UNIVERSITY OF NEVADA RENO

Evolving Cellular Automata Rules for Maze Generation

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering

by

Chad Adams

Dr. Sushil Louis/Thesis Advisor

May 2018

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

Chad Adams

Entitled

Evolving Cellular Automata Rules for Maze Generation

be accepted in partial fulfillment of the
requirements for the degree of

Master of Science

Sushil Louis, Ph.D., Advisor

David Feil-Seifer, Ph.D., Committee Member

Ramona Houmanfar, Ph.D., Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May, 2018

UNIVERSITY OF NEVADA RENO

Abstract

Maze running games represent a popular genre of video games and the automated design of playable mazes thus provides an interesting research challenge for computational intelligence research in games. The ability to automatically create levels would be of particular benefit to maze running games as they require a large number of mazes per game. The automatic generation of mazes removes a large time bottleneck in maze running game development, the manual creation of mazes by expensive designers. We therefore attack the problem of automatically designing mazes using a genetic algorithm.

However a genetic algorithm requires measuring the quality of the generated mazes in order to function. This poses a problem as human players have differing opinions on what makes mazes enjoyable. We therefore first have a human user acting as a fitness function guide an interactive genetic algorithm that generates mazes in order to collect data on what types of mazes human players find interesting. Analysis of the generated mazes by our human experts led to two properties that appear in mazes that players found interesting; path length and number of dead ends. We then created fitness functions based upon these properties to guide a non-interactive genetic algorithm.

To show the viability of our approach we compared genetic algorithm generated mazes against the global optimum found by exhaustive search on a small, though non-trivial maze generation problem. The genetic algorithm generates mazes on average within 90% of optimal, producing a maze of such quality or better 65% of the time while having a floor level of performance at about 80% of optimum. We then expanded our experiment by creating a new probabilistic representation, increasing the overall search space for our genetic algorithm. The probabilistic representation produced mazes that achieved higher fitness scores than even the optimum for our previous experiment. These results provide evidence that genetic algorithms are a viable approach to automatic maze generation in maze games.

"I'm So Meta Even This Acronym."

Randall Munroe (XKCD)

Acknowledgements

This work was partially supported by the Office of Naval Research under Grant N00014-15-1-2015

Contents

List of Tables

List of Figures

Abbreviations

GA	**G**enetic **A**lgorithm
IGA	**I**nteractive **G**enetic **A**lgorithm
CA	**C**ellular **A**utomata

Chapter 1

Introduction

The generation of mazes through automation presents an interesting and difficult research challenge in computational intelligence. Current methods of maze generation for maze running games involve expensive human designers spending a non-trivial amount of time creating each maze. Automated maze generation circumvents the need for human designers to spend that time.

Automated maze generation falls within the larger category of procedural content generation which focuses on automatically creating content for many different game and simulation genres. The automatic generation of content would be especially useful for maze running games due to the many different mazes that need to be included in each game, or with the more modern approach of having a continual stream of mazes added periodically through updates. The ability to generate mazes quickly would greatly increase the number of mazes for players to run through.

Therefore we attack the problem of automatic maze generation using a genetic algorithm that generates mazes. Specifically the genetic algorithm evolves rules for a 2-dimensional cellular automata and repeated rule application produces maze like patterns. Cellular automata describe a discrete model that consists of a grid of cells, each with a finite number of states, and a set of rules applied to all cells that dictate when cells alter their states. More detail on cellular automata will be provided later in this manuscript. However crafting fitness functions to measure the quality of our evolved cellular automata rules presents a unique challenge due to the subjectivity of maze quality.

The subjective nature of what mazes human players would find interesting stands as a key challenge for automated maze generation. Some players prefer quick mazes while others prefer longer and more complicated mazes to run through. Mazes that are too simple, such as the maze in Figure 1.1, may bore the player. Conversely if mazes become

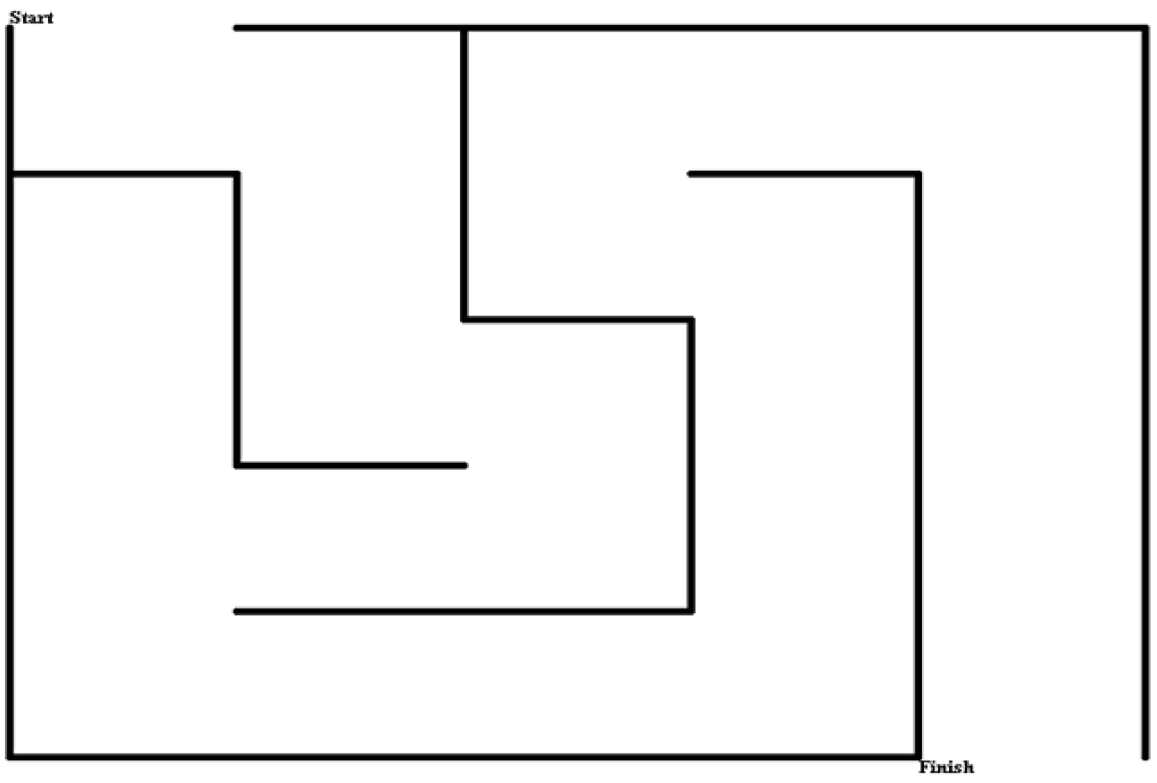


FIGURE 1.1: Simple Maze: not interesting due to being too easy [1]

too difficult (Figure 1.1) the player may lose interest or become frustrated. Therefore in order to formulate fitness functions for a genetic algorithm we required human input on what types of mazes they found interesting.

First, we had human users evaluate the "interestingness" of mazes for an Interactive Genetic Algorithm (IGA) that generated mazes. That is, a human user rates the IGA generated mazes and thus serves as the genetic algorithm's subjective fitness function. Then we had human users play the resulting maze game and rate the top performing mazes from the first, middle, and final generations on a scale of one to five. From the user ratings we were able to extract two properties that were correlated with mazes that users rated as interesting; path length and number of dead ends [2]. We also found that users preferred mazes with fewer different paths from start to end. This work was published in the Proceedings of the Genetic and Evolutionary Computation Conference Companion of 2017 on pages $85 - 86$ [2].

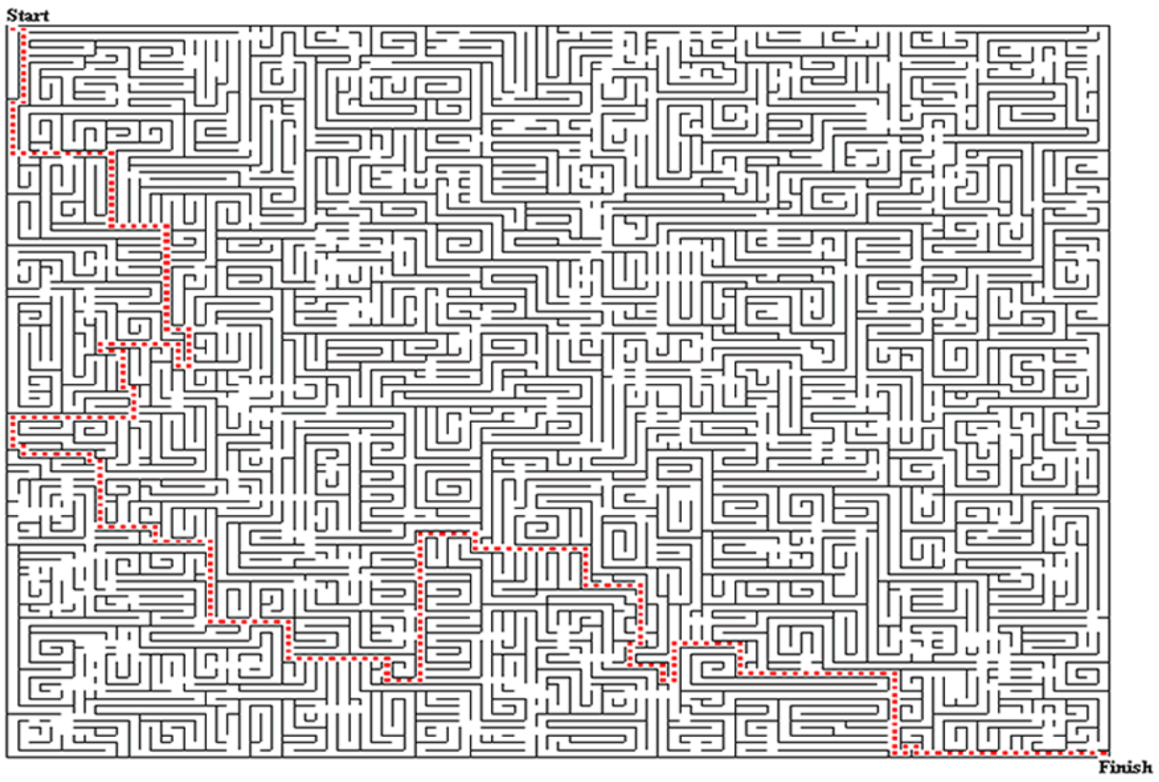


Figure 1.2: Difficult Maze: not interesting due to being too difficult [1]

We then constructed three fitness functions F1, F2, and F3 based upon the results from our interactive genetic algorithm experiment. F1 assigned higher fitnesses to mazes with longer shortest solution paths. F2 gave higher fitnesses to mazes with greater numbers of dead ends. F3 was the sum of F1 and F2. The rule space allowed us to use exhaustive search to find the optimum result for each of our fitness functions as the search space has only 262144, or 2^{18}, possible cellular automata rules in our representation. This allowed us to see how well our genetic algorithm did in comparison to the optimum. The evolved mazes achieved on average 90% of optimum fitness, achieving such quality about 65% of the time while having a floor of 80% optimum fitness.

Next, to allow our genetic algorithm to achieve higher fitness values, corresponding to more interesting mazes, we changed our representation. Our new representation used probabilistic rules to determine whether a given cell would change state. This increased the size of the search space to 2^{126} but the GA was then able to generate mazes with longer path lengths and more dead-ends, that is, mazes that were better than the best possible mazes generatable with our original representation. This result gave us confidence that our approach would scale into large search spaces that are not amenable to exhaustive search. This work was published in the 2017 IEEE Symposium Series on Computational Intelligence (SSCI) [3].

For comparison, the mazes created by our approach achieved overall fewer number of solution paths than achieved in Ashlock's work [4], having on average two distinct solution paths, while Ashlock had four solution paths in most of their generated mazes. Our prior user studies indicate that users may prefer mazes with fewer solution paths, since such mazes tend to be more challenging. The results provide evidence that genetic algorithms evolving cellular automata rules presents a promising approach for automatic maze generation.

1.1 Thesis structure

The next chapter provides background information on maze running games, the approaches used in this research, and related work. We first provide an overview of maze running games and their challenges for automatic maze generation. Next we explain both genetic algorithms and interactive genetic algorithms in detail. Finally we provide a summary of past academic research into automatic maze generation.

Chapter 3 describes our methodology. We represent our mazes through cellular automata rules, whereby repeated application of these rules to a grid results in a maze-like pattern. Employing a region merging algorithm after generating the maze-like pattern ensures the resultant maze contains at least one path from the starting point to the end. We applied the technique of interactive genetic algorithms to find and extract key maze features that human players found “interesting”. The results from our IGA experiment allowed the creation of the fitness functions for our non-interactive genetic algorithm. The first GA experiment utilizes the same cellular automata rule representation as the IGA experiment. The second GA experiment alters the cellular automata rules representation, changing the binary cellular automata rules into probabilistic rules.

Chapter 4 lists the results from our genetic algorithm experiments. We cover the enjoyment scores for the mazes generated by our IGA experiment, the features analyzed in the IGA mazes that correlate to maze 'interestingness', and the quality of solutions generated by our two GA experiments. Maze exemplars from each experiment are also shown and discussed. Comparisons between the binary and probabilistic representations are covered, with the probabilistic representation performing better.

Finally Chapter 5 covers our conclusions from our work. We discuss what our results from each experiment indicate and how the results may be applied in future work. We close with an exploration of possible extensions of the methods presented in this manuscript for future experiments.

Chapter 2

Background

This chapter first details gameplay of maze running games and the design methods for mazes. Next we explain the techniques utilized in our work. We also provide an introduction to genetic algorithms, cellular automata, and their associated terminology. Finally, we review work related to automated maze generation.

2.1 Maze Games and Design

Maze running games have players speed through a maze, attempting to locate a goal or exit. The challenge and fun for maze running games come from the design of the maze. Obstacles, time limits, or hostile agents can add to the challenge of the core maze running game. Past maze running games used human designers to construct mazes by hand. Manual design takes a non-trivial amount of time and requires an expensive designer to perform. Automatic maze generation becomes the natural solution to this issue. In this thesis, we use genetic algorithms to evolve mazes for maze running games.

2.2 Genetic Algorithms

Genetic algorithms (GA) are a heuristic search technique inspired by natural evolution and reproduction and have been often used to attack poorly understood optimization and search problems [5]. First introduced by John Holland, genetic algorithms stemmed from his research on cellular automata in 1975 [6], and fall into the broader research domain of computational intelligence.

GAs attempt to solve problems with an iterative process starting with a population of randomly initialized individuals. Each individual encodes a candidate solution and a fitness function evaluates each individual to rate how well that individual solves the problem. After each individual has been rated by the fitness function, selection chooses individuals with a probability proportional to their relative fitness for further use by the GA. Then crossover recombines pieces of an individual's chromosome with another

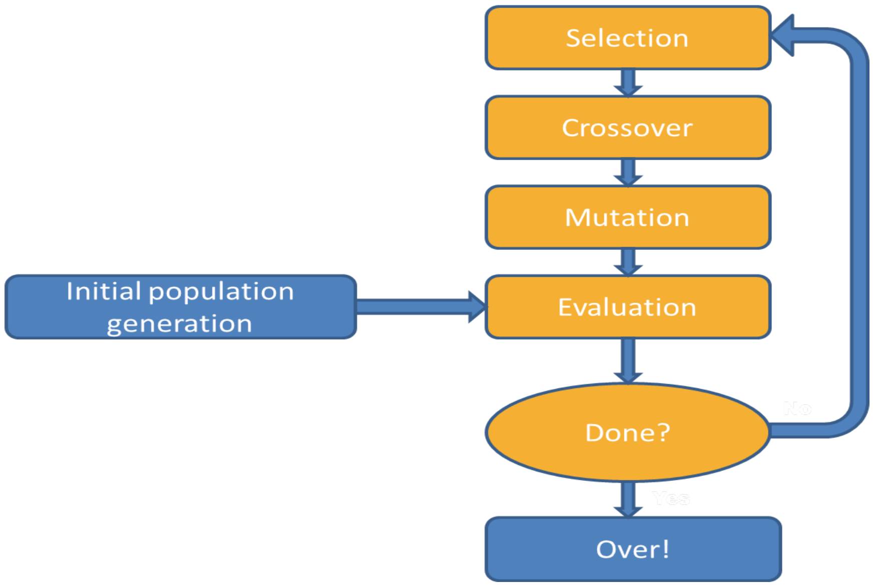


Figure 2.1: Genetic algorithm example diagram

individual to create children in an attempt at achieving individuals with higher fitness ratings. Next mutation iterates through each candidate solution, having a small chance to mutate the candidate solution in order to help prevent premature convergence on a solution with lower than optimal fitness. Fitness functions work by having one or more quantifiable measures on how well a candidate solution performs on the given problem. However, crafting fitness functions for genetic algorithms can run into a key challenge of having a subjective or non-quantifiable fitness measure, like with automatic maze generation. Interactive genetic algorithms are one option to circumvent the issue by replacing the fitness function with human users rating the population of chromosomes.

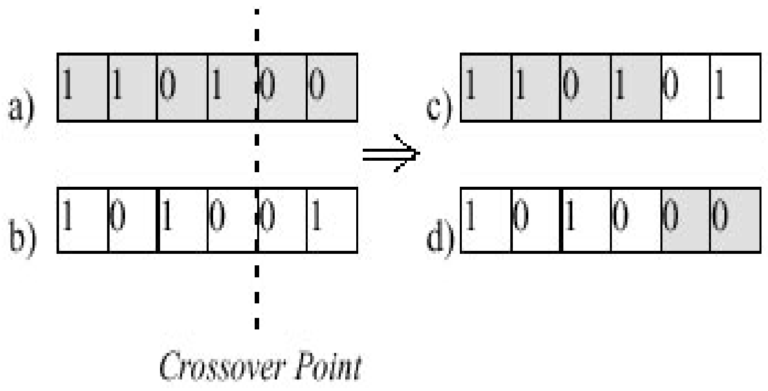


FIGURE 2.2: Single point crossover example

2.2.1 Interactive Genetic Algorithms

Interactive genetic algorithms (IGA) differ from conventional GA's in that they exchange the conventional GA's fitness function for human user ratings to guide evolution [7]. The first interactive genetic algorithm appeared in L. R. Smiths study on designing biomorphs, where an interactive genetic algorithm was used to evolve the forms of Fourier series biomorphs [8].

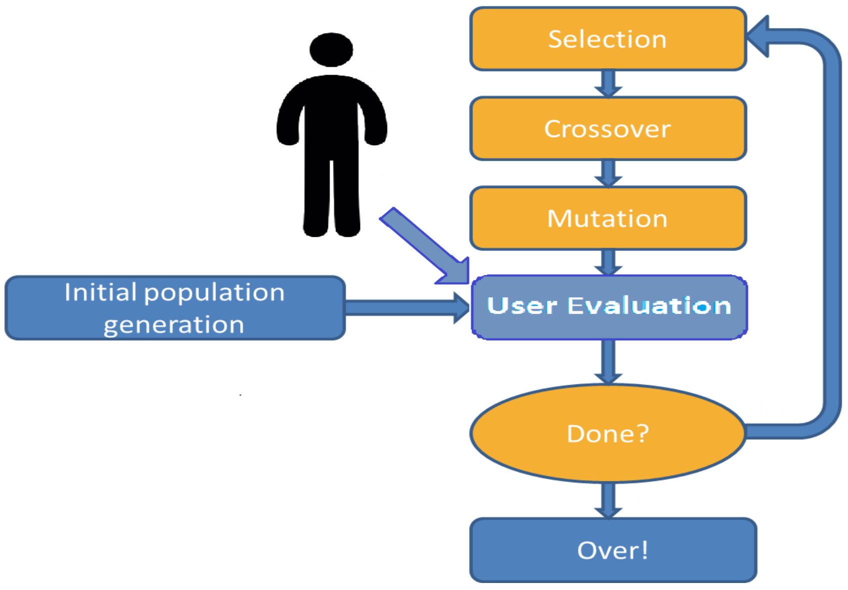


FIGURE 2.3: Interactive genetic algorithm example diagram

Utilizing human interaction as the source of fitness for the genetic algorithm allows the incorporation of subjective qualities in the fitness measure. However, interactive genetic algorithms must carefully balance how much the human users are asked to do, as humans can become fatigued [7].

Cardamone, Loiacono, and Lanzi applied an interactive genetic algorithm to race track generation for the TORCs racing simulator [7]. Users would provide a one to five star rating to the generated tracks that the IGA would use to derive fitness in order to drive evolution. IGAs have also been used to generate Mods for 3D game objects and textures by D.-M. Yoon and Kim K.-J. [9]. However these works remain only peripherally related to our core problem of automated maze generation, the next section discusses previous work related to our own.

2.3 Related Work

The papers by Ashlock [4] and Pech [10] stand as the works closest to our own. Ashlock's work explores a genetic algorithm based approach for generating maze levels. They compare multiple methods for representing evolvable mazes. Of note for our work, their positive indirect representation functions similarly to our cellular automata approach. However Ashlock's representation encodes when contiguous sets of grid cells become filled and cannot empty the cells, while our representation encodes rules for when individual grid cells change state between filled and empty. Their comparisons of differing fitness functions were of particular interest to our research where they noted that lengthening paths to cul-de-sacs did not produce desirable mazes. They also use shortest path length, number of dead ends, and the combination of both.

Pech evolves cellular automata to generate maze levels [10]. Their work differs from ours in that they test evolving binary cellular automata rules dealing with between two (2) and four (4) cell states, what they refer to as flavors, while we test probabilistic rules on binary cell states. They also use a fitness function that measures similarity to a combination of nine target features, each with differing weights, while our approach only looks at shortest solution path length and number of dead ends.

Togelius presents a multi-objective evolutionary approach for procedurally generating StarCraft maps [11] [12]. Togelius' work shows similarity to our work in that it uses a GA to procedurally generate maps. StarCraft map generation relating well to maze creation due to the maze like patterns prevalent in StarCraft maps. All of the works discussed in this section fall within the larger research category of procedural content generation which Togelius provides a helpful survey paper [13].

The work by Johnson [14] presents a method for generating cave levels using cellular automata. Johnson's work details an approach that bears similarity to our own for generating mazes, though they do not evolve the cellular automata rules using a genetic algorithm. Van der Linden's work [15] provides a useful survey of automated methods for generating dungeon maps, a type of maze-like map. Among the methods surveyed are cellular automata and genetic algorithms, which were of particular interest to our own work.

Chapter 3

Methodology

We evolve cellular automata rules using a genetic algorithm. The rules when input into a cellular automaton will generate maze-like patterns after a number of time steps. Using a post-processing operation known as region merging we ensure the maze-like patterns have a contiguous path from the starting position to the ending position of the maze. After region merging, the mazes will have their quality measured by a fitness function to give the genetic algorithm direction for evolution. After each maze receives a fitness rating the genetic algorithm will perform the crossover and mutation operations to generate a new population of cellular automata rules. Each step of the GA will repeat until a desired number of generations are reached.

In the following section we explain cellular automata and the cellular automata rules that our genetic algorithm evolves.

3.1 Cellular Automata

Cellular automata were first invented by J. Von Neuman [16] with the most well known cellular automaton Conway's Game of Life [17]. The term cellular automata defines a discrete model that consists of a grid of cells, each with a discrete number of states, and a rule set that governs when cells change state. The rules determine when a given cell will change state based upon the cell's neighboring cells or neighborhood, with our experiment utilizing the Moore neighborhood shown in Figure 3.1.

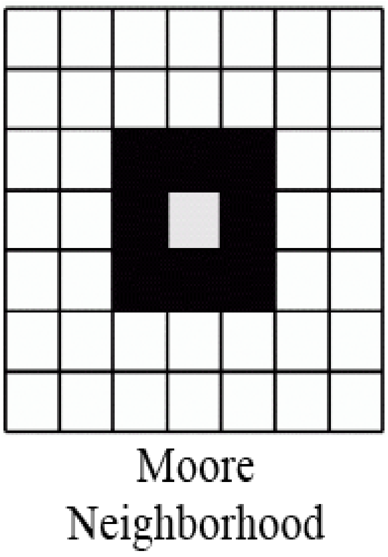


FIGURE 3.1: Cellular automata neighborhood

Each cell of the grid has the rules applied to them once each time step before any cell state changes take place, Figure 3.2 shows an example. The cellular automaton in Figure 3.2 follows the rules laid out in Algorithm 1.

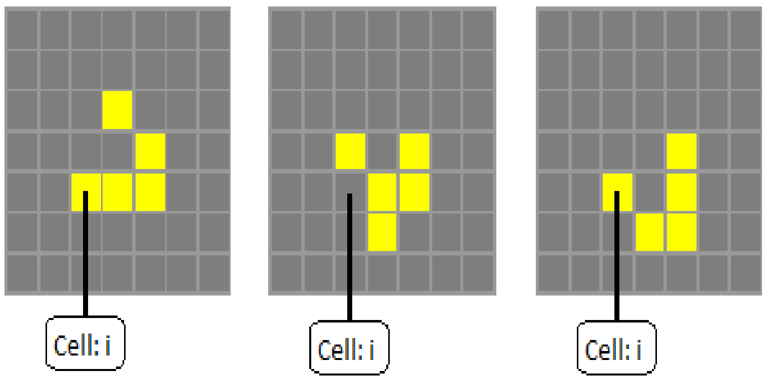


FIGURE 3.2: CA running: Left: Time step 0; Center: Time step 1; Right: Time step 2

Algorithm 1 CA example rules

```
if Cell State == Empty then
   if Filled Cells in Neighborhood == 3 then
      Cell State = Filled;
   end if
else
   if Cell State == Filled then
      if Filled Cells in Neighborhood != 2 or 3 then
         Cell State = Empty;
      end if
   end if
end if
```

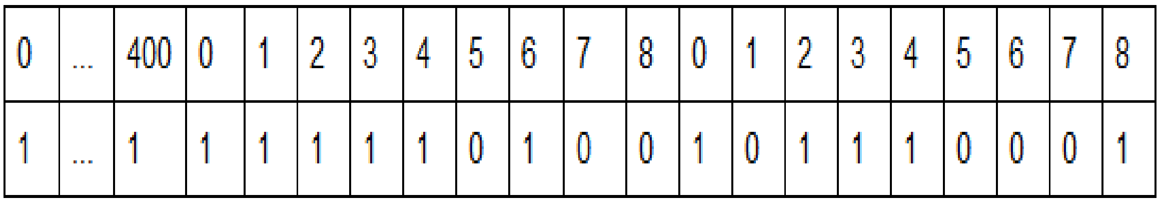
The next section explains our genetic algorithm and our choices during design.

3.2 Genetic Algorithm Design

A genetic algorithm consists of five parts; chromosome representation, selection method, crossover method, mutation method, and fitness function. The decision on the composition of each part depends on the problem attacked by the GA. We utilize fitness proportional selection to provide a balance between exploration of the search space and exploitation of high quality solutions. Next we must define our representation for mazes that our genetic algorithm will evolve. The cave generation tutorial by Sebastian Lague provides a good framework for representing mazes, as he utilizes cellular automata to generate cavern maps [18]. The tutorial describes a method for using cellular automata to generate arbitrarily sized grid layouts for caves and then translating those grids into thee dimensional caves. For our mazes we selected a grid size of 20×20 for a total of 400 cells that our cellular automaton will operate on. We can now calculate the requisite number of bits needed for cellular automata rules in our representation. Our cells have 2 states of filled and empty, and our neighborhood contains 8 cells. The cells swap states based upon the number of filled cells in the neighborhood, which means that each cell can have between 0 and 8 filled neighbors for a total of 9 possible neighborhood states. 2 cells states combined with 9 possible neighborhood states means that our representation must have 18 bits dedicated to the cellular automata rules. In the next section we provide more detail on our representation along with the crossover and mutation methods chosen.

3.3 Representation

The chromosome representation we used thus contains 418 bits. The first 400 bits contain the starting maze state for our 20×20 grid, with a 1 indicating a filled cell and a 0 indicating an empty cell. The remaining 18 bits encode the cellular automata rules that will be applied to the maze. Our chosen chromosome format means that our IGA has a search space of 2^{418}. Figure 3.3 shows our chromosome format along with an example chromosome.



0	...	400	0	1	2	3	4	5	6	7	8	0	1	2	3	4	5	6	7	8
1	...	1	1	1	1	1	1	0	1	0	0	1	0	1	1	1	0	0	0	1

Figure 3.3: IGA chromosome format

Mazes are constructed using our chromosome representation by initializing our grid with the starting maze state contained within our first 400 bits of chromosome and then applying the cellular automata rules in the remaining 18 bits to the grid over a set number of iterations. We found that about twenty iterations were required for the cellular automata to fully expand across the grid. The resultant maze however requires the application of Lague's region merging algorithm [18] as the generated maze-like patterns will frequently contain several disconnected regions.

We use standard fitness proportional selection where the relative fitness of a population member determines the probability of selection for crossover. Our crossover function uses a modified two point crossover strategy with one point of crossover constrained to the first 400 bits of the initial maze state and the second point of crossover falling within the 18 bits of cellular automata rules. We utilize standard mutation where each bit of a chromosome has a small probability to mutate. The fitness function however poses a problem, as we do not know what human players find interesting in mazes. A type of genetic algorithm known as an interactive genetic algorithm lends itself well towards solving this problem as it uses direct human interaction to gauge fitness. The following section details the interactive genetic algorithm experiment we ran in order to determine features correlated to mazes human players find interesting.

3.4 Interactive Genetic Algorithm Experiment

Our interactive genetic algorithm utilizes the the GA components defined in the previous sections: fitness proportional selection, point mutation, modified two-point crossover, as well as the interactive fitness function that defines an IGA [7] [5]. Figure 3.4 displays the full process of our IGA for each user. We had each user evaluate nine mazes per generation from our population of sixteen by manipulating the sliders in our user interface shown in Figure 3.5. Our population size of sixteen and the user evaluating only nine mazes at a time are to limit user fatigue [7]. Afterwards the remaining seven mazes were

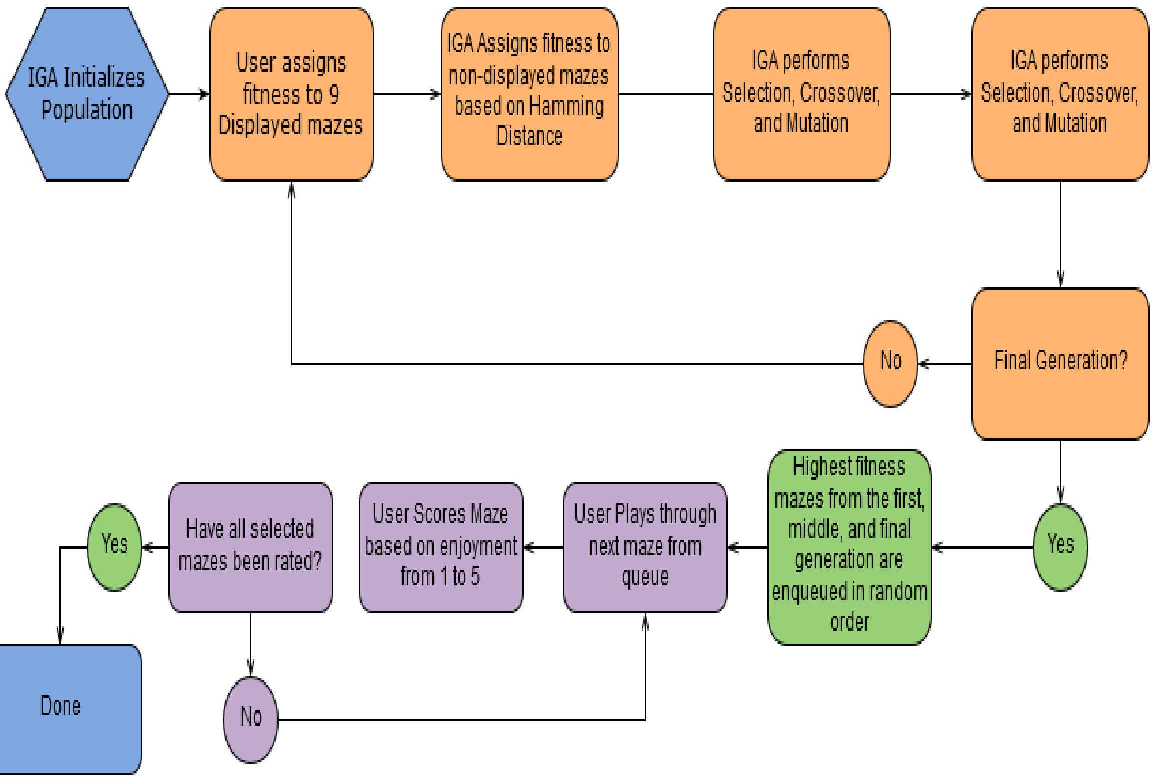

IGA Initializes Population
User assigns fitness to 9 Displayed mazes
IGA Assigns fitness to non-displayed mazes based on Hamming Distance
IGA performs Selection, Crossover, and Mutation
IGA performs Selection, Crossover, and Mutation
Final Generation?
No
Yes
Highest fitness mazes from the first, middle, and final generation are enqueued in random order
User Plays through next maze from queue
User Scores Maze based on enjoyment from 1 to 5
Have all selected mazes been rated?
Yes
No
Done

Figure 3.4: IGA maze evolution and scoring process

then assigned the fitness of the most similar rated maze, with maze similarity measured using the hamming distance between a given unrated maze and each rated maze [1]. The population would then undergo selection, crossover, and mutation with the canonical GA parameters of 70% crossover rate and 1% mutation chance [5]. We ran the IGA for ten generations with a user rating nine mazes per generation. After the final generation of the IGA concluded, we had the users run through three selected mazes. We saved the highest rated maze from the first, fifth, and tenth generation for users to play through in first-person perspective. Figure 3.6 shows the view players would have when running through the selected mazes. The walls not being textured could possibly effect the user's experience, though the textured floor displays depth. The users were then asked to score each maze after their play-through on a scale from one to five on how much the user enjoyed the maze. The ratings gathered allowed us to compare randomly initialized mazes from our first generation to evolved mazes from the fifth and tenth generation to confirm that our IGA evolved improved maze designs.

[1]hamming distance is the differing number of bits between two strings

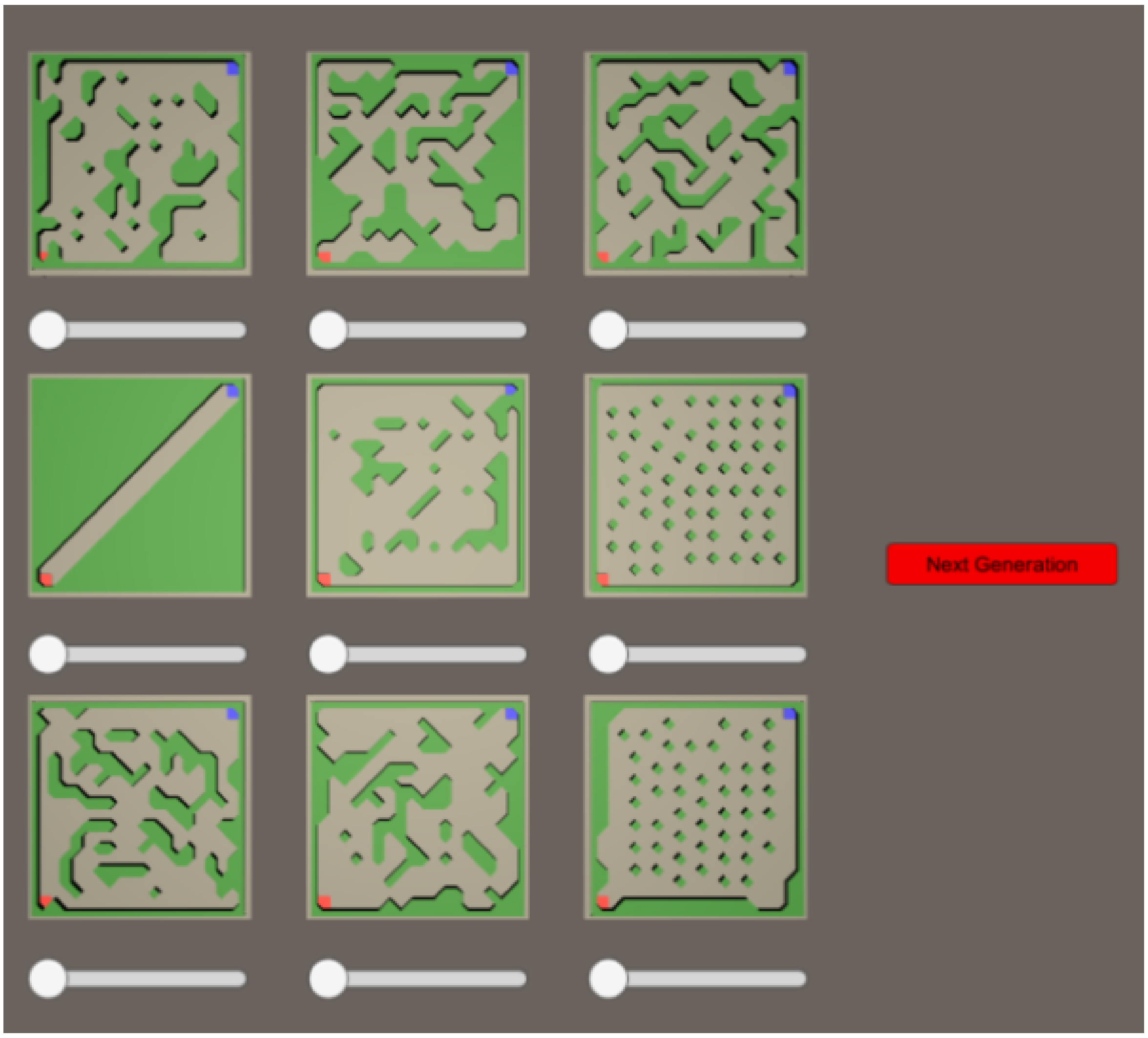


Figure 3.5: Interactive genetic algorithm user interface

3.5 IGA Results

We gathered the top rated mazes from each first, fifth, and tenth generation from 10 different users for 30 total maze exemplars. The users then played through one exemplar maze from the first, fifth, and tenth generation in first-person perspective. After playing each maze the user was asked to give the maze a score on a scale from one to five representing how much the player enjoyed the maze. Users that participated all were experienced with video games in general and several with maze running games. Figure 3.7 shows the results of our enjoyment scored mazes. In the figure the improvement in scores between mazes from the first to middle to final generation can be seen. The results between the first and final generations have a p-value of 0.037 indicating a statistically significant difference between the scores of the two generations as it falls below

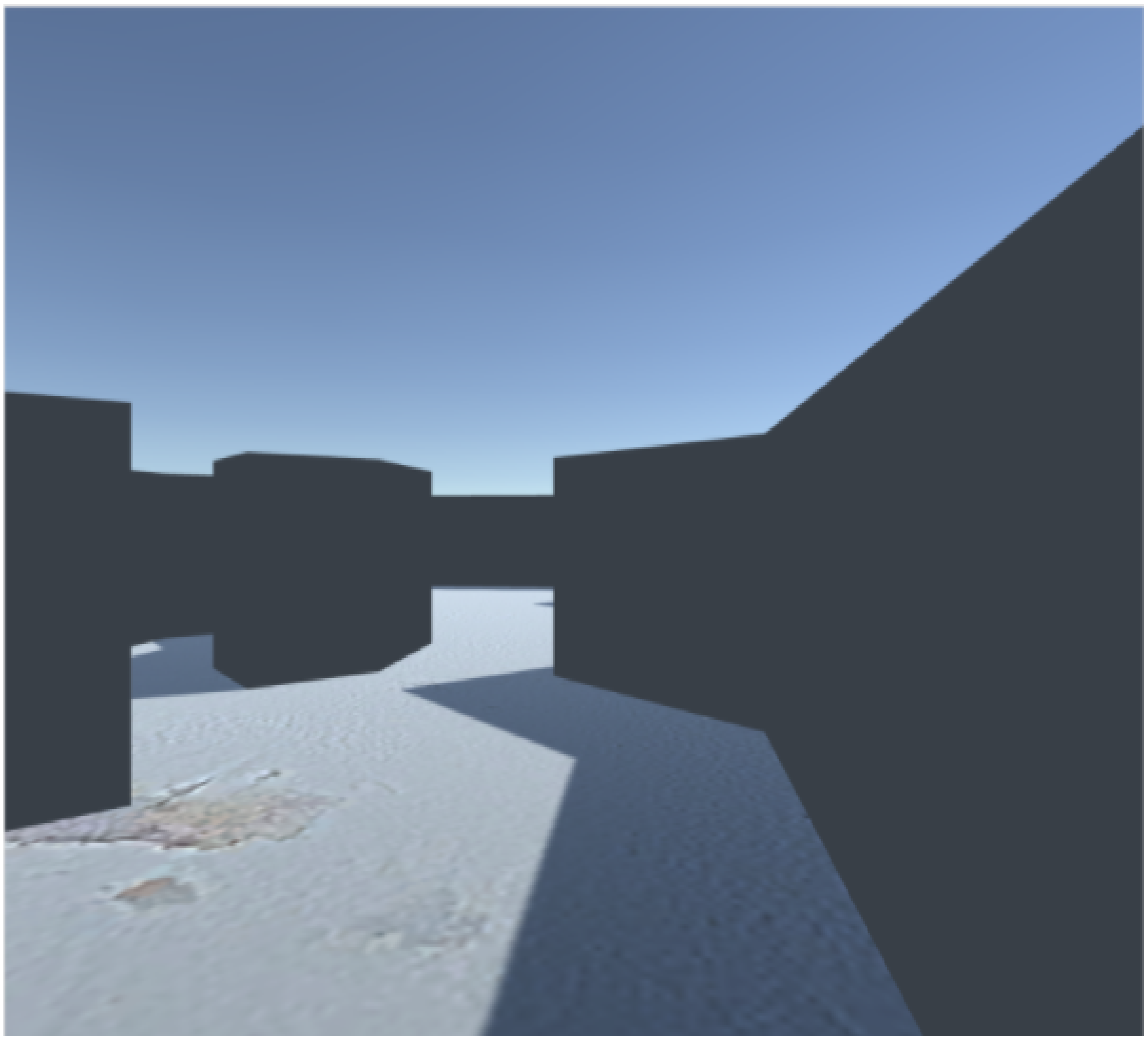


Figure 3.6: First person maze running view

the p-value of 0.05. However we found no statistical difference between the initial to middle generations, or the middle to final generations as they have p-values of 0.289 and 0.283 respectively. From this data we then inspected the 30 exemplar mazes in order to look for features that correlate to higher ratings by our human users. Figure 3.8 shows the average of the maze features we observed sorted by what score the mazes received. The number of solution paths and the number of turns on the shortest solution path both follow a similar, although inverse, trend as the maze ratings increase. Table 3.1 confirms these trends with the displayed correlation coefficients as coefficients close to 1 or -1 indicate a strong positive or negative correlation respectively. The more interesting story comes from the number of dead ends, as it follows a positively correlated trend as the scores increase, with a drop at mazes with a score of 4. Figure 3.9 shows the possible reason for this drop. The left most maze having many trivial dead ends

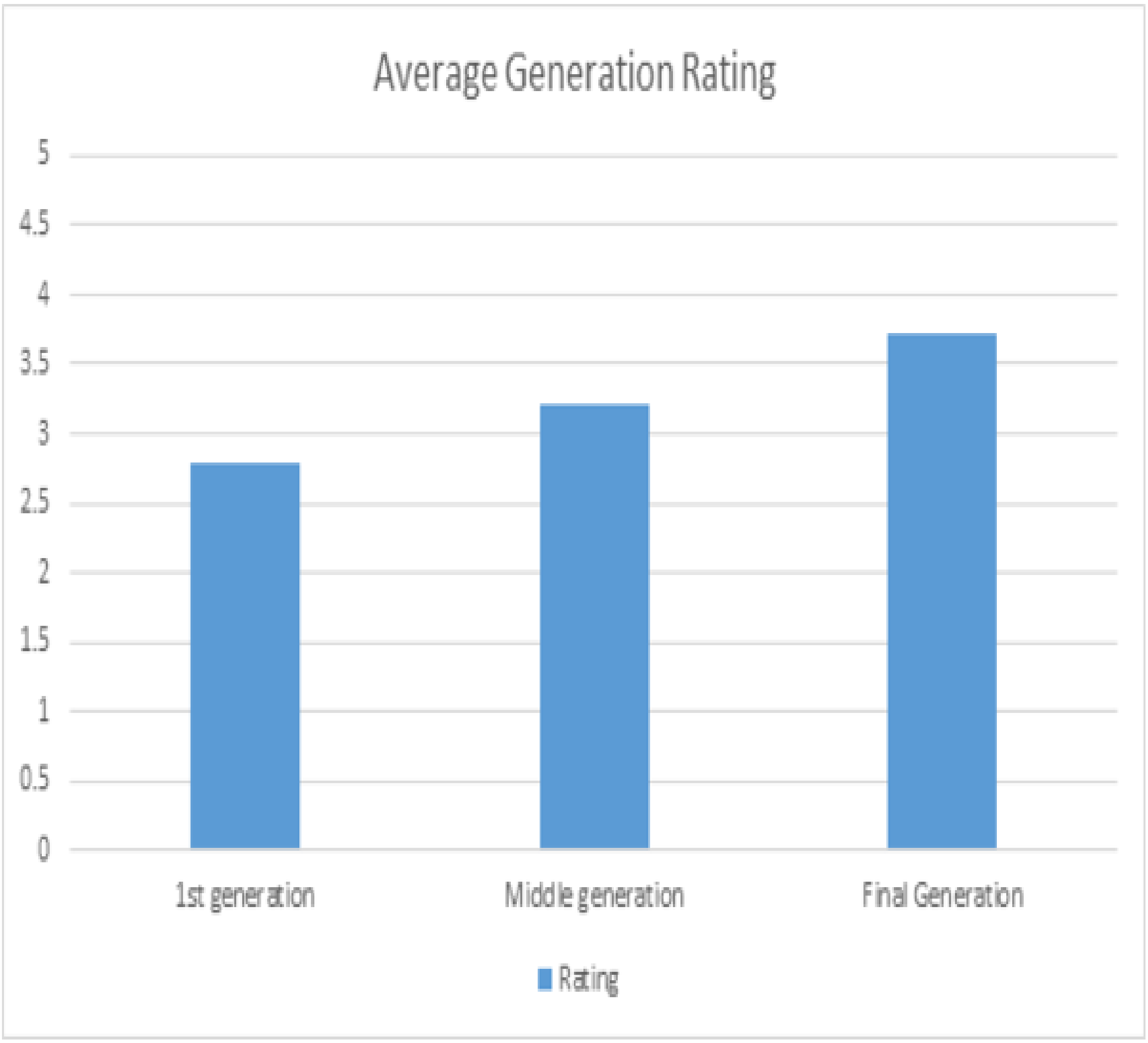


FIGURE 3.7: Enjoyment scores by generation

with few turns on the shortest solution path. While the center maze has very few dead ends but more turns on the shortest solution path. Of note, no mazes that were selected for play-through were rated with a score of 1 by our users, hence the absence of metrics for rating 1 mazes. From our results and the correlation coefficients shown in Table 3.1 we can see that the number of turns on the shortest solution path and the number of dead ends correlate well to how interesting human players would find a given maze.

TABLE 3.1: T-Test values for statistical difference between binary and probabilistic representations

Maze Metrics	**Dead Ends**	**Solution Paths**	**Turns on Shortest Path**	**Wall Sections**
Correlation Coefficients	0.858	-0.863	0.911	0.421

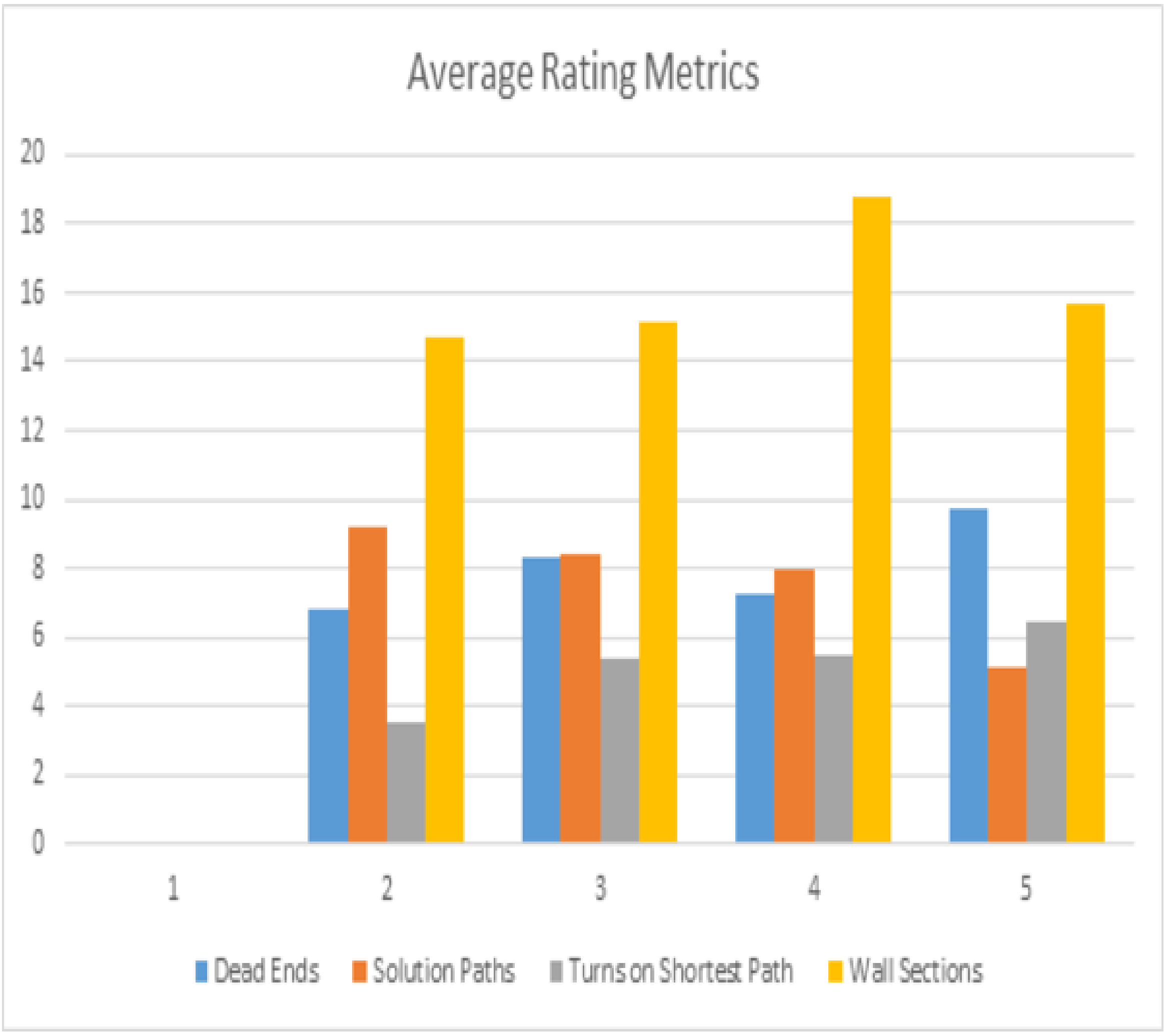


FIGURE 3.8: Maze metrics by enjoyment score

From these results we chose to use the number of dead ends and number of turns on the shortest solution path for our fitness measures in our non-interactive genetic algorithm's fitness functions. The number of turns on the shortest solution path being modified to the length of the shortest solution, as a longer solution path will have more turns as well.

3.6 Evolutionary Cellular Automata

While performing our IGA experiment we found that the cellular automata rules alone had a large impact on the final maze pattern. Therefore we chose to discard the 400 initial maze state bits of our 418 bit representation used by our IGA experiment from the chromosome for our non-interactive genetic algorithm leaving only the 18 bits of our

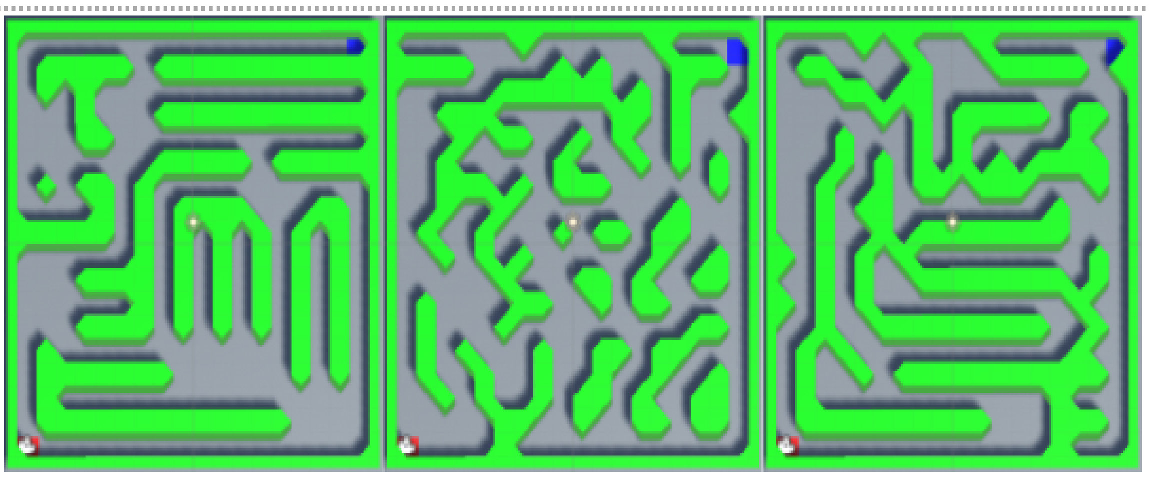


Figure 3.9: IGA maze exemplars
Left: Score 3 - Center: Score 4 - Right: Score 5

cellular automata rules. Choosing this much smaller representation opens the possibility for comparison with the true optimum, since we can use an exhaustive search. The binary cellular automata rules give us 2^{18} or 264144 possible rule sets, which we can exhaustively search reasonably quickly for the true optimum for us to compare our GA against. We will provide more detail on our new representation later in this section.

As we have removed the maze initial state from our representation, we chose to test two fixed starting maze patterns; a blank slate where all cells start as empty and filled center where the center four cells start as filled. The starting map states will then have the cellular automata rules applied to them over fifty iterations as we found that fifty iterations usually suffices to stabilize cell states on the cellular automaton grid. Afterwards we apply our region merging algorithm in order to ensure that there exists a complete path from start to finish, and then measure fitness on the resulting maze. Additionally we raised the grid area to 30×30 cells for our non-interactive genetic algorithm experiments. Increasing the grid area allows the cellular automaton to generate maze-patterns with increased complexity.

The selection method from our IGA does not apply well to the much larger populations used in non-interactive genetic algorithms, so we must pick a new selection method. We chose to use an elitist selection method where n offspring are created from a population of size n, giving us a total of $2n$ individuals, after which the top n individuals are chosen to continue into the next generation [19].

For crossover we will need to also choose a new method as we no longer have the initial maze state as a portion of our representation. We altered our crossover method from the modified two point crossover to half uniform crossover, which crosses over half the differing bits of the parent chromosomes [19]. Since we use a highly elitist selection method, we used a crossover rate of 100% to combat premature convergence and increase search space exploration. Our mutation method requires no alteration and we use a mutation rate of 5%. These crossover and mutation rates were arrived at after some testing. Algorithm 2 defines our genetic algorithm. Algorithm 3 defines how our genetic algorithm constructs and evaluates a maze.

Algorithm 2 Maze generation genetic algorithm

```
InitializePopulation();
EvaluatePopulation();
Generation = 0;
while Generation < MaxGenerations do
    ElitistSelection();
    Crossover();
    Mutate();
    EvaluatePopulation();
    Generation += 1;
end while
```

Algorithm 3 Maze Evaluation

```
Iterator = 0;
while Iterator < 50 do
    ApplyCellularAutomataRules();
    Iterator += 1;
end while
MergeRegions();
GetMazeFitness();
```

3.6.1 Representation

We encode our chromosome as an 18 bit array, nine bits for each cell state in our CA. An example of the binary representation can be seen in Figure 3.10 where cells in the top array indicate the number of filled neighbors the corresponding cell in the bottom array relates to, with a one signifying that the corresponding number of filled neighbors will cause a given cell to become or stay filled and a zero indicating the cell will become or stay empty. Algorithm 4 shows how the example rule set in Figure 3.10 would apply to our cellular automaton.

0	1	2	3	4	5	6	7	8	0	1	2	3	4	5	6	7	8
0	0	1	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0

FIGURE 3.10: Binary chromosome format & example

Algorithm 4 Example Binary Chromosome CA Algorithm

```
if Cell State == Empty then
  if Filled Cells in Neighborhood == 2 or 3 then
    Cell State = Filled;
  end if
else
  if Cell State == Filled then
    if Filled Cells in Neighborhood != 2 or 3 or 4 then
      Cell State = Empty;
    end if
  end if
end if
```

3.6.2 Region Merging

Cellular automata often create disconnected regions in the maze-like patterns which makes region merging a necessary step to ensure a complete path from start to finish. Algorithm 5 illustrates how our region merging algorithm works on a high level. Shown in Figure 3.11, you can see the order that the region merging algorithm will detect each of the shown regions and that same order would be used to merge the regions together along the thick lines. The displayed order of region merging stems from the fact that we follow an algorithm that starts at the lower left of the map and moves from left to right, bottom to top which then keeps the first shortest path found if there are multiple paths tied for shortest.

Algorithm 5 Region Merging

```
Get Disconnected Rooms(Maze,Rooms);
while Rooms > 1 do
  Path Start,Path End = Find Nearest Cells Between Disconnected Rooms(Maze,Rooms);
  Clear Path Between Rooms(Maze,Path Start,Path End);
  Get Disconnected Rooms(Maze,Rooms);
end while
```

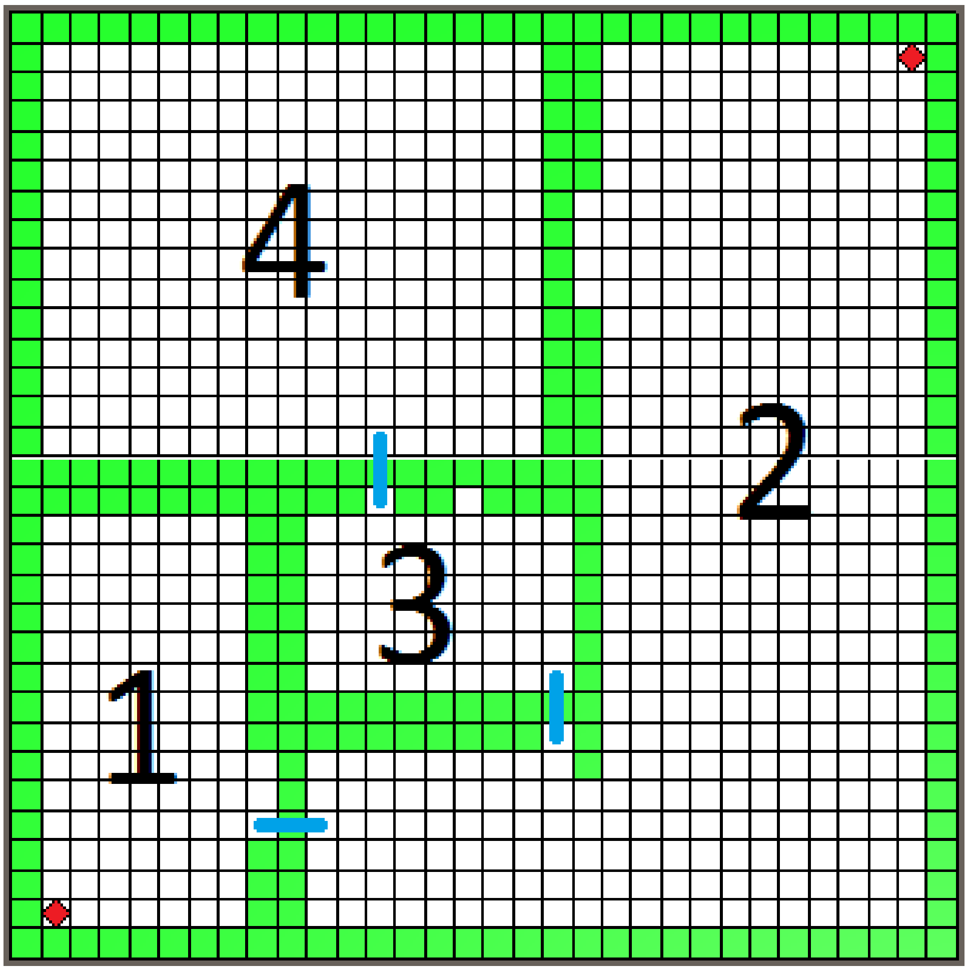


FIGURE 3.11: Region merging example.
Diaomnd: Starting/Ending Cell. Thick Lines: Path taken to merge regions

3.6.3 Fitness Evaluation

We tested three separate fitness functions and compared their performance against each other. The first fitness function ($F1$) we used measured shortest solution path length, the same fitness function from Ashlock's work [4]. We define solution path length to be the fewest number of contiguous empty map cells that connect the starting and end cell, including the end cell, with the fitness as the unweighted total length of the path. The second fitness function ($F2$) used total dead ends, with dead ends defined as a map cell that has no neighboring cell with a longer path length to the entrance cell. $F2$ is thus the number of dead ends. We chose this fitness function as a greater number of false paths makes a maze more interesting to solve, supported by results in our prior work [2]. While similar to Ashlock's work [4], it differs from their fitness function in that we detect individual dead end cells instead of detecting grouped 'cul-de-sac' cells at the end of a path. The third fitness function ($F3$) used the sum of $F1$ and $F2$. We tested these fitness functions on two different starting map states. The first having the map start out as a Blank Slate where all cells start in the empty state which was chosen to minimize the impact that the starting map state had on the final maze layout. The second starting map state tested was where the center four cells started as filled while the remaining start as empty which was chosen to test if a minimally altered starting state would result in significantly different performance. Figure 3.12 shows our fitness functions running on an example maze with the numbers indicating the shortest path length from the given cell to the starting cell; F1 would return a fitness of six, F2 a fitness of four, and F3 a fitness of ten. F2 returns a fitness of four because the ending cell also fits the criteria of a dead-end.

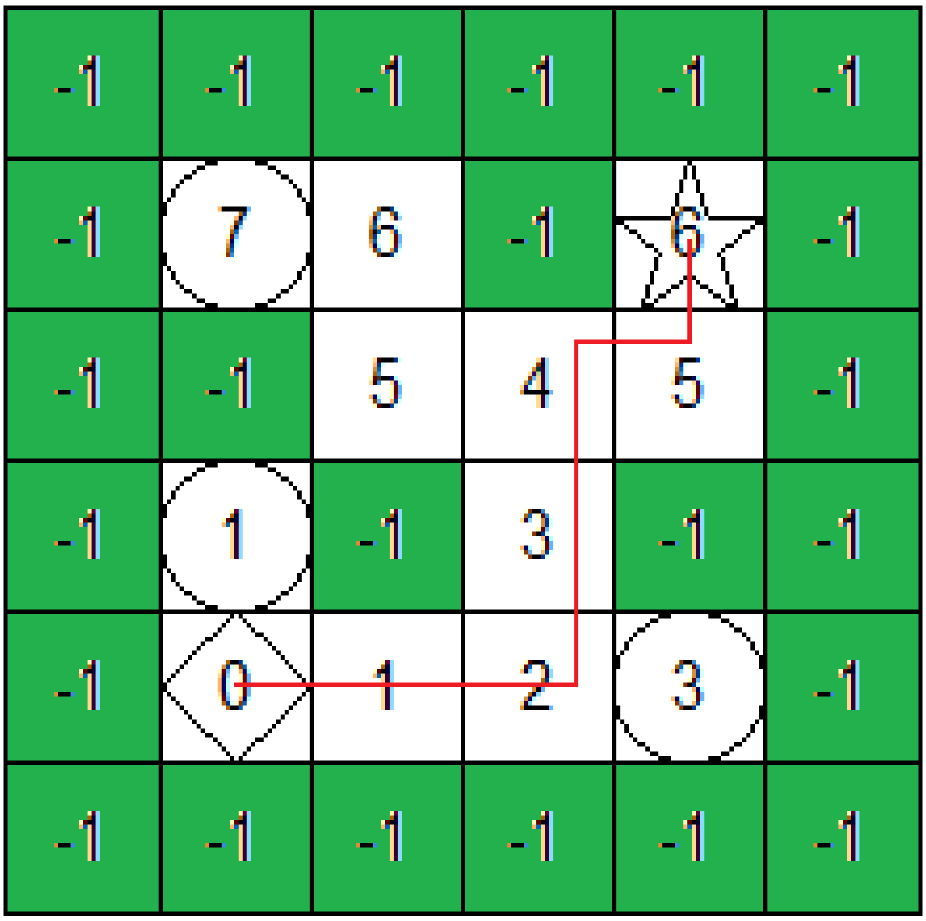


FIGURE 3.12: Fitness evaluation example.
Diamond: Starting cell. Circle: Dead end. Star: Ending cell

To test the scalability of our approach we use probabilistic cellular automata rules in our representation instead of binary rules. In the following section we detail the probabilistic representation we created to test our maze generation method.

3.7 Probabilistic Representation

We created a second representation to test our maze generation approach in a search space not amenable to exhaustive search. Instead of each allele of our chromosome containing a binary integer, they instead encode a probability utilizing a seven bit number that we normalize between 0 and 1.

Figure 3.13 shows an example of our probabilistic chromosome representation. The top row of cells representing the exact number of filled neighbors the corresponding cell in the bottom row relates to and the bottom row encodes the **probability** that the above number of filled neighbors will cause a given cell to change its state. The decoded values shown in the example chromosome of Figure 3.13 indicate that an empty cell has a 100% chance of changing state to filled if it has zero filled neighbors, while a filled cell has a 38.6% chance of changing state to empty if it has one filled neighbor.

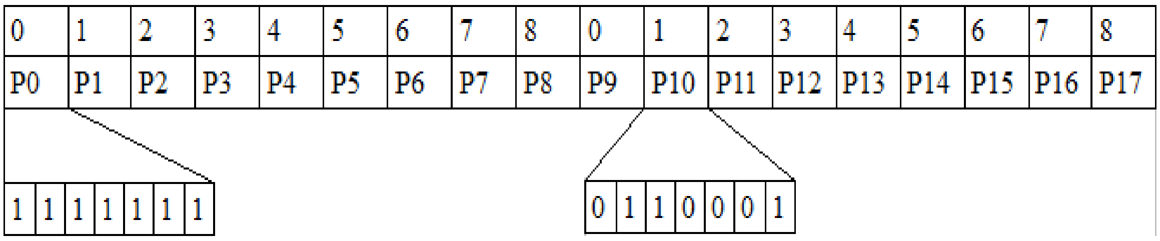

0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16 P17
1 1 1 1 1 1 1
0 1 1 0 0 0 1

FIGURE 3.13: Probabilistic chromosome format & example

We utilize the same selection method for our probabilistic representation as for the binary representation. For the probabilistic chromosome, our GA used a crossover rate of 90% and a mutation rate of 1% after testing other values. The fitness functions remained the same. The results for both the binary representation and probabilistic representation experiments are detailed and discussed in the following chapter.

Chapter 4

Evolutionary Cellular Automata Results

In this chapter we display and discuss the results from our experiments. In the first section we discuss the results from our binary representation experiments. The following section lists our results from our probabilistic representation experiments.

4.1 Binary Representation Results

We ran six separate scenarios, one for each of our three fitness functions on our two starting map states, with 20 runs of each scenario in order to garner the average performance. As an exhaustive search can exhaustively search the space specifically for our binary representation (2^{18}) within a reasonable time frame, we were able to find the maximal performance for each of our fitness function by evaluating all possible combinations of 18 bits.

Shown in Figures 4.1, 4.2, and 4.3 are the top performing mazes generated by our GA on both starting map states using our binary chromosome. Figure 4.1 shows the mazes produced with the fitness function $F1$. The mazes evolved using $F1$ tended to form two symmetrical paths with only one leading to the goal, with several small paths branching off the two main paths. Of note in Figure 4.1, the mazes displayed are identical because they are the optimal solution using our binary representation for our fitness function $F1$.

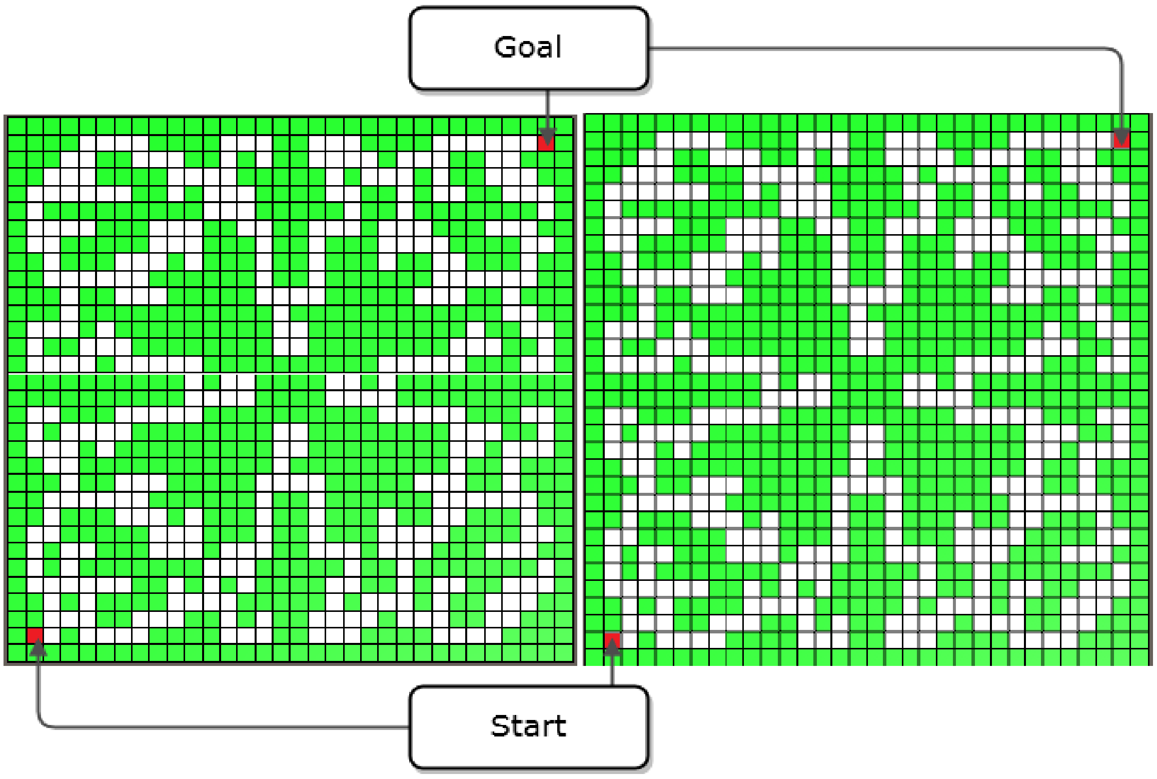


Figure 4.1:
$F1$ Binary cellular automata exemplars. Left: starting with blank initial maze state. Right: starting with a filled center initial maze state

Figure 4.2 contains the exemplar mazes produced by our fitness function $F2$ which awards fitness based upon the number of dead ends within a maze. Mazes produced by $F2$ tend to have mostly diagonal walls with most mazes containing more than one solution path.

Figure 4.3 displays our fitness function $F3$'s expemplar mazes. The mazes generated with $F3$ combine the two noted behaviors of $F1$ and $F2$ albeit to a lesser degree for each. Mostly diagonally connected walls whilst creating long and symmetrical solution paths. With all three figures we can see a pattern emerge for each starting maze state. The mazes produced with the blank slate starting state had a tendency towards having multiple axes of symmetry, with a few minor differences that were created by the region merging algorithm. While filled center starting state also produced symmetrical mazes, they tended to only have a single axis of symmetry.

Our GA was able to produce rules that on average produced mazes that were about 90% of the optimal solutions for our fitness functions as seen in Table 4.1. Table 4.2 contains the average fitness values of our binary representation for the three fitness

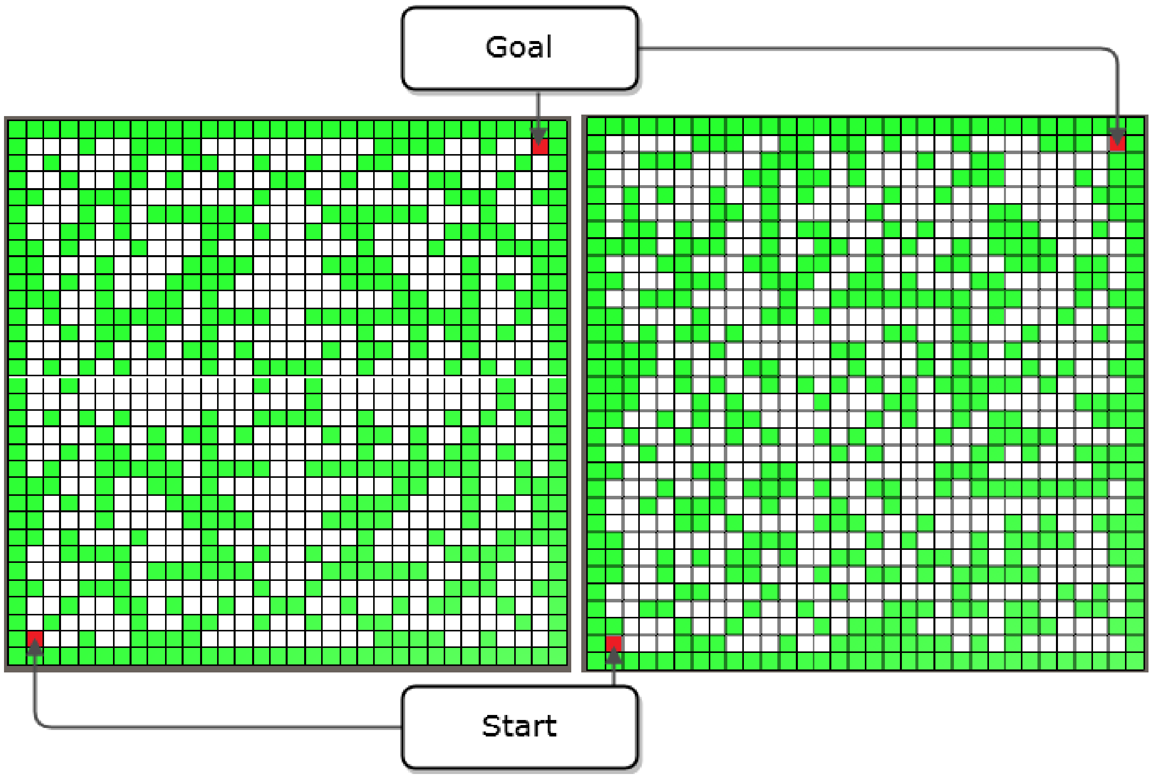


FIGURE 4.2:
*F*2 Binary cellular automata exemplars. Left: starting with blank initial maze state. Right: starting with a filled center initial maze state

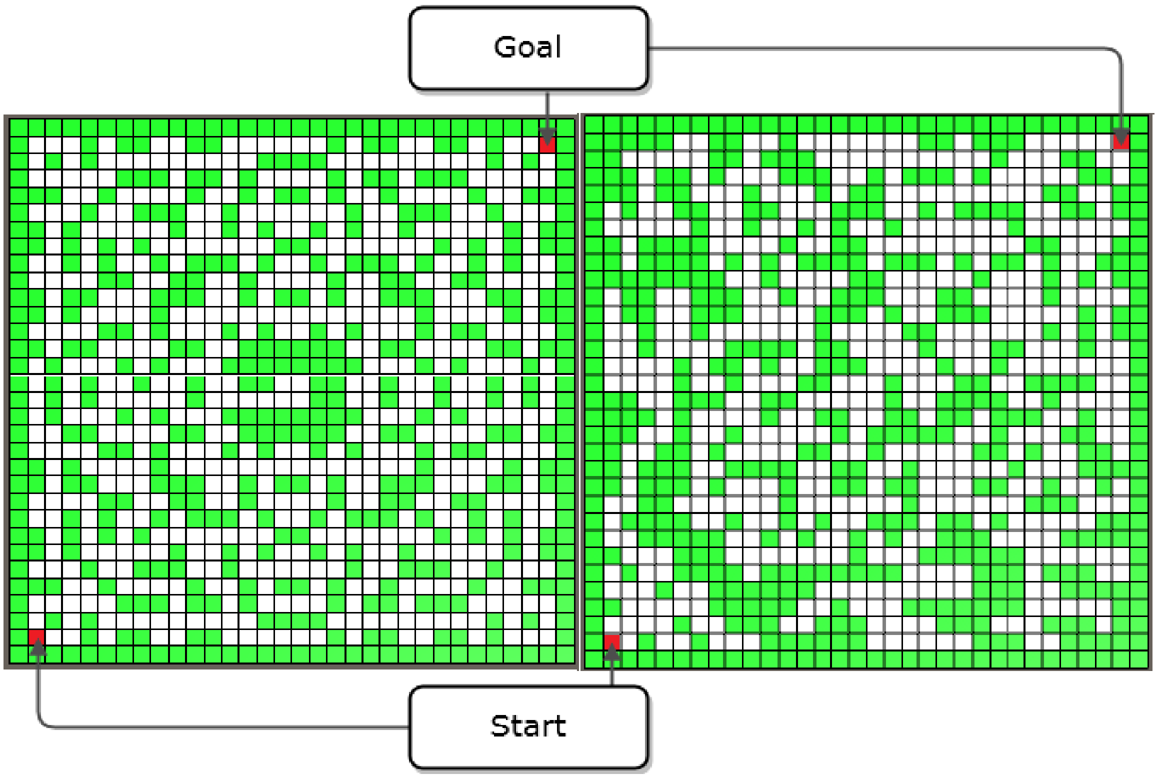


FIGURE 4.3:
*F*3 Binary cellular automata exemplars. Left: starting with blank initial maze state. Right: starting with a filled center initial maze state

Table 4.1: Fitness function average performance using binary chromosome as percent of maximal result

Fitness Function	**Blank Slate**	**Filled Center**
F1: Path Length	92.3% (9%)	88.7% (5.8%)
F2: Dead End Count	91.3% (5.9%)	94.6% (4.6%)
F3: F1+F2	98.4% (3.8%)	90.3% (3.7%)

Table 4.2: Fitness function average performance using binary chromosome on different starting map states

Fitness Function	**Blank Slate**	**Filled Center**
F1: Path Length	97.8 (9.6)	101.1 (6.6)
F2: Dead End Count	113.2 (7.4)	112.6 (5.4)
F3: F1+F2	180.1 (6.9)	186.0 (7.7)

functions on each starting maze state. The values within Table 4.2 representing the average number of cells that were on the shortest solution path, counted as dead ends, or the sum of both. Shown in Figure 4.4 our GA's performance on $F1$ using the 'Blank Slate' starting map state can be clearly seen. Our maximum and average performance follows an expected trend, with both approaching the top line representing the optimum fitness found by exhaustive search. The average minimum performance tells a more interesting story, as it hovers around the minimum possible performance for the entire run. The cause of such a result could likely be due to the sensitivity of the cellular automata to mutations in certain rules, specifically in the case of the first bit mutating to a zero, the bit that encodes the rule of an empty cell swapping to filled if the cell has no filled neighbor cells, which would result in the entire map remaining empty as no starting filled cells will appear. $F1$ has a similar performance curve on both blank slate and filled center starting map states while $F2$ and $F3$ follow a similar performance curve on both starting map states as shown in Figure 4.5. We chose to include only two graphs for this

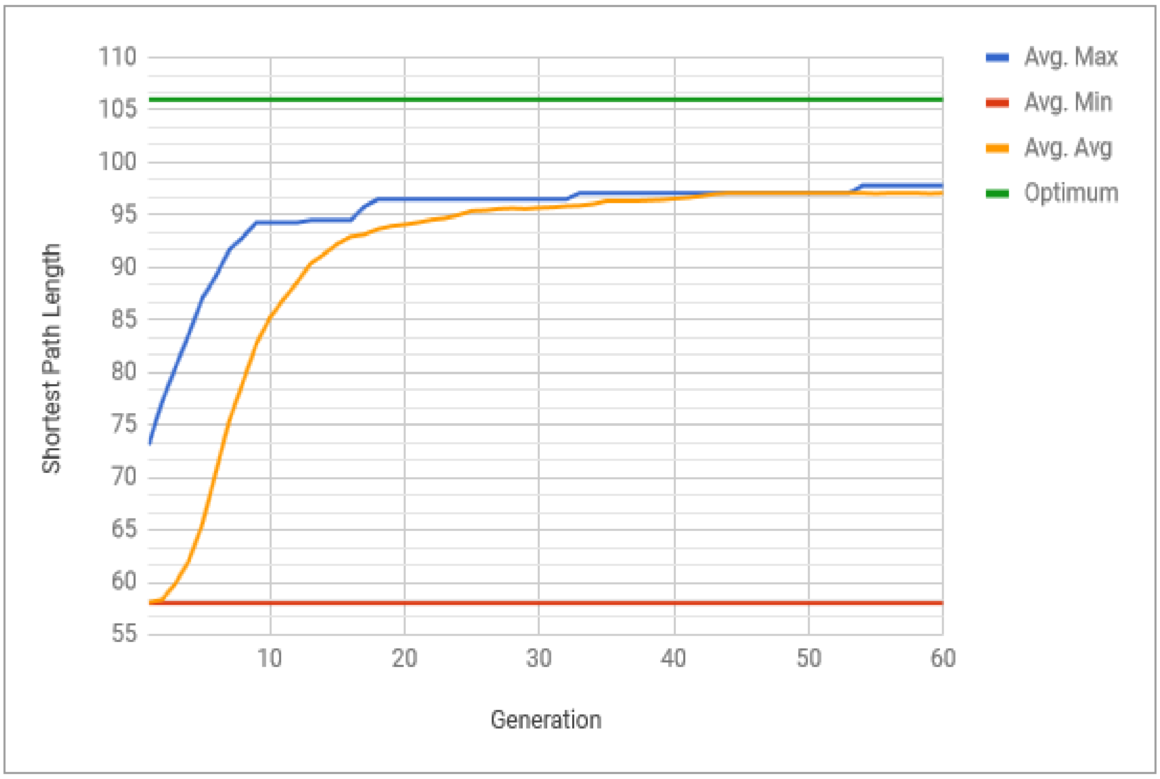


FIGURE 4.4: GA performance using $F1$ on the blank slate starting map state

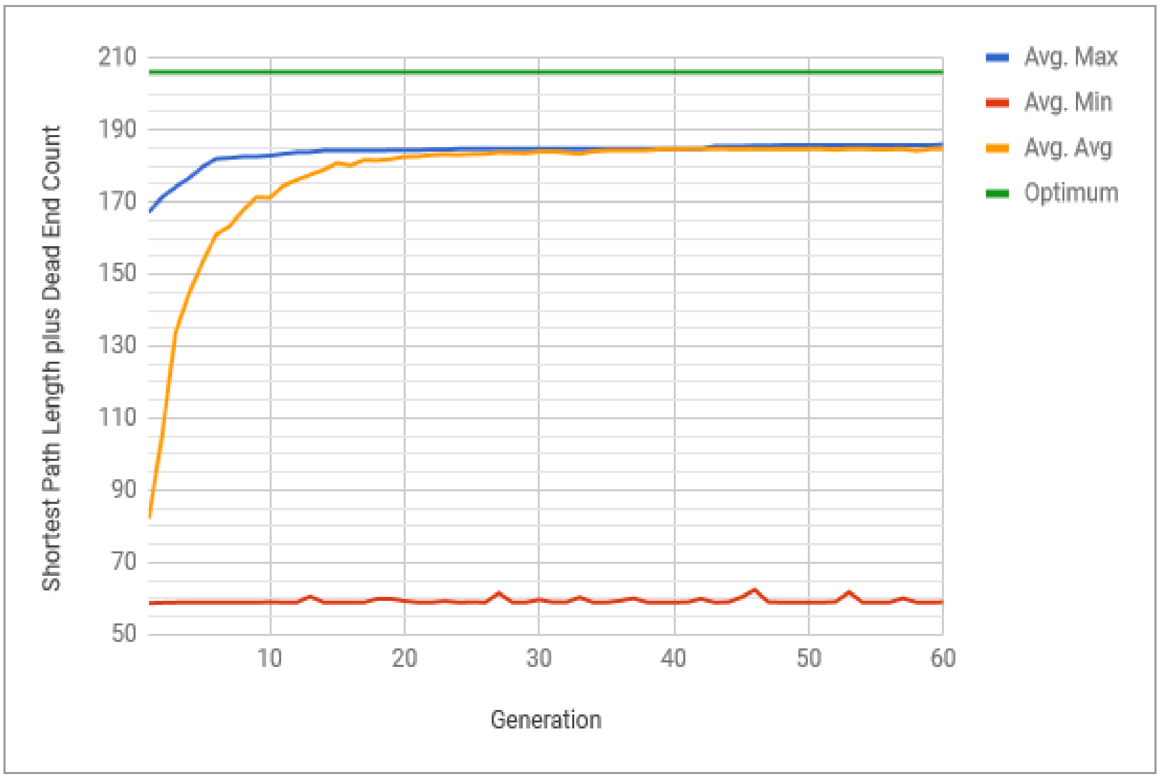


FIGURE 4.5: GA performance using $F3$ on the filled center starting map state

reason. The following section details the results for our probabilistic representation's runs through the same scenarios as our binary representation and compares the two.

4.2 Probabilistic Representation Results

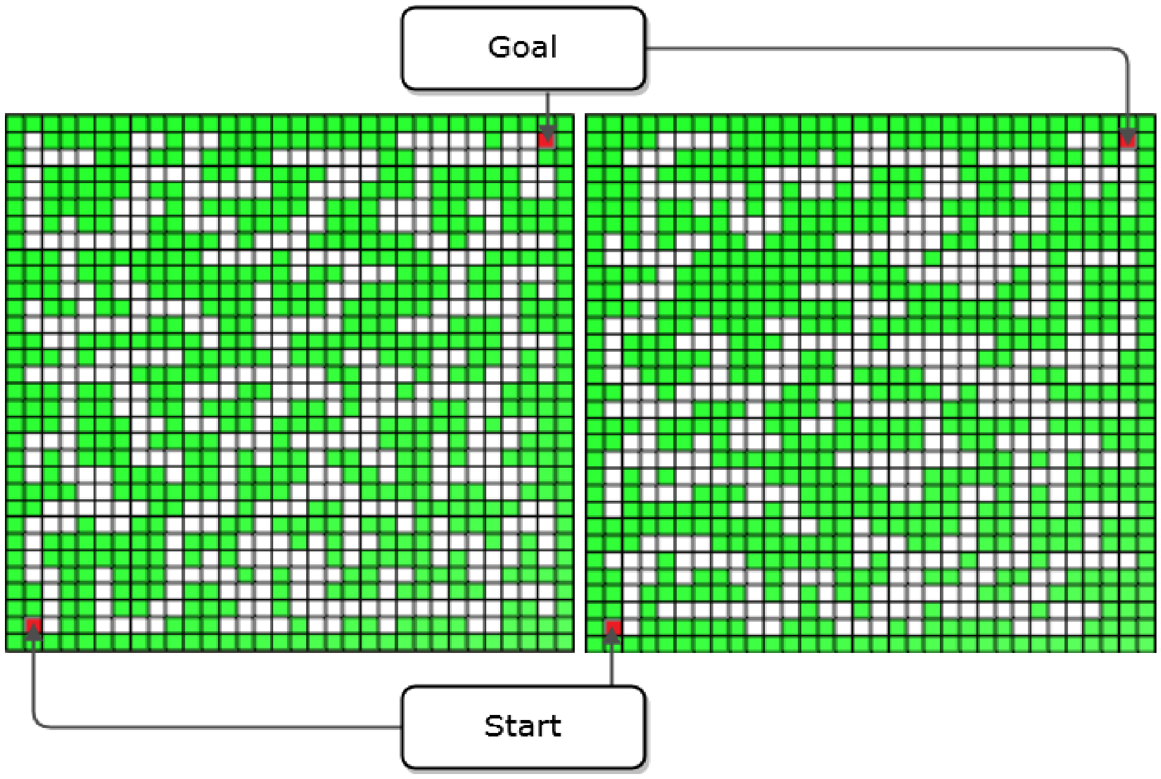


FIGURE 4.6:
$F1$ Probabilistic cellular automata exemplars. Left: starting with blank initial maze state. Right: starting with a filled center initial maze state

Figure 4.6 shows the exemplar mazes for fitness function $F1$ while using our probabilistic representation. The shortest solution path is a single long snaking path, with several offshoot paths of mostly small length. The performance on $F1$ by our probabilistic representation achieved slightly longer paths while using the filled center starting maze state, though visually the mazes follow the same pattern of a single long path.

In Figure 4.7 we can see our probabilistic representation's generated mazes with the fitness function $F2$. The mazes display similar behaviour to the binary representation when using $F2$, having mostly diagonal wall sections, although lacking an axis of symmetry. Although the mazes still contain more than one solution path, the paths tend to diverge from one another less. Figure 4.8 displays our $F3$ probabilistic representation maze exemplars. We can see that the pattern from our binary representation on $F3$ also

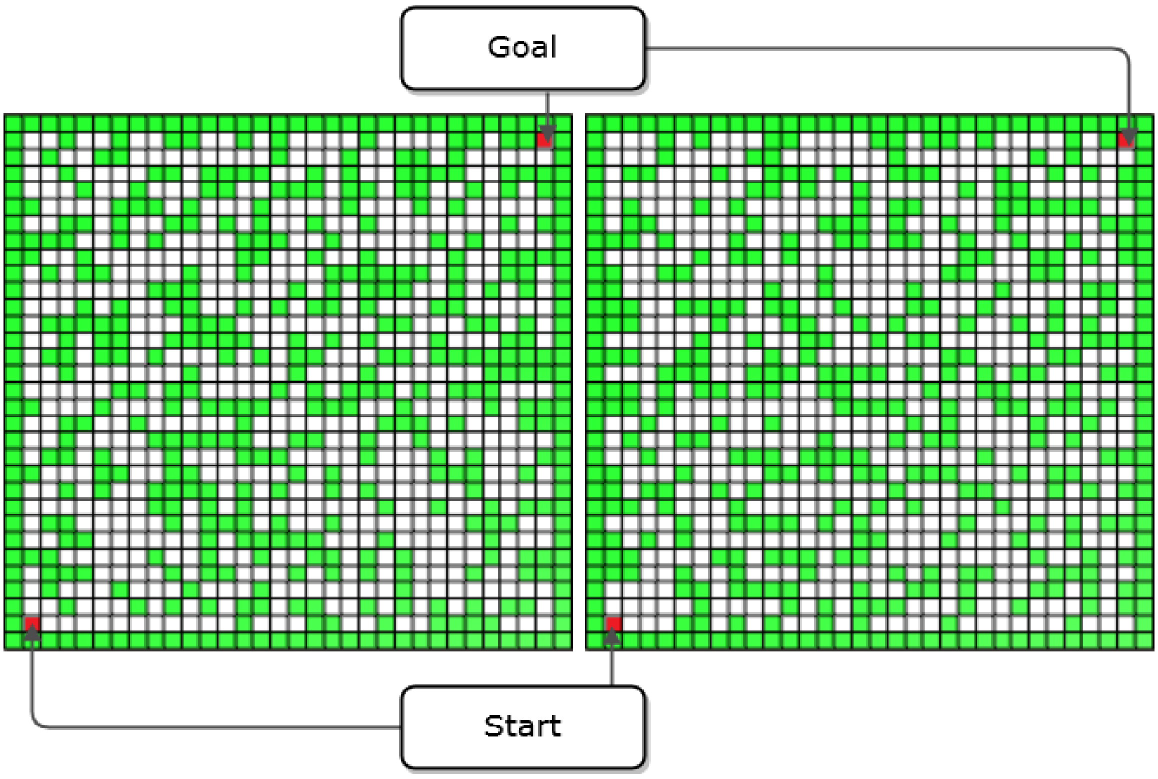


FIGURE 4.7:
*F*2 Probabilistic cellular automata exemplars. Left: starting with blank initial maze state. Right: starting with a filled center initial maze state

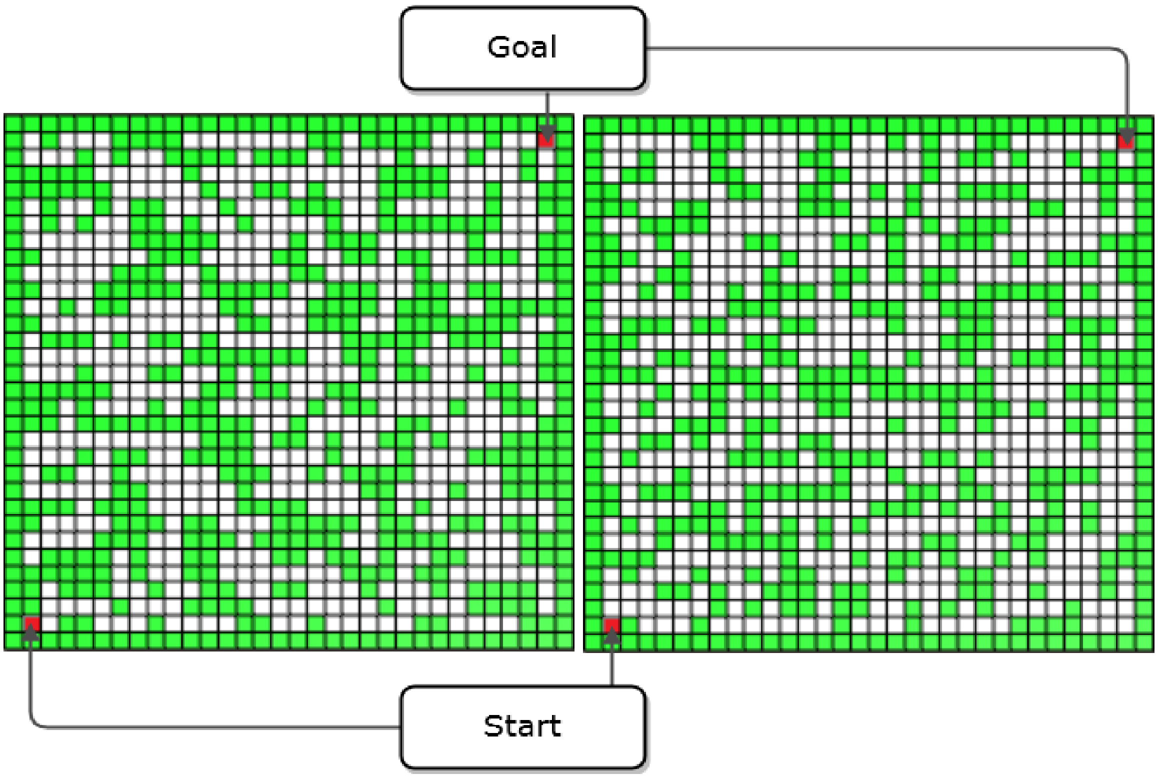


FIGURE 4.8:
*F*3 Probabilistic cellular automata exemplars. Left: starting with blank initial maze state. Right: starting with a filled center initial maze state

holds with our probabilistic representation, where mazes generated from chromosomes evolved using $F3$ show a mix of the pattern tendencies of mazes created with $F1$ and $F2$. Of note in Table 4.3, our probabilistic representation far out performs our binary representation on $F1$ and $F3$, but only slightly on $F2$. This possibly indicates that our method has found a local maximum in the search space. The graphs in Figure 4.9 and

TABLE 4.3: Fitness function performance using probabilistic chromosome on different starting map states

Fitness Function	**Blank Slate**	**Filled Center**
F1: Path Length	132.6 (10.4)	136.8 (11.7)
F2: Dead End Count	116.5 (2.8)	116.4 (3.1)
F3: F1+F2	219.4 (12.1)	218.5 (11.1)

TABLE 4.4: T-Test values for statistical difference between binary and probabilistic representations

Fitness Function	**Blank Slate**	**Filled Center**
F1: Path Length	1.73E-61	4.45E-48
F2: Dead End Count	2.04E-07	2.23E-11
F3: F1+F2	2.85E-62	2.99E-54

Figure 4.10 show the performance curves for $F1$ and $F3$ respectively which follow similar curves both to one another and on both the blank slate and filled center starting map states, with the performance curve for $F2$ keeping with the same trend. We chose to show only two graphs for the same reason as our binary representation, the performance curve graphs for the other fitness function/starting state combinations all look highly similar.

Our probabilistic representation was able to perform better than our binary representation while maintaining similar improvement trends. The mazes generated by our

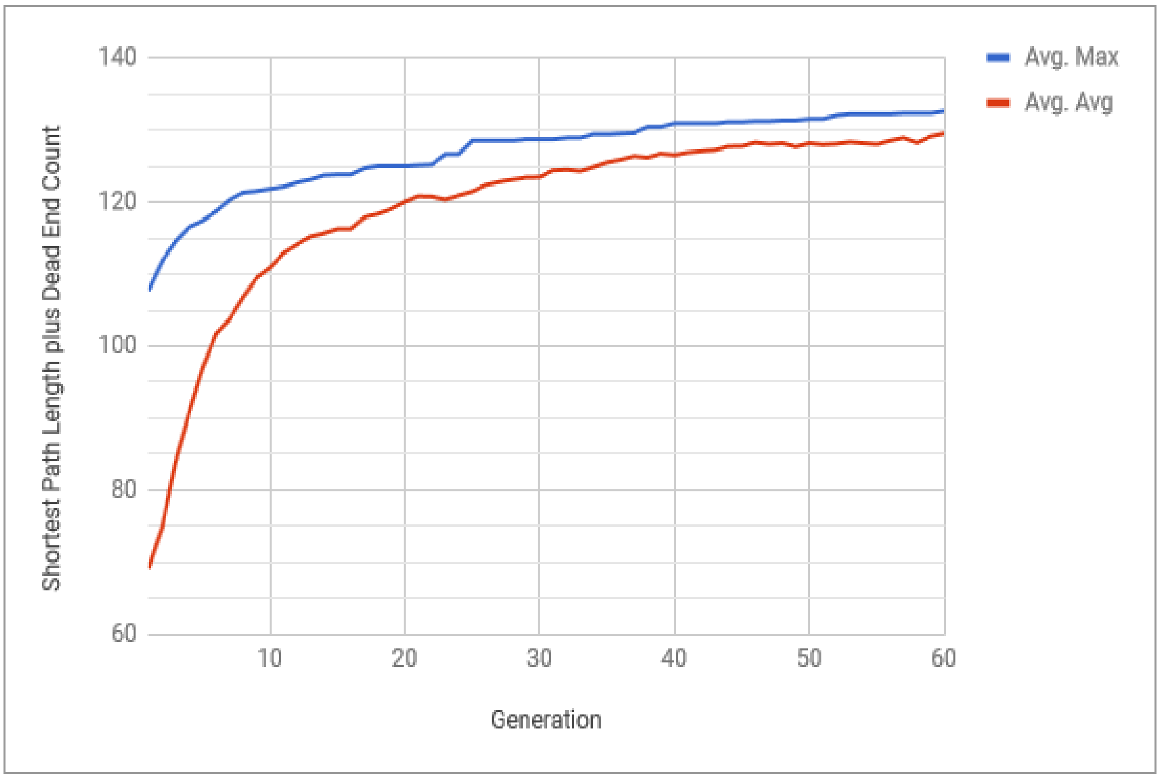


FIGURE 4.9: GA performance using $F1$ on the Blank Slate starting map state

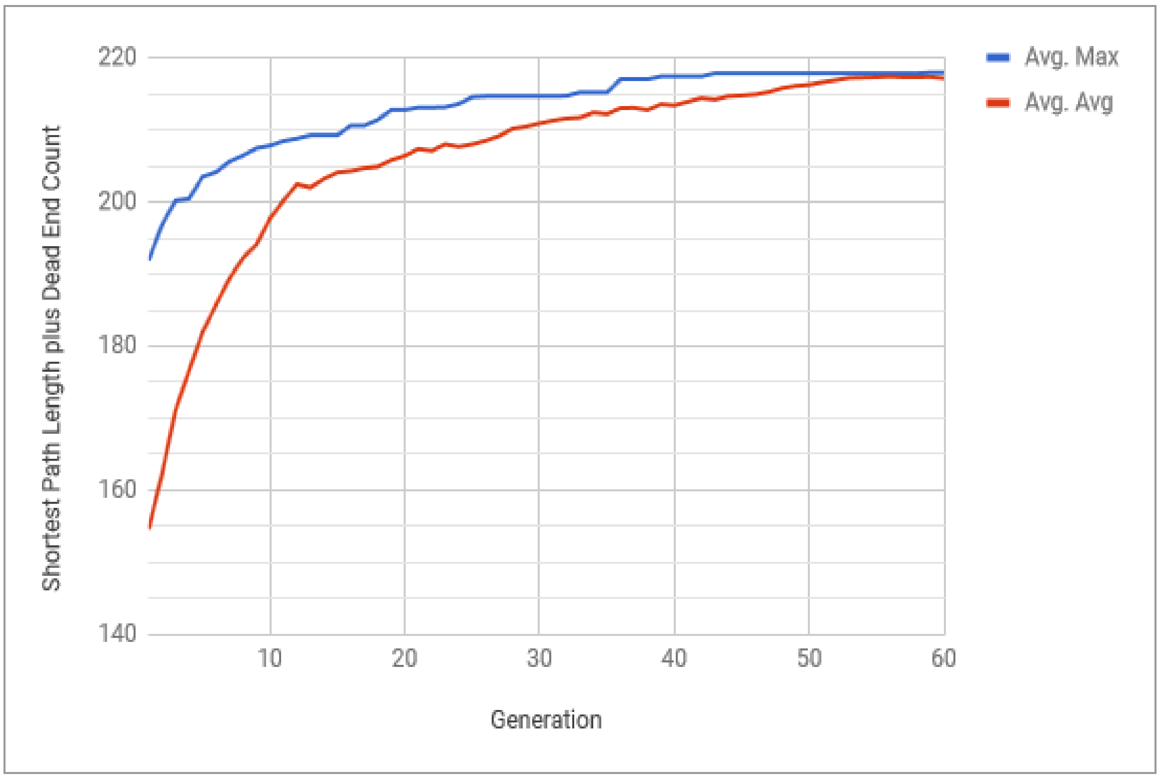


FIGURE 4.10: GA performance using $F3$ on the Filled Center starting map state

probabilistic chromosome were able to further optimize our fitness functions as well as break from the symmetric patterns that the binary representation tended to create. The superior performance of the probabilistic representation over the binary representation can be seen by comparing the values in Table 4.2 with those in Table 4.3. Our results show a marked increase in performance by the probabilistic representation over the binary representation on all three fitness functions. Table 4.4 shows that there exists a statistical difference between the results of the binary representation and our probabilistic representation as they all fall below the p-value of 0.05. Both representations were capable of producing mazes that consistently performed well on the three fitness functions, with the binary representation tending to produce mazes with one or more axes of symmetry while the probabilistic representation having no such tendency. In the following section we discuss the conclusions we have drawn from the results of our experiments and explore future extensions of the work presented in this manuscript.

Chapter 5

Conclusions and Future Work

Our research investigates maze generation for maze running games through the use of evolutionary cellular automata. We experimented with three different representations of mazes: maze initial state plus binary cellular automata rules, just the binary cellular automata rules, and probabilistic cellular automata rules. First we utilized an interactive genetic algorithm to determine several maze features that correlate to human players finding the maze interesting. This was done because what mazes human players find interesting can differ between players, making the creation of a fitness function for a traditional GA difficult. We had 10 different users operate as our IGA's fitness function for one run each. Once all runs of our IGA were completed we had the users run through selected mazes and provide a score afterwards on how much they enjoyed the maze. Through analysis of the mazes that received a higher score we found the maze qualities of shortest path length and total number of dead ends that positively correlated to the the score of the maze.

Therefore we were able to create three separate fitness functions that we could drive the evolution of a genetic algorithm with these two maze qualities. Using results from our IGA experiment we pruned the starting maze state from our representation and instead used two fixed starting maze states. Our representation then became 18 cellular automata rules from which we created two different representations, one where the rules were binary and another where the rules were probabilistic. The results from our binary representation showed that our approach was capable of evolving mazes to within on average 90% of the optimal solution. Mazes generated by our binary representation tended towards symmetrical patterns, which we infer as due to the symmetric maze starting states. Our results for the probabilistic representation exceeded those of the binary representation, outperforming on all metrics. The mazes generated by our probabilistic representation also did not display the symmetrical patterns that the binary representation did. The results of our experiments provide evidence that evolutionary cellular automata can be used to automatically generate interesting mazes for maze running games.

5.1 Extensions and Future Work

There are many directions for future research that our current work can extend into. First, multi-state cellular automata show promise in generating mazes, similar to the work by Ashlock using chromatic mazes [4], and could possibly increase the quality of mazes our approach generates. Multi-state cellular automata provide several advantages, such as allowing more maze features than wall and open space.

Second, three dimensional cellular automata would allow our approach to generate fully $3D$ mazes where the player can move vertically as well as laterally. Higher dimensions could also allow more intricate maze designs after conversion into $3D$ or $2D$ space.

Third, are alterations to the neighborhood space. Either adding or removing from the number of cells in the neighborhood could produce more interesting mazes.

Fourth, we are interested in extending our approach into game formats other than maze running games, such as real time strategy (RTS) map generation for games like StarCraft [11]. Extending into other game genres would would most likely involve cellular automata with greater than two states as well as possibly more than two dimensions.

Bibliography

[1] Mazes.ws. `http://www.mazes.ws/`.

[2] Chad Adams, Hirav Parekh, and Sushil J Louis. Procedural level design using an interactive cellular automata genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 85–86. ACM, 2017.

[3] Chad Adams and Sushil Louis. Procedural maze level generation with evolutionary cellular automata. In *Computational Intelligence (SSCI), 2017 IEEE Symposium Series on*, pages 1–8. IEEE, 2017.

[4] Daniel Ashlock, Colin Lee, and Cameron McGuinness. Search-based procedural generation of maze-like levels. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):260–273, 2011.

[5] D.E. Goldberg. Genetic algorithms in search, optimization, and machine learning. 1989.

[6] John Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, Cambridge, MA, 1992.

[7] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 395–402. ACM, 2011.

[8] Joshua R Smith. Designing biomorphs with an interactive genetic algorithm. In *ICGA*, pages 535–538, 1991.

[9] DuMim Yoon and Kyung-Joong Kim. 3d game model and texture generation using interactive genetic algorithm. In *Proceedings of the Workshop at SIGGRAPH Asia*, pages 53–58. ACM, 2012.

[10] Andrew Pech, Philip Hingston, Martin Masek, and Chiou Peng Lam. Evolving cellular automata for maze generation. In *Australasian Conference on Artificial Life and Computational Intelligence*, pages 112–124. Springer, 2015.

[11] Blizzard Entertainment. *Diablo.* 1997.

[12] Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N Yannakakis. Multiobjective exploration of the starcraft map space. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 265–272. IEEE, 2010.

[13] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[14] Lawrence Johnson, Georgios N Yannakakis, and Julian Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 10. ACM, 2010.

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

[16] John Von Neumann and Arthur Walter Burks. *Theory of self-reproducing automata.* University of Illinois Press Urbana, 1996.

[17] Martin Gardner. Mathematical games: The fantastic combinations of john conways new solitaire game life. *Scientific American*, 223(4):120–123, 1970.

[18] Sebastian Lague. Procedural cave generation tutorial, 2015. URL `https://unity3d.com/learn/tutorials/projects/procedural-cave-generation-tutorial`.

[19] Larry J Eshelman. The chc adaptive search algorithm: How to have safe search when engaging. *Foundations of Genetic Algorithms 1991 (FOGA 1)*, 1:265, 2014.