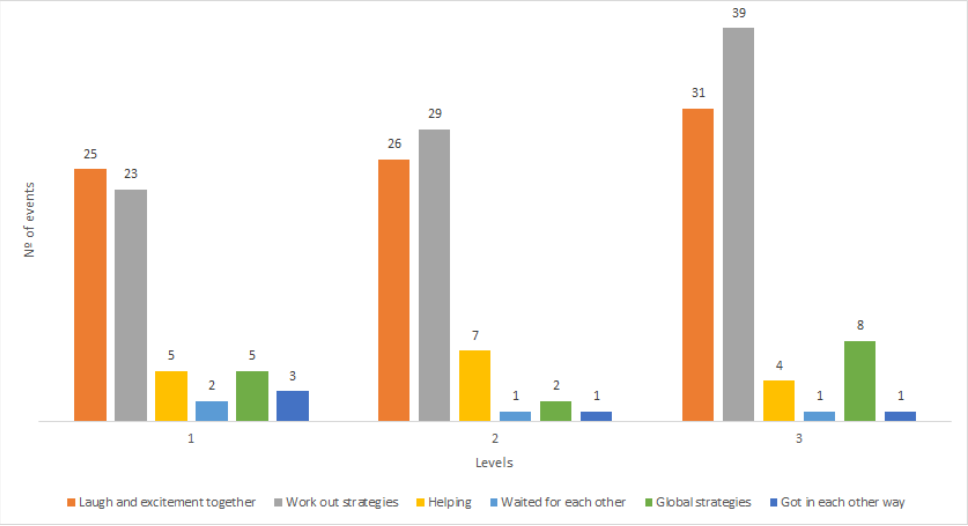


Figure 4.7: World 1 - Comparing all events between levels.

World 2

As for World 2, we will not go into much detail because of the low number of plays. However, we will comment about it and give an overview because, it took place an interesting occurrence in level 3.

There was a team that got ”stuck” (could not finish the level in a regular time) in level 3. They did not understand the puzzle in the level and when they figure it out, they did not had enough coordination to finish the level. This result in a excessive number of events in *Laugh and Excitement together*, *Work out strategies* and *Global strategies*. The more they tried, the more strategies they developed and the more fun they had. Eventually, they finished the level. In total level 3 had 19 events in *Laugh and Excitement together*, 20 in *Work out strategies* and 9 *Global strategies*, these and the rest of the results for World 2 can be seen in Fig. 4.8.

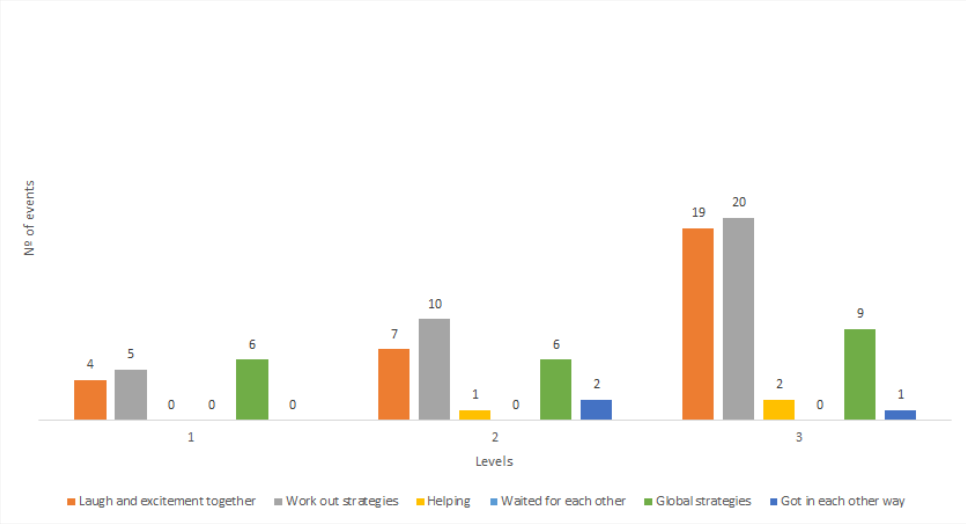


Figure 4.8: World 2 - Comparing all events between levels.

World 3

Finally, the last and most difficult world, World 3. This world had the highest number of events of all worlds, on account of having the most difficult and complex levels. Spectators after watching other teams playing this world, always wanted a shot at completing it.

Starting with the metric *Laugh and Excitement together* level 1 had 22 events with an average of 5,5 events per play, level 2 had 20 events and an average of 5 and level 3, being the least favorite of the 3 levels, had 15 events and an average of 3,5. As we can see, levels 1 and 2 had the best average of all levels (besides the inflated case of World 2). These result can be better observed and compared in Table 4.5 and Fig. 4.9.

Table 4.5: World 3 - *Laugh and excitement together* averages and Standard Deviation.

	Average	**Std. Deviation**
Level 1	5,500	1,290
Level 2	5,000	2,160
Level 3	3,750	2,362

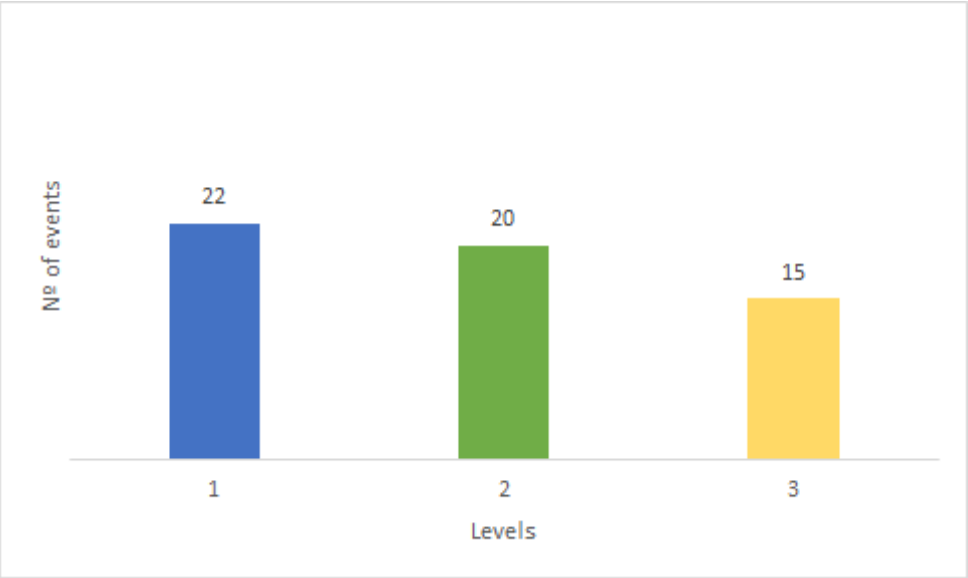


Figure 4.9: World 3 - Comparing the nº of events of *Laugh and excitement together* between levels.

The next metric is *Workout strategies*. Similar to *Laugh and excitement together* we had great results for the first two levels. In levels 1 and 2 there was 27 occurrences with an average of 6,75 events per play, in both levels. For level 3 there was 18 events and an average of 4,5. These results can be compared in Table 4.6 and Fig. 4.10.

Table 4.6: World 3 - *Work out strategies* averages and Standard Deviation.

	Average	**Std. Deviation**
Level 1	6,750	1,707
Level 2	6,750	1,707
Level 3	4,500	2,380

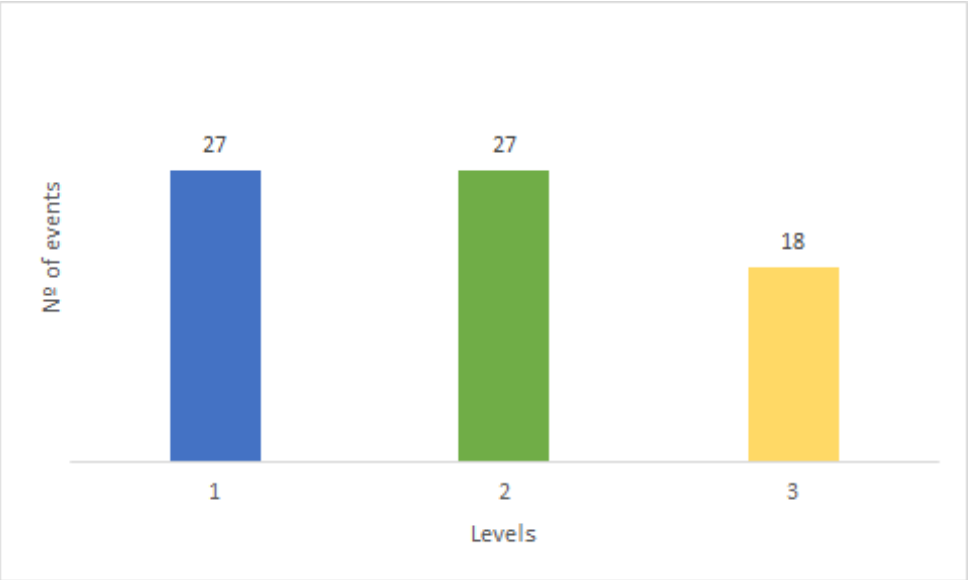


Figure 4.10: World 3 - Comparing the nº of events of *Work out strategies* between levels.

The final metric is *Global strategies*. Almost every team approach the levels with caution and tried devise plans of attack before playing the levels, occasionally some teams paused the game to review theirs strategies. In level 1 we only had 4 events of *Global strategies* with a median of 1 and levels 2 and 3, both had 9 occurrences and 2,25 of median. Thus, all the averages can be seen in Table 4.7 and the number of events in Fig. 4.11.

Table 4.7: World 3 - *Global strategies* averages and Standard Deviation.

	Average	**Std. Deviation**
Level 1	1,000	1,154
Level 2	2,250	1,500
Level 3	2,250	0,957

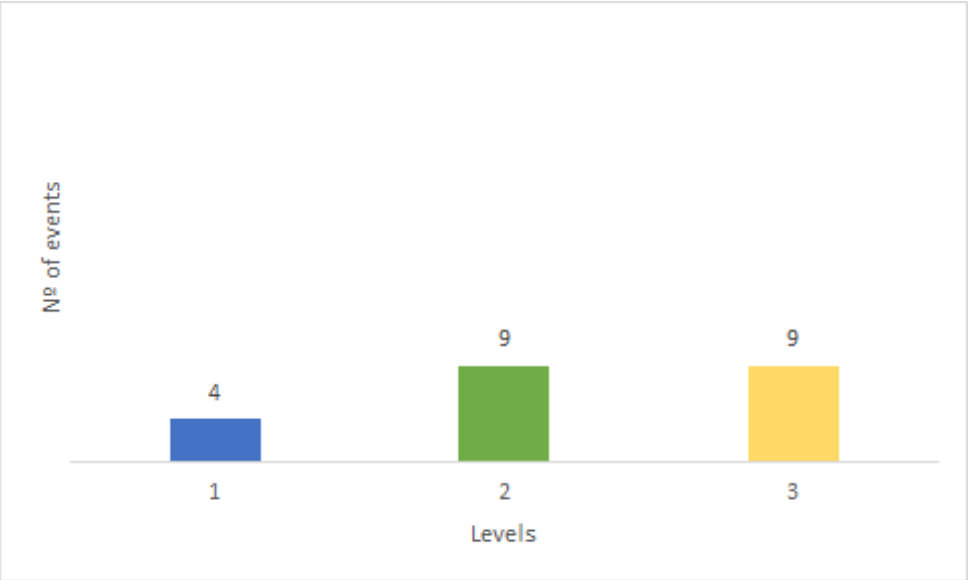


Figure 4.11: World 3 - Comparing the nº of events of *Global strategies* between levels.

In sum, the results correspond to the opinions of the players, the most fun world to play was World 3. In Fig. 4.12 we present an overview of all levels of World 3. In this case, we see an higher amount of events in the first two levels on contrary of Worlds 1 and 2, this is because players did not found level 3 complex in terms of strategies but in terms of skill and coordination.

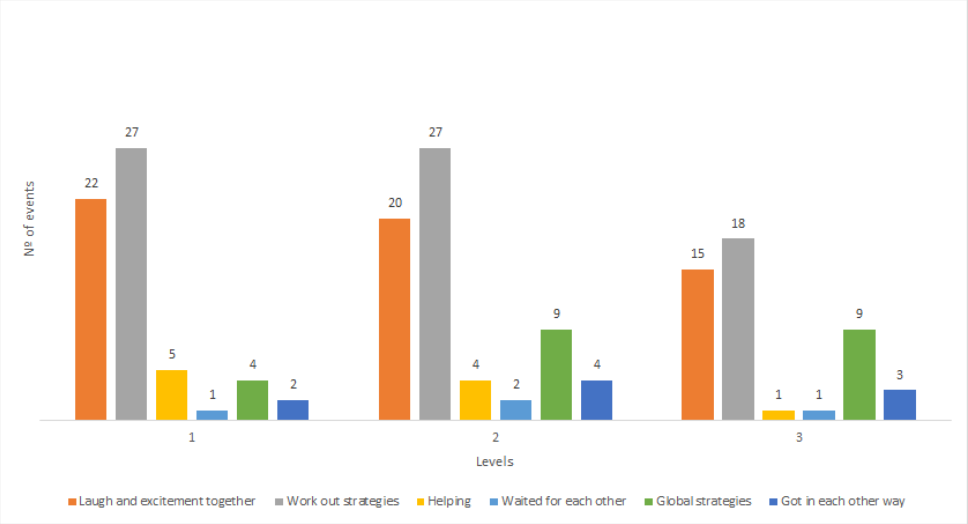


Figure 4.12: World 3 - Comparing all events between levels.

Overall, players liked the levels produced by our tool and were always excited for a new and better challenge. We think we had good results and fantastic reactions from the teams.

4.4 Result significance

In this section we will present the significance of the results that were shown in the last section. We will explain the conclusions that can be drawn from the results of our experiments.

The majority of the participants could distinguish when a level was balanced or not and when a level required cooperation to be finished, this is crucial because we reached our most important goal, the procedural generation of cooperative levels. We also noted an important aspect in the participants, each person as its own definition of cooperation and balance. There are some similarities that can be crossed in all definitions, for example when a level can not be finished without explicit cooperation between the players. However, in some levels players cooperate when it is not necessary or use communication as a form of cooperation and balance. These are some forms of cooperation and level balance that a tool can not predict or generate, although it is fascinating that we could produce some of those results in our experiment. Another essential point, are the results of the levels evaluation from the cooperative metrics. There is an implicit correlation between the level difficulty and the number of occurrences in *Laugh and excitement together* and *Work out strategies*. The harder the level is the more events of the metrics occur. Other influential result was the difference in strategies adopted by the teams. The majority of the teams would jump straight to the game play and plan as they go, while the other teams would plan strategies and a plans of attack before play the level, and sometimes if they were stuck in a level they would pause the game and rethink their strategy. These factors had an huge influence in th *Global strategy* metric.

In sum, we can assume that our tool reach our objective of generating engaging cooperative levels.

5

Conclusion

Contents

5.1 Conclusions 55

5.2 Future work 55

In this chapter, we present the conclusions of our work and future work that could be done to improve the solution that was developed.

5.1 Conclusions

In this document, we proposed to develop a tool that could generate cooperative levels for the game Geometry Friends. We gave an introduction to the problem of the lack of cooperative levels generator and how we could solve it. An extensive research was done, to help define what PCG and Cooperation meant and how could we translate those definitions to a level editor. Also, a survey was done regarding what works were developed in this field and how we could test our tool. The most important work and the inspiration for this paper, was the paper written by Rafael [10] where he tackles similar problems. After the research we created a compilation of all the related and crucial works to our solution. We concluded that the safest approach was to use Monte Carlo Tree Search as the backbone of the generation and Rafael's work as the level evaluation algorithm. Having all the pieces in place the level editor was developed and tested. We had the tremendous opportunity to test our solution at the event MOJO at IST. After MOJO we treated the results and drawn conclusions upon that data. We conclude, that we were successful in creating a tool that can generate interesting, balanced cooperative levels, we hope that can be used to augment the creativity of levels designers or simply to generate more levels for a game.

5.2 Future work

Although, we achieved our objectives, it does not mean that our level editor can not be improved. One of the improvements that can be done is add the colored platforms to the editor. In GF besides the black platforms, exists two other variants, a green one where only the square can pass through and an yellow where only the circle can pass through. This would be a fine addition to the level editor. Another, idea for future work is to eliminate the necessity of the human being to create the platforms and place the gems, with this the level generator would produce and place the gems, doing the generation entirely alone.

Bibliography

[1] Psyonix, “Rocket league,” 2015, (VideoGame).

[2] Rockstar North, “Grand theft auto: San andreas,” 2004, (VideoGame).

[3] Paradox, “Magicka,” 2011, (VideoGame).

[4] Blizzard Entertainment, “Heroes of the storm,” 2015, (VideoGame).

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of monte carlo tree search methods,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[6] Frontier Developments, “Elite: Dangerous,” 2014, (VideoGame).

[7] Nicalis, “The binding of isaac: Rebirth,” 2014, (VideoGame).

[8] Valve Corporation, “Portal 2,” 2011, (VideoGame).

[9] GAIPS INESC-ID, “Geometry friends,” 2009, (VideoGame).

[10] R. V. P. de Passos Ramos, “Procedural content generation for cooperative games,” 2015.

[11] Atari, Inc., “Fire truck,” 1978, (VideoGame).

[12] Namco, Nex Entertainment, “Time crisis,” 1995, (VideoGame).

[13] Wow Entertainment, “House of the dead,” 1996, (VideoGame).

[14] Riot Games, “League of legends,” 2009, (VideoGame).

[15] J. B. Rocha, S. Mascarenhas, and R. Prada, “Game mechanics for cooperative games,” *ZON Digital Games 2008*, pp. 72–80, 2008.

[16] M. Seif El-Nasr, B. Aghabeigi, D. Milam, M. Erfani, B. Lameman, H. Maygoli, and S. Mah, “Understanding and evaluating cooperative games,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 253–262.

[17] Blizzard Entertainment, “Overwatch,” 2016, (VideoGame).

[18] Capcom, “Resident evil 5,” 2009, (VideoGame).

[19] TT Games, “The lego movie videogame,” 2014, (VideoGame).

[20] Nazca Corporation, “Metal slug,” 1996, (VideoGame).

[21] Namco, Bandai Namco Entertainment, Now Production, Namco Bandai Holdings, “Beautiful katamari,” 2007, (VideoGame).

[22] Valve Corporation, “Left 4 dead 2,” 2009, (VideoGame).

[23] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[24] J. Togelius, N. Shaker, and M. J. Nelson, “Introduction,” in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2016, pp. 1–15.

[25] David Braben, Ian Bell, Telecomsoft, Imagineer, Torus, “Elite,” 1984, (VideoGame).

[26] Glenn Wichman, Ken Arnold, Artificial Intelligence Design, Michael Toy, “Rogue,” 1983, (VideoGame).

[27] Cellar Door Games, “Rogue legacy,” 2013, (VideoGame).

[28] 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*, vol. 3, no. 3, pp. 172–186, 2011.

[29] Gearbox Software, Demiurge Studios, “Borderlands,” 2009, (VideoGame).

[30] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, “What is procedural content generation?: Mario on the borderline,” in *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, 2011, p. 3.

[31] Maxis, Tilted Mill Entertainment, Aspyr Media, Full Fat, Infrogames, Nintendo EAD, Babaroga, Track Twenty, “Simcity,” 1989, (VideoGame).

[32] Electronic Arts, Maxis, The Sims Studio, “The sims,” 2000, (VideoGame).

[33] Nintendo Australia, Nintendo Entertainment Analysis and Development, SRD Co., Ltd., “Super mario bros.” 1985, (VideoGame).

[34] A. Summerville, S. Philip, and M. Mateas, “Mcmcts pcg 4 smb: Monte carlo tree search to guide platformer level generation,” in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[35] “Markov chain,” http://mathworld.wolfram.com/MarkovChain.html, Accessed at 16/12/2016.

[36] “Moore neighborhood,” http://mathworld.wolfram.com/MooreNeighborhood.html, Accessed at 3/11/2016.

[37] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[38] M. Magnuson, “Monte carlo tree search and its applications,” *Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal*, vol. 2, no. 2, p. 4, 2015.

[39] I. Millington and J. Funge, *Artificial intelligence for games*. CRC Press, 2016.

[40] J. Fischer, N. Falsted, M. Vielwerth, J. Togelius, and S. Risi, “Monte carlo tree search for simulated car racing.”

[41] TORCS Team, “Torcs,” (VideoGame).

[42] “Euclidean space,” http://mathworld.wolfram.com/EuclideanSpace.html, Accessed at 23/12/2016.

MOJO’s questionnaire

Procedural content generation for cooperative games

This form is to help us gather information about our master thesis in procedural content generation for cooperative games. A tool was made to create those levels and we need your help to determine if we were successful or not.
Thank you for your time!

* Required

1. **Gender:**
   Mark only one oval.

   - ◯ Male
   - ◯ Female

2. **Age:**

   _____

3. **You played as:** *
   Mark only one oval.

   - ◯ the Circle
   - ◯ the Rectangle

4. **What "world" did you played?** *
   A "world" a combination of 3 levels. The world name is in the game.
   Mark only one oval.

   - ◯ World 1
   - ◯ World 2
   - ◯ World 3

5. **Which levels had cooperation?** *
   Check all that apply.

   - ☐ Level 1
   - ☐ Level 2
   - ☐ Level 3

6. **What levels were balanced?** *
   A balanced level is a level that it is not heavily focused on one of the players.
   Check all that apply.

   - ☐ Level 1
   - ☐ Level 2
   - ☐ Level 3

7. **Can you determine which level was done by a human apart from those done by a computer?** *

Only one level was created by a human.

Mark only one oval.

- ◯ Level 1
- ◯ Level 2
- ◯ Level 3

8. **Did you had fun playing?** *

Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ This game is super frustrating.

Figure A.1: Questionnaire presented at MOJO.

B

MOJO's metric table

The following table was created to help with the cooperative metrics data collection at MOJO.

World - X	Laugh and excitement	Work out	Helping	Waited for each	Global	Got in each other	Observations
Level 1							
Level 2							
Level 3							

Figure B.1: Table used to record the number of events during a play session.

C

Levels presented at MOJO

The following figures represent all the levels that were presented and used to test the solution at MOJO.

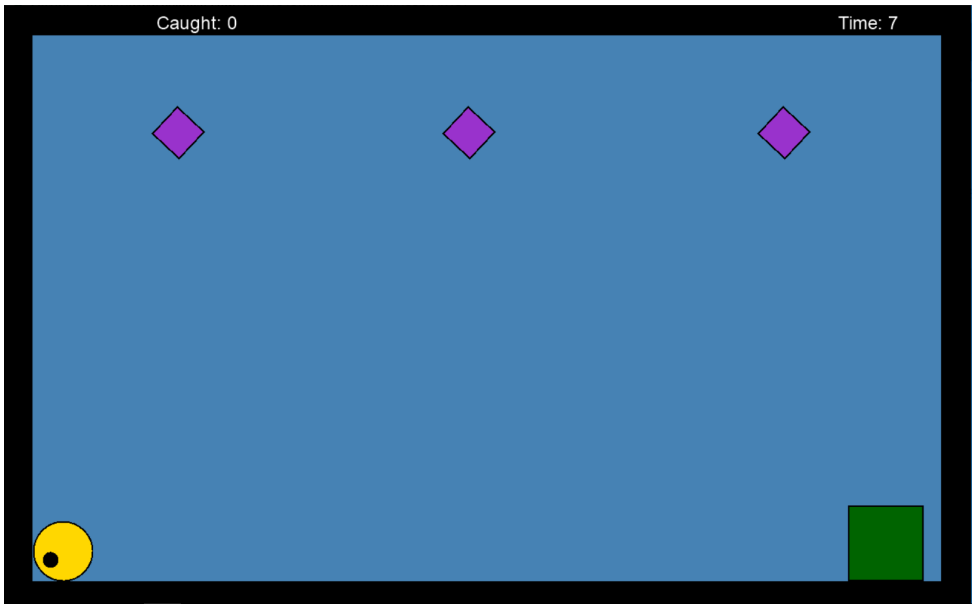


Figure C.1: Tutorial level. This was the first level the played in a game session to help players learn the game and the controls.

C.1 World 1

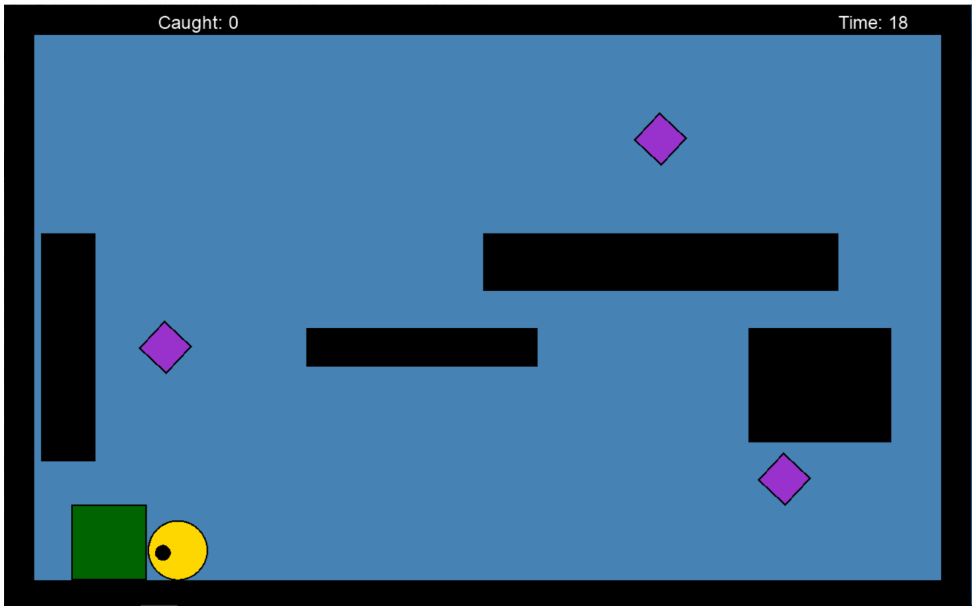


Figure C.2: Level 1 from World 1. This level was generated by our tool.

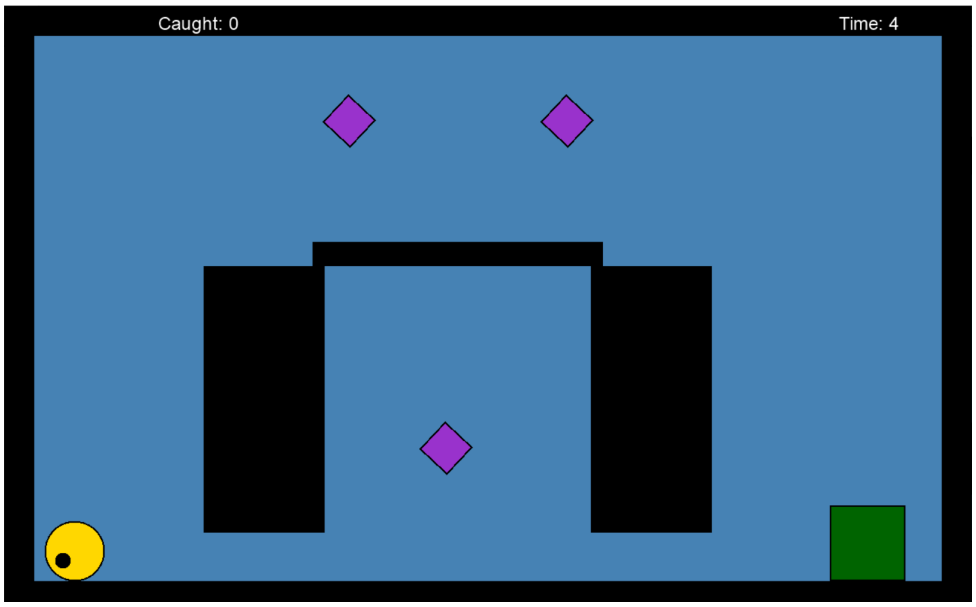


Figure C.3: Level 2 from World 1. This level was picked from the list of levels of Geometry Friends.

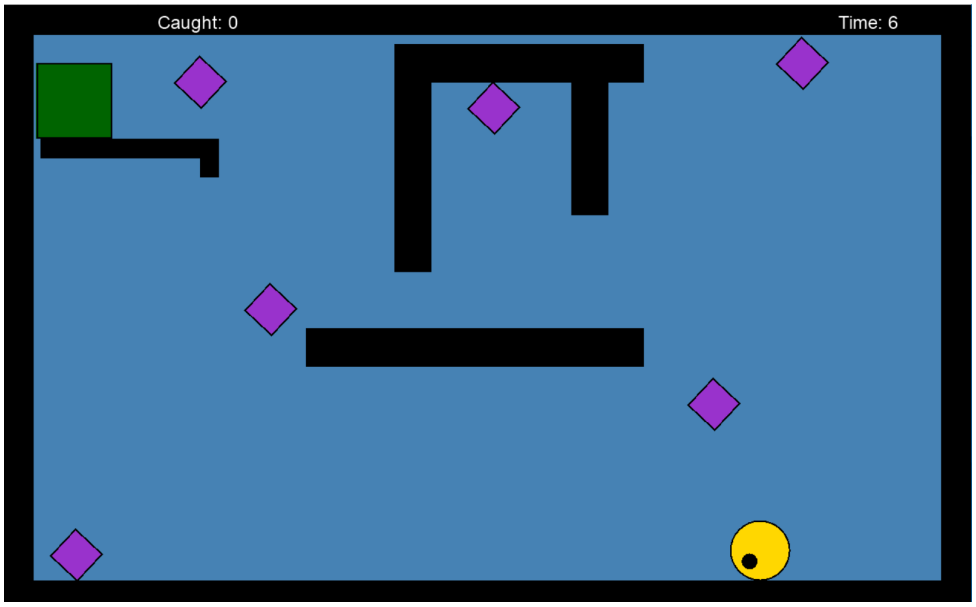


Figure C.4: Level 3 from World 1. This level was generated by our tool.

C.2 World 2

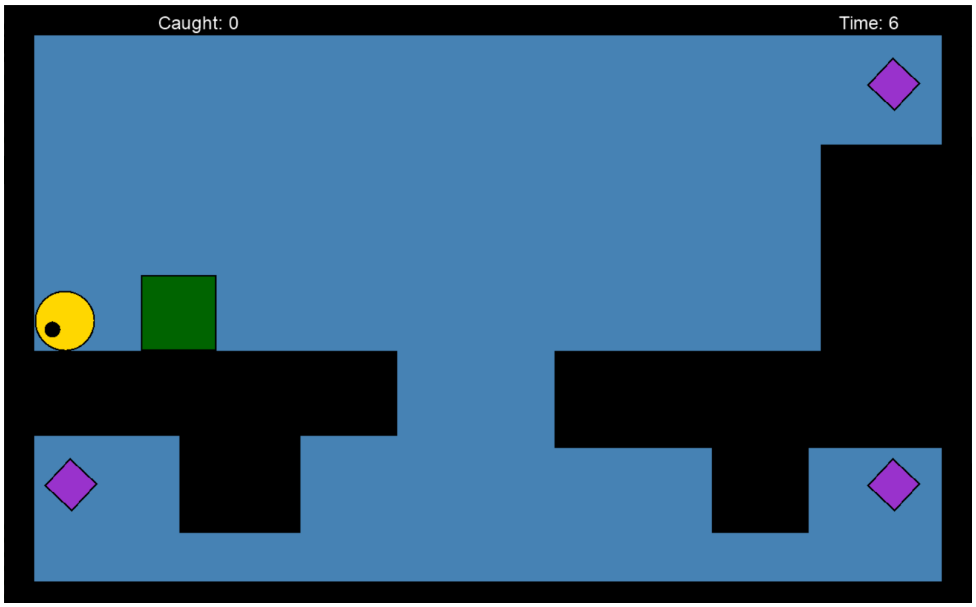


Figure C.5: Level 1 from World 2. This level was generated by our tool.

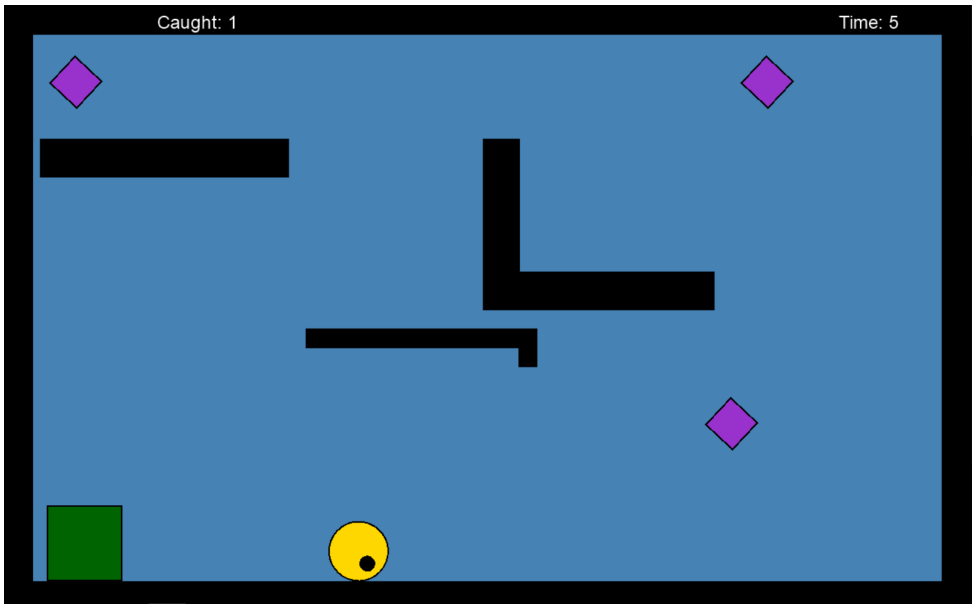


Figure C.6: Level 2 from World 2. This level was picked from the list of levels of Geometry Friends.

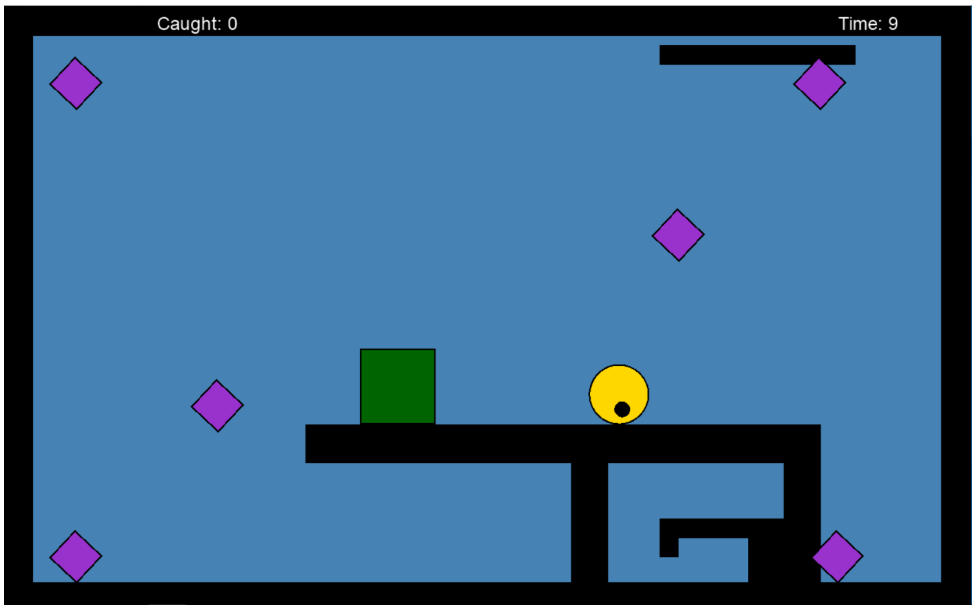


Figure C.7: Level 3 from World 2. This level was generated by our tool.

C.3 World 3

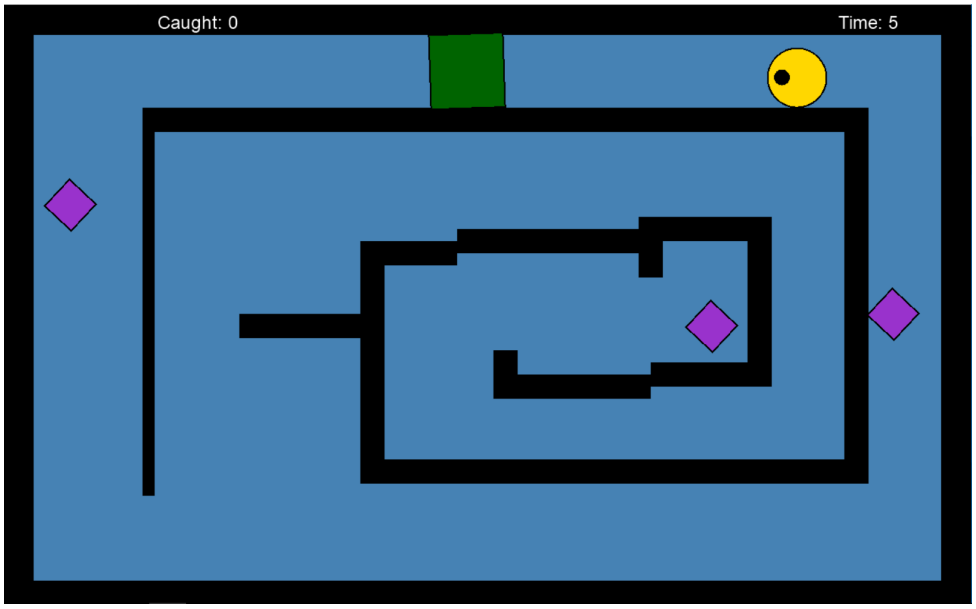


Figure C.8: Level 1 from World 3. This level was generated by our tool.

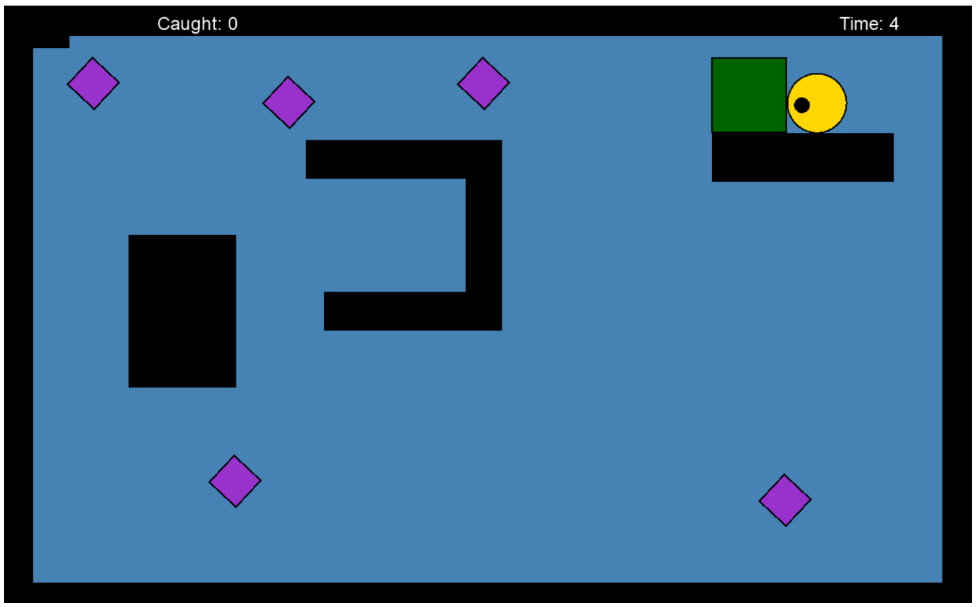


Figure C.9: Level 2 from World 3. This level was picked from the list of levels of Geometry Friends.

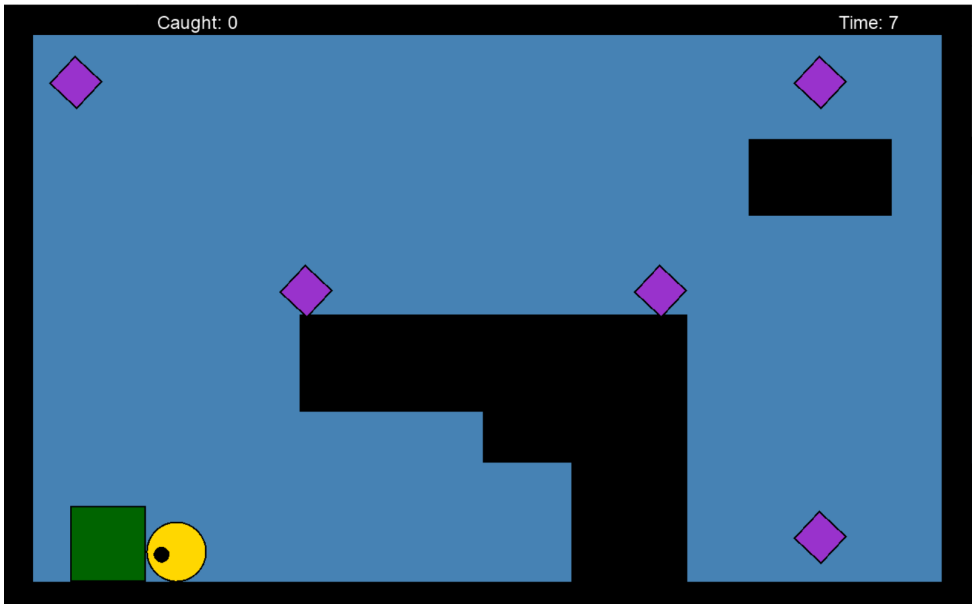


Figure C.10: Level 3 from World 3. This level was generated by our tool.