Automatically Produce Interesting Computer Game Challenges Using Reverse Simulation

A dissertation submitted in partial fulfilment of
the requirements for the degree of
MASTER OF ENGINEERING in Computer Science

in

The Queen's University of Belfast

by
Kelvin McClean

04/05/2018

Acknowledgements

I'd like to thank my supervisor Dr. John Bustard for giving me inspiration and guidance throughout the project.

I'd also like to thank my play-testers, Aaron Bell, Simon Campbell, Robert Finn, Matthew Hawthorn, David McGibbon Alex McMillan, Ryan Pepper, Steven Rodgers, Scott Sinnamon, and Alison Weir. The valuable metrics that they gave provided clarity on the shape that the project would take.

I would like to thank my mother Dawn, and my father Keith, for supporting me throughout my education, and pushing me to achieve more. Without their guidance, I would not be where I am today. I would also like to thank my siblings, Matthew and Lara; whose kinship and friendship have been a solid rock through my life, no matter where I am.

I would like to thank my friends, both in Queen's and out, for getting me through the difficulties of my years here.

And finally, I would like to thank the lecturers and academics at Queen's. Their teaching gave me the knowledge to get this far, and their guidance gave me confidence.

Abstract

Automatically Produce Interesting Computer Game Challenges Using Reverse Simulation

Creating a Taxonomy of Features for Reverse Simulation.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec elementum porta leo, id placerat nulla consectetur eu. Suspendisse aliquet ullamcorper luctus. Fusce eu orci vestibulum, dictum turpis nec, consectetur leo. Integer et est dignissim, sagittis metus a, tincidunt magna. Suspendisse tincidunt nisl ut tincidunt ultrices. Vestibulum gravida urna eget dui aliquam, eu efficitur lectus tincidunt. Cras at euismod purus, eget ornare leo. Proin suscipit at felis eu viverra.

Ut in dictum enim. Quisque bibendum, justo eget vestibulum rutrum, diam nulla mollis ligula, nec cursus ante felis sed lacus. Donec semper, sem vitae auctor eleifend, urna purus sollicitudin quam, vehicula tincidunt leo augue eu magna. Quisque finibus purus nec vulputate dignissim. Aenean varius velit ut turpis ultrices molestie. Duis ultricies ipsum a massa hendrerit consequat. Suspendisse elementum, lectus vitae suscipit elementum, dolor nisi cursus nisi, quis efficitur nibh quam in est. Nam tristique turpis ut dignissim varius. Cras ut interdum urna, a iaculis augue.

Contents

Introduction

The video game industry has grown at an incredible pace over the past 45 years. However, as the industry has grown, expectations have risen, and modern games are expected to conform to standards impossible even five years ago.[1] As these expectations rise, so does the cost of games.[2] It has been shown that manual content generation is unscalable, and alternatives should be examined.[3]

The game development landscape has changed in the past few decades. Marketplaces like Steam and the Xbox arcade allow smaller development teams to release games to a wider audience. In order to have a variety of challenge in their games, a large portion of Indie developers use Procedural Content Generation in Video Games (PCG-G).[4] While this content is necessary in some cases to offer competition to bigger budget titles, it has been shown that proceduaral content lacks the depth of manually created content.[5]

This master's thesis proposes a novel approach to procedural generation; to create challenges through Reverse Simulation. By generating content from a defined ending state, working to the beginning, it should be possible to create more engaging content through the development of a 'perfect solution'. This thesis aims to provide a framework for Reverse Simulation, and to analyse the various archetypes of games that this framework may apply to.

1.1 Motivation

As expectations rise, games relying solely on manual content will become increasingly more expensive to create. Developers must examine alternative methods of content generation. Possible solutions are hybrid systems (Figure 1.1), where designed content is supplemented with areas of procedurally generated plots.

Another solution is wider use of PCG-G. PCG-G can create nearly any aspect of games, from levels in platform games, to 3D assets such as trees and even animals.

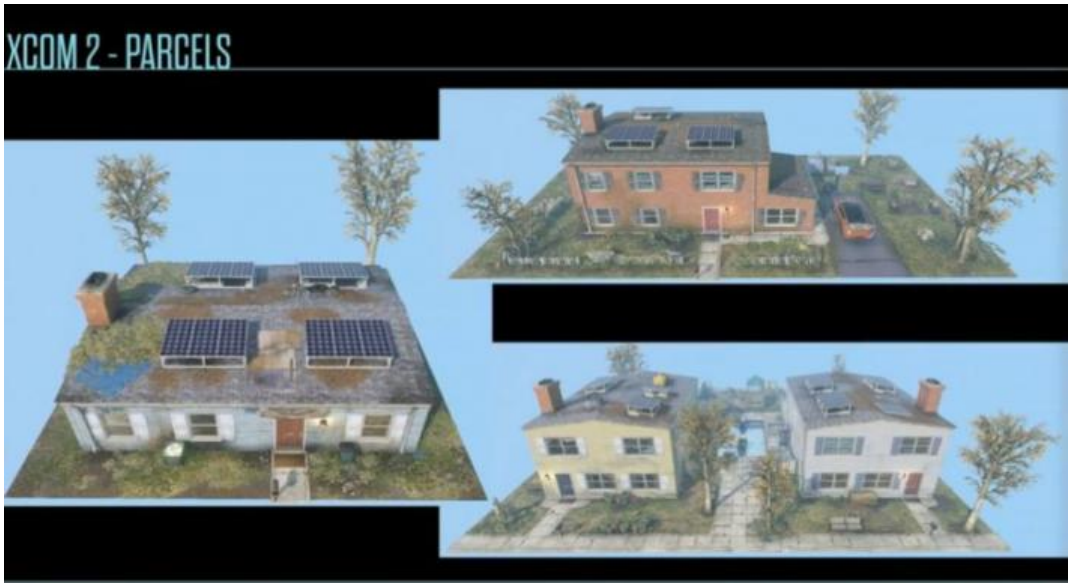

Figure 1.1: XCOM 2 uses a combination of designed levels with procedural plots or parcels.

However, PCG-G has its issues. Hendrikx et al. found that more advanced gameplay elements such as story, puzzles and world design were not implemented as rigidly in PCG-G as more basic elements.[5]

These two issues provide a significant obstacle to the future development of games which will continue to match consumers rising expectations. Reverse Simulation is proposed as a form of PCG-G that may better fulfil the higher level elements of gameplay that standard generation cannot.

1.2 Related Work

While there has been very little work done directly in relation to Reverse Simulation in the context of PCG-G, there is a variety of related content that is relevant to a subsection of the topic. Adrian et al. allow a developer to input difficulty curves, and levels are generated genetically to match that curve.[6] This can lead to interesting challenges through varying the levels of difficulty throughout the level.

Isaksen et al. generated levels then simulated AI to play them in an imperfect fashion.[7] The game is simulated several times, and the difficulty is calculated by analysing the histogram of results.

Hendrikx et al. conducted a survey on procedural component usage in video games. They found that the more complex (or higher up on their game component pyramid) a component was, the less it was matured in video games. For example, they note that procedural puzzles and worlds are not generated as sophistically as procedural textures or buildings. This could explain the lack of diversity in PCG-G games, where content generated on a lower level is sophisticated, but the higher-level gameplay does not take advantage of this.

Horn et al. discuss how to evaluate randomly generated Mario levels, and use metrics such as density and linearity to evaluate the difficulty. These metrics are compared to each other and the generators are evaluated on the range of metrics which they produce. Postma generated Mario game levels using Recurrent Neural Networks, using metrics such as Pattern Density, Pattern Variation and Column Variation to determine how the levels were generated. The levels were then compared by humans to original Super Mario levels. The testers would give scores to each level tested based on enjoyability, difficulty, and aesthetics. It was found that the human testers could not establish the difference between a generated and designed level.

Togelius et al. discuss the nature of PCG, and evaluate whether online generation of content - where a player's actions have immediate consequences – constitute PCG.[8] A rather heavy-handed example is used, where picking up a coin in a generated Mario level causes an enemy to spawn in front of you. They find that offline tools are most often PCG, online tools are likely methods for player-generated content.

1.3 Aim

The aim of this thesis is to examine Reverse Simulation of gameplay, and to analyse whether this Reverse Simulation is suitable for solving the issues presented with PCG-G.

To reach these goals, a very simple game shall be implemented, simulated in reverse and then analysed through an Evaluation Algorithm that will gather metrics that may reveal information pertinent to gameplay. Levels will be selected based on their metrics, and play-tested. The play-testing will determine whether or not the metrics are an appropriate measure of the gameplay, and help determine whether a different assessment measure should be used, or a different form of generation.

Although beyond the scope of this project, with these metrics, a machine learning classification algorithm could determine which combinations of metrics would result in which type of experience. The Reverse Simulation Algorithm could then develop levels, or even gameplay elements to satisfy those metrics.

1.4 Research Questions

The research questions that this thesis aim to answer are:

1. Does Reverse Simulation create interesting gameplay challenges?
2. How well do metrics and Evaluation Algorithms aid the Reverse Simulation?
3. What gameplay elements could be developed through Reverse Simulation?

1.5 Delimitations

This project will focus on a simple game, based on a grid and level based system. While the work should allow general conjecture about the nature of the Reverse Simulation, the results will not provide any evidence for other types of game.

The algorithm will only create levels for the game, and will not touch on larger procedural elements, such as story or world building, nor the smaller elements covered by existing algorithms eg textures, enemies, and items. In Section 5.3, the possible implementations of these components will be discussed.

To determine the quality of the metrics gathered, they will need to be analysed to determine which combination of metrics provides which type of gameplay. The metrics used for evaluating the levels will be measured by a human test. These tests will be performed by students, and the results may not be accurate for other groups.

The game this procedure will be performed for will be a turn based game. While the Reverse Simulation could likely be developed for a real-time game, the algorithm would need to be substantially more sophisticated, and as such, the findings here will only be applicable in a broad sense.

Since the desire of this thesis is to generate a framework for Reverse Simulation, the final result will not be a completed game, or game section. Instead,the difficulties in generating gameplay in reverse will be assessed, as well as the advantages and disadvantages of generating game content in this manner.
Through providing this information, informed decisions can be made on the selection of algorithm for creating impactful gameplay.

1.6 Background

This framework is being developed in Java OpenGL, using the Lightweight Java Game Library,[i] and developed to be compatible with Windows Operating System. As this thesis is exploratory, the code developed is built ground-up from these libraries. This allows more control over the entire loop of the game, which is useful to complete the level generation and evaluation in a timely manner. Development was iterative, beginning with a minimum viable product for the whole system, then progressing into more complex design.

The game is a tile and turn based strategy game. The finished level has a saved 'state', which is currently defined as the lack of any enemies, and the player reaching a certain tile. The player is able to move in one of the four cardinal directions, and landing on top of an enemy's tile kills it. The enemies move back and forward in lines, from one to three steps at a time depending on their colour. Figure 1.2 shows an example of a generated level. The software is built to be easily extensible, and adding new components shouldn't interfere with the Monte Carlo and Reverse Simulation elements.

The following are generated to fill these levels:

- *Walls:* Walls are generated to create a block for player movement. In the current algorithm, these are placed randomly throughout the level, with the number of wall tiles created being used as a metric. This could be extended to include cohesiveness of tiles to get a proxy measure for level aesthetics.

- *Enemies:* Enemies provide a challenge for the player. If an enemy moves on top of the player, the player will die. Similarly, a player can defeat an enemy if they move on top of it. To complete a level, the player must defeat all enemies, then move to the end tile. The number of enemies, as well as the number of enemies with higher step values, is gathered.

[i] https://www.lwjgl.org/

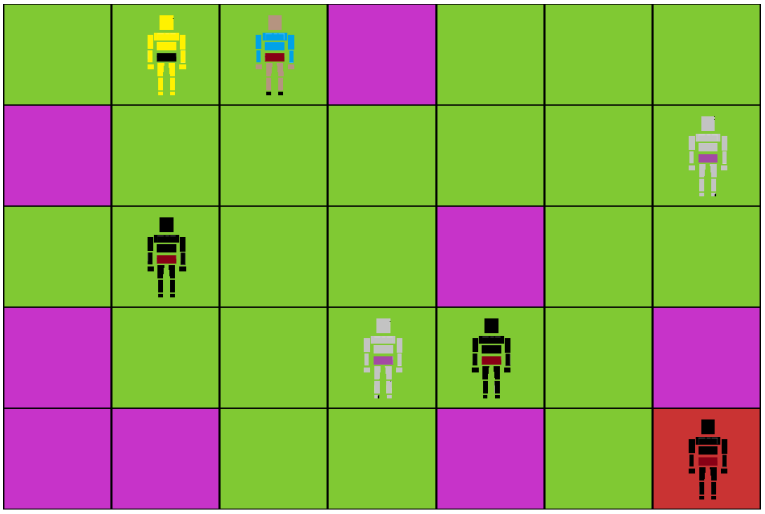

Figure 1.2: A screen capture of a generated level.

- *End tile:* Used to start the Reverse Simulation process. The player must defeat all enemies and move to this tile to complete a level. In the current version of the program, this tile is always in the bottom right corner of the level.

Theory

This chapter provides terminology and background information for the algorithms and other concepts used in this thesis. Reverse Simulation, the Monte Carlo Tree Search Algorithm, and Metric Evaluation are discussed here.

2.1 Reverse Simulation

Reverse Simulation is a proposed way to generate Procedural Content, or PCG for short. PCG is defined by Togelius et al. as "the algorithmical creation of game content with limited or indirect user input".[8]

Procedural Content Generation

PCG is commonly generated through creating a pattern for a level, and then filling in the generated level algorithmically.[i] There are often difficulties with scaling the difficulty of procedural content, as the path the player may take is hard to estimate.[9]

A distinction needs to be established between PCG generated online and offline. Online content is generated as the game is run, while offline content is generated during development.[10]

Applications of Reverse Simulation

Reverse Simulation is procedural generation from a defined end point, using an algorithm to take steps backwards until a satisfactory starting position is found. Reverse Simulation is currently used for very simple games; for example it is used to create Sudoku or Crossword Puzzles from a finished solution.[11]

Reverse Simulation is also used in a variety of fields outside of games. For example, it has been used to discover the source of pollutants in rivers,[12] and to find optimal

[i] http://tinysubversions.com/spelunkyGen/

factory schedules.[13] It has proved useful in finding solutions that may be computationally complex to arrive at procedurally.

Ensuring an Accurate Result

Due to the nature of Reverse Simulation, it requires that every step of the program be considered carefully as to be reversible. For example, if an entity moves algorithmically in a certain fashion, a backwards version of the algorithm must be able to move them back to their original position. During the Reverse Simulation phase, all steps taken should be stored. To ensure an accurate and feasible result is created, these steps should then be played out forwards, in order to test the generated solution.

If the generated forward and reverse algorithms match, all entities will end up in the same position that generation began on.

2.2 Monte Carlo Tree Search Algorithm

The Monte Carlo Tree Search Algorithm is already used in various fields to search complex problems, examining likely paths while allowing for new approaches to be searched so as a better solution may be found. The algorithm is a general one, requiring little to no knowledge of the system it is being implemented in. As the algorithm is so practical, and generalised, it has been implemented in a variety of fields in industry.

Algorithm Background

The algorithm builds a tree from a set of states. From a starting position, the algorithm looks at all possible outcomes, and picks one effectively at random, adding a 'pass-over bonus' to the passed nodes. These nodes will be more likely to be looked at on the next iteration. This continues, until either a solution is found, or the algorithm fails. On subsequent attempts, the algorithm will choose nodes that are most likely to give it success, balanced with the novel approaches of the passed nodes. An example of the searching structure of the algorithm can be seen in Figure 2.1.

The algorithm is able to interrupted at any time, with more accurate results for longer run-times.

Applications of the Monte Carlo Tree Search Algorithm

The algorithm is currently implemented in a variety of different systems today. Most notably, it is used by Google's AlphaGo, which has beaten Go grandmasters.[14] The algorithm is also used in the financial world to find optimal investments and distribute risk.[15] The algorithm excels when there may be a large number of outcomes to search, and it is uncertain initially which may be best.

Modified Monte Carlo Tree Search Algorithm

To improve the accuracy of the metrics, a simple step was added to the algorithm. When adding on the 'win' value, it is adjusted by a multiplier (in this project, this multiplier is kept at 0.99). For example, initially the win value will be one, but on each subsequent addition, it is 99 percent of the previous value.

This multiplier will help the algorithm tend towards paths where the completion state is less steps away. Consider a State where the algorithm must choose between a step where the victory is one step away, or one where it would be two. Both steps have equal wins and attempts. However, with the addition of the multiplier, the step where victory is one step away naturally has a higher 'win' value than the other, and so it is chosen.

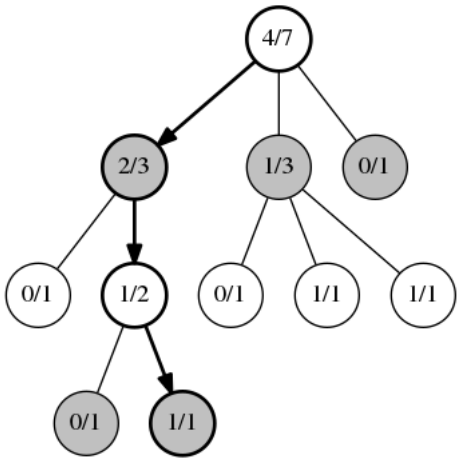

Figure 2.1: An example structure of Monte Carlo Nodes, taken from https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/

2.3 Metric Evaluation

During level generation, Reverse Simulation, and while running a Monte Carlo Tree Search on the created level, metrics are gathered about that level. These metrics measure basic statistics from the level.

Using the Metrics

Metrics are often used by procedural algorithms to determine if a generated level is at an appropriate difficulty. Horn et al. discuss various metrics such as leniency, linearity, density, variation and compression distance, which they use to evaluate the complexity and fun of a generated level.[16]

Metrics are gathered from throughout the level generation, Reverse Simulation, and Monte Carlo analysis. After metric analysis, levels can be generated in an online fashion to fit certain metric requirements, or batch generated offline, and selected based on the requirements of the level.

Evaluating Metrics

The evaluation of metrics is beyond the scope of this thesis, however it is hoped to provide a rough framework. Levels are play-tested by people, and given scores. These scores could be used by a regression algorithm to determine the optimum metrics to be generated for a desired response.

Supervised regression algorithms take a series of inputs, and, given an expected output, will find the line of best fit. When evaluating metrics, and the found line line maps- for example- the perceived enjoyment of a level, the high points on the line can be mapped to the metrics used to generate them. Levels can then be generated that try to fit this definition of enjoyment.

2.4 Iterative Development

Iterative development is used heavily in the modern development life cycle. It's use involves beginning a product in its simplest form; a minimum viable product, or MVP. This MVP should be both quick to implement and easy to extend.

The iterative process comes from improving the minimum viable product. As opposed to developing the entire product in one development cycle, or sprint, small objectives

should be tackled in a systematic fashion. As the minimum viable product is developed early, it can be clearly seen how changes to the system affect the entire process. Jorge Osorio found that using iterative development also made it "easier to incorporate changes to the process" of development.[17]

In projects of larger scale, iterative development is necessary so the developer has several set points in which to get user feedback. As metric evaluation cannot measure the human experience, user feedback would allow comparisons of the metrics to generated to the user results, and direct development moving forward. In "From England to Uganda: Children Designing and Evaluating Serious Games", the authors found that by evaluating their game with the target audience, they were able to gather information that would shape future development of the "Serious Game" created.[18]

This form of user feedback becomes increasingly important when selecting levels based on the emotions they may elicit in players. To ensure that these results are accurate, user testing is needed from an early stage of the development lifecycle.

Method

The methods used to implement and evaluate the Reverse Simulation Algorithm are discussed in this chapter. The method consists of implementing a basic algorithm, followed by iterative development to increase the complexity of the program. The iterative development allows the program to be functionally implemented before implementing additional complexity. This is necessary as the Reverse Simulation Algorithm becomes much more complex as more elements are implemented.

3.1 Development Methods

In implementing the base algorithm, a series of simple steps must be followed. A minimum viable product must be established to use the Reverse Simulation Algorithm on. Following this, the Reverse Simulation Algorithm must be implemented. At this stage, a simple algorithm can be used, such as random movements. After the level has been generated in reverse, the Evaluation Algorithm must be put into place; in this thesis, this evaluating algorithm is the Monte Carlo Tree Searching Algorithm. When these algorithms are in place, metrics can be gathered to determine the quality of the levels generated. With all of the factors in place, iterative development may now be used to make the generated levels more complex.

In each step of the iteration in the iterative development process, the following steps should be taken:

1. Evaluate new features to be added in relation to the Monte Carlo and Reverse Simulation Algorithm.
2. If the algorithms need to be adjusted, make the necessary changes.
3. Evaluate the levels generated by analysing their metrics.

4. Add the new features.
5. Evaluate the levels generated again. Compare results to results created earlier.
6. If at a user-testable stage, get user evaluation, and run machine learning to determine level quality.
7. If the differences are not significant, rework either the features added, or the metrics used; or remove the feature.

This allows development of a project where every decision is going towards answering the chosen research questions. Making changes based on metric evaluation allows examination of the metrics as a tool, and every iteration brings the program closer to a version that utilised its Reverse Simulation and Evaluation Algorithms to their fullest extent.

Step 6 is not followed in the development process of this project, as user feedback was not attained until after development was completed.

The sections below will describe the project after the development process.

3.2 Generating the levels

When generating a level, a set of blocks is generated. In the current algorithm, this set of blocks is placed randomly, based on the max of two randomly generated numbers being higher than that of a single randomly generated number. In a fully developed system, these blocks would be generated in a similar fashion to the rest of the reverse algorithm, where many unused squares would become blocks, with others left as options for the player to take. It was decided that in order to test the Reverse Simulation, as soon as possible however, the blocks in the current system are generated quickly and randomly.

When the level layout is in place, the player starts to move backwards from the end tile. Their movement is tracked so it can be verified after execution. In the current system, to ensure that the player can complete a level quickly- if the corrected solution is used - the player is prohibited from travelling to the same tile more than once.

Each movement that the player takes has a chance of being a move where an enemy would have been killed, had this level been played forwards. On occasions where this is calculated to be true, an enemy is created on the position where the player moved from

(backwards), and will also begin to move in reverse. If the player lands on this enemy again, or the enemy on the player, the turn must be rolled back, as these would signify an enemy killed before it should be, or the player killed. When no options remain for the player, the level is considered generated.

To ensure that the level is fair, the steps that the player and enemies took forwards are played out forwards. If the player makes it back to the ending position, the generation is considered successful.

Issues with Current Generation methods

When the walls are generated, it is possible that the player will be 'boxed in' into the corner of the level. The proposed improvements in wall placement should negate this issue.

In the current Reverse Simulation Algorithm, if a player walks into a dead-end created by walls on either side, level execution must end there. Levels also currently tend to err on the simple side, with several levels being completable in five or less actions. In the seeded data-set, four out of eleven generated levels are in this category. The algorithm could be changed to not let the player enter the same state twice, as opposed to the same tile, although this may cause levels to grow wildly difficult before a natural end-point is found.

As more complex features are added, such as projectiles and enemies that do not move in straight lines, the designer needs to be careful. These features need to be added in a way that they are reversible. For example, if an enemy turns when and only when it hits a wall, it must have turned onto its current path from some wall, or the level must have started before it could have entered its natural cycle. If the enemy could not have entered the state it was killed in, it will fail the verification. Similarly, if an enemy shoots a projectile every few seconds, there may be projectiles still in the air after the enemy dies. These would have to be calculated and added in before the enemy appears in the generation algorithm.

3.3 Metrics Used

After the level is created and evaluated, the following metrics are derived:

- *Minimum Steps:* The least actions found to be necessary to complete the level.

- *Minimum Complexity:* Complexity is measured as change in actions. Moving in a direction one turn, then turning another direction the next is additional complexity, but moving in one direction several times has a low complexity.
- *Number of Walls:* Counts the number of walls generated in the level.
- *Number of Enemies:* Counts the total number of enemies in the level.
- *Number of Two-Step Enemies:* Counts the number of enemies that move two steps every turn.
- *Number of Three-Step Enemies:* Counts the number of enemies that move three steps every turn.
- *Number of Monte Carlo Completions:* The number of times the Monte Carlo Algorithm found a solution to the level.
- *Time:* Time in which the evaluation was completed.

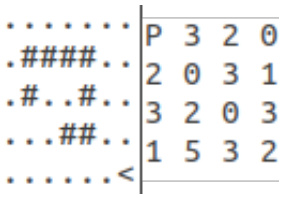

Figure 3.1: The level plan for a designed level. Map shown on the left, Entity location shown on the right.

3.4 Testing Method

To find out if the generated levels are valuable or not, play-tests were performed. Ten generated levels were created to be tested. These levels were first evaluated, and then given to a group of people, who completed the levels and scored them out of ten on 'Enjoyability' and 'Challenge'. These metrics were chosen to be a proxy measure of interest. A level with a good balance of both aspects would be interesting to play.

To have a base-line to compare results to, a set of eleven designed levels were also tested. These levels were entirely hand created, evaulated with the Monte Carlo Tree Search Algorithm, then tested using the same methods as the generated levels. The creation method of these levels can be seen in Figure 3.1.

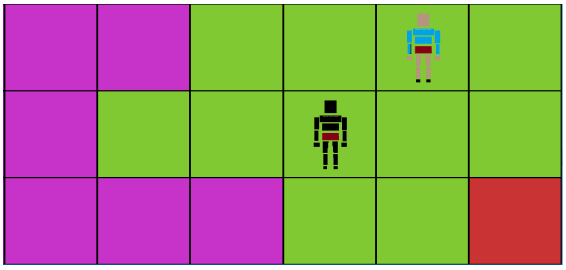

Figure 3.2: A level that cannot be completed. The enemy, moving left and right, will always be in a position where the player cannot kill it without first dying themselves.

In creating these levels, it was noted that while the game is very simple, the limited options available to the player make it easy to accidentally create levels that are impossible to complete. For example, a player may not ever be able to reach an enemy that is positioned in a certain fashion. An example of such a level is shown in Figure 3.2.

After collecting the results from the testers, the averages were found for each level, and compared to the metrics generated. The metrics were analysed to discover any metrics that may be particularly useful in discovering the enjoyability and challenge of a level, and to discover any correlation between the metrics themselves.

The test is also run on a set of designed levels, which are designed to be, in general, more difficult than the generated levels. This will allow a good comparison point for the metrics generated. In future projects, a level that players enjoy playing could be taken and analysed, and its metrics used as goals in generating a new level.

3.5 Experimental Set-Up

Metric Evaluation

All tests were ran on the same machine, with no other non-essential programs running on it. The tests were ran on a 2014 laptop with the following specs:

- OS: Windows 10
- CPU: Intel(R) Core(TM) i7-4600M CPU @ 2.90GHz
- Graphics Card: GeForce GT 750M Graphics Card
- RAM: 2 x 8GB

The program was written in Java, so JDK version 1.8.0_131 was used in development. The program was executed from a JAR file on the CPU.

The metric gathering evaluations ran each level over 1300 times, or until the level had been completed 9 or more times. These parameters were checked every 50 iterations. These limits were put in place instead of having a human operator to terminate each test.

Each attempt lasted either until the player was killed, or until they had taken 300 steps. This limit was again a compromise between evaluation time and correctness in the evaluations.

Thirteen-hundred was chosen as a compromise between giving the evaluation algorithm a chance to complete the level, and time efficiency. Similarly, if the level has been completed 9 times, then most likely a straightforward path has likely been found, and it is not necessary to continue to evaluate the level. As levels take different times to complete, it was deemed more fair to cut levels based on these parameters rather than execution time.

User Testing

Users were divided into two groups, A and B. Testers in Group A tested the generated levels first, and then the designed. Testers in Group B tested designed levels before the generated levels. Users were placed into these groups based on their demographic; an equal number of Game Development Students were put into each group as people non familiar with games, and it was ensured that the groups had the same number of participants.

Users tested the levels on machines familiar with them. Performance is not an issue with the game created, so any machine could be used for fair comparison. The user's local machine was used for convenience.

3.6 Threats to validity

The validity of the study will be analysed under Internal Validity, External Validity, and Experimental Reliability.

Internal Validity

The levels were split into two groups (generated and designed), to avoid bias from repeated testing, the testers were divided into two groups, one of which tested the generated levels first, and the other tested the designed levels first.

The study may suffer from learning effects, where testers were more used to later levels, and rated them differently because of their experience with the other levels.

When evaluating metrics, the experiment may suffer from instrumental change, where the longer the program was running, the more memory it used, causing longer run times of the program.

External Validity

To help mitigate potential selection bias, testers were chosen from two groups; half of the testers were from a group of peers in the Game Development Course at Queen's, and half were other students from a variety of Universities, studying a variety of subjects. The testers experience with PCG-G was varied, with the Game Development students being familiar with PCG-G, and the other students not being familiar with PCG-G, or with games as a medium. This should allow a comparison between people's perception of the new generative methods from the perspective of those familiar and unfamiliar with previous methods. However, as all testers were students, the experiment may not be generalisable to non-students.

It may not be possible to generalise this project to a larger scale. The results are based on the metric evaluation and generalised user feedback received on the project developed. Scaling issues in this project may be caused by the metric evaluation. Monte Carlo Tree Search in its current state can cause the evaluation time to grow exponentially on certain levels, and may need some optimisation to be extended to larger levels. The Reverse Simulation Algorithm will also needed to be expanded heavily for a larger program.

Experimental Reliability

To ensure valid results, a seed was used to ensure the same levels were generated. The same testing experience was given to each member of the group, with none of the testers having seen the levels before they played them. Testers may have compromised this by discussing their experiences to other testers before they played the levels.

Other factors could cause the runtime differences in levels, such as background processes on the computer, or a particular aspect of the used computer's architecture. All metric evaluations through an Evaluation Algorithm were performed on the same computer. The specifications of this computer are outlined in Section 3.5. User Tests were ran on a computer that the user was familiar with, as performance was a non-issue for the in-play project.

Results and Analysis

This chapter presents the results attained from the metric evaluations and the user tests. The results are reviewed and compared, and analysed for any flaws.

Before any evaluation was attained, it was ensured that each level had a completable path, and that each level could be reasonably completed by a human player. Both sets of levels were tested through human trial, with the designed levels additionally being algorithmic verified as completable.

4.1 Metric Evaluation Results

Metric evaluation was ran twice, once on the generated levels, and once on the designed levels. While running, the results were printed to a log file. The full results can be found in Appendix A, and are summarised below in Figures 4.1 and 4.2.

Generated Levels

The full results for the generated levels are presented in Appendix A in Table A.1.

A summarised version of the results can be seen in Table 4.1 below.

	Time (seconds)	Steps	Complexity	Walls	Enemies	2 step	3 step	Monte Carlo Wins
Mean	165.45	21.27	15.91	8.18	4.91	0.82	0.45	23.64
S/D	249.27	20.63	15.92	2.99	2.47	0.87	0.69	13.20
Mode	N/A	25	18	8	7	0	0	16
Lowest	1.084	4	3	0	1	0	0	6
Highest	782.707	74	58	12	8	2	2	51

Table 4.1: Summary of Metric Evaluation for Generated Levels.

While six of the levels in this set took under a minute to evaluate, two of the levels took a over 500 seconds, raising the total time for the evaluation to 30 minutes and 15 seconds. The

The levels generated are quite varied, as shown by the standard deviation of most metrics being similar to the mean. All levels were completed by the Evaluation Algorithm, however one level was only completed six times. The results for this level may not be as accurate as a level that found the solution more often.

Designed Levels

The full results for the generated levels are presented in Appendix A in Table A.2.

A summarised version of the results can be seen in Table 4.2 below.

	Time (seconds)	Steps	Complexity	Walls	Enemies	2 step	3 step	Monte Carlo Wins
Mean	3572.20	129.67	91.44	8.55	5.82	2.09	1.18	13.36
S/D	7566.42	106.15	75.98	4.57	2.75	1.45	1.08	18.11
Mode	N/A	N/A	10.00	6.00	4.00	2.00	1.00	2.00
Lowest	4.37	13.00	10.00	0.00	3.00	1.00	0.00	0.00
Highest	19606.34	254.00	191.00	19.00	12.00	6.00	4.00	49.00

Table 4.2: Summary of Metric Evaluation for Designed Levels.

Similarly to the generated levels, it was found that most designed levels took only a short time to complete, with seven of the levels completing in under five minutes. Two of the levels took 334 minutes each, each over 49 percent of the total time, raising the total evaluation time to 11 hours and 38 minutes. This brought the average time for the designed set of levels to an hour and three minutes.

It should be noted that in this set of levels, the Monte Carlo Tree Search Algorithm did not find a path through four of the levels. As such, their *Step* and *Complexity* scores are not used for calculating the statistics in Table 4.2 below.

Other levels' evaluations only found solutions once, which most likely will result in the metrics generated for those levels becoming mostly inaccurate.

Unlike in the generated levels, no mode value could be calculated for the *Step* and *Complexity*, with each value being unique.

If rounded to the nearest minute, the execution time's mode is 3 minutes.

4.2 User Test Results

Each user was taken individually to play the game from the generated and the designed levels. After completing each level, the user gave the researcher a score out of 10 for the Enjoyability and Challenge found in the level. These results are presented in Appendix B, in two groups, Group A and B, found in Appendix B.1 and B.2 respectively.

Group A

Group A completed the generated tests first, and then the designed set. Their full results are found in Appendix B.1. The results of the group are summarised in Figure 4.3

	Generated	Enjoyability	Challenge	Designed	Enjoyability	Challenge
1		6.4	4.6		6	4
2		5	4.2		6.6	6
3		5.8	4.4		6.4	6
4		5.4	4.6		5.6	4
5		1.4	1		5.6	5.4
6		2	1.6		7	6.8
7		2.6	2		5.8	4
8		4.4	3.6		6.2	5.8
9		4.4	4		6.6	5.6
10		5.4	5.4		6.8	5.4
11		5.6	4.8		7.2	7.4
Total Average		4.4	3.65		6.35	5.49

Table 4.3: Averages of Group A results.

Group A found the designed set of levels both more Enjoyable and more Challenging than the generated set of levels. The most Enjoyable level was the first level that they played, potentially caused by player nervousness with the reviewing system.

As expected, the level that was least Enjoyable and the least Challenging was Level 5, a simple level requiring two steps. Often when seeing this level, players would laugh at the simple nature of it. Despite dying on average 4.3 times on the generated level set, on average, the group found the levels quite easy, with Challenge scores under 5 being common. Any deaths were met with frustration at the player not paying attention, and often not attributed to the difficulty of the system.

Group A found the designed set of levels to be more Enjoyable and Challenging, with Level 11 being the most highest rated level. In this level, the goal was to add as many enemies as possible, to give players an option about how to take the level on. Players died on average 5.4 times in the designed levels. Again however, these deaths often did not attribute directly to a player's decision to call a level Challenging. Players were

more likely to call a level Challenging if they had to spend a time contemplating enemy movements, and thinking about their own moves.

Group B

Group B's full results are shown in Section B.2 of the Appendix, with summarised results shown in Figure 4.4. Group B completed the designed levels first, before moving to the generated levels.

Levels	Generated	Enjoyability	Challenge	Designed	Enjoyability	Challenge
1		6.8	5		6.8	3.4
2		6.4	3.8		6.8	5
3		6.4	5		6.8	6
4		6.6	4.4		6.6	5.4
5		2.6	1		7.2	5.6
6		3.2	1.8		7.6	7
7		4	2.2		6.6	4.8
8		5	3.2		7	6.4
9		6	4.8		7.4	7
10		6.4	5.8		7.2	6.2
11		6.8	5.6		7.8	8.2
Total Average		5.47	3.87		7.07	5.91

Table 4.4: Averages of Group B results.

Group B found that the designed set of levels again to be the more Enjoyable and Challenging, with the final level in particular being found to be the highest rated. In this set of players, there was no significant change in score between the beginning level and others, in either generated or designed level sets.

Interestingly, people in Group B were more likely to give Level 5 of the Generated Levels a higher Enjoyability score, with the average being 2.6 compared to Group A's average of 1.4. Group B on average gave higher results than Group A. This could be due to the level order creating a more interesting scenario, or could be due to the players in Group B being more biased towards higher scores.

Group B players died a similar amount to Group A, dying an average of 5.8 times on the Designed Levels, and dying an average of 3.7 times on the generated levels.

4.3 Discussion of Results

Metric Evaluation Results

Generated Results

As shown in the full results in Table A.1 in the Appendix, the generated levels were quite varied. In this section, the generated levels will be examined with respect to the metrics created for them.

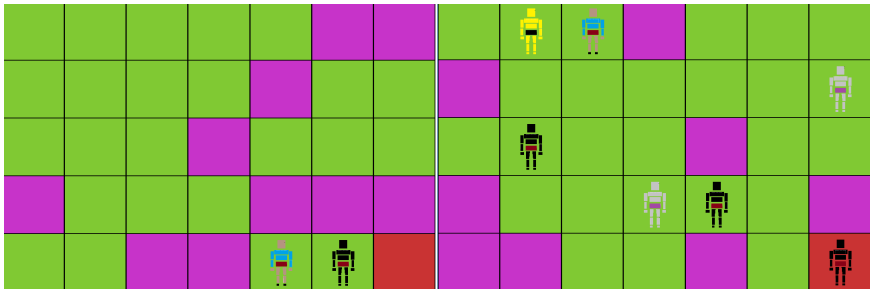

Figure 4.1: Level 5 and 11 from the Generated Levels.

In Figure 4.1, levels 5 and 11 of the Generated Levels are shown, respectively. Level 5 has the lowest associated metrics, with $Steps = 4$ and $Complexity = 1$. Level 11 had the highest associated metrics, with $Steps = 74$ and $Complexity = 58$.

As can be seen from Figure 4.1, these results clearly aren't perfect. Level 5 completed 51 times, however did not get a step count lower than 4. Level 11 also should not need a step count of 74, however given that the evaluation only completed six times, and given the additional length of the level, this error is more expected than the lack of correctness in Level 5.

Level 5 took one second to complete, and Level 11 took 506 seconds to complete. The time taken seems to increase drastically with the number of steps in a level, although this does not follow a strict formula. Level 7, for example, took 125 seconds, but has values of $Step = 5$ and $Complexity = 4$, whereas Level 9, with values of $Step = 10$ and $Complexity = 9$, took only five seconds to finish its evaluation.

Designed Results

As seen from the results in Appendix A, Figure A.2, the designed levels took a substantial amount of time to test, with two levels taking over five hours. In figure 4.2, Level 4 and 5 are compared. Level 4 is completed rapidly by the Monte Carlo Tree Search, finding a solution 49 times in six seconds. Level 5 takes a large time to complete, taking over five hours, and finding a solution two times.

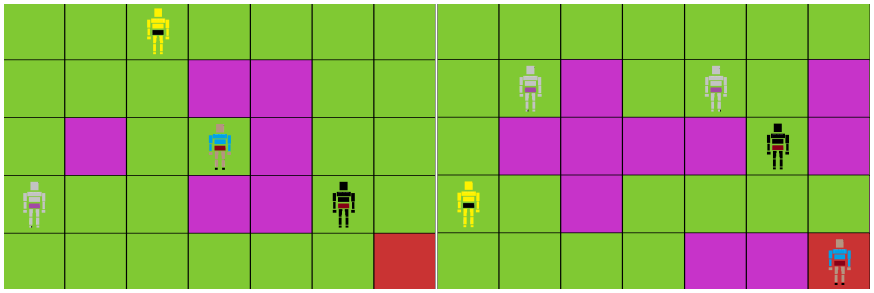

Figure 4.2: Level 4 and 5 from the Designed Levels.

An estimated reason for this disparity is the number of paths of completion each level contains. There is a strictly correct way to do Level 5, whereas Level 4 offers the player the option to take a few paths. As the player has fewer paths that they can take that will lead to success, the player takes actions that take a large amount of turns, but will result in the player dying. This causes memory usage to rise, causing exponentially longer run-time on the computer.

In the Designed Levels, there are two levels that do not find any solution. A solution may be found if the improvements in section 4.3 are implemented. Alternatively, the algorithm could run until a solution is found. The levels in which no solution was found begin with enemies one move away from killing the player, depending on how the player moves. This results in the levels immediately failing in a significant portion of runs, giving the levels a much smaller chance to succeed.

Comparison of Results

By comparing the results of the Metric Evaluation on the Generated Levels to the results on the Designed Levels, it can be seen if the metrics used can distinguish a difference between the two sets. Full results of the evaluations are compared, taken from Appendix A

A significant difference between the Generated Levels and the Designed Levels is the high levels that the evaluated metrics reach. Although several levels have similar values to the Generated Levels, it can be seen that the *Step* and *Complexity* values of the Designed Levels reach much higher values.

Similarly, in the apparent metrics, such as number of walls and all varieties of enemies, on average, the designed levels contained higher numbers. The levels are more difficult for the Evaluation Algorithm to complete, with the average number of Monte Carlo Wins dropping from 23.64 in the Generated metric set to 13.36 in the Designed metric set.

These metrics indicate that there is a quantifiable difference between the levels. The Designed Levels appear to be, on average, more complex than the Generated Levels. Improvements to the Generation Algorithm should be tailored to achieve metrics similar to those found in the Designed Levels.

Improvements to the Metric Evaluation

It is theorised that by adjusting the Modified Monte Carlo Tree Search Algorithm, the former could be more accurately solved, at the cost of the accuracy of the latter. The amount that the algorithm modifies the 'win' cost could be modified, (*i.e. multiplier changed from 0.99 to 0.95*) based on how many times the level has been completed. This would encourage the algorithm to initially discover small deviations that could cut down on steps, and gradually move to larger optimisations.

This may negatively impact larger levels, as they require more steps complete. Having a higher deviance rate early in a level may create levels that are not completed as often, and therefore have less accurate metrics. More accurate results could also be found by letting the algorithm run for a longer time.

In this project, the Monte Carlo Tree Search was implemented in a very generic fashion, with no information on the system. If system information is used in the Monte Carlo Algorithm, such as giving a state a much higher chance of being selected if it would result in an enemy being killed, or any action that would benefit the game being completed successfully, the algorithm could perform much more efficiently.

User Test Results

Full results for the User Tests are presented in Appendix B. In this section, the results are compared to each other, and analysed for any possible biases.

remake graphs with proper legends

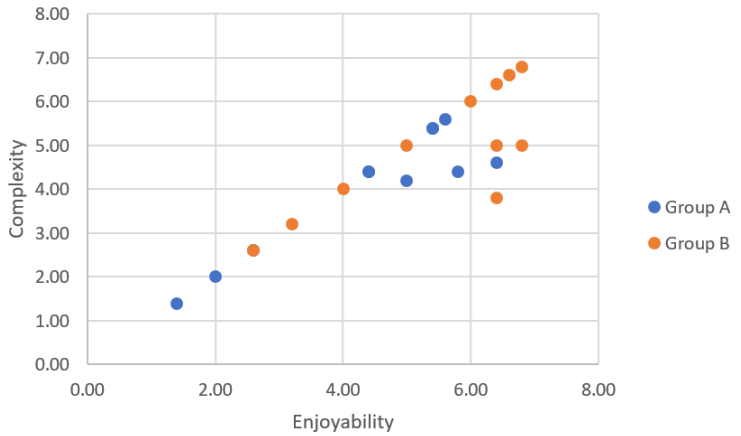

Figure 4.3: User feedback for Generated Levels: Enjoyability against Challenge.

In most cases, Group B gave higher scores to the levels, with the exception of the challenge rating for the first two Designed Levels. This may be due to the change of pace in the levels, with players rating the levels as easier initially before getting acclimatised to the levels. However, Group B contains some bias, with all other levels having a higher

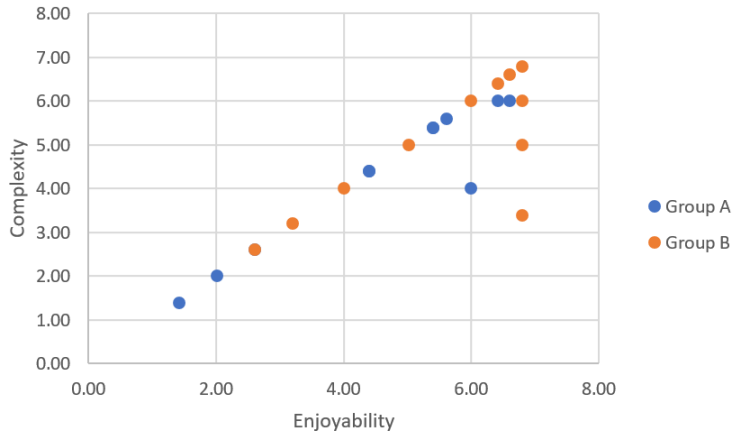

Figure 4.4: User feedback for Designed Levels: Enjoyability against Challenge.

average Enjoyability and Challenge metric from Group B.

This bias was tested by putting the values collected from the two groups into a Mann-Whitney U test. The test confirms the difference of scores between the two groups, with a p-value of 0.02926. This shows that the difference between the two groups is significant. In order to gather a more thorough understanding on why these metrics are higher, a further test should be performed on this subject, using a larger group to average any natural bias that may be present.

Figures 4.3 and 4.4 show the averaged results for the Generated and Designed Levels. It can be seen that there is a high correlation between how challenging a user finds a level, and how enjoyable they find it.

A Pearson Coefficient Correlation Test was used to find the correlation between the two values.[i] The test found an R value of 0.9248, which indicates that the two values have a strong positive correlation. The correlation shows that these two scores are not linearly independent for this game, so if metrics are discovered to model Enjoyability, that model could be transposed linearly to model Challenge.

Comparison Between Results

In this section results and metrics will be examined, and any relationship between the scores will be examined.

The levels found on average to be the most difficult were the levels that the had 0 Monte Carlo wins. An R value of -0.678 was calculated from the Pearson Correlation Coefficient method. This indicates a moderate negative correlation. This correlation remains similar when analysing the Generated and Designed Levels separately, with the Designed Levels having a higher correlation.

Similarly, there is a mild negative correlation between the number of steps that the Monte Carlo Algorithm took, and the Challenge reported by the players, with an R value of 0.5542. This indicates a mild positive correlation. This correlation is significantly stronger for the Designed Levels than for the Generated (R values of 0.6684 and 0.4601 respectively), although this could be due to the lack of data on the designed levels leading to fewer data points being usable.

A mild positive correlation can also be found between the number of enemies in a

[i] The calculator found on http://www.socscistatistics.com/tests/pearson/Default2.aspx was used

level and the reported Enjoyability, with an R value of 0.6576. This correlation is strong in the Generated Levels, although still mild for the Designed Levels.

As these metrics have a mild correlation, no one metric can directly determine how Enjoyable or Challenging a level may be. The following approaches could be taken to find a stronger correlation between metrics and a level's Enjoyability:

- Add additional metrics. Metrics such as independent paths to completion, and level cohesion could be gathered. These metrics may have a more linear relationship with level Enjoyability and Challenge.
- A Machine Learning Algorithm could be developed to find the optimal combination of metrics to approximate Enjoyability and Challenge. As the results have a strongly correlated relationship, a linear model for Enjoyability would also satisfy a Challenge model after a linear transformation.

4.4 Comments on Results

As 10 testers were used, and the only 11 designed and generated levels tested, the sample size for this experiment is not large. In future work, a larger sample size should be used to decrease testing error. As only 10 testers were used, the margin of error for the results found is 30%, with a 95% confidence rate.

When talking to players after they completed their level evaluations, several of them mentioned that the Designed Levels felt more 'purposed', a factor that was not accounted for in the evaluations. If future work is done, metrics should be added to account for the purposed feel of any designed level.

Another issue with the results is the incomplete Monte Carlo Tests for a couple of the levels. Inaccurate results are not a major concern, as a possible machine learning algorithm wouldn't care about accuracy, as long as other results were similarly inaccurate. However, incomplete results mean that that level could not be used at all in a machine learning algorithm. The levels that did not complete were also the levels that users rated as most Enjoyable and Challenging. The Monte Carlo Tree Search Algorithm should be further modified and given system information in order to predict results for these levels without being time prohibitive.

Conclusions

In this section, the research questions will be answered, and any possible future work will be discussed.

5.1 Does Reverse Simulation Create Interesting Gameplay Challenges

Reverse Simulation allows for the creation of unique challenges. Players found the generated levels fun, with the average Enjoyability score for the Generated Levels being over five in seven out of 11 levels. While the Generated Levels didn't meet the same scores as the Designed Levels, the metrics created are also quite low.

Further development of the Reverse Simulation algorithm should focus on creating more complex levels, that require many more steps to complete. By design, the current algorithm requires a maximum of 34 steps to complete any level, as the player should not travel to the same tile twice in the Generation Algorithm. This should be changed to another ending condition, to allow for levels where several dozen steps are needed, with more complex enemy patterns.

To further increase how interesting puzzles can be created from Reverse Simulation, the issue of randomness must be tackled. Even with deterministic action, the scope of the generation algorithm's actions are currently limited by the nature of Reverse Simulation. If an enemy is spawned in a certain position, its movements must allow it to move to that position to be killed if the level is played forwards.

Reverse Simulation in its current form shows promise, and the levels created from the simple algorithm demonstrated were found to be interesting. If extended into a more complete form, the Generation Algorithm could generate levels as interesting, or more interesting, to testers as the designed levels.

5.2 Usefulness of Metric Evaluations

Metric evaluations are essential for the Reverse Simulation method. As discovered by Spronck et al., PCG-G often suffers from difficulty scaling.[9]

Therefore, a method to evaluate levels is necessary. However, there are issues with the current system. The main issue is the time that evaluation takes. For most levels, the Evaluation Algorithm runs for a small time, and a few levels taking the majority of the time. These levels have very specific paths to completion.

A potential solution to this is to try and ensure that the levels generated have more than one specific path through them. Another solution is to improve the Monte Carlo Tree Search Algorithm so less looping paths are chosen: that is, to stop the player from going back and forth in a circle for 300 moves.

The metrics currently gathered are useful for estimating the Enjoyability and Challenge of a level, as shown in Section 4.3, but may not measure how 'purposed' a level felt. In order to measure these factors, additional emphasis should be placed on gathering metrics that measure level Aesthetics, or any factor that may tell us about the emotions that a level elicits.

Metrics are essential in the current form of Reverse Simulation, in order to inform a level selection algorithm which levels might be interesting, and which might be boring. However, if the Reverse Simulation algorithm is developed to a point where all levels created are unique and interesting, the metric evaluation would not be necessary.

5.3 A Taxonomy for Reverse Simulation

This section will examine a taxonomy for PCG-G, as presented by Hendrikx et al. in "*Procedural Content Generation for Games: A Survey*".[5] These elements will be addressed as directed in Alexander's "*A Pattern Language*":[19] from a top-down approach. Large design elements will be discussed initially, and then given detail through the discussion of the smaller elements. This taxonomy has been created as an overview of possible uses for Reverse Simulation.

5.3.1 Game Design

Reverse Simulation of an entire project should begin with a Directing Algorithm. This algorithm will start with a blank slate, and, starting from the point of a topic or genre, create and end point, and work backwards, adding in components as needed.

System Design

System Design is the overall view on how the game is played. For example, a turn-based strategy game will have a different system design to a real-time action game. Sections of System Design are given to the Directing Algorithm before play begins. This provides a focus for the algorithm to start the Reverse Simulation process and start generating the Game World, along with other Game Scenarios and Systems. Generating System Design is one of the main reasons why Reverse Simulation is necessary. If generated forward, some games would be difficult to balance, such as competitive Real-Time-Strategy games. If the System is changed in response to what is necessary to create balance as the game is generated in reverse, this becomes a non-issue.

Write about elements of System Design

Game World

The Game World is "the design of a setting, story, and theme" of a game.[20] The design of the Game World should be based on the System Design, and choose something thematically appropriate. For example, a game in which the mechanics are built around a serious Real-Time-Strategy would not fit a World Design that a mobile-based puzzle game would benefit from.

As World Design is so varied depending on the type of game generated, the choice is not entirely suited to Reverse Generation by itself, and may be better generated through a neural system and given to the Directing Algorithm.

5.3.2 Game Scenarios

Game Scenarios describe the events of the game. This is the space in which the game takes place. Scenarios contain the Story of a game, the Levels that comprise it, and the Puzzles that comprise those Levels.

Story

Story is a difficult element to generate in any form of PCG-G, and would require significant development of secondary systems to make a generated story coherent. A Directing Algorithm could perhaps be provided hooks for a story, where parameters are given to

it that represent certain defined scenarios that should happen. These scenarios could then be expanded into a story by a developer. This hook system however, would be more similar to a standard PCG-G generated feature; designing content around a given template.

Levels
Level generation is extremely popular in standard PCG-G algorithms. In most areas of System Design, a Level is created from a set of Game Systems and Spaces, which should have a defined ending position. From that ending position, a more complete version of the algorithm used in this project should be used.

If the System Design of a game is more open than a traditional level system, it should be broken into interconnecting segments by the Directing Algorithm. To achieve these segments, several iterations of Reverse Simulation should be run, one from every 'connection point' to simulate the desired experience to get to that point. A compromise then needs to be created so the best fit for those segments can be found. If the segments can be parameterised, a regression algorithm would answer this question.

Puzzles
Puzzles are possible to generate through Reverse Simulation, and generating puzzles is the main use for Reverse Simulation in games today.[11] Starting from a solved state, which is created from a Game Space procedure, the puzzle is moved back and tested until the Directing Algorithm finds a sufficiently complex starting state.

5.3.3 Game Systems

Game Systems are how the Game Spaces and Game Bits come together. This could be the simulation of interactions between flora and fauna, or a city simulation. Ultima Online famously generated a Game System using procedural models, which modelled a large ecosystem with complex food chains.[21] This ecosystem failed, as they failed to predict that players would kill every creature that they came across. Reverse Simulation without Metric Evaluation would likely not solve this problem, as Reverse Simulation simply describes a single path, not all potential paths that a player may take. Through Metric Evaluation, however, the generated system could be evaluated for weaknesses. The designer could then plug these weaknesses, and test the system again.

Entity Behaviour is an interesting problem to tackle through Reverse Simulation. If introducing a new Entity, the Directing Algorithm should assign it features based on a

need in the Game Space. The game space could then be stepped back, with the entity placed into it. The space could then be evaluated, and the Directing Algorithm could choose whether or not to alter the features of the entity. The features which the Directing Algorithm chooses should be small, isolated elements, which would represent a simple action that the entity can do, or have done to it, such as attack, be attacked, move by walking, etc. Features should also be able to be adjusted by parameters, to adjust them. For example, the speed the entity walks. These features and parameters would allow complete customisation over the entity. A set of Game Bits should be chosen to match the parameters chosen.

Other systems should be more simple to model through Reverse Simulation. Cities can be modelled efficiently through a combination of Game Space Techniques.

5.3.4 Game Space

Game Space- the maps in which a game takes place- is one of the most common elements to be generated through normal PCG-G techniques. In this project, a simple example of Game Space was generated through Reverse Simulation. It should be possible to create a wide array of Game Spaces based on the principles of Reverse Simulation. To build Game Space, first the final state of the game space should be created. Where does the Directing Algorithm want this space to end? It could be entering a building, completing a dungeon, or even one of several objectives. The space should be played backwards from each space, with difficulty added in as play continues.

5.3.5 Offline and Online generation

When deciding whether content should be generated online or offline, consider the content being generated. In generating content Online, the generated content should provide a relatively similar experience each time. As shown by Togelius et al., unexpected content in a PCG-G can be off-putting.[8] Therefore, the Game Design and some elements of the Game Scenarios should be left to offline generation. Once the baseline for the game is created, other content can be generated online, unless time prohibitive. This also matches Togelius' findings; which state that online content is generally best suited to be player-generated content, with the bulk of the PCG work being done offline.

5.3.6 Game Bits and Derived Content

Reverse Simulation is not suited to generate game components. If needing to generate Textures, Vegetation, or Sounds, traditional generation methods should be used. The Directing Algorithm could algorithmically pick appropriate sections of these to use when generating systems.

Similarly, Derived Content such as leaderboards is not suited to being generated through Reverse Simulation. This content should be designed, and then picked by a Directing Algorithm while generating appropriate Game Systems.

5.4 Future Work

If Reverse Simulation is to be developed into a larger project, the following challenges must be addressed:

1. The differences between generated and designed levels must be analysed further. Interesting puzzles can be made through Reverse Simulation, however more research needs to be done into what differentiates the feeling of PCG games from that of designed games.

2. The evaluation method needs to be adjusted. Metrics are essential to the presented Reverse Simulation method, and a more efficient Evaluation Algorithm should be developed for any created game. This Evaluation Algorithm should use system information and make informed decisions based on the information given.

3. Reversible complexity must be challenged. As games become more complex, Reverse Simulation becomes exponentially more difficult. For example, if enemies can fire projectiles, in some occasions a projectile will be in existence relatively before the enemy is un-killed. Then this enemy must move into a position where it may have fired that projectile at the appropriate time. Doing these simulations may be expensive, with large amounts of reversing the algorithm if a scenario is found to be a dead end, or not to match with the expected end state when going through verification.

4. Randomness in Reverse Simulation should be considered. The ways a pseudo-random decision can be modelled in reverse need to be tackled.

If these challenges are addressed in future projects, a Reverse Simulation solution can be used for a huge variety of topics. Levels could be generated to closely match a level, with some deviation. This would be an alternative to the currently popular 'pattern' system, where the maps are designed, with sections replaced with generated terrain.

Section 5.3 provides a large amount of topics to look into for any future development. The taxonomy is suggested to be a set of topics possible to create through Reverse Simulation. Many of them would have their own set of challenges to implement.

Bibliography

[1] J. Ruggill, K. McAllister, R. Nichols, and R. Kaufman, *Inside the Video Game Industry*. November 2016.

[2] Superannuation, "How much does it cost to make a big video game?." `https://kotaku.com/how-much-does-it-cost-to-make-a-big-video-game-1501413649`, 2014.

[3] A. Iosup, "Poggi: generating puzzle instances for online games on grid infrastructures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 158–171, 2011.

[4] H. Yihua, *Procedural Game Environment Generation in Independent Game Development: Case Study and Simulation Approach*. PhD thesis, Auckland University of Technology, 2015.

[5] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, no. 1, p. 1, 2013.

[6] S. Bakkes, S. Whiteson, G. Li, G. V. Vişniuc, E. Charitos, N. Heijne, and A. Swellengrebel, "Challenge balancing for personalised game spaces," in *Games Media Entertainment (GEM), 2014 IEEE*, pp. 1–8, IEEE, 2014.

[7] A. Isaksen, D. Gopstein, and A. Nealen, "Exploring game space using survival analysis.,"

[8] J. Togelius, E. Kastbjerg, D. C. Schedl, and G. N. Yannakakis, "What is procedural content generation?: Mario on the borderline," in *PCGames '11*, 2011.

[9] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Difficulty scaling of game ai," 2004.

[10] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[11] "Reverse sudoku generator? exists? : Software." `http://forum.enjoysudoku.com/reverse-sudoku-generator-exists-t4067.html`, May 2006.

[12] T. Yagi and D. Kitazawa, "Application of reverse simulation to the pollution problem in a bay," in *OCEANS, 2012-Yeosu*, pp. 1–6, IEEE, 2012.

[13] A. Reynolds and G. McKeown, "Construction of factory schedules using reverse simulation," *European journal of operational research*, vol. 179, no. 3, pp. 656–676, 2007.

[14] S. Nair, P. Kundzicz, K. An, and V. Kumar, "Monte carlo tree search and alphago." `http://www.yisongyue.com/courses/cs159/lectures/MCTS.pdf`.

[15] O. Jauri and T. Penttilä, "No limits to using monte carlo for modelling." `http://www.theactuary.com/features/2013/01/no-limits-to-using-monte-carlo-for-modelling/`, 2013.

[16] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of procedural level generators in the mario ai framework," 2014.

[17] J. A. Osorio, M. R. Chaudron, and W. Heijstek, "Moving from waterfall to iterative development: An empirical evaluation of advantages, disadvantages and risks of rup," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pp. 453–460, IEEE, 2011.

[18] G. Sim, J. C. Read, P. Gregory, and D. Xu, "From england to uganda: Children designing and evaluating serious games," *Human–Computer Interaction*, vol. 30, no. 3-4, pp. 263–293, 2015.

[19] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press, August 1977.

[20] B. Brathwaite and I. Schreiber, *Challenges for Game Designers*. Rockland, MA, USA: Charles River Media, Inc., 1 ed., 2008.

[21] L. Hutchinson and UTC, "War stories: Lord british created an ecology for ultima online, but no one saw it." https://arstechnica.com/gaming/2017/12/an-afternoon-with-lord-british-creating-ultima-onlines-unknown-virtual-ecology/, Dec 2017.

Appendices

Metric Evaluation Results

Levels	Time (seconds)	Steps	Complexity	Walls	Enemies	2 step	3 step	Monte Carlo Wins
1st	54.53	25	18	0	8	2	0	16
2nd	782.707	21	15	10	7	2	1	11
3rd	51.167	36	25	8	6	1	0	33
4th	64.387	25	18	8	5	1	0	11
5th	1.084	4	3	10	1	0	0	51
6th	50.48	4	3	8	2	0	0	26
7th	125.381	5	4	12	3	0	0	34
8th	20.439	6	4	9	2	0	2	16
9th	5.569	10	9	9	7	0	0	30
10th	157.74	24	18	8	7	1	1	26
11th	506.428	74	58	8	6	2	1	6

Table A.1: Results of Monte Carlo Tree Search Metric Evaluation on the Generated Levels.

Levels	Time (seconds)	Steps	Complexity	Walls	Enemies	2 step	3 step	Monte Carlo Wins
1st	22.335	43	31	0	4	2	1	47
2nd	10.341	21	10	7	4	2	0	14
3rd	4.369	13	11	19	3	1	0	18
4th	6.002	17	10	6	3	1	1	49
5th	19606.338	223	168	10	4	2	1	2
6th	334.888	NULL	NULL	6	8	2	1	0
7th	467.488	142	104	10	6	1	1	9
8th	408.311	254	191	11	8	3	2	2
9th	241.065	202	138	9	5	2	1	1
10th	18108.081	252	160	8	7	1	1	5
11th	84.991	NULL	NULL	8	12	6	4	0

Table A.2: Results of Monte Carlo Tree Search Metric Evaluation on the Designed Levels.

User Test Results

B.1 Group A

Group A	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10	Level 11
	6	4	5	4	2	4	4	6	4	5	4
	8	4	6	7	1	1	1	3	4	5	8
	5	5	6	5	1	2	4	5	5	6	5
	5	5	5	4	2	2	2	3	4	5	5
	8	7	7	7	1	1	2	5	5	6	6

Table B.1: Group A Enjoyment of Generated Levels

Group A	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10	Level 11
	5	2	4	2	1	2	2	3	4	4	3
	5	6	3	7	1	2	3	4	4	7	8
	3	5	6	4	1	2	3	5	5	7	5
	6	3	5	5	1	1	1	3	4	4	4
	4	5	4	5	1	1	1	3	3	5	4

Table B.2: Group A Challenge of Generated Levels

Group A	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10	Level 11
	7	6	7	6	6	7	5	7	6	7	6
	7	8	8	6	7	9	8	9	8	7	8
	5	7	6	5	5	6	5	4	5	6	8
	6	7	6	7	5	7	6	5	7	7	5
	5	5	5	4	5	6	5	6	7	7	9

Table B.3: Group A Enjoyment of Designed Levels

Group A	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10	Level 11
	2	5	6	3	3	8	2	7	4	5	7
	7	6	7	4	7	10	6	9	7	7	6
	5	8	6	4	8	7	4	4	6	5	7
	4	6	6	5	5	5	4	5	7	5	8
	2	5	5	4	4	4	4	4	4	5	9

Table B.4: Group A Challenge of Designed Levels

B.2 Group B

Group B	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10	Level 11
	7	7	7	8	4	5	5	7	7	8	9
	8	7	7	8	4	5	6	7	8	9	8
	4	4	5	4	1	2	3	3	4	4	4
	7	6	6	6	2	2	3	4	5	5	6
	8	8	7	7	2	2	3	4	6	6	7

Table B.5: Group B Enjoyment of Generated Levels

Group B	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10	Level 11
	4	6	7	6	1	3	3	4	7	8	6
	6	4	7	5	1	3	4	5	6	7	6
	3	4	4	3	1	1	1	4	5	5	5
	5	2	3	4	1	1	2	2	3	4	6
	7	3	4	4	1	1	1	1	3	5	5

Table B.6: Group B Challenge of Generated Levels

Group B	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10	Level 11
	5	6	6	6	7	8	8	8	8	8	9
	7	8	8	7	9	9	7	8	8	9	9
	5	4	4	5	5	6	4	4	5	4	5
	9	8	8	7	7	7	6	7	8	7	7
	8	8	8	8	8	8	8	8	8	8	9

Table B.7: Group B Enjoyment of Designed Levels

Group B	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10	Level 11
	3	4	8	7	8	9	6	8	7	8	9
	4	7	7	6	7	8	5	7	8	7	9
	3	4	5	4	4	6	4	5	7	5	7
	3	4	5	5	4	5	4	6	7	5	7
	4	6	5	5	5	7	5	6	6	6	9

Table B.8: Group B Challenge of Designed Levels

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC4006 – RESEARCH AND DEVELOPMENT PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Research Dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Kelvin McClean Student Number: 40111604

Project Title: Automatically Produce Interesting Computer Game Challenges Using Reverse Simulation

Supervisor: John Bustard

Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

By submitting your dissertation you declare that you have watched the video on plagiarism at http://www.qub.ac.uk/directorates/sgc/learning/WritingSkillsResources/Plagiarism/ and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.

Student's signature Kelvin McClean *Date of submission* 04/05/2018