Procedural Content Generation for Cooperative Games

Pedro Borges

Instituto Superior Técnico, Lisboa, Portugal

May 2018

Abstract

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.

1. Introduction

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. That's why we believe that developers and level designers need more tools to create more cooperative driven games. Most topics covering Procedural Content Generation (PCG) do not take cooperation into consideration. Typically cooperative VG which utilize PCG do not have the PCG affect cooperation between players. For example, the game The Binding of Isaac: Rebirth [13], 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. Most academic research is about PCG or cooperative gaming, but never the two combined. 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. Videogames like Portal 2 [22] the players are forced to cooperate with one another to complete the levels, but the levels are all made by hand and the game do 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 [13] 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. Having all this in consideration the focus of this work is to develop a tool that will automatically generate levels and test them based on cooperation. Our main focus is the development of a complete level editor that, given some inputs, can generate a cooperative level. This document consists of several sections. The first two sections is where the two main themes are explored and explain, a section about Cooperation in games and one about PCG. After, there is a section about Monte Carlo Tree Search (MCTS), the backbone of our solution. Furthermore, we present the solution developed. A section on how it was tested. And we finish the document with a Conclusion and Future Work sections.

2. 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.

Geometry Friends (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.

3. Cooperation in games

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 [6] 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, they can create their own goals. We used this definition to guide our creative process and to establish what cooperation is.

With cooperation defined, we can now show how it can be translated to videogames. 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. The Design Pattern created by Rocha et al. and later expanded by El-Nasr et al. are the following:

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 it self 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 [2] where Mercy can only heal or give a damage boost to her teammates.

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 [22], 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.

Camera Settings - they identified three design choices for camera settings in the shared screen cooperative 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 [4] and The Lego Movie Videogame [20].
- One character in focus - where the camera focus on one character.

- 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 and Metal Slug [12] are good examples of games that use this type of camera configuration.

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 [11] 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. While doing our research we stumble upon a curious example, in the game Heroes of the Storm [1] 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. This example is the truest form of shared characters.

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 Non-Playable Characters (NPCs). The Hunter, Smoker, Jockey and Charger of Left 4 Dead 2 [21] 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.

Evaluating cooperation

Beside expanding the cooperative design patterns, Rl-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.

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.

Worked out strategies - 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.

Waited for each other - was created to complement the *Helping* metric. 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.

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. These types of metric are important, however the metrics we will create will be slightly different from these. CPMs are made

for evaluating cooperation based on human observation, the metrics we will develop are for computers to use and to create an automated process for the evaluation, without the need of human observation.

4. 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 [17] in the first chapter [18], 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.".

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. [17] 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 [17] 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 (in this case a solution is content). 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 a 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 between procedural generated content or 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.

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. [19] that was later revised in the book [17] in the first chapter [18]. 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 generated content can be discard or exchanged for other content. For example, games like Borderlands [7], 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*).

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 [5], 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.

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 therefor is classified as cooperative. And the focus of our document is the 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.

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.

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 [3]. 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:

1. Selection - selection is the step were the next node to be expanded is chosen. This is done using an evaluating function that helps in the choosing process. A node is considered expandable if it is a non-terminal node (if with this node the game does not end) and has unexpandable children.

2. Expansion - this is the function that expands the tree, adding new nodes. The child node that is added is picked from a set of available actions.

3. Rollout - also known as simulation or playout. From the new node, a simulation is played until a end state is reached or a predefined condition is reached. The simulation consists of choosing moves at random.

4. Backpropagation - end all the other steps are done it is time to do the backpropagation. This is the step that updates all the values from the new node to the root. Every node has information about the number of wins that were simulated in that subtree, and the number of simulations done in that subtree. These are the two values that backpropagation updates.

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 (UCT) was introduced [8]. As stated by Magnuson in [9], 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 the nodes that give the best value will be picked, avoiding a greedy approach. UTC tries to balance out these two concepts by giving unexplored nodes a boost. The evaluation function is represented by the following equation:

$$UTC(n) = u_i + C + \sqrt{\frac{\ln N}{n_i}} \quad (1)$$

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 [3] 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 [10] 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 must time and does not scale properly it will create a bottle neck and delay all other actions.

6. Implementation

As mention previously, this work consists in developing a cooperative level generator. We decided that the game Geometry Friends was the best choice to develop the tool and test our solution, because of its cooperative nature, being developer friendly already having an array of tools at our disposal and even the source code and it has the option to import and export levels. To develop our tool we used the programming language C# and the Windows Form Application (WFA) functionality that is built in the compiler Visual Studio. We also used XML to export the levels to the GF game.

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. This editor supported import and export of levels to XML and an evaluation tool.

To develop a MCTS algorithm and our level generator we defined 4 properties:

1. The actions - what actions does the algorithm does to construct the level. In our generator, we defined that an action is ”place a platform in X an Y”
2. A deterministic playing field - knowing the exact result of doing an action and the exact state of the playing world. The world is a rectangular grid with 33 lines and 52 columns, this way we have a coordinate system;
3. The game over condition - when does the game ends. It could be a win or a loss. The game ends when all the platforms are placed in the world;
4. The winning condition - when a game ends we also need to distinguish if it ended on a win or a loss. Our winning condition depends on a evaluation of the generated level.
   (a) Evaluation method - encapsulated in the winning condition, we need a method to evaluate the each action result. This method was based on Rafael et al. work [6]. where they developed a method to evaluate cooperation using a flood field tactic.
   (b) Score - as any game, when we win a game a score is given. We created a 4 point score system. Where 0 points the game is not playable; 1 point the level is

playable, however does not contain cooperation; 2 points, the level is cooperative driven; and the highest score, 3 point, happens when a level has cooperation and is balanced.

The idea of the first implementation was to develop the based 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 mention 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 **??**, 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, on 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 the evaluation of the level, 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 work.
- Grid - it is a group of cells. The grid represents the game world and 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. As mention previously, an iteration of MCTS as 4 steps and accordingly to our implementation they are defined as followed.

Selection
As already explain, this step is where the next node, to expand or simulate, 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 and explained in the Section 4.2.3. With this search method we can wide our search 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. A node could only be expanded 50 times. This step always returns a 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. This step always returns a new node child node.

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. 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. 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.

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.

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

7. MOJO and Data analysis

For this experiment, six levels 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 and generate cooperative levels even if a person lacks level design skills. 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. After all the preparations were made, we were ready to test the levels. We had the opportunity to make our tests at *Montra de Jogos do IST* (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 over the semester. It was the perfect moment to gather data and have at our disposal different types of people to help us test our program.The experiment could 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).

Table 1: Percentage of players that evaluated each levels as cooperative

	Level 1	Level 2	Level 3
World 1	100%	100%	100%
World 2	100%	100%	100%
World 3	87,50%	100%	87,50%

Table 2: Percentage of players that evaluated each levels as balanced.

	Level 1	Level 2	Level 3
World 1	71,43%	92,86%	78,57%
World 2	100%	50%	50%
World 3	50%	50%	62,50%

4. Finally, they respond 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.

Questionnaires Results

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. In Table 1 we can observe the the percentage of players that evaluated a level as cooperative and in the the Table 2 the results for balance.

Cooperative Metrics Results

As mention 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 paper we will focused primarily on *Laugh and excitement together*, *Work out strategies* and *Global strategies*. Everyone of the CPMs are important, yet duo 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 the game for the first time. Some of the occurrences that were detected happened because a player had

Table 3: World 1 metrics - Average

	Level 1	Level 2	Level 3
Laugh and Excitement together	3,571	4,333	4,428
Work out strategies	3,285	4,142	5,571
Global strategies	0,714	0,285	1,142

Table 4: World 1 metrics - Std. Deviation

	Level 1	Level 2	Level 3
Laugh and Excitement together	1,133	1,751	1,902
Work out strategies	1,380	2,035	1,988
Global strategies	0,755	0,487	1,463

Table 5: World 3 metrics - Average

	Level 1	Level 2	Level 3
Laugh and Excitement together	5,500	5,000	3,750
Work out strategies	6,750	6,750	4,500
Global strategies	1,000	2,250	2,250

Table 6: World 3 metrics - Std. Deviation

	Level 1	Level 2	Level 3
Laugh and Excitement together	1,290	2,160	2,362
Work out strategies	1,707	1,707	2,380
Global strategies	1,154	1,500	0,957

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. Next, we will present the three most important metrics. We will present an overview of the results of all their levels.

World 1 - for the metric *Laugh and excitement together* Level 3 had the best results with a total of 31 events, followed by Level 2, 26 events and finishing with Level 1 with and 25 events. The same happened with *Work out strategies*, Level 3 having 39 events, Level 2 had 29 and Level 1 with 23. For *Global strategies* the highest number of occurrences was in Level 3, with 8. Followed by Level 1 with 5 events occurrences and last, Level 2 with 2 events. The average for this world and all its levels can be seen in Table 3 and its standard deviation in Table 4.

World 2 - had the lowest number of plays from all the worlds. Ranking with the highest number of events in *Laugh and excitement together* is Level 3 with 19 occurrences, next is Level 2 with 10 events and last is Level 1 with 5. Once again Level 3 had the highest number of events for *Work out strategies*, with 20 events, Level 2 with 10 and Level 1 with 5. For *Global strategies*, Level 1 and 2 had the same number of occurrences with 6 each, behind by Level 3 with 9 events.

World 3 - in the last world, the Level 1 for the metric *Laugh and excitement together* had the highest number of occurrences with 22 events, closely followed by Level 2 with 20 events and Level 3 with 15 events. In *Work out strategies* Level 1 and 2 had the same number of events with 27 and Level 3 had 18 events. *Global strategies* also had a tie, Levels 2 and 3 tied with 9 events each and Level 1 had 4 events. In the Table 5 there is displayed the averages for this world, followed by the Table 6 with the standard deviation.

8. 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 that 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 et al. [6] 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 draw conclusions upon that data. We conclude, that we were successful in creating a tool that can generate interesting, balanced cooperative levels, that can be used to augment the creativity of levels designers or simply to generate more levels for a game.

8.1. 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.

References

[1] Blizzard Entertainment. Heroes of the storm, 2015. (VideoGame).

[2] Blizzard Entertainment. Overwatch, 2016. (VideoGame).

[3] 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*, 4(1):1–43, 2012.

[4] Capcom. Resident evil 5, 2009. (VideoGame).

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

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

[7] Gearbox Software, Demiurge Studios. Borderlands, 2009. (VideoGame).

[8] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[9] M. Magnuson. Monte carlo tree search and its applications. *Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal*, 2(2):4, 2015.

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

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

[12] Nazca Corporation. Metal slug, 1996. (VideoGame).

[13] Nicalis. The binding of isaac: Rebirth, 2014. (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*, pages 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*, pages 253–262. ACM, 2010.

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

[18] J. Togelius, N. Shaker, and M. J. Nelson. Introduction. In N. Shaker, J. Togelius, and M. J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, pages 1–15. Springer, 2016.

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

[20] TT Games. The lego movie videogame, 2014. (VideoGame).

[21] Valve Corporation. Left 4 dead 2, 2009. (VideoGame).

[22] Valve Corporation. Portal 2, 2011. (VideoGame).

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
FPS	First Person Shooters
AI	Artificial Intelligence
WFA	Windows Form Application
IST	Instituto Superior Tecnico