Investigating Puzzle Difficulty Assessment:

Minimum Solution Length is the Best Indicator of Logic Maze Difficulty

CMP504 Report

Stephen Jones

2004792

Abstract

This project investigates the effectiveness of different types of predictors for estimating the difficulty of logic maze puzzles. Puzzle difficulty assessment would be required for the development of procedurally generated puzzle games, as well as being useful for reviewing and improving upon hand-crafted content. Previous research has shown that a variety of puzzle properties can be used as predictors to create a linear model for puzzle difficulty; predictors arising from human solving methods have shown particular promise. The human-assessed difficulty of puzzles was determined using data from online volunteers, then tested for correlation with predictors produced by two distinct computer solvers: one based on a brute force algorithm and the other based on a human-inspired expert system combined with a depth-first search. It is found that in the case of logic mazes, the length of the shortest solution is by far the most effective single predictor, to the extent that no combination of predictors substantially outperforms it alone.

Keywords

Depth-first search, difficulty, expert system, logic maze, procedural content generation, puzzle games

Contents

1 Introduction

Puzzle games as a genre owe much of their popularity to *Tetris* (1984), the most ported video game of all time (Guinness World Records, 2008). A reason for *Tetris*' status as one of the greatest video games of all time is its elegant use of randomness. Since the pieces given to the player are randomly decided, the game is infinitely replayable as no two games of *Tetris* will be identical; there is always something more to learn and each challenge you face will be slightly different (Shaver, 2017).

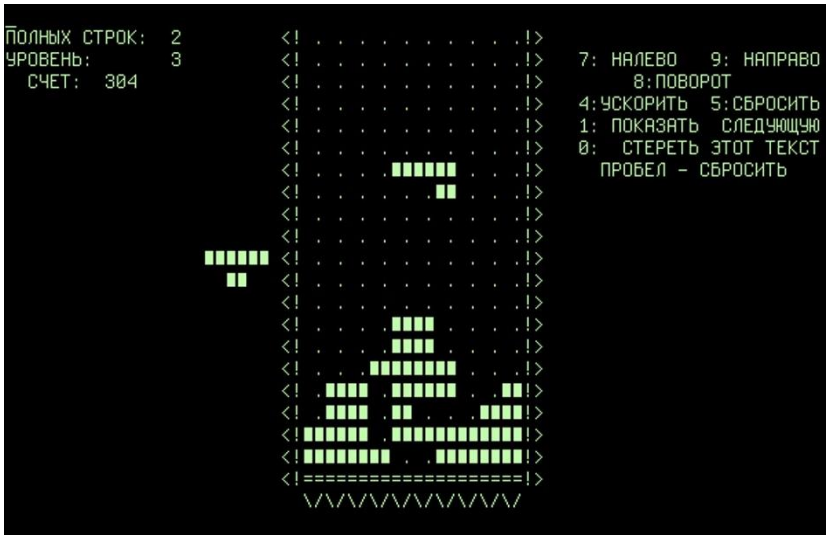


Figure 1-1: Tetris (1984), one of the greatest video games of all time (Chaplin, 2007). Image credit: Wikipedia

Since Tetris, the complexity of games has markedly increased. Procedural content generation (PCG) is an important tool used in the production of many genres of games since the 1970s (*Advanced Dungeons and Dragons*, 1977; *Beneath Apple Manor*, 1978). The term covers a wide variety of automated techniques for the creation of game content including game files, maps, systems and puzzles (Hendrikx et al., 2013). PCG has several important benefits as a content generation method including greater replayability and a reduction in development time (Togelius et al., 2011).

Despite the advantages of PCG, it is not utilised as much for the development of puzzle games as it is for many other genres. De Kegel and Haahr (2020) identified four main unsolved challenges with the procedural generation of puzzles:

1. The creation of a high-quality progression throughout the game. This involves both ensuring an appropriate difficulty curve and the gradual introduction of new concepts, with specific puzzles intended to familiarise players with those concepts.
2. The development of PCG techniques that are not limited to the production of a very specific puzzle type.
3. The accurate assessment of procedurally generated puzzles for difficulty and variety.
4. The development of PCG techniques which produce puzzles that are aesthetically pleasing to humans.

It is the third of these challenges, particularly the estimation by a computer of the human perception of puzzle difficulty, that this project is concerned with. This is a difficult problem for multiple reasons. Current difficulty estimation techniques are often limited to specific puzzle rule sets. Additionally, superficially similar puzzles can be vastly different in difficulty (Ashlock & Schonfeld, 2010).

Aponte et al. (2011) note how important accurate puzzle difficulty assessment is to achieve in order to create an enjoyable game experience, since a novice player faced with an overwhelming challenge too soon is likely to become frustrated, while an experienced player presented with too many similar puzzles will soon grow bored. By fitting the difficulties of puzzles to the desired difficulty and learning curves of the game, a more compelling experience can be created.

1.1 Study Objectives and Outline

According to the research of van Kreveld et al. (2015), the difficulty of a puzzle can be modelled as a linear combination of numerical properties of the puzzle: that is, a weighted sum of these properties. These properties will henceforth be called *predictors*. By collecting data on the human perception of the difficulty of a set of puzzles and performing a multiple linear regression, the weights can be determined. This produces a formula to estimate the difficulty of other, similar puzzles.

There are very many properties that might be indicative of difficulty for any given type of puzzle. This project's aim is to build on previous work by both extending the method of van Kreveld et al. to a new type of puzzle, a logic maze, and by comparing the effectiveness of different combinations of predictors at difficulty estimation for this new type. These properties come in 3 types:

- *Type A* predictors are those that are readily apparent by looking at the puzzle or otherwise trivial to compute, such as the puzzle's size. The properties considered by previous work have mostly been of this type.
- *Type B* predictors require knowledge of the puzzle's solution(s), such as the minimum number of "moves" required to complete the puzzle. To be sure of these values for procedurally generated puzzles, a brute-force approach might need to be used; for human-designed puzzles they might be easier to determine.
- *Type C* predictors concern the steps required to solve a puzzle, such as a measure of how far the solver must think ahead while solving the puzzle in order to make the most difficult deduction required.

To complete the project's objectives, the following three steps were performed.

Firstly, a set of puzzles had to be created to be used for the analysis. 30 logic maze puzzles were used in this project. Secondly, data on the difficulty of these puzzles had to be collected. This was achieved by creating a website which volunteers could use to solve the puzzles and report how difficult they found each one. Thirdly, both a brute force algorithm and a human-inspired solver had to be designed and implemented. These two tools could then be used to determine the puzzles' type B and C predictors, allowing a suitable difficulty formula to be determined.

In summary, this project's question is: which properties of puzzles are the best for difficulty estimation? This question will be answered for a specific puzzle type using two solvers, each of which generates a different type of property, and comparing both the efficiency of the solvers and the accuracy of the metrics each produces.

1.2 Report Overview

The Background section of this report will cover the previous research in the area of puzzle difficulty estimation in more detail as well as the benefits and impacts of this research. It also explains some of the technical concepts used in this project.

Next, the Methodology section will detail the design of the applications created as part of this project. It will also cover the type of data collected and how this data was analysed and assessed, according to the project's objectives.

The Results section will present and analyse the collected data.

The Discussion section will consider the question of efficiently computing minimum solution length, which is found to be the most important predictor of logic maze difficulty.

Finally, the Conclusions section will draw conclusions from the project's results, note the project's limitations and what suggest could be looked into for future work in the area.

2 Background

This section starts by covering the motivation behind this project and the possible applications of work on puzzle difficulty assessment. Next, there is a review of previous research in the area. Lastly, the section ends with explanations of technical structures used in the project: the logic maze (the type of puzzle being investigated), the expert system and the depth-first search.

2.1 Benefits and Applications

Puzzle games are an important area of research not only for games designers but those in education and medicine (Perrotta et al., 2013). Puzzle difficulty assessment is important for the creation of puzzles, either procedurally or by hand. Procedural content generation (PCG) sees little use in puzzle games but could become a greatly beneficial tool for their development if certain challenges, among them automated difficulty estimation, can be overcome (De Kegel & Haahr, 2020).

2.1.1 Applications of Puzzle Games

While *Tetris* was commonly frowned upon during the 1980s due to its addictiveness (Plank-Blasko, 2015), more recent research has shown it and a variety of other puzzle games to have several applications in both medicine and education.

Tetris was demonstrated to increase grey matter in the brain (Haier et al., 2009), to be more helpful in treating amblyopia (‘lazy eye’) than the traditional method of an eyepatch (Sengpiel, 2014), and to aid in dealing with post-traumatic stress disorder after road accidents (Holmes et al., 2009). Numberlink puzzles involving connecting numbers in a grid with non-overlapping paths were used by Nef et al. (2020) who found a significant difference in the performance of patients of Huntington’s disease and healthy adults of the same age.

In education, puzzle games are a natural choice as they allow for players to both train particular skills and simultaneously learn or revise specific information (Perrotta et al., 2013) (Zirawaga et al., 2017).

2.1.2 Benefits of Procedural Content Generation

Procedural content generation has been a central part of a variety games for over four decades, from dungeons in *Rogue* (1980), to 3D terrain in *Minecraft* (2011), to the personalities of non-player character adversaries in *Middle-earth: Shadow of Mordor* (2014).

Replayability is a key benefit of many applications of PCG. While a game’s basic goals and progression may be the same each time it is played, it can remain enjoyable when played multiple times because differences in the game world or characters created by PCG lead to different decisions and gameplay in each playthrough (De Kegel & Haahr, 2020). Puzzle games can often suffer particularly from a lack of replayability, when the same puzzles are presented to the player each time: playing a puzzle a second time does not allow the player to show mastery of the puzzle rule set, only memory of that puzzle’s solution. Because of this, better methods of applying PCG to puzzle games would be a significant boon to the genre.

Replayability is even more important in educational games than recreational ones. In an educational game, introducing a skill for the first time is not the only important aspect; students also need to be

Figure 2-1: Procedural content generation is central to many games including *Minecraft* (2011), but it is rarely used in the puzzle genre (De Kegel & Haahr, 2020). Image credit: Mojang Studios

able to practice skills they have learned in previous sessions. This cannot be done using the same puzzles each time (Dong & Barnes, 2017).

A second benefit of procedural generation is that it can vastly reduce the development time of a game. Murray (2003) estimated that creating one hour worth of high-quality educational content for a game by hand can take an expert designer as much as 300 hours. With procedural generation, huge amounts of content can be produced with little or no developer time and effort after the initial tool is created. Part of the reason for this time is that it is difficult for a human to imagine very many variations on the same theme, meaning that even an expert can have trouble creating puzzles that are both similar enough to form a coherent recreational or educational experience, but different enough to provide an interesting and varied challenge (Togelius et al., 2011). As such, procedural generation can improve not only the quantity but also the quality of game content produced in a given timeframe.

Finally, De Kegel and Haahr (2020) speculate that procedural generation might be the key to creating a truly user-tailored experience in puzzle games. By allowing the parameters of a procedural generator to vary according to the specific actions and aptitudes of a player, puzzles and gameplay might be produced that is unique for each person, which could lead to particularly satisfying gameplay for a wide variety of players. Pelánek (2016) demonstrated the feasibility of using an Elo-like system for this purpose in educational games, where students are matched with tasks according to a rating derived from their performance on previous tasks.

2.1.3 Benefits of Difficulty Estimation Outside PCG

The automated estimation of puzzle difficulty is not only useful in the production of procedurally generated puzzles but also in the curation of hand-crafted puzzle games.

It can be hard for a designer to impartially assess the difficulty of a puzzle they created, since they already know the solution. Similarly, a playtester for a puzzle game might have less useful feedback on the second iteration of a puzzle that has been updated according to previous comments, due to their memory of the original puzzle. By applying an automated difficulty estimation tool to the problem, designers could reduce the number of iterations and playtests required to create a high-quality puzzle of a desired difficulty (van Kreveld et al., 2015).

Some games involve a way for players to create puzzles or levels and share them with others to play. With the potential for a huge number of puzzles created in this way, it is unfeasible for developers to assess these puzzles for quality or difficulty. Any difficulty rating for these puzzles is generally provided by players in most current implementations, but automated difficulty estimation could be helpful as an alternative way or to augment player feedback for classifying and moderating these puzzles (Hicks, 2013).

2.2 Previous Research on Puzzle Difficulty Estimation

Several approaches have been used in the past to estimate puzzle difficulty, with varying degrees of success.

In this field of research, one of the first things that must be done is to decide how to determine a ground truth for the difficulty of a game or puzzle. Most researchers have used human participants. Some (e.g. Jarušek & Pelánek, 2011; András et al., 2013) measured the time taken for participants to complete a task. However, van Kreveld et al. (2015) noted that time taken can be an unreliable indicator of difficulty, especially when a wide domain of challenges is being studied; they chose instead to ask participants for their opinion on the difficulty of each challenge. As opposed to this more subjective approach, Aponte et al. (2011) noted the lack of any precise mathematical definition of difficulty in video games research and aimed to create one. Their definition depends on the chance of success at that task as a function of player ability.

2.2.1 Use of Genetic Algorithms

A genetic algorithm is an algorithm to solve an optimisation problem that is based on the principle of biological evolution. It uses an iterative process of testing, selection and breeding to optimise a population of genomes, each coding for an individual in the solution space of the problem to be solved (Sivanandam & Deepa, 2008).

While genetic algorithms have seen some success in both puzzle solving and puzzle difficulty estimation, they tend to be less computationally efficient than other methods. Mantere & Koljonen (2007) found a genetic algorithm to be more reliable but less efficient than previous methods tried at solving Sudoku puzzles. They noted that the average number of generations required to solve a puzzle correlated well with difficulty but that upper and lower bounds did not, meaning that to get an accurate estimation of difficulty the algorithm might need to be applied many times. Ashlock & Schonfeld (2013) used the time taken by and number of failures made by an evolutionary algorithm to assess the difficulty of Sokoban (box-pushing) puzzles. They found this approach moderately successful, but that it was easy to generate a model that was overfitted to a specific style or structure of puzzle.

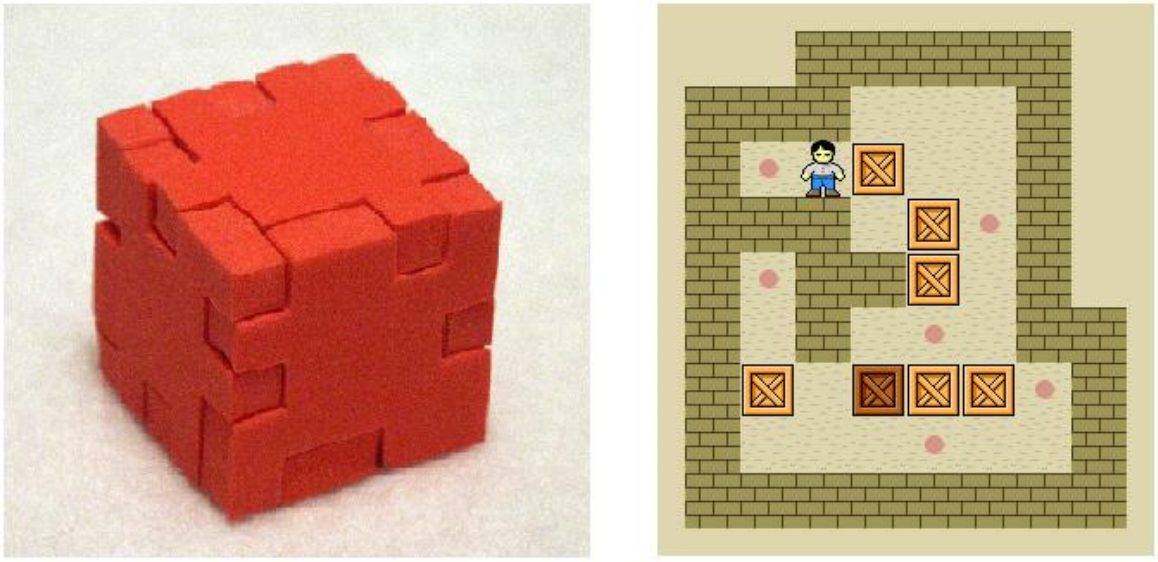


Figure 2-2: Happy Cube and Sokoban, other types of puzzle that have seen similar attempts to estimate their difficulty (András et al., 2013; Jarušek & Pelánek, 2011). Image credit: Wikipedia

2.2.2 Use of an Expert System and Depth-First Search

Jing et al. (2009) found an expert system (see section 2.4) based on rules of logical deduction and a guessing strategy of a depth-first search (see section 2.5) to be just as reliable and significantly more computationally efficient at solving nonograms than either a genetic algorithm or a pure depth-first search. The algorithm is faster in both the average-case and worst-case than either of the other options. A similar algorithm was chosen to estimate type C predictors in this project.

2.2.3 Difficulty Predictors

According to van Kreveld et al. (2015), the difficulty $D(P)$ of a puzzle P can be estimated as a linear combination of predictors $a_i(P)$, i.e.

$$D(P) = w_0 + \sum_i w_i a_i(P)$$

where w_i is the weight of the property a_i. These weights are considered constant for puzzles of the same type. By collecting data on the difficulty of puzzles of three different types, they determined weights for different predictors in each type. This allowed them to estimate the human-assessed difficulty of new puzzles using the above formula. The average error of the estimations varied by puzzle type between 0.40 and 1.01 on a 1-10 difficulty scale. The approach was most successful with Flow puzzles, in which coloured points on a grid must be connected by non-overlapping paths; the predictors used for Flow puzzles were a mixture of type A and type B, as defined in section 1.1.

This was partially based on earlier work by Browne & Maire (2010) who used a similar approach to estimate how enjoyable rulesets for abstract games created by a genetic algorithm would be for humans. The predictors used were determined by analysing games played under that ruleset and included the length of a typical game, the probability of a draw and the probability of recovering from a losing position. This approach resulted in the invention of Yavalath, which has since been

commercially published. However, this was less reliable than later attempts to use the same method for estimating difficulty, with several games that were predicted to be enjoyable being unsuccessful with playtesters.

2.2.4 Use of Type C Predictors

Jarušek & Pelánek (2011) explored the correlation between various predictors and difficulty in Sokoban puzzles. They decomposed puzzles into several sub-tasks and used predictors based on the way these sub-tasks must be interleaved during solving to predict difficulty. They also used a computational model to simulate the way a person traverses the state space of a Sokoban puzzle to estimate difficulty with similar accuracy. These methods are based on different type C predictors, and both were more accurate than those they attempted based on type A and B predictors. They suggested that future improvement in difficulty prediction will similarly rely on type C predictors.

András et al. (2013) created graphs to represent possible routes taken while assembling Happy Cube puzzles. They then used features of that graph, such as the way in which backtracking was required, to estimate the cube's difficulty. Conversations with solvers of these puzzles confirmed that a trial-and-error approach involving backtracking (another example of a depth-first search) was a common strategy. Their approach was more closely correlated with the time taken by solvers than the difficulty ratings for Happy Cube puzzles given by their manufacturers.

2.2.5 Summary

Previous research on puzzle difficulty assessment indicates that a variety of puzzle types have difficulties that are linearly correlated with different predictors, with all three types of predictors proving effective depending on the type of puzzle. Other approaches have been tried, especially genetic algorithms, but they have been shown to be slow and unreliable at solving logic puzzles compared to a method based on human reasoning. An expert system using a depth first search is a human-inspired technique that has been shown to be both reliable and efficient at solving logic-based puzzles. The successful use of type C predictors in the past shows that studying the solving process of a puzzle can provide an insight into its difficulty. However, type A and B predictors have also been used, just as successfully, for other puzzle types and genres.

2.3 Logic Mazes

The style of puzzle used in this project is called a logic maze. A logic maze is a type of logic puzzle invented by Robert Abbott and first published in 1962 (Gardner, 1962). In a logic maze, the solver must find a route through a maze which conforms to certain rules. Logical deduction is typically used to find this route, hence the name logic maze. The logic mazes used here are similar in structure to the Flow puzzles investigated by van Kreveld et al. (2015), which were the type of puzzle most successfully modelled by their approach, as they both involve linking points on a grid using non-overlapping paths.

The Witness (2016) is a puzzle game involving several kinds of logic mazes in which the player must deduce various rules according to symbols present in the maze. The type of logic maze chosen for this project is equivalent to one of the simplest types present in *The Witness*. In this type of maze, the solver must find a single path along the dotted lines, linking the start and end points (the grey squares at the top left and bottom right of the maze). The path must not meet or cross itself and it must pass through all of the checkpoints (crosses).

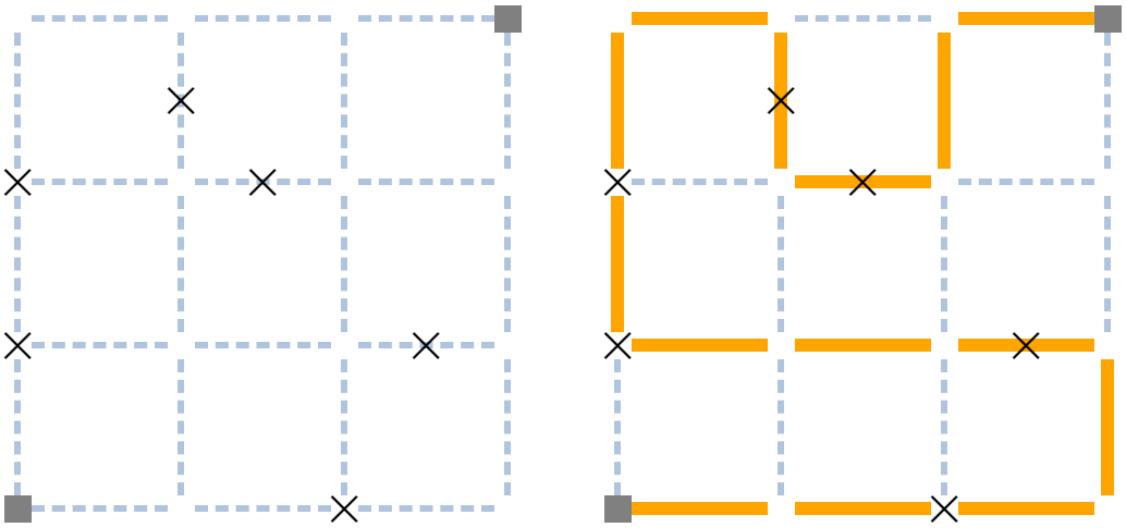


Figure 2-3: A logic maze of the type used in this project – blank (left) and with one of the two possible solutions marked (right).

Unlike many types of logic puzzle, it is not necessary for there to be a unique solution to this type of logic maze, but only for there to be at least one solution. The solver is asked to find any one solution to the maze obeying its rules. The example given in Figure 2-3 has exactly two solutions.

2.4 Rule-Based and Expert Systems

A *rule-based system* is a simple form of artificial intelligence which applies human-crafted rules to manipulate and interpret data. A rule-based system is comprised of a *rule base* and a *database*; the system applies the *production rules* in the rule base to the *data* in the database using an *inference engine*. An *expert system* is a type of rule-based system which uses rules designed to codify the knowledge of a human expert in a specific field. The rules are designed by an expert in the task that the system is designed to perform; this individual is called the *knowledge engineer*. An expert system was used in this project to compute type C predictors for logic mazes.

A rule-based system's production rules are logical statements in an "if-then" form which state what the system should do given specific data. When a rule's hypothesis ("if") is satisfied, the rule is *triggered*. In the simplest implementation of a rule-based system, any rule which is triggered is immediately *fired*, meaning that its conclusion ("then") is executed, in some way changing the data. A more complex implementation uses an *arbiter* or *conflict resolver* to resolve conflicts when two rules are triggered simultaneously before executing any rules. Once one or more rules have been fired, the process repeats (Millington, 2006).

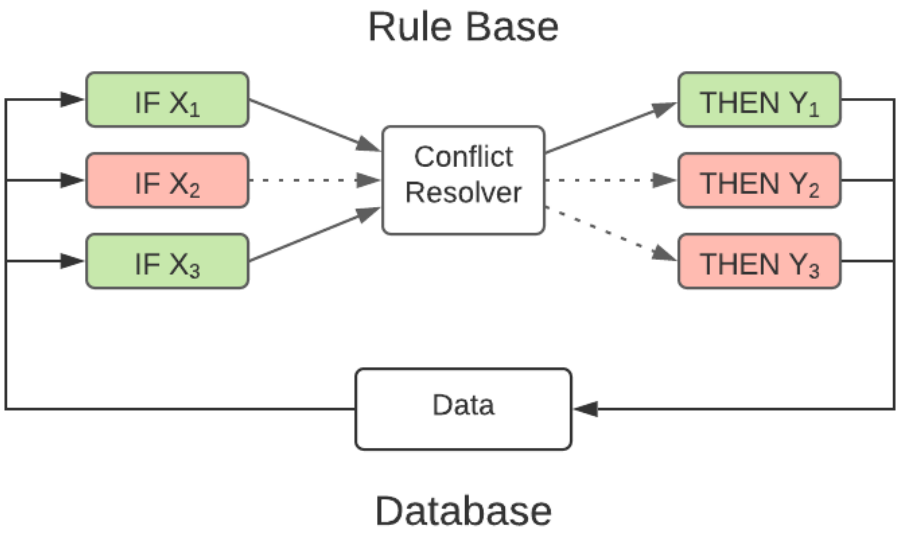


Figure 2-4: The architecture of a rule-based system. In this example the system has 3 rules. Both hypotheses X_1 and X_3 are satisfied (green), so both rules 1 and 3 are triggered (solid arrow). The conflict resolver determines that only rule 1 should be fired (green). After the conclusion Y_1 has been applied to the data, the process will repeat on the updated data.

Expert systems are particularly suited for repetitive tasks that nonetheless require detailed knowledge of the problem space to perform. It is also important that the task should be both qualitative and reasonably narrow in scope. By codifying the expertise of the knowledge engineer into rules that can be applied by the artificial intelligence, the system emulates the decisions of an expert but can be applied to a much larger volume of scenarios.

The main drawback of an expert system is its lack of scalability, which occurs because each rule utilised must be meticulously designed by the knowledge engineer. While a human expert could learn from experience and respond creatively in a new situation, the expert system relies on maintenance from the engineer in order to improve its effectiveness or scope (Tripathi, 2011).

An expert system was chosen for this project because it is perfectly suited to this application. This was mainly because of the way it emulates the methods of a human solver. By simulating the way a human approaches a puzzle, it was hoped that an expert system could provide an insight into the difficulty of a puzzle according to humans by estimating type C predictors. This is why other types of rule-based systems would not be appropriate: it is important that the rules used are the same as deductions that a human might make. Another reason for using an expert system is that the domain of solving a particular type of logic maze is extremely narrow, is qualitative, and the solving process can be broken down into simple, explainable steps which can in turn be codified by simple rules. Since the field is entirely static, the issue of scalability does not arise. Finally, this approach has proven to be highly computationally efficient compared to other methods (Jing et al., 2009).

2.5 Depth-First Search

A depth-first search is an algorithm by which a tree is traversed by exploring an entire branch of the tree before moving onto another part of the tree. This is as opposed to a breadth-first search, in which the tree is traversed in order of distance from the root node. A depth-first search is equivalent to a trial-and-error approach by which a course of action is chosen and not abandoned until it is either successful or proven to not lead to any successful outcome. This has been shown to be a method by which humans solve some puzzles (András et al., 2013). It was chosen in this project to simulate the way in which humans might guess at part of the solution to a puzzle when they cannot otherwise make progress.

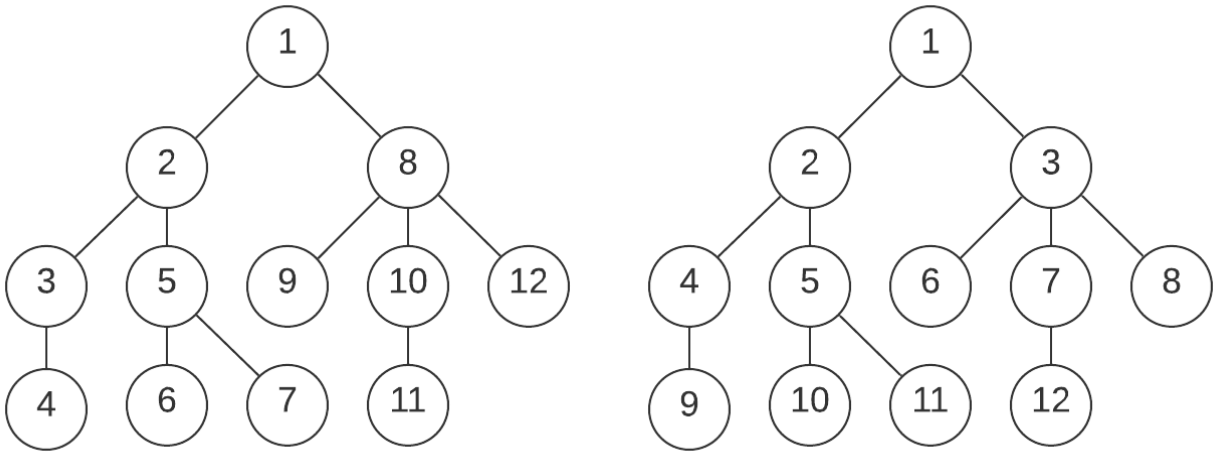


Figure 2-5: The order that nodes are considered in a possible depth first search of a tree (left) and a possible breadth-first search of the same tree (right).

3 Methodology

3.1 Definitions

This section defines terms used in this report to talk about logic mazes.

The maze is considered as a *graph* (that is, a set of *vertices* connected by *edges*). The vertices are arranged in a square grid, with vertices that are adjacent horizontally or vertically being connected by an edge. The bottom left and top right vertices in the grid are called the *endpoints* of the maze. The *size* of the maze is equal to the number of vertices along each side of the grid.

A *path* in the maze is a sequence of vertices where successive vertices in the path are connected by an edge. A path *includes* a vertex if that vertex is in the sequence, and it *includes* an edge if some pair of successive vertices in the sequence are connected by that edge. The *endpoints* of a path are the first and last vertices in the sequence.

A *loop* is a path whose endpoints are the same vertex.

Some vertices and edges in the maze have *checkpoints*, and some edges have *breaks*. A checkpoint is *satisfied* by a path if that path includes the vertex or edge of that checkpoint. A break is *satisfied* by a path if that path does not include the edge of that break.

A *solution* to the maze is a path which (a) contains no repeated vertices, (b) has endpoints which are the endpoints of the maze, and (c) satisfies every checkpoint and break.

The expert system used in this project was designed to find a solution to a maze in a human-like way. At any given point during this process, it considers each edge and vertex to be:

- *included*, if it is known or guessed that it will be included in the solution
- *excluded*, if it is known or guessed that it will not be included in the solution
- *unknown*, if it is not known or guessed whether it will be included in the solution

An *end* is a vertex with exactly 1 included edge connected to it. An end is *isolated* if there is no path from it to any other end which only includes unknown edges.

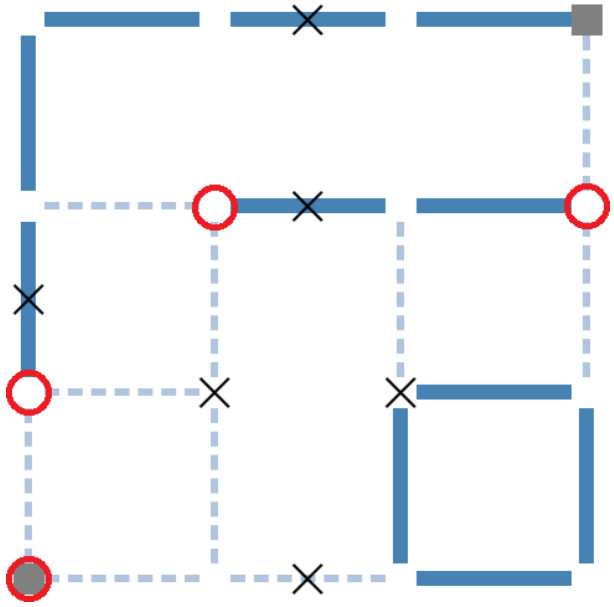


Figure 3-1: Examples of definitions.

This puzzle is of size 4, with 16 vertices and 24 edges. The 11 included edges are drawn as solid lines, the 10 unknown edges as dotted lines and the 3 excluded edges are not drawn. Checkpoints are drawn as crosses, and grey squares mark the endpoints of the puzzle.

At the bottom right there is a loop comprised of four included edges.

The red circles indicate ends. Only the end on the right side of the puzzle is isolated.

3.2 Maze Generation

30 logic mazes were created for use in this project. These were made by randomly generating a suitable path through a grid with a C++ application, then placing checkpoints and breaks appropriately by hand. The puzzles used can be found in Appendix A.

3.2.1 Type A Predictors

Type A predictors for each maze could be generated immediately since they are (by definition) trivially apparent from the unsolved maze. The type A predictors collected were:

A_1: The size of the puzzle: that is, the number of vertices to a side of its grid. While there is no fundamental reason why they must be, for simplicity's sake all mazes used were square.

A_2: The number of vertices in the puzzle. This is equal to the size squared.

A_3: The number of edges in the puzzle without breaks. Edges with breaks were not counted since they were hidden in the puzzles presented to participants.

A_4: The number of checkpoints on vertices.

A_5: The number of checkpoints on edges.

A_6: The total number of checkpoints ($A_6 = A_4 + A_5$).

3.3 Data Collection

In order to train the AI, a large amount of data on the difficulty of suitable puzzles was required. This data was collected from members of the public using a dedicated website, built using the Django framework for Python and hosted on pythonanywhere.com. The project was approved by Abertay University's Research Ethics Committee. Because of the terms of this approval, participants were required to be at least 16 years of age. In addition, all data was stored entirely anonymously, and it was possible to opt out of the process at any point.

The website was advertised both publicly on Twitter and by word of mouth.

3.3.1 Website Structure

The website's main purpose was to present puzzles to the volunteer users, allowing them to solve the puzzles and rate their difficulty. It was also necessary to brief and debrief users, ensuring that informed consent could be given for collecting the data they provided.

On reaching the site, users were presented with a welcome screen which provided information about the project. From there they could choose to start a session.

On starting a session, users were presented with three questions. These questions were designed to give context to the user's subjective difficulty assessment of the puzzles. Next, users were presented with a tutorial on the type of logic maze being used. The questions and tutorial can be found in Appendix B.

After completing the tutorial, they were presented with each of the puzzles one at a time. Upon completing a puzzle, users were asked to rate its difficulty on a Likert scale between 1 (very easy) and 10 (very difficult). Users were also given the option to skip a puzzle that they found too difficult. In this case, the difficulty rating was considered to be 10.

The appearance of the puzzles on the website was the same as their appearance in the figures of this report.

As well as the difficulty rating given by the user, the website recorded the following data about how the user solved the puzzle:

- The time it took for the user to complete the puzzle (in milliseconds)
- The number of times the user clicked an edge before completing the puzzle
- The length of the solution the user found

It was hypothesised that participants might improve at the puzzles as they completed more of them, consequently rating their difficulty as lower. Because of this, the puzzles were presented in a different random order for each user, so that the average difficulty rating assigned to a puzzle would still be meaningful even if this was the case.

Once they completed all 30 puzzles, users were thanked and debriefed. They were asked not to do the same puzzles again, but because of anonymity it is not possible to tell if anyone disregarded the request. This was considered acceptable since it is unlikely it had a significant adverse effect on the quality of data that was collected.

3.4 Brute-Force Solver

To provide additional metrics, a brute-force solver for the mazes was implemented using C++. This program systematically considers each possible path through the maze, making a note of each which satisfies every checkpoint.

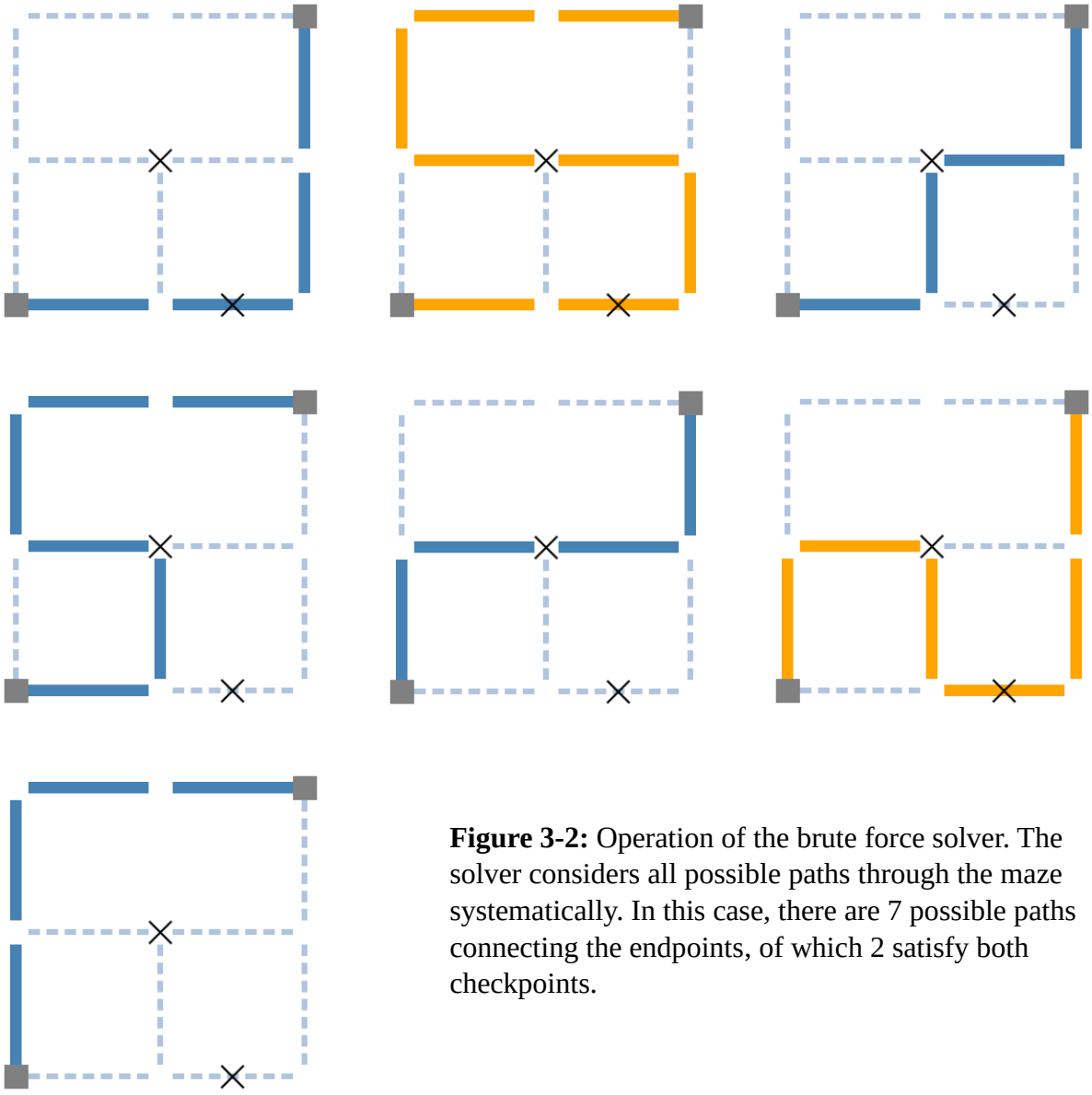


Figure 3-2: Operation of the brute force solver. The solver considers all possible paths through the maze systematically. In this case, there are 7 possible paths connecting the endpoints, of which 2 satisfy both checkpoints.

3.4.1 Type B Predictors

The brute force solver measured 3 type B properties, which depend on the maze's set of solutions.

B_1: The number of distinct solutions.

B_2: The least number of edges in any solution ("minimum solution length").

B_3: The minimum across all solutions of the maximum separation (measured in edges) between two consecutive checkpoints or endpoints along the path.

B_3 is intended to be a measure of how counter-intuitive the solution to a maze is. Since it is natural to want to link nearby checkpoints with a short path, it is possible that mazes with checkpoints that are not close together on the path are harder to solve.

3.5 Human-Inspired Solver

An expert system was chosen as a way of simulating human reasoning when solving a puzzle This solver was also implemented as a C++ application. As described in section 2.4, an expert solver operates by repeatedly applying the rules in its ruleset to data in its dataset. In this case, the data in question is a puzzle, which starts completely unsolved. Applying the rules of the system to the puzzle iteratively builds a solution. The operation of this system is summarised in Figure 3-3 and detailed in the following sections. Appendix C gives a full example of the solver's operation.

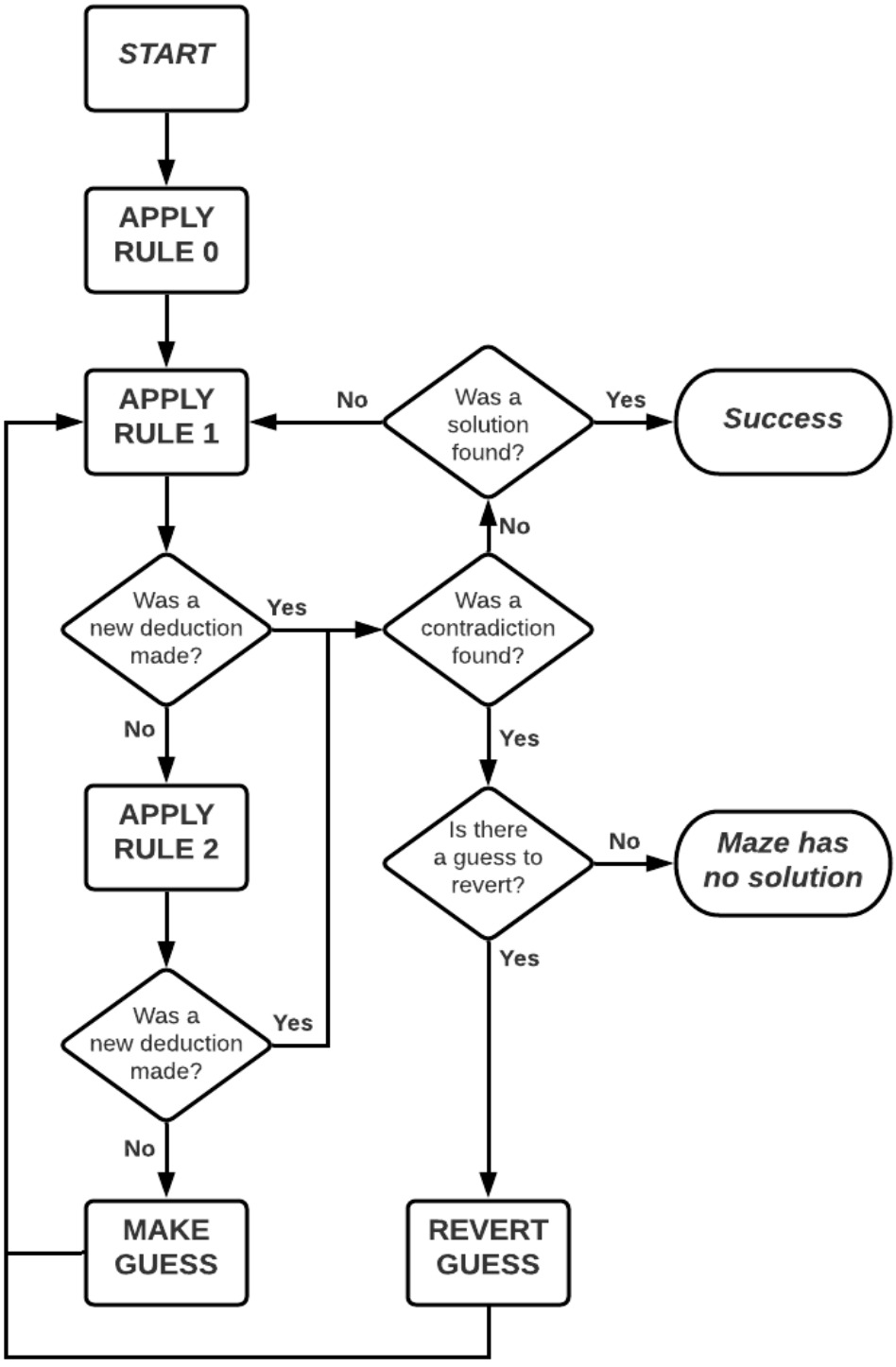


Figure 3-3: The structure of the human-inspired solver.

3.5.1 Rules

Rule 0: The solution includes all checkpoints and endpoints and excludes all breaks.
IF an unknown edge contains a checkpoint, THEN that edge is included.
IF an unknown vertex contains a checkpoint, THEN that vertex is included.
IF an unknown vertex is one of the maze's endpoints, THEN that vertex is included.
IF an unknown edge contains a break, THEN that edge is excluded.

Rule 0 is applied just once, at the very start of the process, since unlike the other rules it uses only the visible clues given in the puzzle and not any prior deductions.

Rule 1: An included vertex has exactly 2 included edges.
IF a vertex has 2 included edges, THEN all other edges connected to that vertex are excluded.
IF a vertex is included and has only 2 edges that are not excluded, THEN those edges are included.
IF a vertex has fewer than 2 edges that are not excluded, THEN that vertex is excluded.

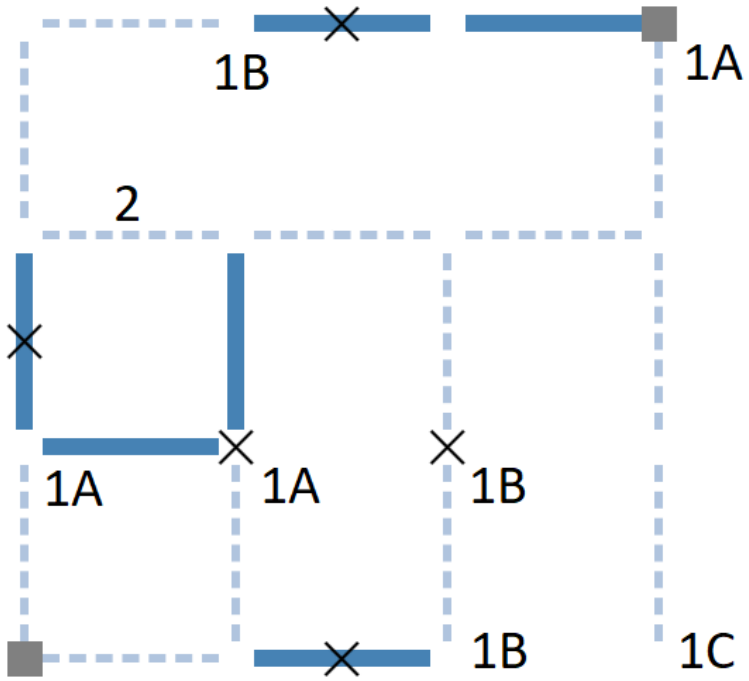


Figure 3-4: Rule examples

This figure shows a hypothetical arrangement of included, excluded and unknown edges and vertices at some point while solving a puzzle.

At the vertices marked 1A there are 2 included edges, so the unknown edge at each vertex could be set to excluded. Note the extra included edge at endpoints for the top right vertex.

The vertices marked 1B are all included, either because of one of their edges being included or because of a checkpoint. In each case there are only 2 non-excluded edges, so an unknown edge at any of these vertices could be set to included.

The vertex marked 1C can be set to excluded since it only has 1 edge that is not excluded.

The edge marked 2 would form a loop of included edges if it were included, so it can be excluded.

Rule 1 considers the number of included and excluded edges at each vertex. This is calculated differently at the endpoints of the maze, which should only have 1 included edge. This special case is handled by always considering endpoints to have 1 more included edge than they actually do. This can be imagined as additional “dangling” edges connected only at one end, one edge to each endpoint, which are always included; the task of the maze is then to connect these dangling edges to each other.

Rule 2: There can be no loop of included edges.
IF there are two vertices which are the endpoints a path of included edges and are also connected by a single unknown edge, THEN that unknown edge is excluded.

3.5.2 Automatic Rules

Whenever an edge is set to included, the vertices at each end are also set to included (if they were not already). Whenever a vertex is set to excluded, all its edges are also set to excluded (if they were not already). This happens automatically and is not considered a separate deduction.

3.5.3 Checking for Contradictions and Solutions

After any deduction is made, the current state of the grid is checked to see if it has any contradiction or solution.

A contradiction is when the current arrangement of included and excluded edges and vertices are incompatible with any solution, whether any unknown edges or vertices are chosen to be included or excluded. A contradiction is detected whenever:

- Any vertex has more than 2 included edges
- A vertex is excluded but has at least 1 included edge
- A vertex is included but has fewer than 2 included or unknown edges
- Any set of included edges forms a loop
- There is an isolated end (that is, some part of the path is cut off from the rest)
- There is a path of included edges connecting the endpoints of the puzzle, but a solution has not been found

These conditions were chosen because they are very easy for a human to identify, according to casual conversations about the solving process.

When checking for a solution, there is no need for every edge not on the path to be specifically excluded; unknown edges are considered excluded for this purpose.

3.5.4 Guessing

Any set of rules which only make deductions about edges or vertices which are certainly included or certainly excluded will inevitably fail to find a solution to a maze with multiple solutions, since there will always be some edges and vertices that are included by some solutions but excluded by others. Because of this, the expert system was given the ability to make a guess about the solution and continue deduction from there.

If none of the system's rules lead to a new deduction, the system chooses an unknown edge and guesses that it is included. This edge could be chosen in one of two ways:

1. Every unknown edge has an equal probability of being chosen.
2. An end is chosen at random, with each end having an equal probability of being chosen. The edge is then chosen at random from the unknown edges connecting to that end, with each edge having an equal probability of being chosen.

The second method was chosen for the analysis, since extending a pre-existing partial solution is a more human-like approach to taking a guess than picking any edge completely at random.

After setting the chosen edge to included, the system continues to apply the rules in the same way as before until it either is forced to make another guess or it finds a contradiction. If a contradiction is found, the most recent guess and all deductions leading from it are reverted. This means that every edge and vertex that was deduced to be included or excluded since the most recent guess is set back to being unknown. Finally, since letting the originally guessed edge be included led to a contradiction, that edge is deduced to be excluded.

The system keeps track of which deductions were made from each guess by marking each edge and vertex with a "priority" value equal to the number of guesses that were made before a deduction was made about that edge or vertex. An edge or vertex with priority 0 was deduced to be included or excluded with certainty before any guesses were made. If the n^{th} guess leads to a contradiction and is reverted, the originally guessed edge is marked as excluded with priority $n - 1$.

3.5.5 Type C Predictors

Type C predictors describe the process of reaching a solution, rather than the solution itself. Because the guessing method this solver uses is random, it can produce very different results on different attempts to solve the same maze. Because of this, the following predictors were estimated by running the solver 100 times and taking the mean of the measured values:

C_1: The number of times Rule 1 was applied

C_2: The number of times Rule 2 was applied

C_3: The number of guesses made

C_4: The number of guesses reverted

C_5: The maximum number of unreverted guesses at any point during the process

C_6: The maximum number of deductions between making a guess and discovering that the guess leads to a contradiction (if no guess led to a contradiction, this is 0)

3.6 Analysis

The first step was to ensure that a reasonable measure of ground truth for puzzle difficulty could be established from the collected data. This process involved comparing difficulty ratings given to puzzles with the time taken and the number of clicks made in the process of solving the puzzle to find which was the most appropriate indicator of difficulty. The possibility of a learning effect, by which participants would improve as they solved more puzzles, was also considered. Finally, a way of condensing the 100 or so data points for each puzzle into a single number – the "true" difficulty of the puzzle – had to be determined. Investigations indicated that the simplest approach was the best: the difficulty of a puzzle was taken to be the mean difficulty rating assigned across all participants. This process is explained in more detail in Section 4.2.

The next part of the analysis was to investigate which of the collected predictors was the most effective. Firstly, the predictors were tested individually using Pearson's correlation test (Pearson, 1895) to determine whether each was correlated with the difficulty of the puzzle, and for those that were correlated to compare how strongly they were correlated. After this, combinations of multiple predictors were tested with multiple linear regressions, with these combinations compared with each other as well as the results of the single predictors. Type B and type C predictors were harder to compute than type A predictors and were computed using different methods, so the main point of comparison was whether type B predictors or type C predictors performed better. Because of this, type B and type C predictors were not combined with each other in this analysis, but each was also combined with type A predictors (since those were trivial to compute).

Two main potential problems were considered for the analysis of multiple predictors:

1. Many of the predictors were expected to not only be correlated with difficulty but also to be correlated with each other. For example, most of the puzzles had a similar density of checkpoints, meaning that the number of checkpoints (predictor A_6) should be correlated with the puzzle's size (predictor A_1). To account for this, in a test where size and number of checkpoints were both considered would use a corrected predictor: in this case, the number of checkpoints would be divided by the total number of edges and vertices in the maze to get a new predictor, checkpoint density, that is not related to size.
2. Adding more predictors to a model will always increase the R^2 value that results, because using more degrees of freedom allows a closer fit. To see when this happens, R^2 values were adjusted to account for the number of variables (UCLA, 2021). A predictor has only meaningfully improved the model if the adjusted value R^2 value increases.

Once the most effective set of predictors to use had been determined, the model was tested using cross-validation, by which part of the dataset is used to compute appropriate weights for each predictor and the rest of the dataset is used to test the resulting formula. The average difference between predicted difficulty and actual difficulty was computed; this value was compared to the values obtained by van Kreveld et al. (2015) on other types of puzzles.

Finally, in case participants' different subjective difficulty scales affected these results in any way, the model was also tested using data corrected for this using the z-score of each difficulty rating for the player that gave it instead of the raw difficulty rating. The z-score z of a raw score x is calculated as $z = (x - \mu)/\sigma$, where μ is the mean of the participant's ratings and σ is the standard deviation (Kreyszig, 1979: p. 880).

4 Results

138 people began a session on the website and did not opt out of the process. 124 of these people completed at least 1 puzzle, and 93 completed all 30 puzzles. It is not possible to know how many people began the process and opted out since no data was stored on them. This is slightly higher than the number of volunteers in previous studies in this area e.g. van Kreveld et al. (2015). Between them, the participants completed a total of 3,043 puzzles. 2,790 puzzles were completed by those who completed all 30 puzzles, and 253 were completed by those who did not.

The best definition of the ground truth of difficulty for each puzzle was established. Next, each potential predictor was analysed separately, followed by sets of predictors together. The best model found was then tested. Finally, scaling difficulty values for each participant was considered.

4.1 Participant Demographics

A summary of participants' responses to the questionnaire is given in Figures 4-1, 4-2 and 4-3. Raw data can be found in Appendix D.1. Participants who completed no puzzles were excluded.

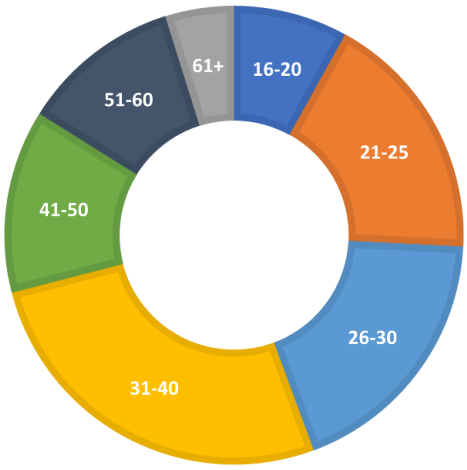


Figure 4-1: Participants' responses to the question "How old are you?"

Figure 4-2: Participants' responses to the question "How often do you solve puzzles?"

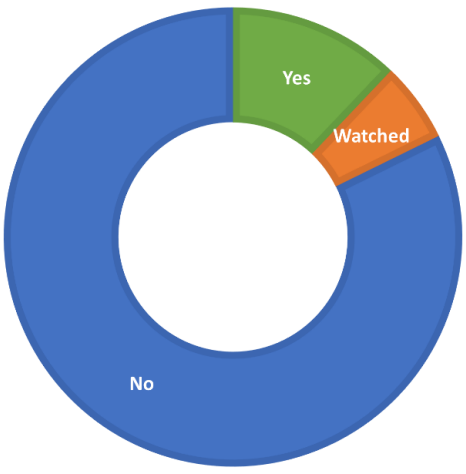


Figure 4-3: Participants' responses to the question "Have you played The Witness (2016)?"

The label "Watched" corresponds to the answer "No, but I've watched someone else play it."

4.2 Exploratory Analysis

This section explains several explorations made on the data before analysing any predictors, with a particular focus on choosing an appropriate measure for the ground truth difficulty of each puzzle.

4.2.1 Deciding between Difficulty Rating, Time and Clicks

The difficulty of a puzzle is inherently subjective. Several objective values that could correspond to difficulty were collected, but they are not without their own problems.

The time taken for a participant to solve a puzzle was in each case collected. This is problematic as a metric due to a high quantity of outliers. One user spent just under 36 hours on a puzzle (`bbeya`) before deciding to skip it, and almost 20 hours on another (`4674c`) which they assigned a difficulty of 4. It is unlikely that they spent the whole time attempting to solve the puzzles in question. While this is the most extreme example, many of the time values recorded were unreasonably high.

The reasons for this were most likely the casual nature of the data collection interface and the fact that there was no visible timer or other specific indication that participants were being timed. As a result, time measurements cannot be considered reliable in many cases, although (as Figure 4-4 shows) it was still correlated with the user-assigned difficulty.

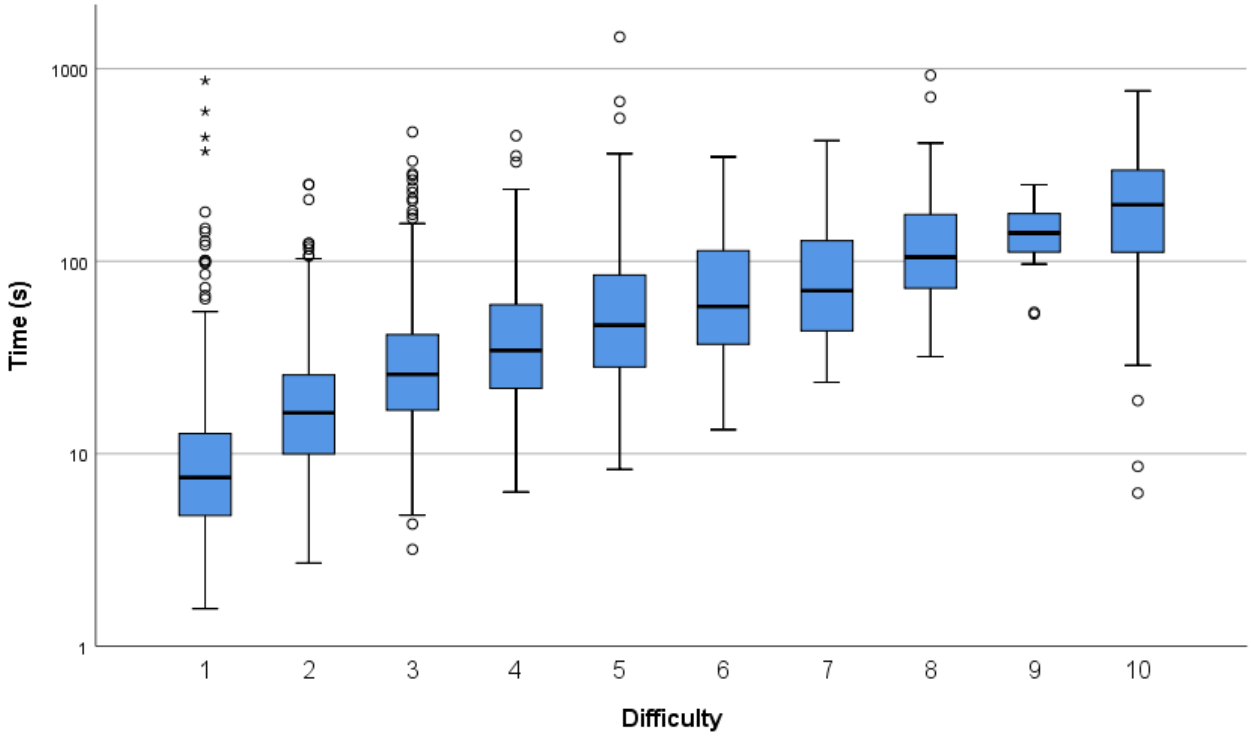


Figure 4-4: Time taken to solve a puzzle (log scale) for each difficulty rating. 13 outliers do not fit on the axes used.

The number of times a participant clicked any edge before solving the puzzle was also recorded. In this case, the problem is that different users had very different habits. Some were very deliberate and clicked no more edges than necessary to create the solution for most puzzles, while others clicked much more frequently. One user recorded 2396 clicks over the 40 minutes 39 seconds it took them to solve all 30 puzzles – just under one click every second!

Because of these marked differences in behaviour between users which are difficult to correct for, it is also not ideal to use the number of clicks for assessing how difficult a user found a puzzle.

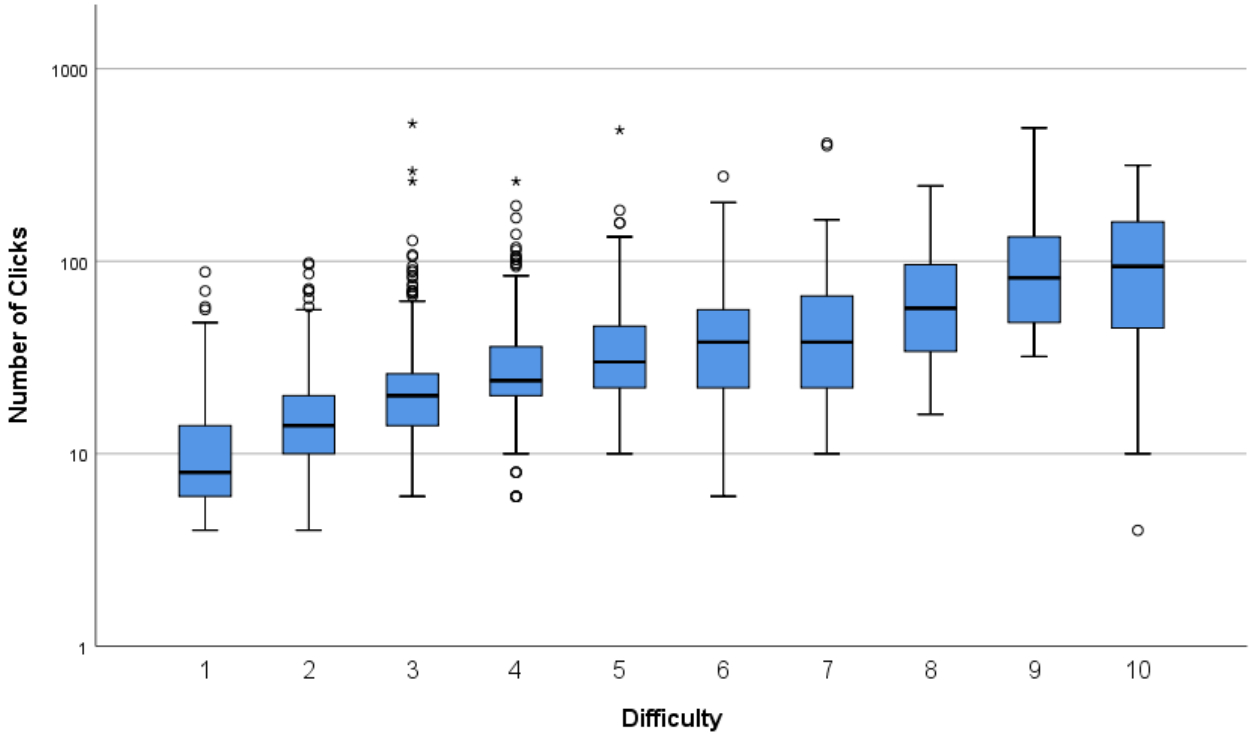


Figure 4-5: Number of clicks taken to solve a puzzle (log scale) for each difficulty rating.

The charts above needed to be plotted with logarithmic scales because time taken and number of clicks seem to grow roughly exponentially with difficulty. These results indicate that difficulty rating assigned, time taken and number of clicks are all correlated with each other, so it is reasonable to use any of them as a basis for a metric for a ground truth of difficulty. However, there are reasons that time taken and number of clicks can be considered less reliable between participants than difficult rating assigned. To be a useful metric, either would likely have to be pruned of extreme outliers and use a geometric mean rather than an arithmetic mean (due to the roughly exponential relationship). For this reason, the user's difficulty rating was used for the rest of this analysis.

4.2.2 Learning Effect

Since participants were asked to attempt 30 puzzles of the same type in succession, it was hypothesised that they would improve at them over time. If so, this may have led to the same puzzle receiving a lower difficulty rating from a user if it is seen near the end of the process than it would have done if it was seen near the beginning, when the participant has had less time to learn.

A learning effect was checked for by looking for a correlation across all puzzle results between the position at which the puzzle appeared for a participant (first puzzle to appear, second puzzle, etc.) and the difficulty rating assigned by that participant. Since the learning effect should be monotonic, but not necessarily linear – users should not ever get worse at the puzzles as they solve more puzzles but the rate at which they learn may increase or decrease as they solve more puzzles – Spearman's rank correlation test was used (Spearman, 1904).

Spearman's rank correlation coefficient was calculated as $r_s = -0.024$ with $p = 0.190$. Since $p > 0.05$, the data does not indicate a significant correlation between the position at which a puzzle appeared and the difficulty rating assigned. Figure 4-6 shows the results.

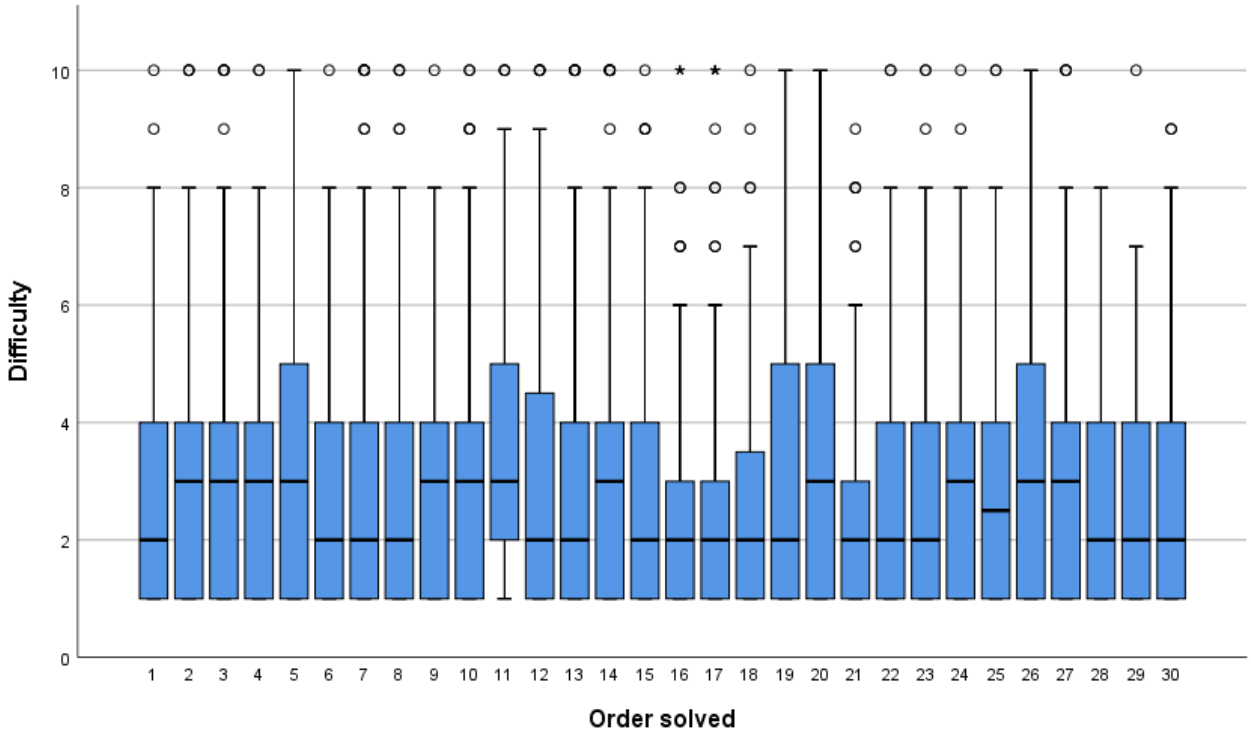


Figure 4-6: The distributions of difficulty rating assigned by users for their first puzzle, second puzzle and so forth. There is no significant correlation between the position at which a puzzle appeared and the difficulty rating assigned.

A correlation between position at which a puzzle occurred and the natural logarithm of the time taken to solve the puzzle was also tested for. The logarithm of time taken was taken because time taken seems to be exponentially related to difficulty as explained above. In this case, Spearman's rank correlation coefficient was $r_s = -0.102$ with $p < 0.001$. This indicates a weak but significant negative correlation, so participants did become quicker at solving the puzzles as they progressed. A linear regression indicated that participants sped up, on average, by a factor of 1.48 between their first puzzle and their last, with $R^2 = 0.010$. The very low value of R^2 indicates that there are other, much more important factors that determine the time taken to solve a puzzle. This is to be expected: a larger and more difficult puzzle seen at the end of the process will take longer to solve than a small, simple puzzle seen at the beginning. These results are shown in Figure 4-7.

The results of these tests indicate that while participants did improve at solving the puzzles, the difficulty ratings they assigned did not significantly change. This may be because participants did not improve enough over the course of 30 puzzles for the difference to show up on a relatively coarse 1-10 scale. Alternatively, participants may have accounted, consciously or subconsciously, for their own improvement as they progressed when assigning difficulty ratings. In either case, there is no need to correct for any learning effect in difficulty scores or to discard any of the data.

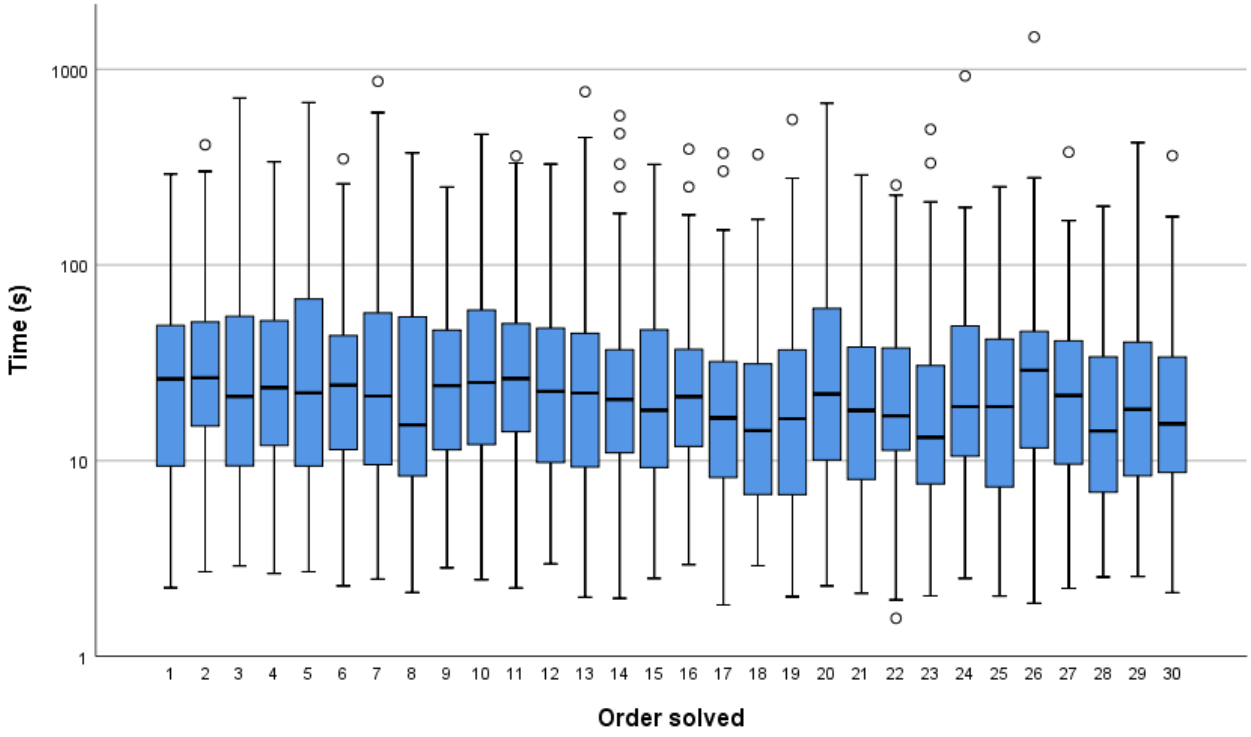


Figure 4-7: The distributions of time taken by users to solve their first puzzle, second puzzle and so forth. There is a weak negative correlation between the position at which a puzzle appeared and the time taken to solve it, indicating a slight improvement by users as they solve more puzzles.

4.2.3 Data Summary

Figure 4-8 shows the distributions of difficulty rating assigned to each puzzle. They are sorted in ascending order of the mean difficulty rating across all participants.

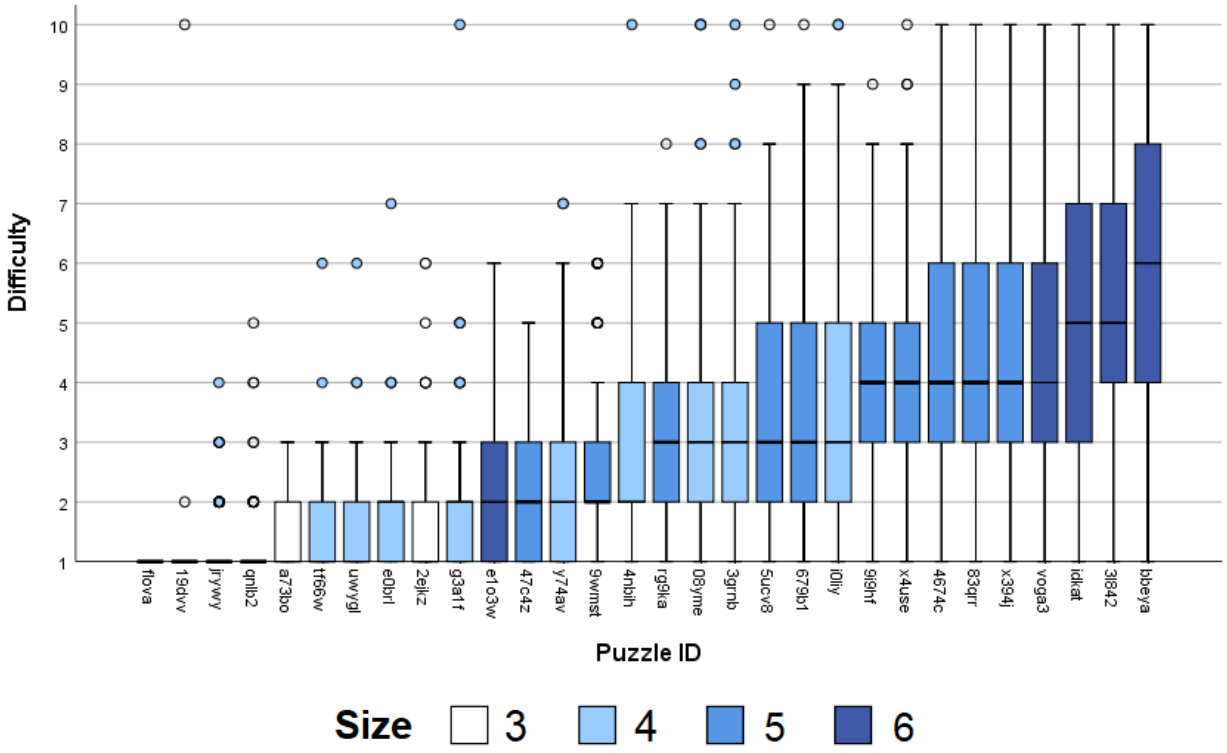


Figure 4-8: Distributions of difficulty rating assigned to each puzzle, ordered by ascending mean. Colour indicates the size of the puzzle.

This figure demonstrates several important points:

1. While there is plenty of overlap between the difficulty ratings assigned to different puzzles, difficulty ratings assigned by different participants are generally very consistent with each other, given that the values of the 1st and 3rd quartiles increase monotonically with the mean.
2. The distributions of difficulty ratings for each puzzle usually have a positive skew. This is likely because most of the puzzles used were relatively easy, and difficulty ratings cannot go below 1. For a particularly hard puzzle, a negative skew would be expected.
3. Taking the median difficulty rating as the “true” difficulty of each puzzle would not be appropriate as there are not enough distinct values to distinguish enough between puzzles; 90% of the puzzles have a median difficulty rating of 1, 2, 3 or 4.
4. The size of a puzzle (indicated by the colour of its boxplot) is an important factor in determining puzzle difficulty, shown by the fact that the boxes generally get darker towards the right. However, it is far from the only contributing factor.

Since the median is not an appropriate way of condensing user difficulty ratings into a single value for ground truth, the mean was used instead.

4.3 Single Predictors

Now that a suitable value for the true difficulty of each puzzle has been established, the correlation between various predictors computed for each puzzle and this difficulty value was explored. Table 4-1 shows the results. The raw data for the most important predictors can be found in Appendix D.2.

Most type A predictors were strongly correlated with difficulty. This simply reaffirms that size is a good predictor of difficulty, since the type A predictors used here were also strongly correlated with each other, with each predictor measuring the size of the puzzle in a slightly different way.

For the type B predictors, minimum solution length was by far the most successful predictor. This is reasonable to expect, as a longer solution means more for the player to hold in their mind to solve the puzzle. However, it was not expected that this predictor would be so strongly correlated with difficulty. Other type B predictors were less successful than type A predictors.

Scatter plots of the two best predictors (B_2 and A_6) are shown in Figures 4-9 and 4-10.

Type A	**Predictor Description**	**R^2**	**p**
A_1	Size	0.622	<0.001
A_2	Vertices	0.616	<0.001
A_3	Edges	0.644	<0.001
A_4	Checkpoints on vertices	0.177	0.021
A_5	Checkpoints on edges	0.583	<0.001
A_6	Checkpoints (total)	0.649	<0.001
Type B	**Predictor Description**	**R^2**	**p**
B_1	Distinct solutions	0.012	0.567
B_2	Minimum solution length	0.870	<0.001
B_3	Minimum maximum checkpoint separation	0.390	<0.001
Type C	**Predictor Description**	**R^2**	**p**
C_1	Mean rule 1 deductions	0.544	<0.001
C_2	Mean rule 2 deductions	0.383	<0.001
C_3	Mean guesses made	0.124	0.056
C_4	Mean guesses reverted	0.216	0.010
C_5	Mean maximum guess depth	0.040	0.289
C_6	Mean maximum deductions between guess and its reversion	0.168	0.025
C_7	Mean total deductions	0.482	<0.001
C_8	Proportion of guesses made that were reverted	0.380	<0.001

Table 4-1: Results of correlations for single predictors. Rows highlighted in red are insignificant results at the 5% level. Rows highlighted in yellow would in isolation be considered significant but due to the multiple hypothesis tests carried out, it is possible that one or more of these might have arisen by chance.

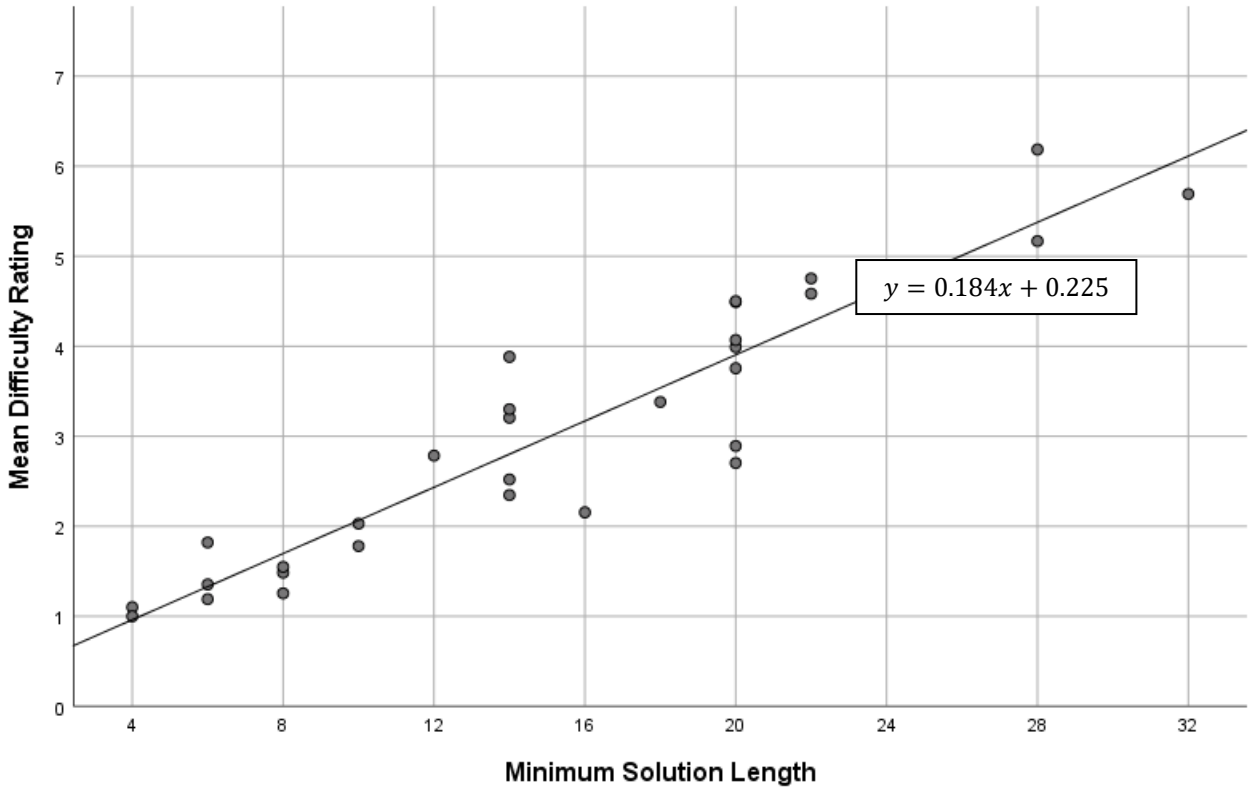


Figure 4-9: Mean difficulty rating against minimum solution length. Minimum solution length was the most strongly correlated single predictor of difficulty ($R^2 = 0.870$).

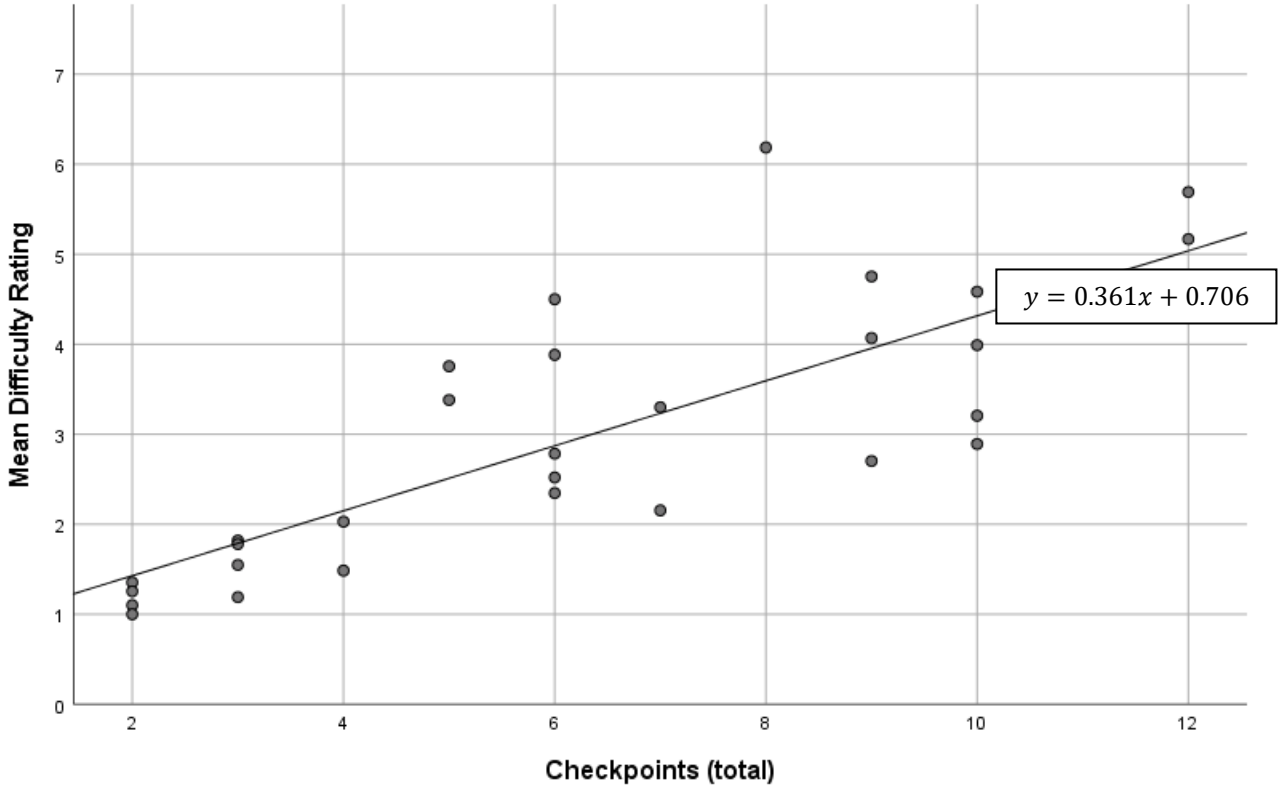


Figure 4-10: Mean difficulty rating against total checkpoints. Total checkpoints was the most strongly correlated type A predictor of difficulty ($R^2 = 0.649$).

Type C predictors were generally less successful than type A predictors. This is probably because size is such a powerful indicator of difficulty, as it is strongly linked with minimum solution length. Predictor C_8, which superficially should be unrelated to size, was still moderately correlated with difficulty, indicating that there may still be some merit in using type C predictors in combination with type A or B predictors.

4.4 Multiple Predictors

The results of the single predictor analysis already indicate a very accurate formula for estimating difficulty, but minimum solution length cannot be the only factor determining difficulty – after all, puzzles with the same minimum solution length in the dataset sometimes had quite different mean difficulty ratings. Because of this, combinations of predictors were analysed to see if any could outperform it. In particular, combinations of type A and B predictors as well as combinations of type A and C predictors were analysed. This is because type A predictors are trivial to compute, so it is reasonable to use them in any model, but type B and C predictors are more difficult to compute, so it is less reasonable to combine them into a single model.

4.4.1 Corrected Predictors

Many of the predictors used were highly correlated with each other. For example, the number of checkpoints in a maze is closely linked with the size of the maze, since a larger maze requires more checkpoints to create a reasonably interesting puzzle. Because of this, certain corrected predictors were computed to try to account for these links.

A_7: *Vertex checkpoint ratio* is calculated as the number of vertex checkpoints divided by the number of vertices. That is, $A_7 = A_4 / A_2$.

A_8: *Edge checkpoint ratio* is calculated as the number of edge checkpoints divided by the number of edges. That is, $A_8 = A_5 / A_3$.

A_9: *Total checkpoint ratio* is calculated as the total number of checkpoints divided by the total number of edges and vertices. That is, $A_9 = A_6 / (A_2 + A_3)$.

B_4: *Corrected minimum solution length* is calculated as minimum solution length divided by the length of the shortest possible solution on any puzzle of that size. That is, $B_4 = B_2 / 2(A_1\text{-}1)$.

B_5: *Corrected minimum maximum checkpoint separation* is calculated as minimum maximum checkpoint separation divided by the length of the shortest possible solution on any puzzle of that size. That is, $B_5 = B_3 / 2(A_1\text{-}1)$.

4.4.2 Predictors Used

8 sets of predictors were tested and their adjusted R^2 values were compared. Adjusted R^2 is calculated as

$$(R^2)_{adj} = 1 - \frac{(1 - R^2)(N - 1)}{N - k - 1}$$

where N is the number of observations (here 30) and k is the number of predictors (here between 1 and 4). This is to account for the fact that adding larger numbers of predictors may result in fitting to the noise in the dataset rather than any real trend (UCLA, 2021).

The sets of predictors used and the results are shown in Table 4-2 and Figure 4-11.

Rank	Predictors	Adjusted R^2
1	A_3, B_2	0.872
2	B_2	0.865
3	A_9, B_2	0.864
4	A_3, B_4	0.783
5	A_3, B_5	0.693
6	A_3, C_6, C_8	0.690
7	A_1, A_3, A_9, $(A_9)^2$	0.683
8	A_3, A_7, A_8	0.621
9	C_7, C_8	0.505

Table 4-2: Results of multiple difficulty predictors. No set of predictors performed substantially better than maximum solution length (B_2) alone.

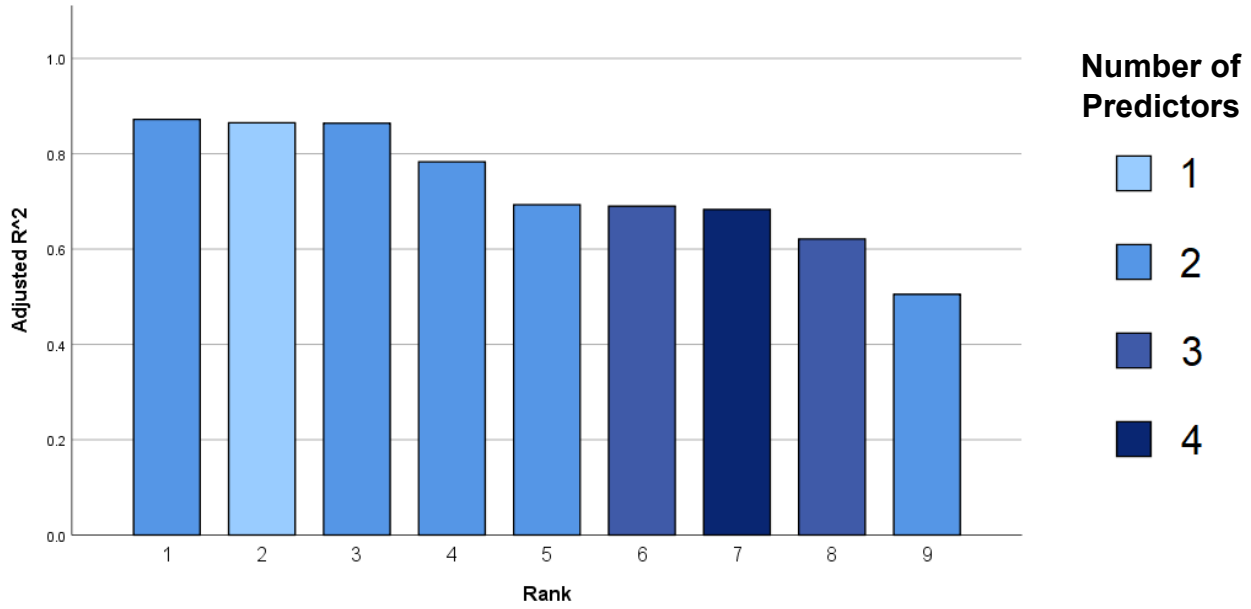


Figure 4-11: Results of multiple difficulty predictors. The colour of the bar indicates the number of predictors used.

Every one of the most successful combinations of predictors involved minimum solution length in some way, and none substantially outperformed minimum solution length alone. Also, using multiple predictors makes a model more vulnerable to anomalies in the data, as well as causing potential problems with correlated predictors (Tabachnick & Fidell, 2007: p. 5). Because of this, it is best to use minimum solution length as the sole predictor of difficulty instead of any combination of predictors. Other than that, combinations of type A predictors with other type B predictors or type C predictors were more successful than combinations of several predictors of the same type. Most of these performed better than single type A or C predictors.

4.5 Cross-Validation

To test how well the model performs in practice, 5-fold cross-validation was carried out. For this, the mazes were split into 5 sets of 6. Each set contained one maze of size 3, two of size 4, two of size 5 and one of size 6. Other than that restriction, mazes were assigned to sets entirely at random.

For each of the 5 sets, a model for difficulty was produced from the results of the other 4 sets (*training data*), then tested on the original group (*testing data*). The main model ("Minimum solution length") was produced as before, with a linear regression using minimum solution length as the sole predictor. For comparison, the same process was applied to two much simpler models. The first ("No predictor") assumes that all puzzles have the same difficulty and so predicts each puzzle in the testing data to have difficulty equal to the mean of the difficulties of all puzzles in the training data. The second ("Size") assumes instead that all puzzles of the same size have the same difficulty and so predicts each puzzle in the testing data to have difficulty equal to the mean of the difficulties of all puzzles of the same size in the training data.

The difference between the predicted difficulty and actual difficulty for each puzzle is called the *error* for that puzzle of the model. The distribution of errors of each model is shown in Figure 4-12. The model utilising minimum solution length was clearly the most successful, as it has maximum, first quartile, third quartile and minimum all closer to 0 than either of the other two models. The model utilising no predictor was the least successful.

The *mean absolute error* of each model is given by the mean across all puzzles of the absolute value of the error of the model. The mean absolute error of each model is shown in Figure 4-13. This again shows that the model using minimum solution length was the most successful, with a mean absolute error of 0.51.

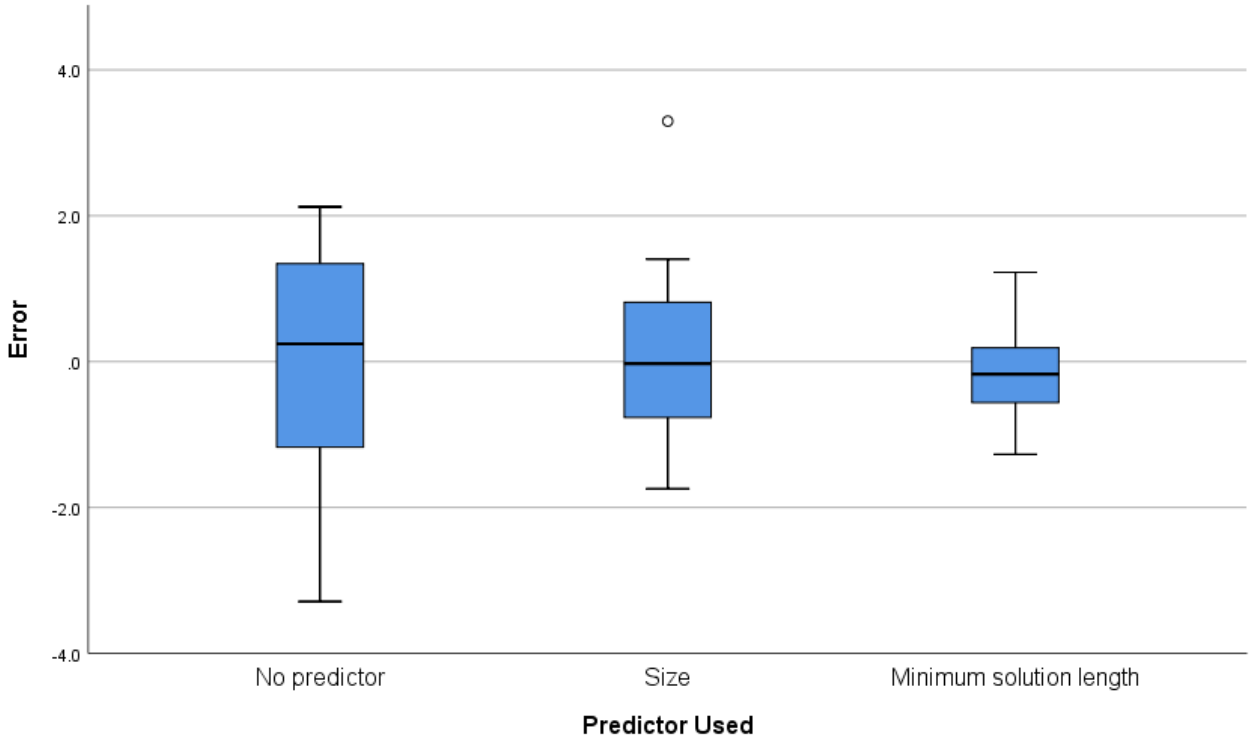


Figure 4-12: The distributions of error of each model. Unsurprisingly, using minimum solution length was the most successful approach and using no predictor was the least successful. The outlier in the plot for size is the puzzle with ID `e3o1w`, which is a particularly easy puzzle of size 6.

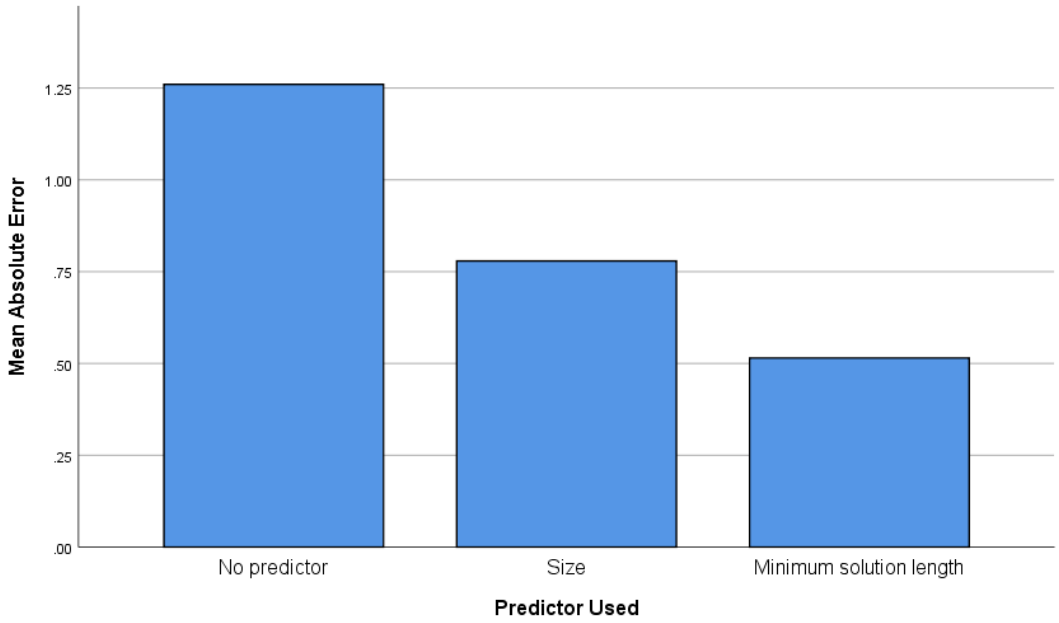


Figure 4-13: The mean absolute error of each model.

4.6 Adjusting for Player Skill

While participants roughly agreed on the order of the puzzles' difficulty, some of them consistently rated puzzles easier than others. This may be because a more skilled or experienced participant will find the same puzzle easier than one with less aptitude or experience, and as a result will assign it a lower difficulty rating.

To correct for this difference, the z-score of each difficulty rating for the participant can be used instead of the raw difficulty rating. This is computed as:

$$z = \frac{x - \mu}{\sigma}$$

where x is the raw difficulty rating, μ is the mean of the participant's raw difficulty ratings and σ is the standard deviation (Kreyszig, 1979: p. 880). Because of this calculation, it is necessary for this analysis to only consider the 93 participants who completed all 30 puzzles, as the mean and standard deviation for participants who solved different sets of puzzles are not comparable. The distribution of z-scores will have a mean of 0 and a standard deviation of 1.

The distribution of scaled difficulty ratings for each puzzle is shown in Figure 4-14. The puzzles are arranged in ascending order of (unscaled) mean difficulty. This figure indicates that scaling the difficulty ratings in this way does not much change the order of difficulty. There is still plenty of overlap between the distributions, indicating that disagreements on how difficult a puzzle are caused by more than differing subjective rating scales between participants.

Testing for a correlation between mean scaled difficulty and minimum solution length gave very similar results to using the mean unscaled difficulty ratings. R^2 was 0.872, compared to 0.870 for the unscaled difficulty ratings. Figure 4-15 shows the relationship.

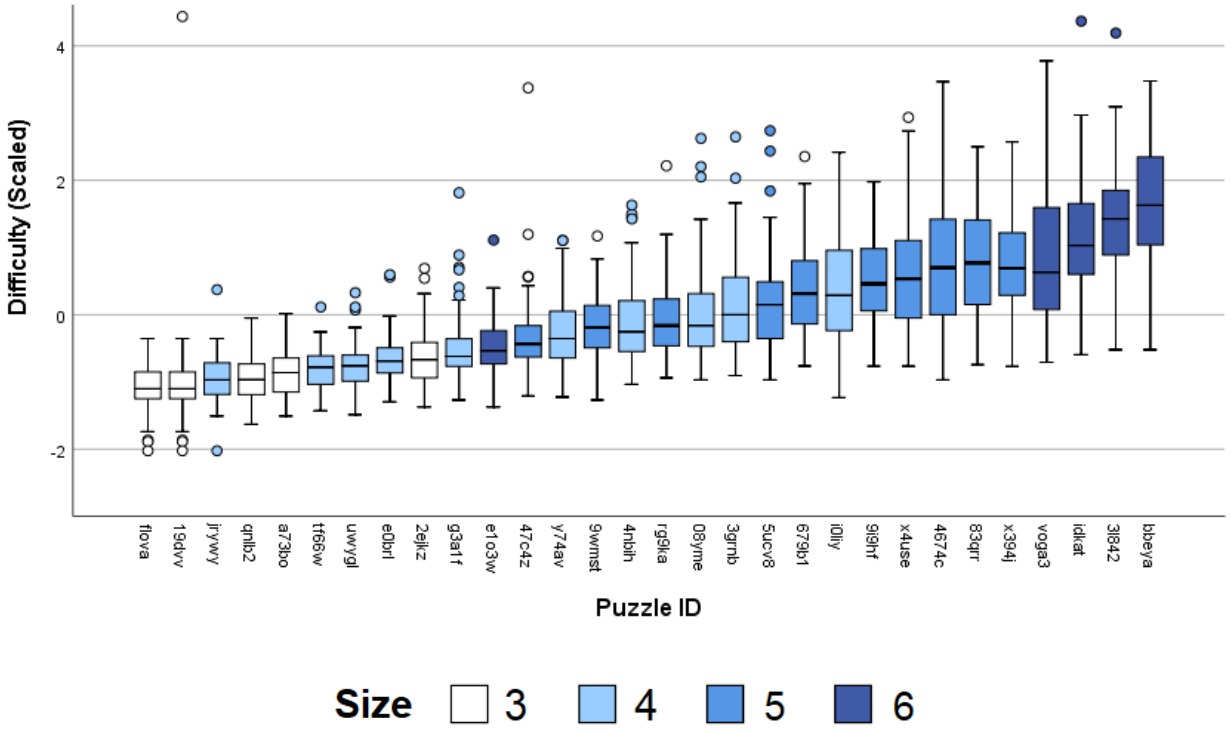


Figure 4-14: The distributions of scaled difficulty rating across all participants by puzzle.

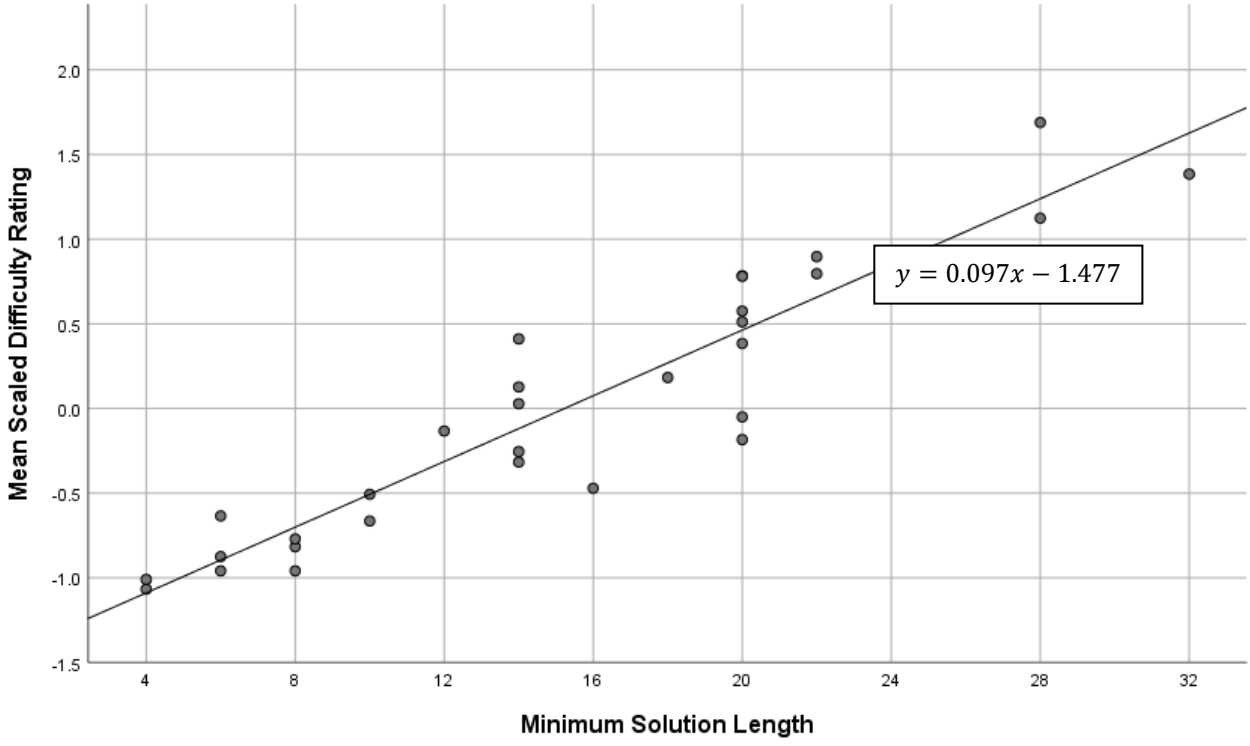


Figure 4-15: The mean scaled difficulty rating against minimum solution length. Minimum solution length was still the most strongly correlated single predictor of difficulty ($R^2 = 0.872$).

Performing cross-validation also led to very similar results, as shown in Figures 4-16 and 4-17. The mean absolute error in scaled difficulty was 0.22 using minimum solution length, compared to 0.65 using no predictor and 0.40 using size alone: this is a reduction of 66% in mean absolute error compared to using no predictor. This value was 60% when using unscaled difficulty values.

To determine whether this difference in error reduction was significant, the absolute error for each puzzle using the minimum solution length model was compared between the scaled difficulty data and the unscaled difficulty data. Each of the sets was scaled according to the mean absolute error using no predictor on that dataset. Since the absolute value of error was taken, we cannot assume normality. Because of this, a Wilcoxon matched pairs test was performed (Wilcoxon, 1945). The result of the test was Z = -0.586; p = 0.558. This shows that the increase in error reduction when using the scaled difficulty data compared to the unscaled difficulty data was not significant.

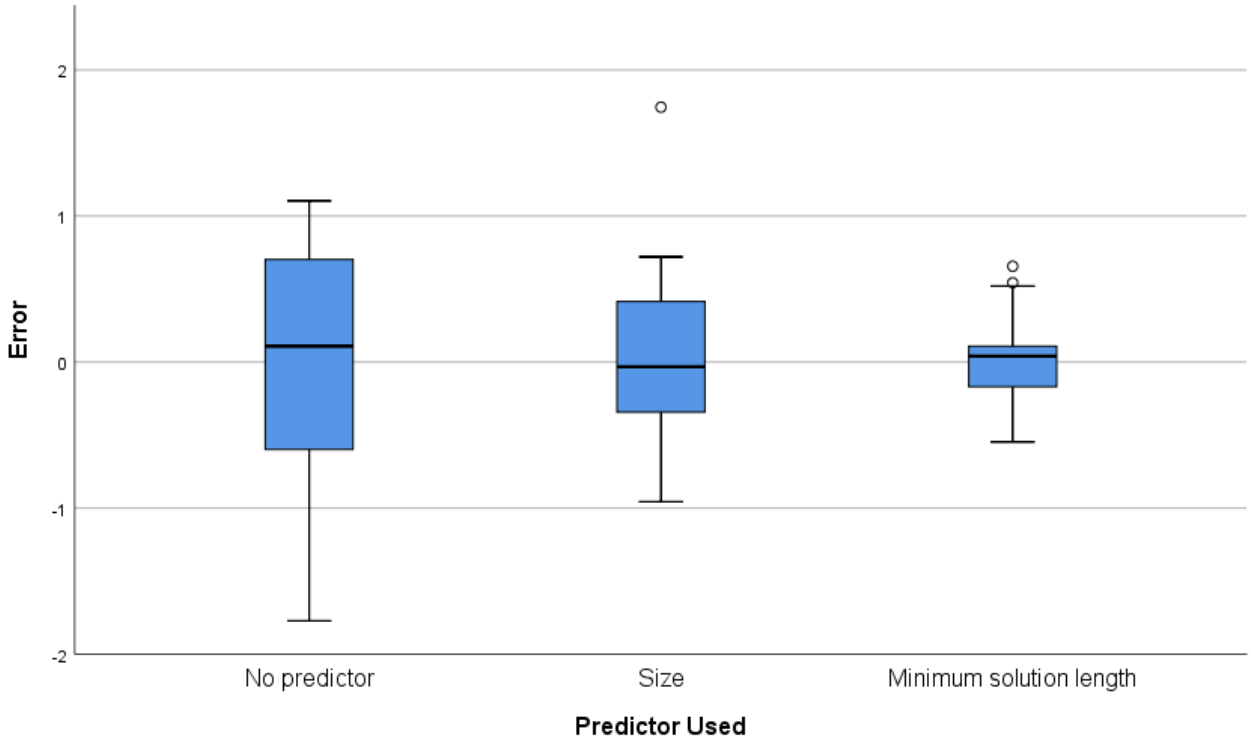


Figure 4-16: The distributions of error across all puzzles using cross validation on the scaled data, for each of the three models.

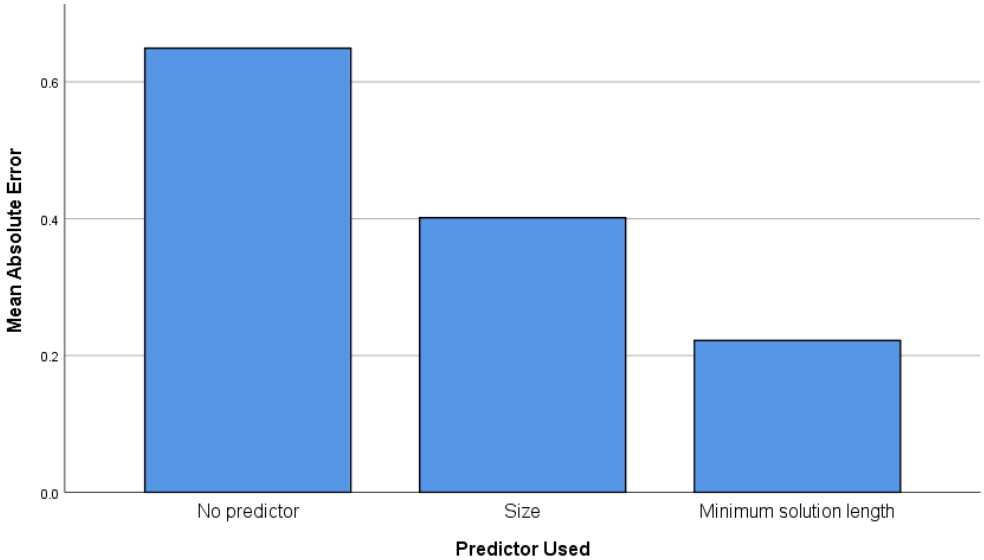


Figure 4-16: The mean of the absolute value of error using cross validation on the scaled difficulty data for each of the three models.

5 Discussion

The results of this project indicate that of all the difficulty predictors considered, minimum solution length is by far the best for this type of logic maze, performing better even than combinations of other predictors. The result for mean average error in predicted difficulty on a 1-10 scale was 0.51.

In comparison to these results, van Kreveld et al. (2015) used a similar approach (but with more predictors) for three types of puzzle. They found mean average errors of 0.40 for Flow, 0.92 for Move and 1.01 for Lazors, which is consistent with the results obtained by this project.

However, it should be noted that an approach not accounting for any feature of the puzzles whatsoever and simply predicting a difficulty value equal to the average difficulty of previously seen puzzles had a mean absolute error of 1.26. This is an extremely trivial approach, and yet it did not perform much worse than a much more complicated formula on another puzzle type, Lazors. It is probable that the Lazors puzzles used by van Kreveld et al. involved a wider spread of difficulties than here. However, this result should caution us about drawing conclusions from statistics about errors without comparing the proposed method with another method that should be unimpressive, such as the no-predictor method considered here.

Clearly from previous results, this method is not equally applicable to all puzzle types; it would be very important to test it on the puzzles in question before using it in any practical application.

5.1 Estimating Minimum Solution Length

Since minimum solution length was by far the best predictor of difficulty for logic mazes, finding a way to calculate minimum solution length efficiently is desirable. The brute force solver algorithm will always find the exact value of the minimum solution length but is not efficient. On the other hand, the human-inspired solver is much quicker to execute but it only finds one solution, and that solution will often not be of minimal length.

Jing et al. (2009) found that a similar system of an expert system pairing logical rules with a depth-first search was significantly more efficient at solving nonograms than the previous approaches of either a genetic algorithm or a pure depth-first search (analogous to the brute force solver used in this project). It is reasonable to suppose that such systems will be an efficient way of solving similar types of puzzles that are dependent on making many small logical and/or arithmetical deductions (such as Sudoku and their variants). By optimising the method and refining the way in which it makes guesses towards finding a solution of minimal length, this approach could be a highly

Table 5-1: Performance of solver 1 (brute force) and solver 2 (human-inspired) by size of puzzle, mean value over 10,000 applications of the solver to each maze in the dataset. Measurements were taken using:

- Windows 10 Home 19042.1110
- Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
- 16.0 GB Memory

Size	Solver 1 time (ms)	Solver 2 time (ms)
3	0.083	0.031
4	1.014	0.119
5	62.379	0.324
6	452.801	0.795

efficient way of estimating maximum solution length. This is especially true for larger puzzles, since the human-inspired solver scales better with puzzle size than the brute force solver.

As shown in Table 5-1, the human-inspired solver is significantly faster than the brute force solver and it scales better with the size of the puzzle. The human inspired solver took 26 times as long to solve a size 6 puzzle as a size 3 puzzle, whereas the brute force solver took 5500 times as long. For a size 5 puzzle, the brute force solver took 190 times longer to apply than the human-inspired solver. For a size 6 puzzle, it took 570 times longer. Hence if puzzles of size 5 or larger are used, it is much more feasible to apply the human-inspired solver many times than to apply the brute-force solver even once.

In a puzzle type with only one solution, minimum solution length would be easier for the human-inspired solver to find. In this case where many of the puzzles have multiple solutions, the probability of finding a solution of minimal length varies from puzzle to puzzle. Figure 5-1 shows the distribution of these probabilities.

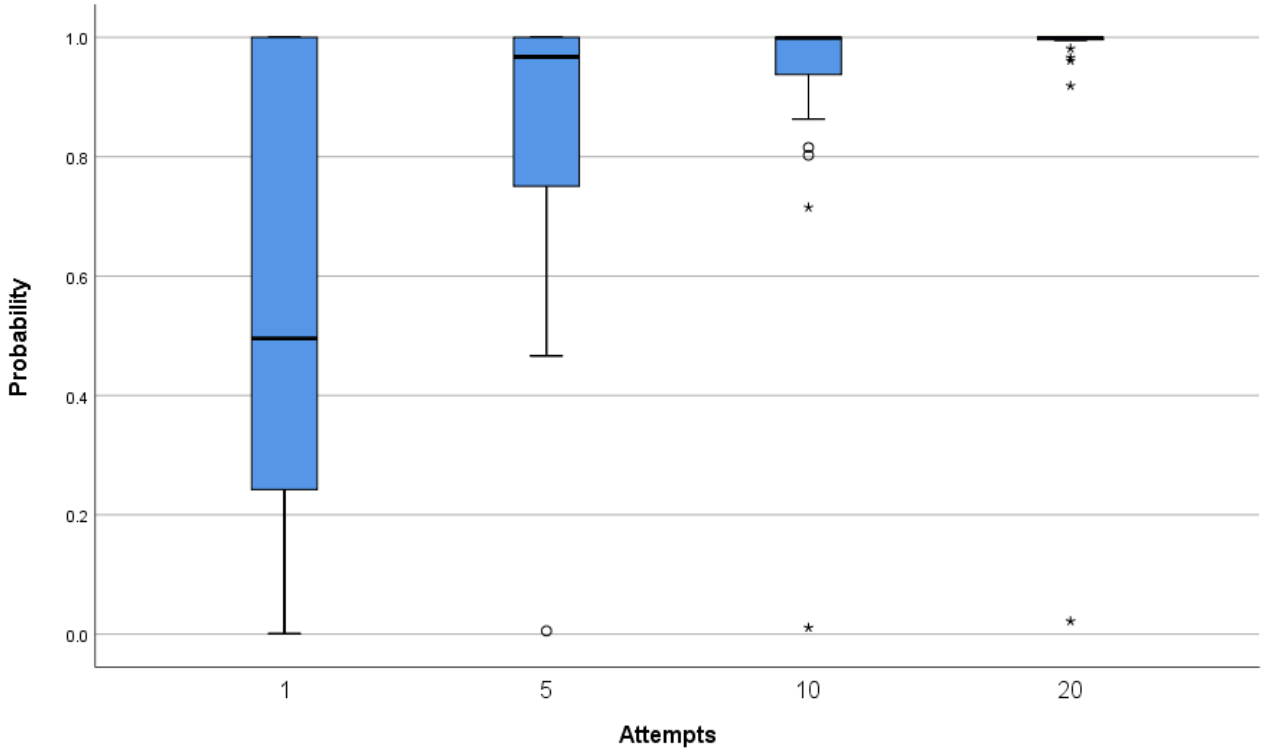


Figure 5-1: The probability of finding the real value of minimum solution length using the human-inspired solver within various numbers of attempts, summarised over all the puzzles in the dataset. The very low probability outlier is the maze with ID `e1o3w` (see Appendix A); this maze has a very high number of possible solutions (1576) which makes it unrepresentative of this type of puzzle; in a real application it is unlikely that puzzles with this many solutions would be used. Every other puzzle used had a >90% probability of finding a minimal solution within 20 attempts, and 25 of the 30 puzzles had >99%.

While neither approach is particularly optimised towards finding the minimum solution length and improvements could certainly be made to both algorithms now that minimum solution length is known to be the optimal predictor, it is unlikely that any optimisations would benefit the brute force solver enough to change this balance.

6 Conclusions

The main conclusion of this project is that minimum solution length, a type B predictor, is the most accurate predictor for difficulty in the type of logic maze used. It is inefficient to compute using a brute-force method but can be efficiently estimated using an expert system.

Jarušek & Pelánek (2011) suggested that one of the keys to computerising difficulty estimation is further study into how humans solve puzzles. In this case, however, type C predictors were less successful than type B predictors. It may be that the system employed here did not sufficiently simulate a typical human solving method to evaluate type C predictors accurately enough to outperform the measured type B predictors. This may have been exacerbated if the solving method employed by participants for these puzzles varied more than supposed. On the other hand, type C predictors combined with type A predictors outperformed type A predictors alone, which suggests that they may have captured some aspect of difficulty which could be used once dominant predictors such as puzzle size were accounted for.

The simplicity of the best performing difficulty formula may have been due to the type of puzzle investigated, a logic maze, which is simpler than other puzzle types that have been considered in previous research such as Sokoban. Van Kreveld et al. (2015) found that their method was most successful with Flow puzzles, which bear substantial similarity to the logic mazes used here, and less successful with Move puzzles which are much more like Sokoban. Taken together with the results of this research, this implies that there may be no single most effective method for puzzle difficulty evaluation and that it is important to take puzzle type (and in particular, puzzle complexity) into account.

The expert system was efficient enough that it would be feasible to execute it many times to assess a puzzle if necessary. However, it will always prove advantageous to optimise such a system so that it can be run more times, improving its reliability. In the case of a procedurally generated puzzle game, where a selection of generated puzzles must be assessed for difficulty so that one can be presented to the player, the solver may need to be executed thousands of times for every puzzle that the player sees.

Collecting data using a website required a large time investment to set up but was an effective way of accumulating a large dataset. Initial feedback showed that it was difficult to navigate on mobile devices, which may have put off some potential participants; a mobile-friendly site unfortunately could not be implemented within the time constraints of the project. The main disadvantage of the website is that the casual nature of participation meant that time taken to solve each puzzle was not as reliable an indicator of puzzle difficulty as it might have been in more controlled conditions.

6.1 Future Work

Minimum solution length cannot be the only contributing factor to difficulty, but it was such a dominant predictor that in a dataset of this size it could not feasibly be corrected for. To investigate secondary predictors that could be used in conjunction with minimum solution length, a new set of puzzles would have to be created in which minimum solution length is held constant. Since size was also a very successful predictor, it should likely also be kept constant. This would also help alleviate the problem that using too many predictors simultaneously can have where the model produced becomes over-fitted to the specific dataset used and does not generalise well; by keeping one predictor constant, the effects of varying another can be independently investigated. Indeed, when

designing research into any puzzle type that has been previously considered (as opposed to more exploratory research such as this project), it would be advantageous to determine which previous predictors have been most successful and to hold them constant.

It will always be helpful to extend previously attempted methods to new puzzles to assess their effectiveness. Doing this will allow a better understanding of what makes puzzles different from each other to be developed.

Design and implementation of the human-inspired solver was quite laborious as well as specific to a narrow category of puzzles. An interesting direction for future research would be to devise an artificial intelligence system that can discover logical rules like the ones used in this project by itself. It would be especially useful if it can be shown that the rules it discovers are similar to those used intuitively by human solvers.

More detailed study of human puzzle-solving methods is important to make significant progress in this area. This research should particularly concern how much the methods different people use to solve the same style of puzzle varies and whether these differences affect the solvers' perceptions of puzzle difficulty.

It would also be useful to investigate the differences in solving methods between different genres of puzzles. This would allow puzzles to be better classified by how they are solved, which may provide insight into which method of difficulty evaluation will be the most successful.

Finally, by studying how puzzle solving methods and perception of puzzle difficulty vary according to demographics such as age and experience at the genre, research could be conducted on how to vary the estimation of type C predictors based on the target demographic of the difficulty estimator. This would be a first step towards personalised procedural puzzle generation.

References

András, S., Sipos, K., & Sóos, A. (2013). *Which is harder? - Classification of Happy Cube puzzles.* Deutsches Institut für Internationale Pädagogische Forschung (DIPF).

Aponte, M.-V., Levieux, G., & Natkin, S. (2011). Measuring the level of difficulty in single player video games. *Entertainment Computing*. 2(4), 205–213.

Ashlock, D. & Schonfeld, J. (2010). Evolution for automatic assessment of the difficulty of Sokoban boards. *IEEE Congress on Evolutionary Computation*, 1–8.

Browne, C., & Maire, F. (2010). Evolutionary Game Design. *IEEE Transactions on Computational Intelligence and AI in Games*. 2(1), 1–16.

Chaplin, H. (2007). 'Is That Just Some Game? No, It's a Cultural Artifact.', *New York Times*, 12 March.

De Kegel, B. & Haahr, M. (2020). Procedural Puzzle Generation: A Survey. *IEEE Transactions on Games*. 12(1), 21–40.

Dong, Y. & Barnes, T. (2017). Evaluation of a Template-Based Puzzle Generator for an Educational Programming Game. *Proceedings, the Thirteenth AAAI Conference on Artifical Intelligence and Interactive Digital Entertainment*. 13(1), 172–178.

Gardner, M. (1962). Mathematical Games. *Scientific American*. 207(4), 134–135.

Guinness World Records (2008). *Guinness World Records: Video Gamer's Edition 2008.* Time Inc. Home Entertainment.

Haier, R. J., Karama, S., Leyba, L., & Jung, R. E. (2009). MRI assessment of cortical thickness and functional activity changes in adolescent girls following three months of practice on a visual-spatial task. *BMC research notes*. 2, 174.

Hendrikx, M., Meijer, S., Van der Velden, J., & Iosup, A. (2013). Procedural Content Generation for Games: A Survey. *ACM Transactions on Multimedia Computing, Communications and Applications*. 9(1), Article 1, 1–22.

Hicks, A. (2013). BOTS: Harnessing Player Data and Player Effort to Create and Evaluate Levels in a Serious Game. *Educational Data Mining 2013*.

Holmes, E. A., James, E. L., Coode-Bate, T., Deeprose, C. (2009). Can Playing the Computer Game "Tetris" Reduce the Build-Up Flashbacks for Trauma? A Proposal from Cognitive Science. *PLoS ONE*. 4(1), e4153.

Jarušek, P., & Pelánek, R. (2011). 'Difficulty rating of Sokoban puzzle', in Ågnotes, T. (ed.) *Proceedings of the Fifth Starting AI Researchers' Symposium*. Amsterdam: IOS Press, pp. 140–150.

Jing, M.-Q., Yu, C.-H., Lee, H.-L., & Chen, L.-H. (2009). Solving Japanese puzzles with logical rules and depth first search algorithm. *Proceedings of the Eighth International Conference on Machine Learning and Cybernetics*. 2962–2967.

Kreyszig, E. (1979). *Advanced Engineering Mathematics* (4th ed.). Hoboken, NJ: Wiley.

Mantere, T., & Koljonen, J. (2007). Solving, rating and generating Sudoku puzzles with GA. *IEEE Congress on Evolutionary Computation*. 1382–1389.

Millington, I. (2006). *Artificial Intelligence for Games*. San Francisco, CA: Morgan Kaufmann.

Murray, T. (2003). An overview of intelligent tutoring system authoring tools: Updated analysis on the state of the art. In: T. Murray, S. Blessing and S. Ainsworth eds. *Authoring Tools for Advanced Learning Environments.* Dordrecht, Kluwer Academic Publishers. pp. 491–544.

Nef, T., Chesham, A., Schütz, N., Botros, A. A., Vanbellingen, T., Burgunder, J.-M., Müllner, J., Müri, R. M., & Urwyler, P. (2020). Development and Evaluation of Maze-Like Puzzle Games to Assess Cognitive and Motor Function in Aging and Neurodegenerative Diseases. *Frontiers in Aging Neuroscience*. 12, Article 87.

Pearson, K. (1895). Notes on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*. 15, 240–242.

Pelánek, R. (2016). Applications of the Elo rating system in adaptive educational systems. *Computers & Education*. 98, 169–179.

Perrotta, C., Featherstone, G., Aston, H., & Houghton, E. (2013). Game-based Learning: Latest Evidence and Future Directions. *NFER Research Programme: Innovation in Education*. Slough: NFER.

Plank-Blasko, D. (2015). 'From Russia with Fun!': Tetris, Korobeiniki and the ludic Soviet. *The Soundtrack*. 8(1), 7–24.

Sengpiel, F. (2014). Plasticity of the Visual Cortex and Treatment of Amblyopia. *Current Biology*. 24(18), 936–940.

Shaver, M. (2017). What Makes Tetris so Incredibly Addictive? https://tetris.com/article/44/what-makes-tetris-so-incredibly-addictive (accessed 04 August 2021).

Sivanandam, S. & Deepa, S. (2008). *Introduction to Genetic Algorithms*. Berlin: Springer.

Spearman, C. (1904). The Proof and Measurement of Association between Two Things. *American Journal of Psychology*. 15, 72–101.

Tabachnick, B., & Fidell, L. (2007). *Using Multivariate Statistics* (5th ed.). Boston, MA: Pearson.

Tripathi, K. (2011). A Review on Knowledge-based Expert System: Concept and Architecture. *Artificial Intelligence Techniques – Novel Approaches and Practical Applications (IJCA special issue)*. 19–23.

UCLA (2021). Regression Analysis: SPSS Annotated Output https://stats.idre.ucla.edu/spss/output/regression-analysis/ (accessed 20 August 2021).

Van Kreveld, M., Loffler, M., & Mutser, P. (2015). Automated Puzzle Difficulty Estimation. *IEEE Conference on Computational Intelligence and Games 2015*. 415–422.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*. 1(6), 80–83.

Zirawaga, V. S., Olusanya, A. I., & Maduku, T. (2017). Gaming in Education: Using Games as a Support Tool to Teach History. *Journal of Education and Practice*. 8(15), 55–64.

Games

Blow, J. (2016). *The Witness*. Thekla, Inc.

Gygax, G. (1977). *Advanced Dungeons & Dragons*. TSR Inc.

Monolith Productions (2014). *Middle-earth: Shadow of Mordor*. Warner Bros. Interactive Entertainment.

Pajitnov, A. (1984). *Tetris*.

Persson, M. (2011). *Minecraft*. Mojang Studios.

Toy, M., Wichman, G. & Arnold, K. (1980). *Rogue*. Epyx, Inc.

Worth, D. (1978). *Beneath Apple Manor*. The Software Factory.

Acknowledgements

I would like to thank my supervisors, Craig Stark and William Kavanagh, for their invaluable advice and support throughout this project. I am also grateful to my friends and family for many helpful conversations about the work, especially Jonathan Jones and Ruth Dixon.

Finally, I would like to thank everyone who took the time to participate in this project by providing data, especially those who shared and retweeted links to the website and got it seen by many more people than I dared hope for.

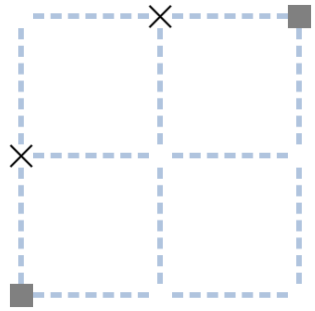
Appendix A: Puzzles

30 hand-made puzzles were used for this project. Details of each are given in this appendix.

3 × 3 Mazes

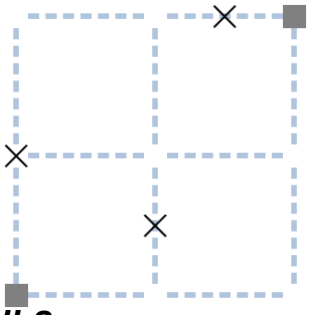
19dvv

Mean difficulty rating: 1.100



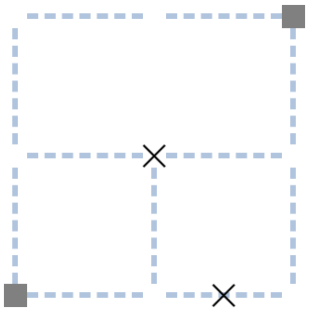
2ejkz

Mean difficulty rating: 1.820



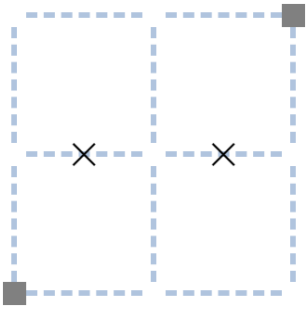
a73bo

Mean difficulty rating: 1.354



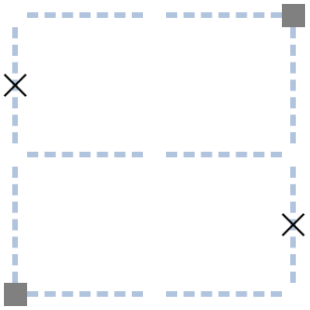
flova

Mean difficulty rating: 1.000



qnlb2

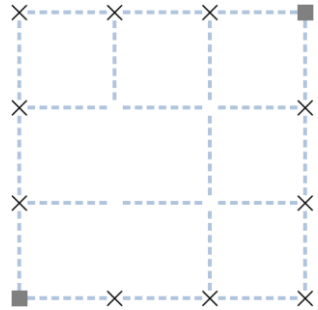
Mean difficulty rating: 1.255



4 × 4 Mazes

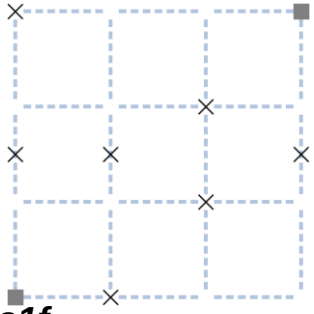
08yme

Mean difficulty rating: 3.206



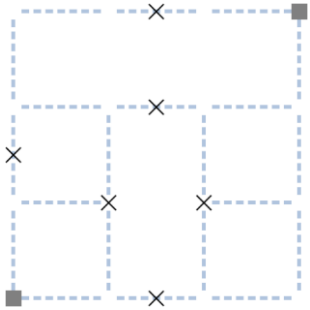
3gnrb

Mean difficulty rating: 3.300



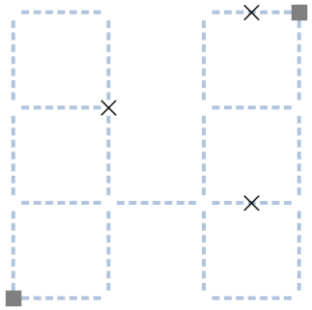
4nbih

Mean difficulty rating: 2.784



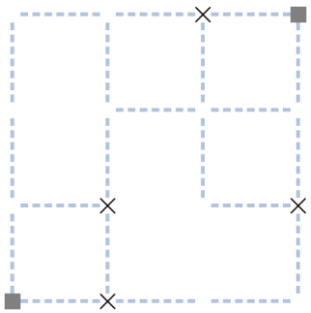
e0brl

Mean difficulty rating: 1.780



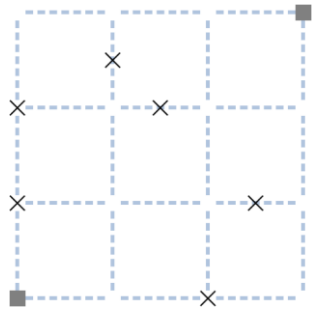
g3a1f

Mean difficulty rating: 2.029



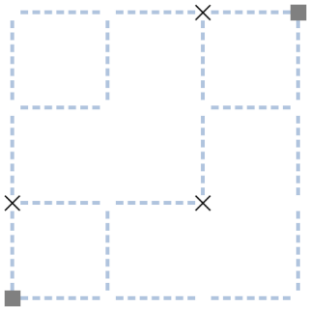
i0liy

Mean difficulty rating: 3.882



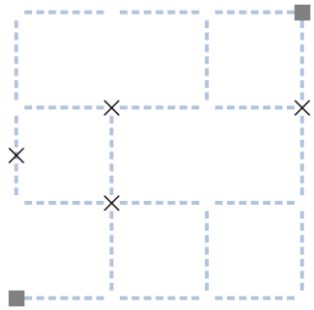
jrywy

Mean difficulty rating: 1.190



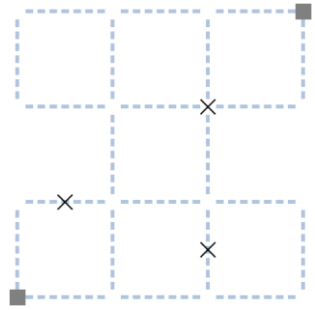
tf66w

Mean difficulty rating: 1.485



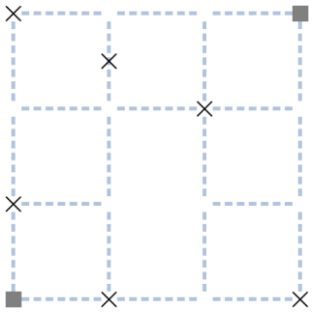
uwygl

Mean difficulty rating: 1.548



y74av

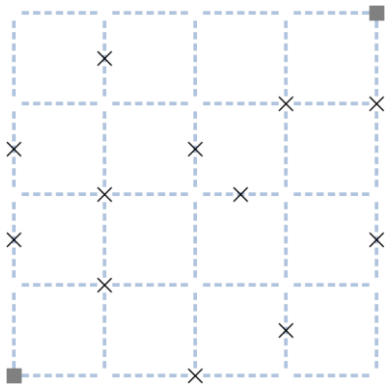
Mean difficulty rating: 2.520



5 × 5 Mazes

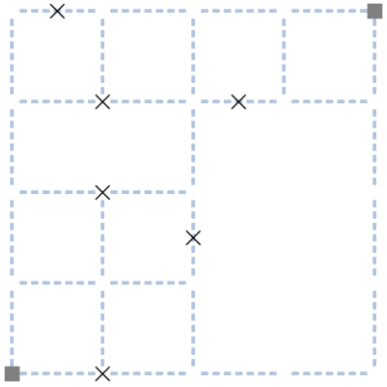
4674c

Mean difficulty rating: 4.490



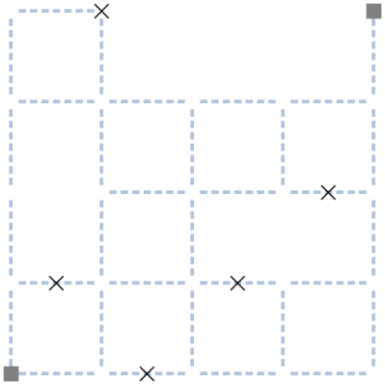
47c4z

Mean difficulty rating: 2.346



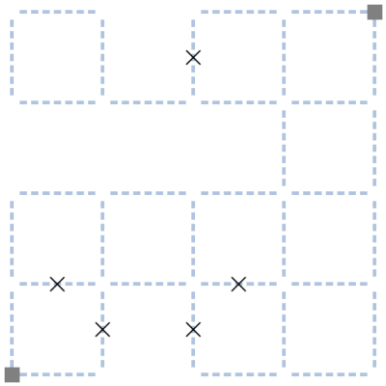
5ucv8

Mean difficulty rating: 3.381



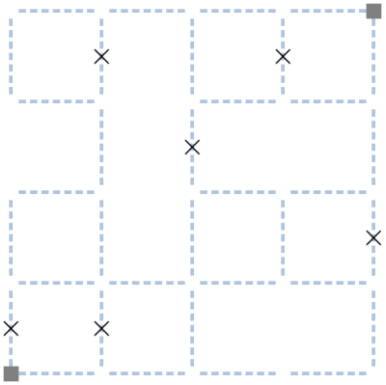
679b1

Mean difficulty rating: 3.755



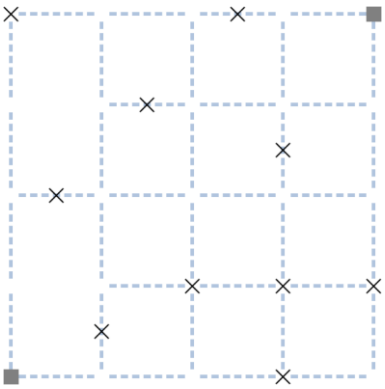
83qrr

Mean difficulty rating: 4.500



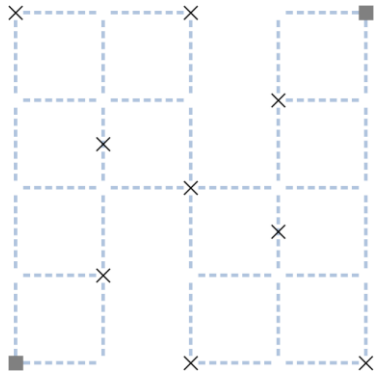
9i9hf

Mean difficulty rating: 3.990



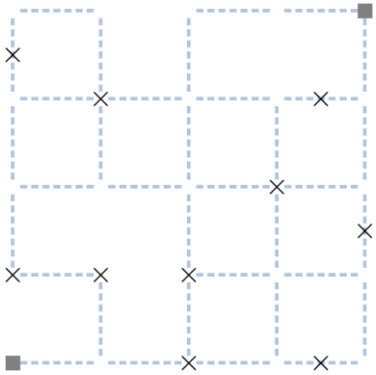
9wmst

Mean difficulty rating: 2.703



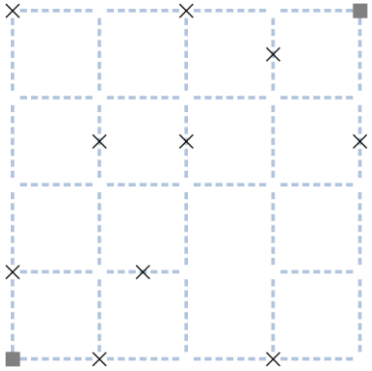
r9gka

Mean difficulty rating: 2.892



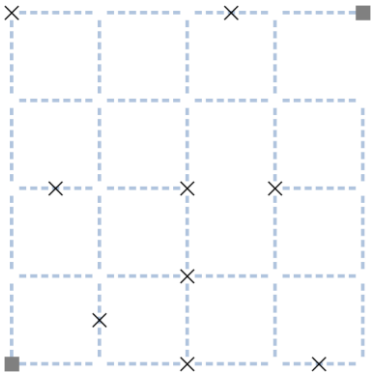
x394j

Mean difficulty rating: 4.584



x4use

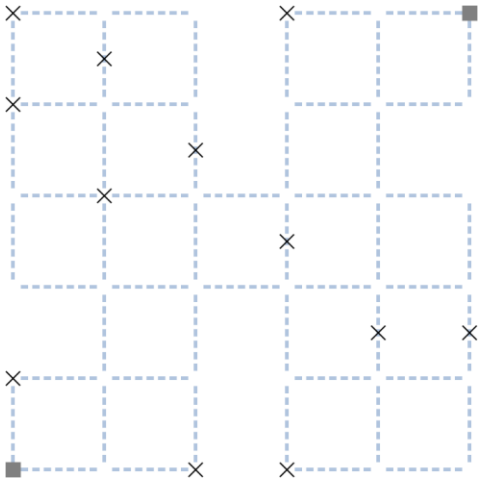
Mean difficulty rating: 4.069



6 × 6 Mazes

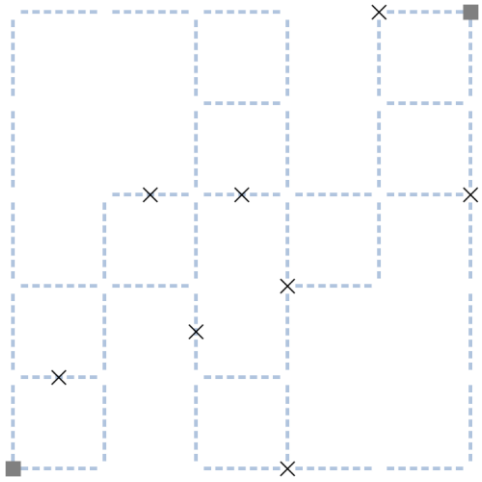
3l842

Mean difficulty rating: 5.691



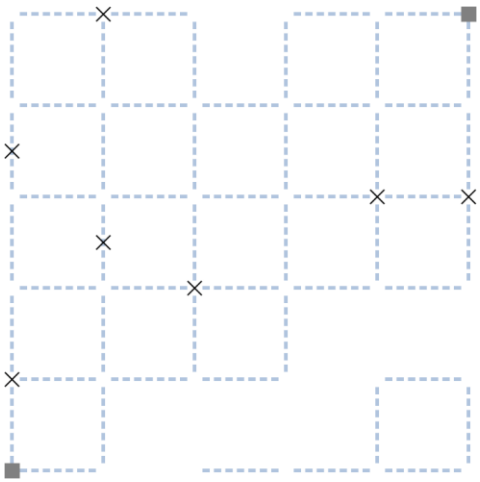
bbeya

Mean difficulty rating: 6.186



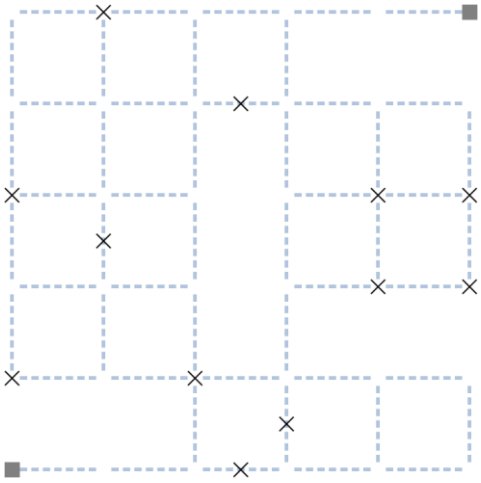
e1o3w

Mean difficulty rating: 2.154



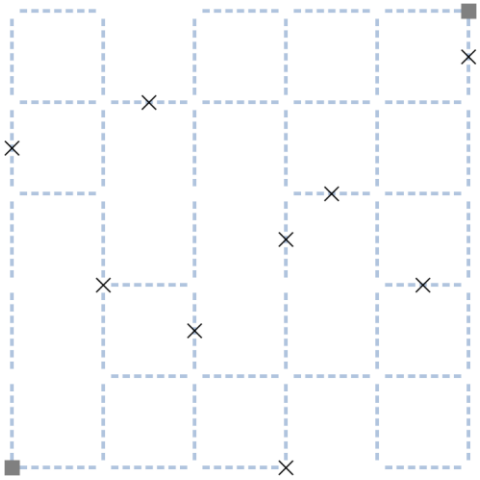
idkat

Mean difficulty rating: 5.168



voga3

Mean difficulty rating: 4.753



Appendix B: Website

B.1 Questions

1. How old are you?

(a) 16 – 20 (b) 21 – 25 (c) 26 – 30 (d) 31 – 40
(e) 41 – 50 (f) 51 – 60 (g) 61 or over

2. How often do you play puzzle games or solve puzzles, either online or on paper (for example: sudoku, mazes, logic puzzles or word puzzles)?

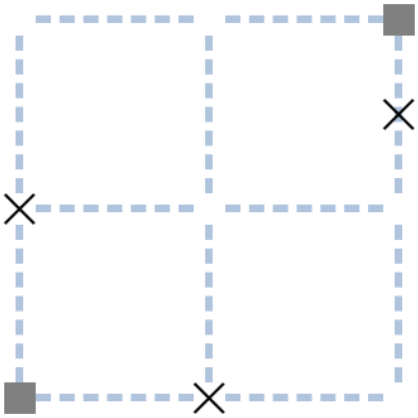
(a) Less than once a month (b) Once a month or so (c) Once a week or so
(d) Several times a week (e) Every day

3. The game The Witness (2016) *contains similar logic mazes to the ones used in this project. Have you played that game?*

(a) Yes
(b) No
(c) No, but I've watched someone else playing it

B.2 Tutorial

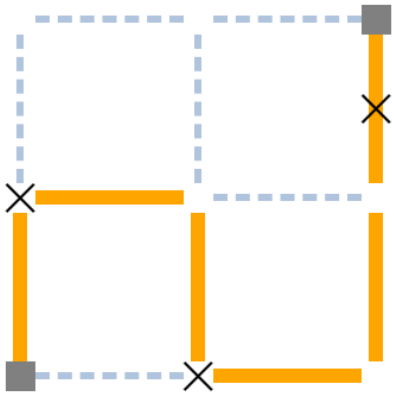
These puzzles are called logic mazes. For each puzzle, you will see a grid such as this one:



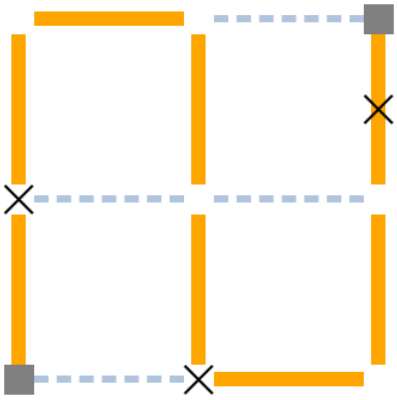
The aim of these puzzles is to make a single path linking the grey squares in the bottom left and top right corners of the maze, by following the dashed lines. The path must also pass through every checkpoint (×). The path must not have any branches or loops and it cannot touch or cross itself.

To draw an edge in the path, click on one of the dashed lines. To remove an edge, click on the edge. Once you find a solution, the path will change colour.

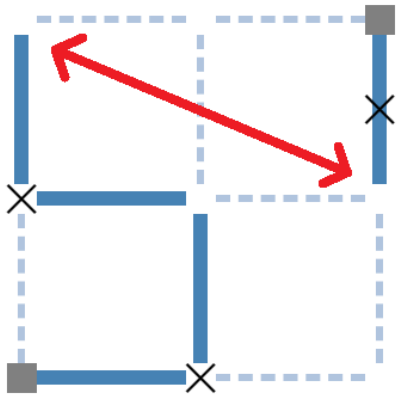
Examples



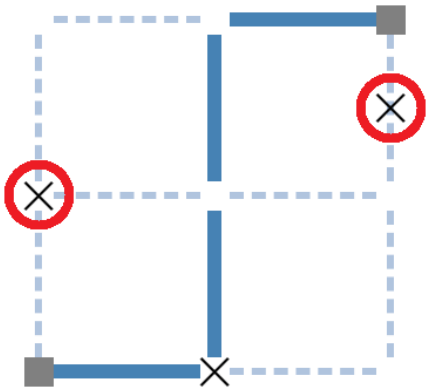
This is a correct solution. The orange line connects the start and end points (grey squares) and passes through each checkpoint.



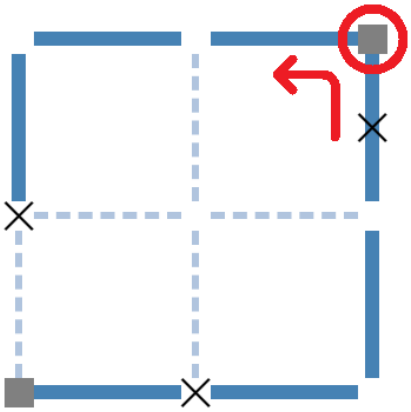
This is another correct solution. There may be alternative solutions for each puzzle – you only need to find one.



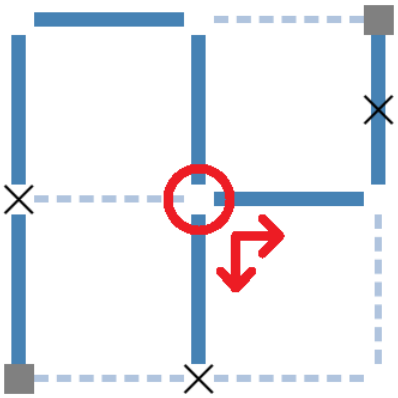
This is not a solution because it does not connect the grey squares.



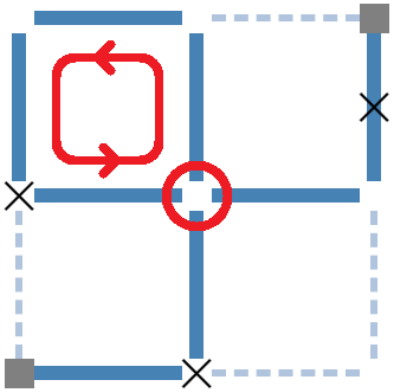
This is not a solution because the path misses two of the checkpoints.



This is not a solution because the path continues beyond the grey square in the top right.



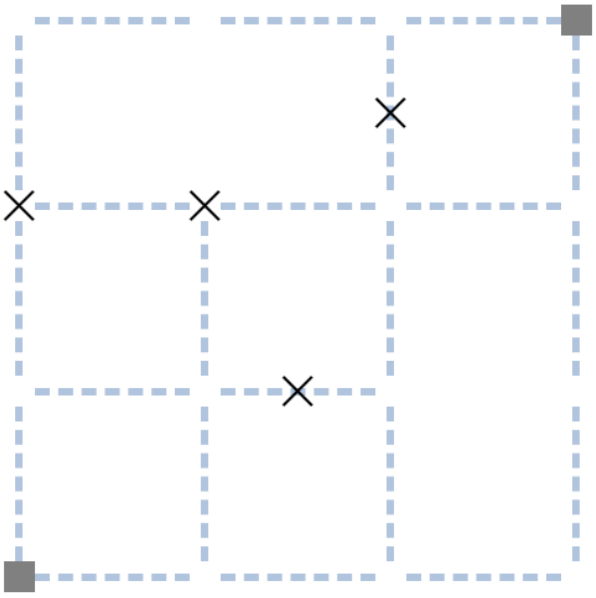
This is not a solution because the path branches.



This is not a solution because the path touches/crosses itself, forming a loop.

Get Started

To show that you understand how the puzzles work, please complete this maze. Remember, you click on dashed lines to put them into your path, or click on solid lines to take them out. The path will change colour when you get it right.



Appendix C: Human-Inspired Solver Demonstration

This appendix demonstrates a possible set of steps taken by the human-inspired solver used in this project to solve a logic maze. Throughout this appendix, co-ordinates will be used to refer to vertices and edges of the puzzle. For vertices, these coordinates are written XY where X and Y are the x and y coordinates of the vertex on the grid, starting at (0,0) in the bottom left corner of the grid. Figure A3.1 shows these co-ordinates. For a horizontal edge, the co-ordinates are XYh where XY is the co-ordinates of the edge's left vertex. For a vertical edge the co-ordinates are XYv where XY is the co-ordinates of the edge's bottom vertex.

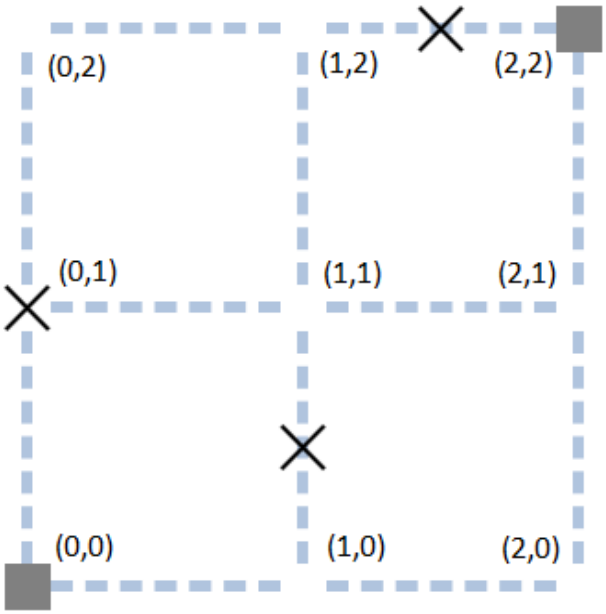


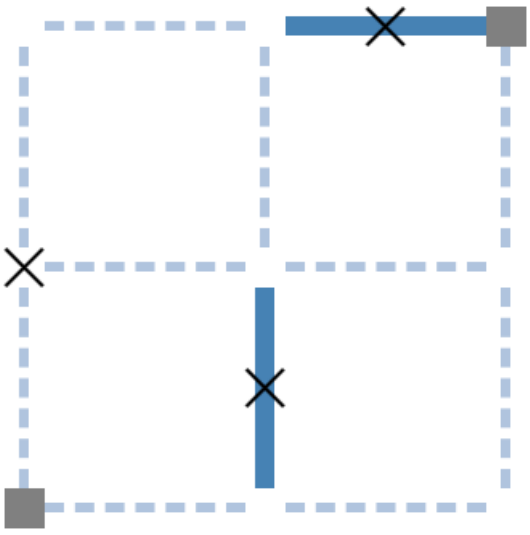
Figure C-1: Vertex co-ordinates on the puzzle to be solved.

Recall the solver's rules:

Rule 0: The solution includes all checkpoints and endpoints and excludes all breaks.
Rule 1: An included vertex must connect to exactly 2 included edges.
Rule 2: There can be no loop of included edges.

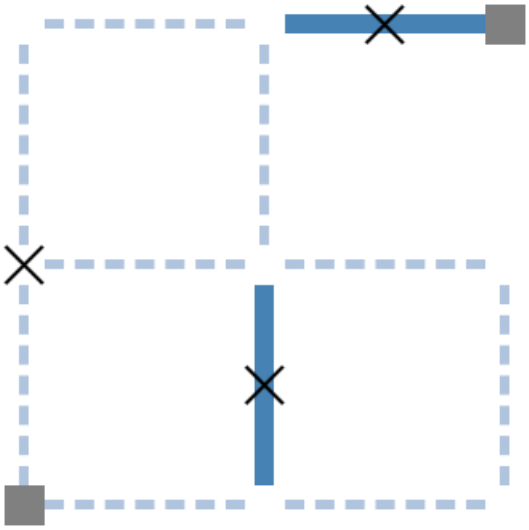
Step 1

Before starting the main loop, the system applies Rule 0. Vertices 00, 01 and 22 are all set to included (endpoints and checkpoints). Edges 10v and 12h are set to included (checkpoints), which automatically also sets vertices 10, 11 and 12 to included. There are no breaks in this puzzle.



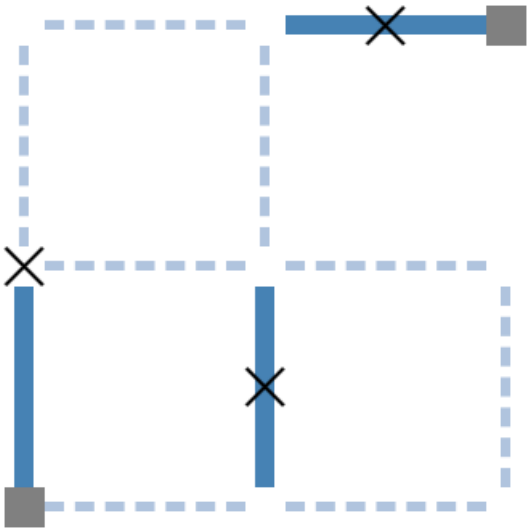
Step 2

Rule 1 is applied at 22. Since 22 is an endpoint, it is always considered to have 1 extra included edge, so it already has 2. Therefore 21v must be excluded.



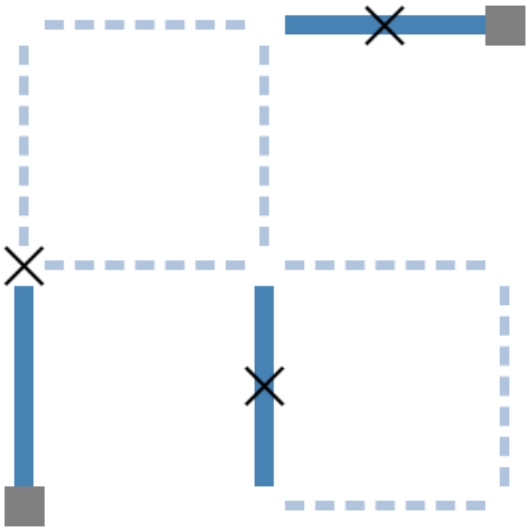
Step 3

Neither Rule 1 nor Rule 2 can be applied here, so the system makes a guess. It chooses a random end from the 4 possibilities (00, 10, 11, 12) – here it chooses 00. It then chooses a random unknown edge at that end – here it chooses 00v.



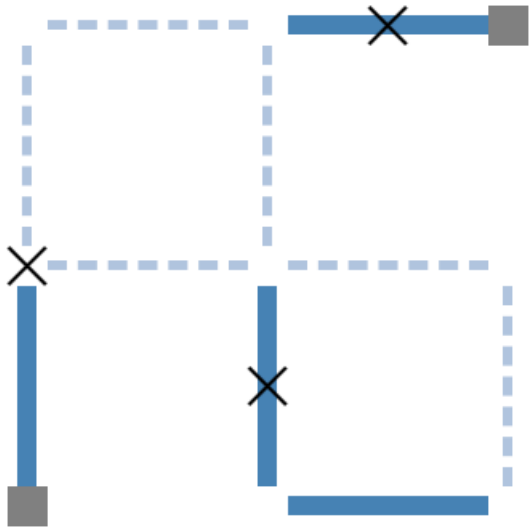
Step 4

Rule 1 is applied at 00, setting 00h to excluded.



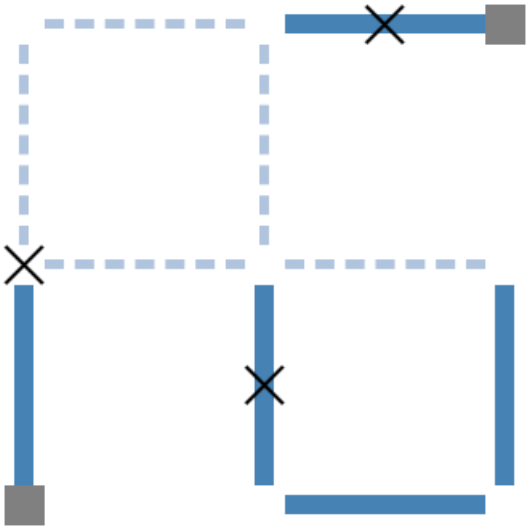
Step 5

Rule 1 is applied at 10, setting 10h to included; 10 is known to be included and has only 2 edges that are not excluded, so both most be included.



Step 6

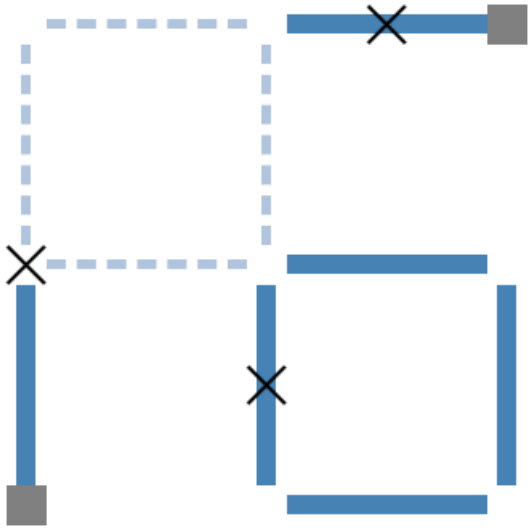
Rule 1 is applied at 20, setting 20v to included.



Step 7

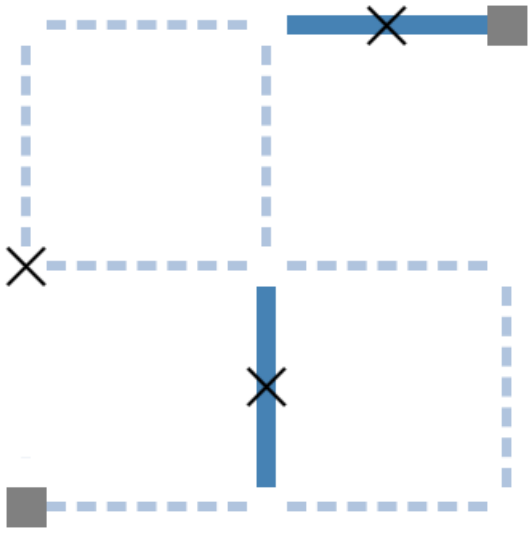
Rule 1 is applied to 21, setting 11h to included. A contradiction is detected after this deduction due to the loop that was formed.

Note that Rule 2 also triggered but did not fire. If Rule 2 had fired, 11h would have been set to excluded; a contradiction would then also have been detected because the end at 21 would have been isolated. Hence, the result of either possibility would be much the same.



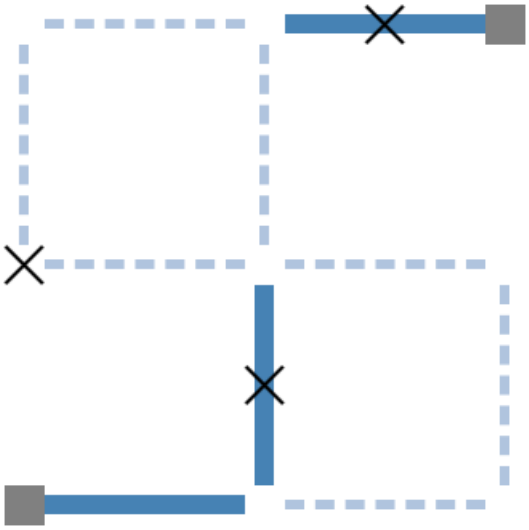
Step 8

The guess made at step 3 is reverted. This causes 00h, 10h, 11h and 20v to be set back to unknown and 00v, the original guess, to be set to excluded.



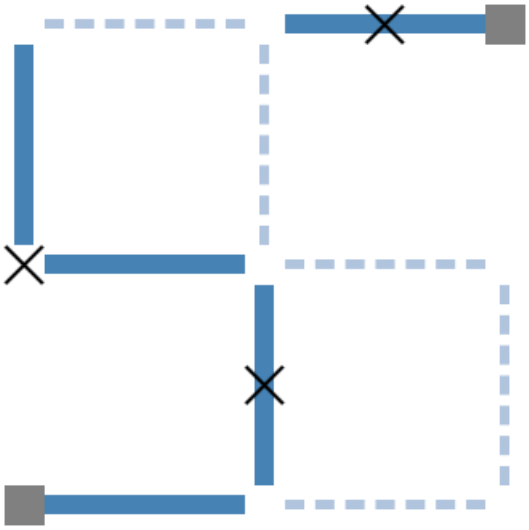
Step 9

Rule 1 is applied at 00, setting 00h to included.



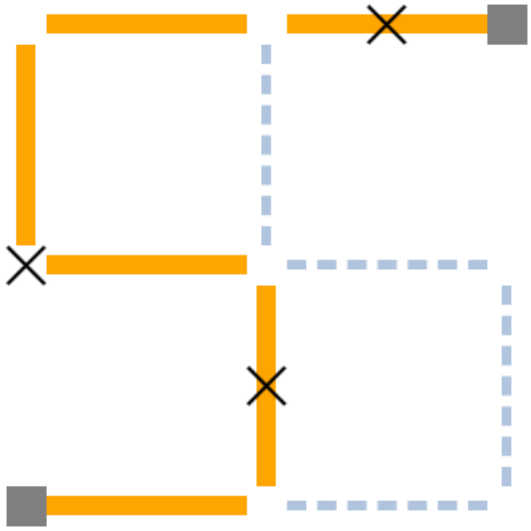
Step 10

Rule 1 is applied at 01, setting both 01h and 01v to included. 01 is known to be included because of its checkpoint.



Step 11

Rule 1 is applied to 02, setting 02h to included. At this point a solution has been found; recall that unknown edges are considered to be excluded for the purposes of checking for a solution.



Predictors

During this process, the following predictors would be measured:

C_1: Rule 1 was applied 8 times

C_2: Rule 2 was applied 0 times

C_3: A guess was made 1 time

C_4: A guess was reverted 1 time

C_5: The maximum number of unreverted guesses was 1

C_6: Rule 1 was applied 8 times

C_7: The maximum number of deductions made before needing to revert a guess was 4

The length of the solution found was 6. In this case, the minimum solution length B_2 is also 6, so the solver would also have successfully identified that predictor.

Appendix D: Data

D.1 Demographics

Q1: *How old are you?*			Q2: *How often do you solve puzzles?*			Q3: *Have you played* The Witness?		
Response	**n**	**%**	**Response**	**n**	**%**	**Response**	**n**	**%**
16 – 20	10	8.1%	Less than once a month	34	27.4%	Yes	15	12.1%
21 – 25	22	17.7%				No	102	82.3%
26 – 30	23	18.5%	Once a month or so	29	23.4%	No, but I've watched someone else play it	7	5.6%
31 – 40	33	26.6%	Once a week or so	25	20.2%			
41 – 50	16	12.9%	Several times a week	18	14.5%	*Note: Some questions and answers are paraphrased here. For the exact wording, see Appendix B.1.*		
51 – 60	14	11.3%						
61 or over	6	4.8%	Every day	18	14.5%			

Table D-1: Participant responses to the questionnaire. Responses from participants who solved 0 puzzles are excluded.

D.2 Puzzle Data

ID	N	Difficulty (unscaled)	Difficulty (scaled)	A_1	A_2	A_3	A_6	B_1	B_2	B_3	C_7	C_8
08yme	102	3.206	0.028	4	16	22	10	2	14	3.0	16.5	0.11
19dvv	100	1.100	-1.010	3	9	12	2	6	4	2.0	8.8	0.00
2ejkz	100	1.820	-0.635	3	9	12	3	1	6	2.5	8.7	0.55
3grnb	100	3.300	0.127	4	16	24	7	3	14	2.5	20.1	0.27
3l842	97	5.691	1.384	6	36	54	12	1	32	5.5	49.1	0.67
4674c	98	4.490	0.784	5	25	40	12	9	20	3.0	28.6	0.09
47c4z	104	2.346	-0.317	5	25	31	6	11	14	3.5	32.7	0.16
4nbih	102	2.784	-0.133	4	16	21	6	3	12	3.0	17.0	0.19
5ucv8	105	3.381	0.183	5	25	33	5	12	18	4.5	31.1	0.13
679b1	102	3.755	0.384	5	25	35	5	1	20	8.0	24.0	0.57
83qrr	100	4.500	0.781	5	25	35	6	4	20	5.0	39.4	0.39
9i9hf	101	3.990	0.512	5	25	38	10	12	20	2.5	31.2	0.11
9wmst	101	2.703	-0.184	5	25	36	9	16	20	2.5	26.9	0.03
a73bo	99	1.354	-0.875	3	9	11	2	2	6	2.5	8.4	0.00
bbeya	102	6.186	1.689	6	36	46	8	2	28	10.0	42.2	0.38
e0brl	100	1.780	-0.665	4	16	21	3	2	10	4.0	23.6	0.25
e1o3w	104	2.154	-0.471	6	36	53	7	1576	16	3.0	47.1	0.08
flova	106	1.000	-1.067	3	9	12	2	2	4	1.5	7.7	0.00
g3a1f	104	2.029	-0.507	4	16	21	4	11	10	3.0	21.2	0.09
i0liy	102	3.882	0.411	4	16	24	6	2	14	2.5	23.9	0.45
idkat	101	5.168	1.124	6	36	52	12	20	28	4.0	70.2	0.51
jrywy	105	1.190	-0.959	4	16	20	3	5	6	2.0	18.0	0.08
qnlb2	102	1.255	-0.959	3	9	10	2	1	8	3.0	7.0	0.00
rg9ka	102	2.892	-0.050	5	25	35	10	3	20	3.0	30.8	0.25
tf66w	99	1.485	-0.817	4	16	21	4	5	8	2.0	17.3	0.10
uwygl	104	1.548	-0.770	4	16	22	3	4	8	3.0	18.4	0.27
voga3	97	4.753	0.897	6	36	52	9	2	22	3.0	95.3	0.73
x394j	101	4.584	0.796	5	25	39	10	11	22	3.5	35.9	0.18
x4use	101	4.069	0.575	5	25	38	9	18	20	3.5	34.9	0.20
y74av	102	2.520	-0.255	4	16	23	6	4	14	3.0	19.0	0.08

Table D-2: Data and the most important predictors collected for each of the 30 puzzles used. N is the number of participants who solved the puzzle.