Procedural Content Generation for Cooperative Games

Rafael Passos Ramos
Instituto Superior Técnico
Universidade de Lisboa, IST-Taguspark
Lisboa, Portugal
rafael.ramos@tecnico.ulisboa.pt

***Abstract*— A popular topic more now than never, procedural content generation is an interesting concept with applications in numerous content creation areas, such as in game development. On the other hand cooperation in games is also an increasingly researched theme and cooperative games have become more common and more massive. A noticeable amount of research has been done on both these subjects, however, little work has been done on a joint effort on both of these areas. This work attempts to start filling the gap, linking together concepts worked on both cooperative games and procedural level generation and what makes them interesting. Existing research is compared on both areas and what ways a cooperative level generator can be conceptualized are evaluated. In this work a procedural cooperative level generator is also implemented, for the game Geometry Friends. Finally limitations and set backs are looked upon and possibilities of improvement and future work is discussed, aimed at the subject of Procedural Level Generation for Cooperative Games.

***Keywords*— Procedural content generation; Cooperative play; Procedural level generation; Procedural generation of cooperative challenges**

I. Introduction

One of the first videogames, Pong [1] was a multiplayer game. The idea that two, or more people can play the same game has been around since the beginning of gaming. Multiplayer in these first videogames had a competitive nature and players were pitted against each other. Then games like Joust [2] came along and introduced cooperative play. Cooperative play and cooperative challenges occur when two or more players work together in order to complete their objectives. Cooperation in videogames has since shown itself in many faces. Teams competing over victory, multi-player puzzle solving and playing against the computer. Games today have also grown in size, demanding ever more resources, people and time.

First appearing in the videogame world as a mean to compress content into a smaller disk size, procedural content generation had impact in the videogame world. Games like Elite [3] and Rogue [4] use procedural generation make their game world bigger that what they could realistically build. Elite uses procedural level generation to store 256 star systems in each of the eight galaxies present in Elite (a number reduced from the original 2^{48} galaxies the algorithm allowed). Using a single seed they also ensure you will always find the same planet whenever you go to the same place. Rogue generated a whole new dungeon and object placement every time a game was started. More recently, procedural content generators have been used to create all sorts of contents, like location focused side-quests in Skyrim [5] or even procedural weapon generation in Borderlands [6].

A few games today advertise having procedural generated levels and cooperative levels at the same time. However upon further look these games featured little cooperative mechanics, and generated levels did not have cooperative challenges. An example of this is the game CloudBerry Kingdom [7].

Figure 1. An example of level generation in Cloudberry Kingdom.

While there is no doubt levels are indeed randomly generated, even if the game is advertised as cooperative players cannot help each other in completing the level. In fact players cannot interact with each other in any way. The extent of cooperation in this level is limited to the fact that players win together, at the same time. The first player to reach the end of the level wins the game for everyone. It can be seen as a race to see which player can win first which is in its essence competition between players and not cooperation. As of the day this was written, no cooperative games in the market featured procedural level creation that generated cooperative challenges. Procedural generators can and are being used in cooperative games, they just aren't producing cooperative content. Only rocks, trees and textures. As such there is motivation to create a procedural level generator that produces levels with cooperative challenge.

This work's purpose is to explore how a procedural content generator could be applied to a cooperative game and to create an instance of a procedural cooperative level generator.

II. Previous work

A. Procedural Cooperative Level Generators

Research on what had been made before on generating cooperative levels led to mostly dead ends. A couple of forum threads were found but the projects were abandoned before any results were posted. The theme also sounds related to Cook et al.'s ANGELINA [8] [9]. ANGELINA is about cooperative coevolution of games. Its results are impressive, however their work is about cooperating algorithms creating single player platform games, which are not what is being looked at in this work. As such it was decided to look at research from both separate fields, cooperation in games and procedural level generation.

B. *Procedural Level Generators*

One of the several generation methods that can be used is an evolutionary generator. These generators intend to reach increasingly good solutions over several generations, in a method akin to natural selection. An initial population is generated and evaluated. Using our level example, levels that more closely match the desired fitness function are selected as the basis of the next generation. How this is done varies, but an example would be selecting the top fitness half of the generation and interbreeding these remaining levels. Interbreeding would mean that the resulting levels would have parts of both the parent levels. Most evolutionary algorithms also allow mutations to occur during interbreeding so new features can emerge.

In [10] Shaker et al. explore an evolutionary approach to generating puzzle levels. They use a technique combining an evolutionary algorithm with a grammatical representation of the level, which they call Grammatical Evolution (GE) In their implementation a level is a one-dimensional list of objects that can have proprieties such as its position in the map, orientation and effects. The evolutionary part of the algorithm uses the parent levels to create new levels by mixing parts of the parent levels into a new level. Mutations can also occur. These include the more drastic inclusion or removal of an object or the less drastic change in an object's proprieties. As a fitness function Shaker et al. desired to use playability something another of their work was trying to automate through a simulation-based fitness function that plays the level to show it can be solved. However this wasn't available so Shaker et al. used a linear combination of a set of conditions as a fitness function. The conditions included four different objects' positioning parameters defined by arbitrarily chosen placement rules that fit the game's level design, such as judging where rockets where aimed at and the predefined distance to exist between objects. Each condition would apply a penalty to the function, effectively pulling the level away from being desirable to the evolutionary part of the algorithm.

Compton et al. introduced a way to segment level design into small patterns in [11]. They define a pattern as an agglomerate of simple game components, such as a jump challenge or a single enemy. Compton et al. identify a few categories of patterns. Basic paters are composed of a single component, by itself or in repetition, with no variation occurring. Complex patterns are repetitions of the same component with tweaks and variations occurring. A compound pattern alternates between two basic patterns composed of different components. Finally composite patterns are components placed so close to each other that require a different approach than normal. Compton et al. refer that the sequential use of these patterns usually lead to lineasrt experiences and propose using a cell based solution to create something akin to a level tree with different branches. A cell would be nothing more than a grouping of patterns into a graph, permitting layers to follow different paths.

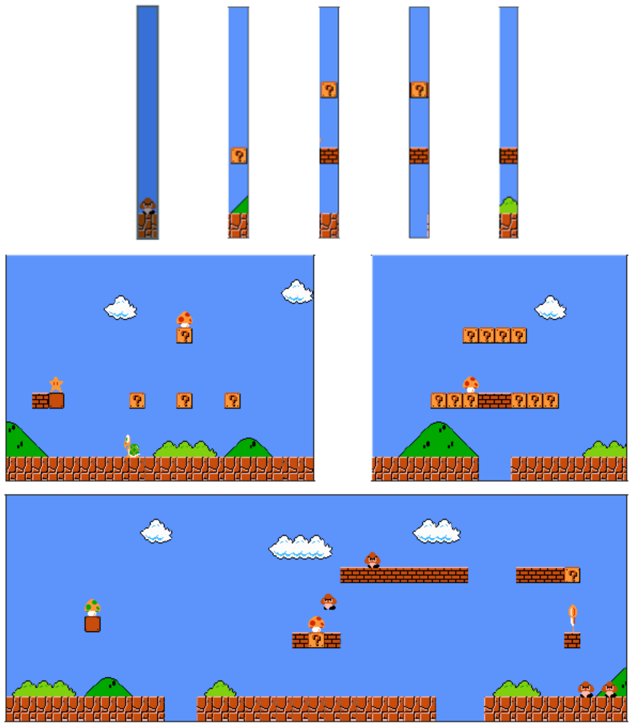
Figure 2. Micro (top), Meso (middle) and Macro-patterns (bottom) identified by Dahlskog et al. in their work.

In [12] and [13] Dahlskog et al. have extended the previous work on patterns by defining meso-patterns and macro-patterns. Meso-patterns are local combinations of micro-patterns (all previous patterns defined by Compton et al.), which consist of small challenges to the player. They notice these usually take up a screen in Super Mario. We believe meso-patterns end up being what Cells were meant be in Compton et al.'s work) Macro-patterns are combinations of several meso patterns, usually with the length of an entire level, with an objective. These new patterns could be used to control a steady increase in difficulty or to teach players game mechanics, where a pattern would first appear in a simpler form and then more complicated. Dahlskog et al. further explore this concept, basing themselves on the game Super Mario Bros. They start by manually identifying meso-patterns and macro patterns that usually combined 3 meso-patterns. From that they create an automatic analysis that reads any Super Mario Bros level encoded in a specific simple file and returns a list of micro-patterns together with their frequencies and order they appear in all meso-patterns. They also adapted this process of extracting patterns to the game Infinity Mario Bros a SuperMario Bros clone than featured possibly infinitely extending levels, procedurally generated. Being able to extract patterns from both these games' levels Dahlskog et al. proceeded to use them as a basis for an evolutionary algorithm. Their approach uses a fitness function that rewards the presence of meso-patterns and uses an evolutionary strategy to create new levels. These levels would evolve to have patterns matching to one of the existing Super Mario Bros levels or would from new hybrid patterns, producing ever-changing, interesting levels.

C. Cooperation in Games

A good amount of research has been made on cooperation in games over the last decade. Here we highlight a few, concerning defining and section cooperation in games. Rocha et al. [14] set out to define characteristics of cooperation in games. This happened when massively cooperative games online were starting to become widespread specially with the launch of a new genre Massive Multiplayer Online Role Playing Games (MMORPGs) such as World of Warcraft, by Blizzard Entertainment. Other games were also mentioned like Valve's Counter-Strike and Team Fortress 2, and Lego Star Wars, a game that rekindles the focus on cooperative, not competitive, gaming. It is wise to notice that, at the time, titles like Portal 2 and Left4Dead and Magicka, all cooperative gaming powerhouses, did not exist. Rocha et al.'s identified a few design patterns for cooperative games. These were intended to be used when designing a cooperative game. These included *Complementarity*, *Synergy between abilities*, *Shared goals*, *Synergy between goals* and *Special rules for players in the same team*. These are to be used as guidelines for creating new cooperative games, and the manner of how they are implemented will decide if the cooperative part of the game is interesting and fun. As a mean to prove this Rocha et al. developed Geometry Friends, a 2D platformer, focusing on *Complementarity* and *Shared goals*. Tests with players supported that the cooperative part of the game was interesting and fun.

In [15] El-Nasr et al. set out to phantom how to evaluate cooperative games. In their work they expand on Rocha et al.'s design patterns by adding the following: *Camera Setting*, *Interacting with the same object*, *Shared Puzzles*, *Shared Characters*, *Special characters targeting lone wolf*, *Vocalization* and *Limited Resources*. After providing new design patterns and updating some of the previous El-Nasr et al. define a set of metrics with which to evaluate whether a game is promoting cooperation and whether this cooperation is positive. They call them Cooperative Performance Metrics (CPMs) and are associated with certain events. A description of each follows:

Laughter or excitement together: These events occur when all players laughed or demonstrated excitement at a situation occurring in the game.

Worked out Strategies: These events occur when players gather to discuss solving shared challenges or dividing a problem into smaller parts that individual players can resolve. It is a metric that marks complexity and/or good cooperative challenges.

Helping each other: This event occurs when a player is helping another with understanding controls or game mechanics, but not when players are helping each other in the game. This last one falls into the Worked out strategies category.

Global Strategies: This refers to events where players take up different roles to complement each other's abilities.

Waited for each other: This metric refers to the amount of events that occur where one player has to wait for another, be it from differences in player skill or level design.

Got in each other's way: These events happen when players decide to do an action that makes it difficult or impossible for other players to do an action they wanted or needed to do.

These CPMs are useful when evaluating cooperative games and are evaluated by observing participants in game. In their work El-Nasr et al. had a groups of children play several games and recorded the game sessions. Two independent researchers were then asked to go through recorded footage and verify when CPM events occurred.

III. Cooperative Level Generator

This work's objective is to have a functioning level generator that generates levels with cooperative challenges. As research into making a procedural level generator take into account cooperative play has not been found a solution must be looked upon existing procedural generator and cooperative play research in order to see what it takes for a procedural level generator to include cooperative play.

The generation methods seen in current cooperative games that employ procedural level generators do not take into account cooperation and as such are forced to make game mechanics such as having players not being able to interact with each other in any way which ensure cooperation or the lack of it won't interfere with level completion. To fix this ways to address solving cooperative challenges have been looked upon. A cooperative challenge is basically a puzzle that requires player A to be at said place or do said action while player B does another action. Puzzles with more than one piece already exist and are often used in levels. However this does not take two players actions into account. The best way out would be having a set of cooperative agents traverse a level's space and wherever they interacted we would have them generate a cooperative challenge. However agents that can solve cooperative challenges are not yet available and another solution must be looked for. Players' actions can be abstracted into general moves: Character X jumping results in character X being in another place. This way of thinking can be used to also abstract both players into a single entity controlling both players' actions. This combined player would only exist for the generator but allows us to generate a level in a similar way to what would be used for a single-player level. The real trick is to know how to design interesting cooperative puzzles. This is something this work does not work upon (and we suggest the reader to further explore cooperative puzzle generation, as it is an interesting subject). Instead, designer levels can be analysed and cooperative puzzles can be extracted. Using the aforementioned pattern system, patterns can be extracted from already made designer levels that have cooperation and be use as guidelines to generate new cooperative patterns. Whether if generated levels are cooperative has still to be analysed as joining cooperative patterns does not guarantee the level remains cooperative but it is a good place to start from because, since levels we are drawing patterns from are cooperative, the bigger patterns we extract are going to be of cooperative challenges. To produce a cooperative level the patterns have to be linked together. Again, linking cooperative patterns cannot be guaranteed to keep the patterns cooperated and this is something that hasn't been looked into and would require experimentation so this obstacle must be avoided. A way this can be done is to separate the cooperative big patterns, meso or macro patterns. Considering a 2D grid of blocks a pattern guaranteed to be cooperative would be inserted into a block. This block would have input and output for each connection it had with other blocks. This input and output would define what players can enter or exist the block through the connected point. As such we could have two cooperative blocks that caused different players to exit through different sides be connected with non-cooperative blocks. This way players can solve a cooperative challenge, have the non-cooperative challenges in the way to the next cooperative challenge. It is important that these blocks fit together as puzzle pieces, since non-valid connections would deem the level invalid and impossible to complete.

The approach above allows us to extract patterns and use them as puzzle pieces, connecting them to each other. For variation level zones that have cooperative challenges need to be generated. The above approach would work if we simply mimicked the patterns without doing changes. A more interesting approach would require generating the level zones with cooperative challenges. This can be done by using the aforementioned patterns as fitness. This process has to be adapted to each game, since game mechanics are different and a gimmicky thing, but as a general rule what has to be done is to generate levels or level portions, sizing them to macro or meso-patterns respectively. Then the generate levels can be compared to the patterns and their likeness can be used as the fitness function the algorithm uses to know what levels are better. The use of an evolutionary approach is suggested. A valid level generator, that ensured levels generated where solvable, would be used to generate an initial population. The population would be evaluated and the parents for the next generation would be selected. After being generated, the new population would be tested for validity, dropping impossible ones out. This new population would then be evaluated and new parents would be selected. This would go on as desired.

IV. Testbed game

The game chosen to serve as a test platform is Geometry Friends, a game initially developed by Rocha et al. in their work [14]. In its' current host [16] Geometry Friends is described as a 2D platform game where two players cooperate to gather all diamonds in each level in the least amount of time. The game features two characters, a circle and a rectangle with complementary abilities. The circle rolls, being able to move if it is on a surface, and is able to jump a fixed amount and expand, changing its weight. The rectangle is able to slide in both directions, even mid-air, and can change its shape by stretching vertically or horizontally, allowing it to go through thinner gaps than the circle can. In this game players are to work together and the characters are fixed, as in there is always the circle player and the rectangle player. The game is being further developed and new features were added since Rocha et al. first created it. It is also currently being used as a platform for a cooperative AI competition. In this work we will develop an external procedural level generator that creates levels with cooperative challenge for this game.

V. Level Generator Implementation

Summarizing the general solution would consist of a few steps. First a representation of the level is required that allows us to compare it with a patterns. Then the patterns need to be extracted from existing or designer-made example levels. These patterns are to be used in the evaluation part of the algorithm. This is followed by generation an initial population of valid levels. If a more complex approach cannot be used the generated levels are try to closely match existing patterns. A more complex approach would be for instance evolutionary generation. In that case an initial valid population is to be generated and compared to the patterns. The closer to the patterns the more fitness a level has. The fitter levels are to be used in producing a new generation.

Attempts to implement this to Geometry Friends were met with a few setbacks, as the previous proposal was meant for games with bigger levels. The small level space meant we would not find macro patterns and only one, or maybe two, meso-patterns fit per level. There were recommendations to simplify what was to be done and as such pattern identification was postponed. The first attempt developed had a system of nodes and links. A node would represent an area where players could be and each node would link to another. A link would store information on what players could go from one node to the other. Due to the level size nodes were first assumed to have a base platform and nothing else. This was to later change. Analyzing the position of each node in the level we could make a graph of where the players could travel to from each node. Links that required cooperative challenge, in Geometry Friends case this was a cooperative jump, were marked. A pair of nodes were chosen from these links to be the origin node and the objective node. In this situation both players were required to reach the origin node so that the circle could reach the objective node. Then links that allowed both players to reach the origin node were marked and starting locations (starting nodes) for both players would be chosen using the marked links. Then links that players could reach from the starting nodes were marked as possible diamond locations. A diamond would be placed near the objective node and then a random number of diamonds would be placed on nodes marked for diamonds, with preference to nodes that required cooperation to be reached. This version of the algorithm produced vertically interesting levels when generating its node positions. It could also read an existing level and assign nodes to platforms. This had a couple of problems, while the levels were generally good for the circle player they were uninteresting to the rectangle player. Also the algorithm had problems dealing with obstacles that separated players and space exclusively reachable by the rectangle, something that would make the level more interesting for the rectangle player. To solve these issues we modified the algorithm with an added discrete approach.

This approach separates available level space into cells. It then marks cells as occupied when platforms are present in the node. Each cell's Moore neighborhood (the surrounding 8 cells) is analyzed for occupied cells and the cell is labeled as fitsCircle and/or fitsRectangle. Separate areas of labeled cells are then identified, using a two-pass connected component labeling algorithm. These new labels and areas are used to improve the previous algorithm's ability to place diamonds, and sectoring out nodes. Nodes in different labeled cells cannot be linked and diamonds can only be placed where a player that can reach the node can fit. This improved the resulting levels and allowed more diverse levels to be loaded in and generated upon. However, many problems remained and diamond placement was far from perfect, sometimes generating in impossible places. Also while the platforms the algorithm could generated made interesting vertical levels for the circle with at least one cooperative challenge, they failed to produce a proper level diversity and are still not able to properly mimic what patterns could do. At this point another recommendation for simplification was received along with a suggestion to segment the problem. To simplify things down again the focus shifted from generating the entire level to generating a valid and interesting level for any valid input level base provided, with cooperative challenges present whenever possible. A level base is the platforms position and size and the starting position of both players. This change in focus was taken as an opportunity to clear things up.
The final version of the algorithm dropped the node system in its entirety and focused on the cell grid. Instead of using the cell grid as an assisting tool for the node system this would be build up for only the discrete cell grid. The algorithm start by doing what its precedent did: cells are marked occupied when platforms are present and cells where players can fit are labeled as fitsCircle and fitsRectangle. This is followed by three reachability phases where we verify what can be reached.
The first one is rectangle reachability. Cells are labeled reachesRectangle when selected for analysis, if the cell is labeled fitsRectangle. The first one selected was the rectangle's starting cell. After labeling the cell the algorithm decides what cells to analyze next. It first verifies if an occupied cell is present under the current analyzed cell. If there isn't the rectangle is in free fall and the algorithm analyzes the 3 cells adjacent below the current cell. If it isn't in mid-air the cells to the side checked for fitsRectangle. If found they are analyzed. If not the rectangle faces an obstacle. The rectangle can overtake some obstacles so we verify the cell diagonally up for the label fitsRectangle. If it is present we analyze that cell. This accounts for the rectangle's stair climbing ability.

This is followed by the circle reachability phase. This phase is similar to the rectangle's only instead of checking the sides for available cells when the circle is on the ground it also attempts to jump and the label is replaced by reachesCircle. A maxJumpStrength is given to the 3 adjacent cells above the current cell to. In this case the value was arbitrarily chosen as 24, since it fit in-game behavior. The 3 cells are then analyzed and when an analyzed cell has a maxJumpStrength bigger than 0 it means the circle is still jumping in that cell. So the 3 adjacent cells above the current cell are analyzed after being given the maxJumpStrength of the current cell subtracted by 1. This mimics the jump arc of the circle.

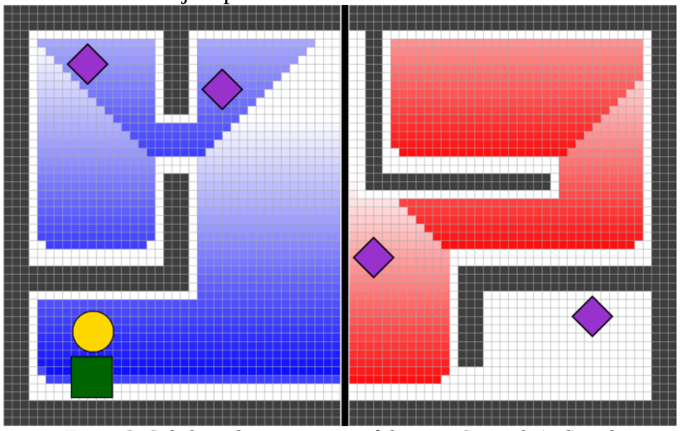
Figure 3. Side by side comparison of the jumpStrength (red) and coopJumpStrength (blue) values. Notice the lower value (less intense blue) near the bottom platform indicating a non-cooperative jump.

The final reachability phase is the cooperative reachability phase. Here we label reachesCoop to cells following a similar way to the previous reachability phase. The difference is that whenever a cell is analyzed where the reachesRectangle label is present the 3 adjacent cells above the current cell are given a maxCoopJumpStrength value of 30. This accounts for the rectangle in its stretched position.

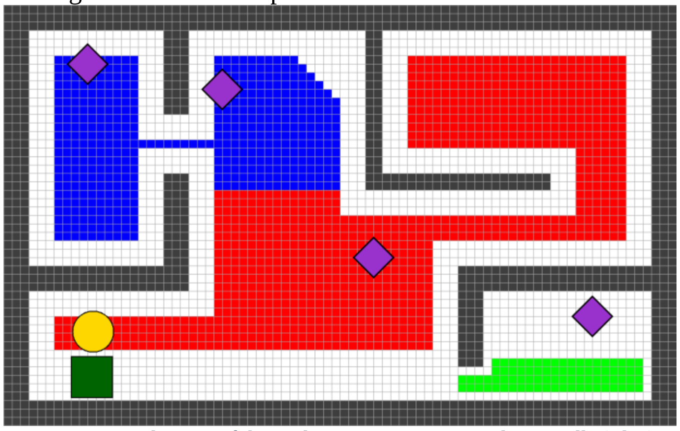
Figure 4. Visualisation of the exclusivity zones coopExclusiveCells (Blue), circleExclusiveCells (red) and rectangleExclusiveCells (green).

With these reachability maps done we proceed do the diamond placement. All cells are ran through one last time. This time we are filling cell exclusivity lists. The cell will be added to the rectangleExclusiveCells list if the cell has the reachesRectangle flag but not the reachesCircle. The cell will be added to the circleExclusiveCells list if the cell has reachesCircle but not reachesRectangle. Finally, the cell will be added to the coopExclusiveCells list if it has the label reachesCoop and not the reachesCircle label. This provides use with the 3 exclusivity zones that make up the points of interest for each level.
When generating a level, besides providing a base level we also provide a seed, number of cells to generate and two desired heuristic values that can go from -10 to 10. These are balance

and collaboration. Balance is the heuristic that measures how biased to a player a level is. A level with a balance of -10 will have the rectangle collect all the gems leaving the circle with nothing to do. A balance of 10 has the opposite effect. A balance of 0 means that there is an equal amount of diamonds to capture for both the circle and rectangle. A collaboration of -10 means there are no diamonds to be collected that require cooperation. A collaboration of 10 means all diamonds require collaboration to be collected.

These given information is used to decide where to place the number of diamonds to generate. The seed number is used as the seed for a pseudo random number generator, so that the algorithm is deterministic and results can be recreated. Then two random numbers are generated, the bias and coop. The coop value is first compared to the collaboration value. If coop is smaller the diamond will be placed in a cooperative exclusive area, in a random cell from the coopExclusiveCells list. If the coop is greater the bias value is compared with the collaboration value. If the bias value is smaller than the collaboration value a diamond is placed in the circle exclusive zone, in a random cell from the circleExclusiveCells list. If the bias is greater tha collaboration a random cell is picked from the rectangleExclusiveCells list. Whenever a list is empty or the diamond isn't able to be placed a new attempt to place the diamond is done, with new random bias and coop values. This process is repeated until no more gems are left to generate.

At this point the level is generated. At this point, before the algorithm stops it evaluates the generated level. Diamonds in each exclusivity zone are counted. Balance starts at 0 and each diamond in the circle exclusivity zone will push the balance value closer to 10. Each diamond in the rectangle exclusivity zone push the balance value closer to -10. Diamonds in the cooperative exclusivity zone aren't counted for balance but are counted for collaboration. The number of diamonds in the cooperative exclusivity zone is compared to the total number of diamonds to give us the collaboration value. This gives us heuristic values for the generated level.

This described the generation of a single level. To further improve results another option is available in the program developed. Regenerate until matching heuristics uses the algorithm to generate levels until one matches the desired heuristics, with a margin defined as (1).

$$margin = 0.1 + 0.1 * max\left(0, \frac{\text{timeout}}{1000} - 5\right) \tag{1}$$

This these values where designed with a maximum timeout value of 10000, starting from 0. It is design as to start increasing the margin after 5000 iterations have occurred, every 1000 iterations. This allowed a levels acceptable levels that didn't quite match the heuristics to be accepted. This margin seems lax, but generated levels with a margin of 30% still matched the expected results for the wanted heuristics. This brute force generation method was in place to be later replaced by an evolutionary method but remains there as a shell in which features would later be added.

VI. Evaluation

This work's objective is to create a procedural cooperative level generator that created levels with interesting cooperative challenges. A generator was created, and the levels it created had cooperative challenges whenever possible. When a level separated both players physically, the generator would produce a non-cooperative level with its balance defined by the balance heuristic provided. To understand if the levels produced where interesting, and a match to human designed ones an experiment was had. For this we had two hypothesis.

H1: Cooperative levels generated by the algorithm required coordinated effort to be successfully completed.

H2: Levels generated by the algorithm are indistinguishable from levels created by a person.

A. *Experiment*

The experiment consisted on having pairs of participants play a series of 9 levels. After completion participants would fill out a questionnaire. This questionnaire had a set of affirmations pertaining levels 2 to 7 that players where asked classify from 1 to 6 whether they agreed with the affirmations. 1 meant Completely Disagree, 2 meant Mostly Disagree, 3 was Partially Disagree, 4 was Partially Agree, 5 meant Mostly Agree and 6 meant Completely Agree. The 6 point Likert scale was chosen as it does not include an indifference choice, making participants express either disagreement or agreement. This helped as the affirmations could correspond to yes or no questions. As participants played through levels an independent researcher would record CPM events occurred during each level and how long participants took to complete the level. Time to completion was important because it had an influence in the number of events that occurred. The recorded CPM events were *Laughter or excitement together*, *Worked out Strategies*, *Helping each other*, *Global Strategies*, *Waited for each other* and *Got in each other's way*.

There existed 3 series, each created by a different algorithm or a person. The A series was generated using the cooperative algorithm. The B series was generated using a non-cooperative algorithm. The C series was designed by an independent researcher. Levels designed by the independent researcher were used as the level base for both A and B series. A pair of participants only played a single series, and where not told which. The series featured levels in the same order and the same design but differently placed diamonds.

B. *Participants*

Our sample consisted of 30 participants, the majority of which belonged to the male gender (N=27) and the average age is approximately 22 (M=22.4 and SD=7.050). This experiment was conducted at IST – Taguspark's campus – so all participants were related to this facilities (teachers and students).

Overall, the majority of participants reported playing less than an hour per week (8 participants). The next biggest group was participants who reported playing 3 to 7 hours weekly (7 participants) followed by 1 to 3 weekly hours (6 participants). The remaining participants reported the largest amount of hours per week reported spending more than 3 hours daily on gaming

(5 participants) and lastly the less common group was participants who reported 1 to 3 hours daily (4 participants).

The majority for participants, concretely 56.7%, reported to sometimes play platform games. Participants who said they often play platform games represent 10% of the sample and 33.3% reported not playing platform games at all.

C. Results

In order to test our first hypothesis, we verified A series' mean report for the affirmation "Cooperation was required to complete the level." for levels 2 to 7. Levels 2 (M = 5.4, SD = 0.699), 4 (M = 5.5, SD = 0.707) and 5 (M = 4.2, SD = 1.687) reported that cooperation was indeed required for completion. Level 3's results were the lowest of the group (M = 1.7, SD = 1.567). This was expected as the level featured both players separate by a wall, with no possible interaction. The level 6 (M = 3.9, SD = 2.132) had particularity that is mistakes were not made the players could individually collect all the diamonds. However if a mistake happened, cooperation was required to get back in track. Lastly, level 7 (M = 3.1, SD = 2.025) did not require cooperation per se, but considering some player styles cooperation would be preferred for completion.

"Foi necessária cooperação para completar o nível"

A Series	N	Mean	Std. Deviation
Level 2	10	5.40	.699
Level 3	10	1.70	1.567
Level 4	10	5.50	.707
Level 5	10	4.20	1.687
Level 6	10	3.90	2.132
Level 7	10	3.10	2.025

Table 1. Reported results from the 6 point Likert scale from the affirmation "Cooperation was required to complete the level" for levels 2 to 7 from questionnaires for the A series.

The aforementioned tendency to use cooperation when possible is more notable in the B series which levels were made by the non-cooperative generator. This series can be argued to be easier, there were no diamonds in hard to reach places that only with cooperation could be collected. That didn't mean however that players did not cooperate. Levels were completed in far shorter time, being the only series where players often solved a level in under than 10 seconds and at worst players took 300 seconds to complete a level whereas in the other two series players took around 600 seconds at times, for the A series and 800 seconds for the C series. These higher finish times were recorded in level 2 and 5 for both series. This is mainly because players still chose to cooperate, given the choice. We observed participants outright choosing to cooperate to collect gems or make jumps even in cases where cooperation was not required, and at times prejudicial. Because of this players reported requiring cooperation in a more uniform way in the non-cooperative series, with reported values for the non-cooperative levels on other series, 3 (M = 3.0, SD = 2.582), 6 (M = 4.9, SD = 1.663), and 7 (M = 4.3, SD = 2.163), being higher. Level 3, where players are separated, remains the only level where participants on average didn't feel cooperation was required. While these results show us that players felt that cooperation was required in all level series, with less accentuated differences between levels in the non-cooperative levels, observation of gameplay seemed to suggest a lot less cooperation was actually done in the B series. Results from CPM events recorded (which we will explore further below) seem to agree with these last observations.

"Foi necessária cooperação para completar o nível"

B Series	N	Mean	Std. Deviation
Level 2	10	5.50	.850
Level 3	10	3.00	2.582
Level 4	10	5.40	1.265
Level 5	10	5.20	1.619
Level 6	10	4.90	1.663
Level 7	10	4.30	2.163

Table 2. Reported results from the 6 point Likert scale from the affirmation "Cooperation was required to complete the level" for levels 2 to 7 from questionnaires for the B series.

"Foi necessária cooperação para completar o nível"

C Series	N	Mean	Std. Deviation
Level 2	10	5.80	.632
Level 3	10	1.00	.000
Level 4	10	5.40	.699
Level 5	10	4.60	1.075
Level 6	10	3.40	1.897
Level 7	10	2.70	1.494

Table 3. Reported results from the 6 point Likert scale from the affirmation "Cooperation was required to complete the level" for levels 2 to 7 from questionnaires for the C series.

For CPMs events we focused on the results of Worked out strategies events for levels 2 to 7 in each series. This event was selected, as along with Global Strategies, is the event that only occurs with cooperative play. Global Strategies was not used as the game Geometry Friends forces it to exist a single event per level, as players always used the same complementary characters. Let's start with results for the A series. As before, Levels 2 (M = 2.0, SD = 1.764), 4 (M = 2.8, SD = 1.814), 5 (M = 2.8, SD = 1.814) are levels that require cooperative action and reported a higher amount of events. Level 3 (M = 1.0, SD = 0.667) reported the lowest amount of events. Level 6 (M = 1.6, SD = 0.516) and 7 (M = 1.2, SD = 0.789) also presented lower amounts of events.

Worked out strategies - A series

	N	Min	Max	Mean	Std. Dev.
Level 2	10	0	5	2.00	1.764
Level 3	10	0	2	1.00	.667
Level 4	10	1	6	2.80	1.814
Level 5	10	1	6	2.80	1.814
Level 6	10	1	2	1.60	.516
Level 7	10	0	2	1.20	.789

Table 4. Results from CPM event recording during play sessions for levels 2 to 7 of the A series.

The B series' result is highlighted by the maximum number of CPM events recorded in each level, which compared to the other two series' results are generally low, with the exception of level 6 and 7. We believe the exception is due to player's being more used to the non-cooperative levels in the B series, whereas in the other two series, these levels would pose less of a challenge compared to the previous ones. The positioning of the diamonds in the B series might also have influenced this, as it made the expected path for the circle to be less obvious. Overall these results show us that the B series had less cooperative events occur, even if participants felt that the levels were cooperative.

Worked out strategies - B series

	N	Min	Max	Mean	Std. Dev.
Level 2	10	1	3	2.00	.943
Level 3	10	0	2	.60	.843
Level 4	10	1	2	1.40	.516
Level 5	10	1	3	2.00	.667
Level 6	10	1	3	1.80	.789
Level 7	10	1	3	2.00	.667

Table 5. Results from CPM event recording during play sessions for levels 2 to 7 of the B series.

The C series has the highest mean values among all series recorded. It also has the highest minimum values for recorded CPM events. We believe this is due to the fact the series was design, with intent, by a person and as such the challenges were more refined and required greater interaction.

Worked out strategies - C series

	N	Min	Max	Mean	Std. Dev.
Level 2	10	2	6	3.80	1.549
Level 3	10	2	3	2.20	.422
Level 4	10	2	3	2.20	.422
Level 5	10	2	6	3.80	1.549
Level 6	10	1	3	2.00	.667
Level 7	10	0	2	1.60	.843

Table 6. Results from CPM event recording during play sessions for levels 2 to 7 of the C series.

In order to test H2, we decided to use an ANOVA to compare all 3 series to understand if people could distinguish between algorithm developed levels and levels designed by a person. Results suggest that participants weren't able to distinguish apart from each other. Our perspective is, as the levels' bases were exactly the same participants mainly looked to the platform placement and overlooked overlapping diamonds and the unorganized location of diamonds (compared to when levels were designed by the external researcher). The null hypothesis being rejected supports there not being a meaningful difference between both distributions however this does not support any qualitative analysis. In any case, these results confirm hypothesis 2. (See table 7). Ideally, the levels generated by the non-cooperative generator wouldn't be identified as designed by a person, as the generator is quite simple and random in nature, for example overlapping diamonds in numerous occasions. This would contrast to levels designed by a person which would be identified as such, if no effort to disguise this was made. As such it was desirable that levels generated by the cooperative generator would be identified as being designed by a person, with results close to the ones from the levels designed by a person and different from the results of the non-cooperative level generator.

ANOVA - H2

		Sum of Squares	df	Mean Square	F	Sig.
Level 2 was designed by a person.	Between Groups	1.067	2	.533	.224	.801
	Within Groups	64.400	27	2.385		
	Total	65.467	29			
Level 3 was designed by a person.	Between Groups	3.800	2	1.900	.809	.456
	Within Groups	63.400	27	2.348		
	Total	67.200	29			
Level 4 was designed by a person.	Between Groups	1.067	2	.533	.193	.826
	Within Groups	74.800	27	2.770		
	Total	75.867	29			
Level 5 was designed by a person.	Between Groups	4.067	2	2.033	.714	.499
	Within Groups	76.900	27	2.848		
	Total	80.967	29			
Level 6 was designed by a person.	Between Groups	4.200	2	2.100	.708	.502
	Within Groups	80.100	27	2.967		
	Total	84.300	29			
Level 7 was designed by a person.	Between Groups	4.067	2	2.033	.640	.535
	Within Groups	85.800	27	3.178		
	Total	89.867	29			

Table 7. Reported results from the 6 point Likert scale's results from the affirmation "The level was designed by a person." for levels 2 to 7 from questionnaires.

VII. Conclusions

For our hypothesis 1 we wanted to see if players felt cooperation was required to complete levels generated by the algorithm. As seen in table 1 for the A series, players felt cooperation was required to complete levels 2, 4 and 5 all of which contained cooperative challenges, in the form of diamonds placed where the circle could only reach when assisted by the rectangle. The presence of Worked out strategies events in the A series (table 4) confirms that participants had to work together in order to overcome obstacles. However, participants also found levels in the non-cooperative B series to require cooperation in order to be completed (table 2). We can see in table 5 that the amount of cooperative events was far inferior to the other two series (table 4 and 6). This tells us players did not have to cooperate as often. The levels in the B series did not require cooperation to be completed however participants chose to cooperate in order to complete the level, as we did not restrict players to however they decided to play the game. When players were separate (level 3, all series) participants felt cooperation was not required. Otherwise, more often than not players felt the levels were cooperative. This allows us to conclude that, given a non-competitive game, where players have the same goal, when possible players will cooperate if cooperation is not restricted. The fact that the game requires two players and they have a common objective seems to be enough to cause them to cooperate.

As we can see in table 7 results the responses to "the level was designed by a person" affirmation in each level in the questionnaire confirm our hypothesis H2 that the players are not able to distinguish levels generated by the algorithm from levels generated by a human. We think this is due to the fact to the level's base similarities. Levels had the same platform layout but different diamond positioning. This result tells us that diamond positioning alone do not cause people to identify games as made by a person, in Geometry Friends.

A. Contribuitions

We proposed a general solution to procedurally generating levels with cooperative challenges. Patterns with cooperative challenges are to be analyzed from pre-existing designer made levels. A level generator is to create the first generation of levels to be evolved into levels that closer match the cooperative patterns. Levels closer matching to patterns have higher fitness. Fitness can also be influenced by other metrics, such as balance and proportion of cooperative challenges to non-cooperative challenges.

We created a procedural level generator that generates levels with cooperative challenges for the game Geometry Friends. The algorithm analyses a level's base composed of the level's platforms and player position. An editor is provided to ease visualization and edition of this level's base. From the exclusivity zones found, the algorithm then proceeds to place diamonds in the level to promote cooperative play and ensure the presence of cooperative challenges, when possible. We prepared a heuristic system to further improve results by the current generator and to later be adapted into a better, evolutionary method.

B. Limitations and Opportunities

The testbed game used, Geometry Friends has an important limiting factor to our idealized pattern approach. Since the levels are screen sized, we cannot have macro patterns in this game. Furthermore, actual level space is vertically limited to less than two circle jumps. If we had used a cooperative game that allowed bigger levels we could have explored pattern identification and use in evolving building blocks for the level. Our first approach featured nodes as a means of abstracting a zone into a single block, however the size restriction of the level meant there was no actual space to do anything other than place a single platform in the area the node was assigned. And so we decided, in this work to move away from implement it in Geometry Friends.

C. Extendibility

As described in chapter V, the algorithm would benefit from being expanded to have an evolutionary part. The current algorithm generates a level and verifies whether it matches the desired heuristic values. If it doesn't it is discarded and a new one is generated, with no regards to previous generations. The margin system implemented allows for an increasingly diversity of values to be accepted, so long as they are close enough and a mechanism of varying the amount of diamonds to be generated is in place, which facilitates finding a solution however we believe this is not the best solution. What was intended was for the algorithm to generate a population of levels, and evolving the levels that closer matched the heuristics with few generations. We believe this would greatly improve results but not as much as the next suggestion.

The above suggestion was planned and should be easy to implement. As a means to further improve the generator we believe two things would have great impact. First would be base level generation. The current generator takes in an already made level. It is currently a mixed-authorship algorithm, with a human designing the base level. Used as such it can definitely produce good results, however taking the step to turn it into an algorithmic generation that used human help would allow a more massed use of the algorithm, such as automatic level generation in runtime. Using Geometry Friends as an example, for this we would need a generator that could create cooperative challenges such as jumps and gaps that the rectangle had to fill for the circle to pass over. We developed a simple one in the first generator, but it only generated platforms and so would generally have at most 2 cooperative challenges, and they would always be a jump. This would quickly become uninteresting for the rectangle. More diverse levels would be required.

The second is the automatic generation of cooperative puzzles. What sets cooperative games like portal is the puzzles present in the level. While our theorized solution would be able to match already made and analysed cooperative puzzle by identifying them as patterns and work towards them and it would be able to create hybrids of existing patterns it cannot create new patterns by itself. As such an algorithm that can create cooperative puzzles will certainly produce more interesting results, in the long run. Generated cooperative puzzles could then be used as a base to produce more interesting cooperative levels. This seems to be, however, a much more difficult task than what we have done here.

References

[1] Kent, S., And then there was Pong, in *Ultimate History of Video Games*, pp. 38-39

[2] *Joust,* William Electronics, Inc. (1982) [Arcade game]

[3] *Elite,* Acornsoft, Firebird, Imagineer (1984) [Videogame]

[4] *Rogue*, Toy, M., Wichman, G., Arnold, K., Lane, J. (1980) [Videogame]

[5] *The Elder Scrolls: Skyrim,* Bethesda Game Studios, Bethesda Softworks (2011) [Videogame]

[6] *Borderlands*, Gearbox Software, 2K Games (2009) [Videogame]

[7] *Cloudberry Kingdom*, Pwnee Studios (2013) [Videogame]

[8] Cook, M., *ANGELINA*, Available at (last accessed 19/12/2014).

[9] Cook, M., Colton, S., Gow, J., Initial results from co-operative co-evolution for automated platformer design in Applications of Evolutionary Computation Lecture Notes in Computer Science, Vol. 7248 (2012), pp. 194-203

[10] Shaker, N., Sarhan, M. H., Al Naameh, O., Shaker, N., Togelius, J., (2013) Automatic Generation and Analysis of Physics-Based Puzzle Games in (2013) 2013 IEEE Conference on Computational Intelligence in Games (CIG) pp. 1-8, Niagara Falls, ON, IEEE

[11] Compton, K., Mateas, M., (2006) Procedural Level Design for Platform Games in (2006) Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE), Marina del Rey, California, AAAI (Association for the Advancement of Artificial Inteligence)

[12] Dahlskog, S., Togelius, J., Procedural Content Generation Using Patterns as Objectives (2014) in Applications of Evolutionary Computation Lecture Notes in Computer Science (2014), pp. 325-336

[13] Dahlskog, S., Togelius, J., A multi-level level generator (2014) in IEEE Conference on Computational Intelligence and Games 2014 (2014) Dortmund, Germany, IEEE

[14] Rocha, J.B., Mascarenhas, S., Prada, R., Game mechanics for cooperative games (2008) in Zon Digital Games 2008 (2008) pp. 72-80, Porto, Portugal, Centro de Estudos de Comunicação e Sociedade, Universidade do Minho

[15] El Nasr, M.S., Aghabeigi, B., Milam, D., Erfani, M., Lameman, B., Maygoli, H., Mah, S., Understanding and evaluating cooperative games (2010) in CHI ’10 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (2010), pp. 253-262, New York, NY, USA, ACM (Association for Computing Machinery)

[16] http://gaips.inesc-id.pt/geometryfriends/?page_id=6 (last accessed 7-10-2015)