A Model for Measuring the Difficulty of a Maze and a System for Generating Mazes of a Custom Difficulty using a Neural Network and Modified Prim's Algorithm

Liam Peachey
ljpeach18@earlham.edu
Department of Computer Science
Earlham College
Richmond, Indiana

ABSTRACT

Mazes as a problem domain have wide applications, from art and games to testing decision algorithms. While there has been focus on the topology of mazes, recent work on generating specific levels of difficulty of maze is more scant. Additionally, existing models for analyzing a maze's difficulty may not accurately reflect difficulty for both humans and computers, with some mazes classified as more difficult appearing easier to a human solving the maze. I propose research exploring different methods for analyzing and classifying the difficulty of mazes, producing a unified model that reflects similar difficulty for humans and computers, followed by designing a neural network that guides maze generation to produce mazes of a specified difficulty rating.

.

1 INTRODUCTION

Mazes are a well known and easily understood problem domain in AI research. They have been produced throughout history for purposes of entertainment, defense, and decoration. The structure of a maze impacts the ease of which a human or algorithmic agent may navigate the maze. Either a human or an AI could easily solve a maze that is just a straight line to the goal, but it might be easier for a human to spot overarching patterns within a maze than for an algorithm. How would such a maze be classified? Models exist by which the difficulty of a maze may be estimated, but these have primarily been aimed at algorithmic agents [6]. Bellot et al.'s approach integrates McClendon's difficulty analysis, and produces a measure of how fun a maze is, integrating some of the ways humans scan mazes [1].

This research intends to improve upon and unify models by which the difficulty of a maze may be estimated in a way that takes both difficulty for humans and for an AI into account. I will also produce an algorithm that will generate mazes using a desired difficulty of the output maze to guide generation through the use of a neural network to transform a difficulty rating into a series of parameters that will affect how the maze is generated. The neural network will be trained using varying measures of difficulty, with the resulting mazes being tested using Gabrovšek's method for measuring the difficulty of a maze through the use of agents to determine which difficulty models produce the most accurate ratings [2].

This paper is structured to provide a detailed outline of the proposed project. The first section details the background of the field and techniques that will be employed in the research. The second section will provide a rough plan of the design and implementation ideas. The third section will outline major risks within the project. The final section will provide a timeline for the project.

CS388 Methods for Research and Dissemination, , Earlham College

2 BACKGROUND

Mazes at their cores are cell graphs. The edges of the graph can be traversable, or not. Those that are not are considered walls. In a typical maze, cells consist of four edges, forming a grid. However, this representation can be bent to create other structures as well [5]. For the purposes of this research, I will be focusing on the standard four neighbor cell graph. Perfect mazes are defined as mazes that have a singular path connecting any two cells [1]. While not all mazes are required to have this attribute, it does allow for easy maze generation through the use of spanning tree generating algorithms [3]. As a result, maze generation typically makes use of existing algorithms for generating these spanning trees.

As mazes are well known, there are already a number of algorithms to generate mazes. The Recursive Backtracking method chooses a cell on the graph, marks it visited, adds it to a stack, is marked as visited, and then chooses a random neighbor that has not been visited to join with. That neighbor becomes the next active cell, and the process repeats until there are no unvisited neighbors around the active cell. At this point, the current cell is abandoned, and a new cell is popped from the stack. Once the stack is empty, the algorithm halts [7]. This method is essentially a depth first traversal.

Kruskal's algorithm works by separating the graph into sets and then joining them back together one cell at a time. When two cells are joined, they become part of the same set. Cells are only joined with cells in different sets. Once the whole graph is part of the same set, the algorithm completes [7].

Prim's algorithm is similar to the recursive backtracking algorithm. It also makes use of a frontier to connect new cells. Instead of popping the next cell off the stack however, it randomly chooses a cell from within the frontier, and it then adds that cell to the "visited" or "in" group, and adds unvisited neighbors to the frontier [7]. This is the approach that this research will be building off of.

Bellot et al. outlines existing methods for assessing the difficulty of mazes, and produces their own model for calculating how fun a maze is [1]. McClendon's difficulty ratings, which Bellot et al. used and refined in their model, is based on the complexity of the solution path multiplied by the complexity of all branches in the maze [6]. Bellot et al.'s approach takes into account the number of what they refer to as "non-significant" walls [1]. According to them, mazes with a low number of non significant walls and the highest difficulty as rated by the McClendon model are the most fun.
Finally, Gabrovšek uses a number of agents to explore a maze, using their performance to determine which algorithms produce the most difficulty mazes [2]. Bellot et al. also produced two hybrid algorithms that they used for generating mazes that they determined to be more fun [1]. However, this and many other works failed to create systems of generating mazes of a specified difficulty. Susanto et al. produced a genetic algorithm for generating a type of maze [8]. Their mazes are not the usual cell graph type, and are specific to the game they developed. They encoded patterns into genes that would then be used within the genetic algorithm [8]. Their method also seems to be suited to generating higher complexity levels. The goal of our algorithm is to produce mazes of a specific complexity without long training cycles at runtime, so a genetic algorithm like this one would not be a good fit.

3 DESIGN AND IMPLEMENTATION

This project will make use of a neural network to process a difficulty rating, outputting parameters for an altered Prim's algorithm maze generator. Once the parameters are determined, the algorithm will generate a maze, which will be assessed by a difficulty model. This result will be used to grade the neural network.
The overall framework of the project is outlined in Figure 1. Input difficulty is the desired difficulty of the maze. This is passed to the maze generator neural network (MGNN) which will return a set of parameters for the main algorithm in an attempt to produce a maze that matches. The goal difficulty and generated maze is then passed to the difficulty model, which analyzes the maze difficulty. The MGNN then uses this value to grade, and proceeds to the next generation. This describes the learning loop. After this is completed, the algorithm may be used to generate a pool of mazes for agent functions to complete. The performance of these functions is then compared with the desired difficulty of the maze, as well as the difficulty model's assessment of the maze, and compiled as performance data.
The difficulty model will factor in the work of Bellot et al. to create measure of how fun a maze is, as well as re-examining McClendon's difficulty formula, which is already factored into Bellot et al.'s formula, to look for additional factors by which to determine a maze's theoretical difficulty [1, 6]. Additionally, I will be using Gabrovšek's method of employing maze solving agents to determine maze difficulty and complexity [2]. I will be adding additional agents however, including heuristic searches, such as A* search, greedy, etc. for more accurate measures as to how a computer may solve a maze as well as how to push the model further.
The neural network will use the difficulty model I develop in the first few weeks of the project as feedback. Parameters will include but may not be limited to the coordinates of the first cell in the maze that is visited, number of starting points, the "age" of the last visited cell to explore again from when hitting a dead end, directional biases, and maximum ratio of certain types of intersection (outlined by Kim and Crawis) to the maze's size [4]. Other factors will likely be added or removed during the course of the project. The core of the algorithm will be based on Prim's algorithm for maze generation [7]. Prim's algorithm makes use of a frontier built through cell discovery, which should allow for more control over determining how the maze will grow.
This work will be evaluated by comparing the desired difficulty rating of mazes generated, actual difficulty ratings as based on the models used, and the performance of the agent functions used later on. We will test multiple difficulty models for training the MGNN, and analyze all of them to see which works best with the Prim's algorithm system we develop.

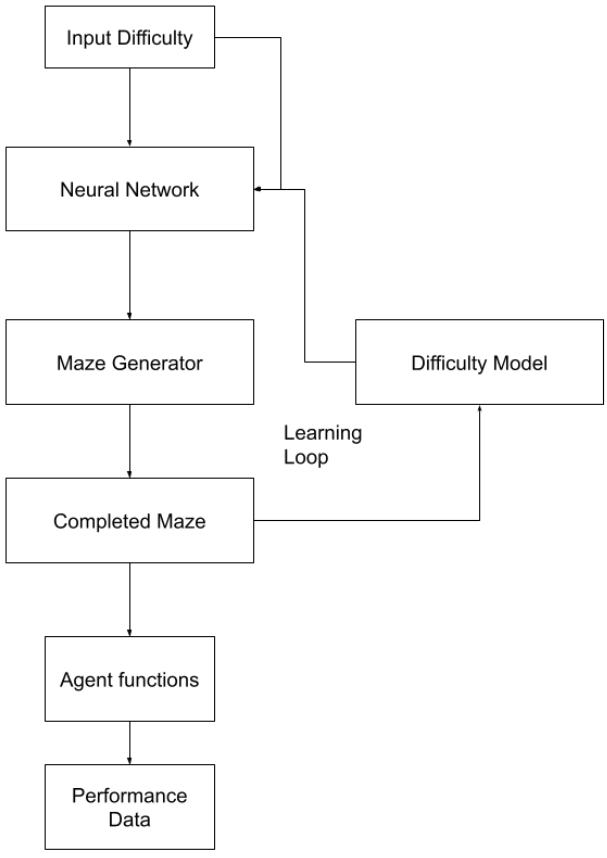

Figure 1: Project Framework

4 MAJOR RISKS

Neural networks are fairly new to me, so I will need to be able to reacquaint myself with them, and properly implement this when the time comes, and it will likely involve time during other sections to learn before beginning the development period. Additionally, as

the neural network will be based on the difficulty model, the model must be complete and fairly functional before I can fully implement begin work on training the neural network. I plan to make my work as modular as possible, so I can potentially circumnavigate this issue by having a backup maze difficulty model that I can substitute. This solution would also provide additional data for assessing the performance of my model. Finally, implementing additional maze generation and solving algorithms will take time and effort to understand and write. Working them into the overarching structure of my code may also be challenging. I have allotted time early on to work on these so that if they take longer than expected, or if unexpected issues arise from them I should be able to compensate.

5 TIMELINE

Week 1	1. Build maze domain 2. Implement McClendon and Bellot et al.'s difficulty and fun models. 3. Implement Prim's algorithm, recursive backtracking, and hunt and kill algorithms.
Week 2	1. Tweak difficulty models, and factor in other ways of measuring mazes 2. Implement A*, greedy, and random walk agents.
Week 3	1. Generate a set of mazes from each algorithm 2. Run difficulty analysis on mazes 3. Have agents complete the mazes, record details on performance of each for comparison with the predicted difficulty 4. Submit first draft of paper analyzing accuracy of model
Week 4	1. Tweak difficulty model as needed 2. Implement maze generation algorithm that will be used with the Neural Network 3. Implement additional agents
Week 5	1. First release of software, limited to difficulty model 2. Begin work on neural network using difficulty model to measure fitness
Week 6	Begin training the algorithm
Week 7	1. Tweak parameters and neural network 2. Train additional neural networks using other difficulty models
Week 8	1. Generate mazes using algorithm 2. Run and collect data from agent functions on generated mazes 3. Perform analysis on expected difficulty, actual difficulty, and difficulty according to performance.
Week 9	1. Record findings and update/expand paper to match current state of findings 2. Submit second draft of paper
Week 10	1.Tweak algorithms/do more training 2. Implement more agents/Implement other difficulty analysis techniques
Week 11	Collect additional data and compare difficulty model with other analysis techniques
Week 12	Begin work on poster. Gather data and visuals.
Week 13	Work through final touches, refine software for ease of use/testing
Week 14	First draft of poster due
Week 15	1. Add final touches to paper, software, and poster 2. Submit final draft of paper 3. Release final software 4. Present Poster

6 ACKNOWLEDGEMENTS

Thank you to David Barbella for his time, patience, feedback and willingness to help throughout the course of producing this proposal.

REFERENCES

[1] Victor Bellot, Maxime Cautrès, Jean-Marie Favreau, Milan Gonzalez-Thauvin, Pascal Lafourcade, Kergann Le Cornec, Bastien Mosnier, and Samuel Rivière-Wekstein. 2021. How to generate perfect mazes? *Information Sciences* 572 (2021), 444–459.

[2] Peter Gabrovšek. 2019. Analysis of maze generating algorithms. *IPSI Transactions on Internet Research* 15, 1 (2019), 23–30.

[3] Paul Hyunjin Kim. 2019. *Intelligent maze generation.* The Ohio State University.

[4] Paul Hyunjin Kim, Jacob Grove, Skylar Wurster, and Roger Crawfis. 2019. Design-centric maze generation. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*. 1–9.

[5] Xue Li and David Mount. 2016. Spherical Maze Generation. (2016).

[6] Michael Scott McClendon et al. 2001. The complexity and difficulty of a maze. In *Bridges: Mathematical Connections in Art, Music, and Science*. Citeseer, 213–222.

[7] Ms Shivani H Shah, Ms Jagruti M Mohite, AG Musale, and JL Borade. 2017. Survey Paper on Maze Generation Algorithms for Puzzle Solving Games. *International Journal of Scientific & Engineering Research* 8, 2 (2017), 1064–1067.

[8] Evan Kusuma Susanto, Rifqi Fachruddin, Muhammad Ihsan Diputra, Darlis Herumurti, and Andhik Ampuh Yunanto. 2020. Maze Generation Based on Difficulty using Genetic Algorithm with Gene Pool. In *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)*. IEEE, 554–559.