Universiteit Leiden
The Netherlands

# Opleiding Informatica

A Further Look

into Mouse Mazes

Tim Merckens

Supervisors:
dr. W.A. Kosters & dr. J.M. de Graaf

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

February 12, 2019

## Abstract

MouseMaze is a game where "Squeaky" tries to escape from a user designed maze. The user can place walls on a grid to create this maze, after which "Squeaky" will walk through it according to specific rules based on how often he has visited the neighbouring squares before, breaking ties by choosing down and right over left and up.

In this thesis we have looked at finding the optimal mazes for different dimensions and we have looked into proof for mazes without loops. In all cases we want "Squeaky" to stay in the maze as long as possible. Research into different kind of loops has also been done. For $2 \times 2$ loops, Squeaky will at most turn around once on the first visit, but will always continue with his path when it arrives at the entrance of the loop for the second time. The optimal mazes have been found for mazes with a dimension of $7 \times 7$ or lower by using an algorithm that generates all possible mazes. A concrete proof and formulas have been found to allow for calculating the amount of steps "Squeaky" needs to exit any loop-less maze.

# Contents

# Chapter 1

# Introduction

We will look at analysis done on the game Mouse Maze. Traditionally when looking at mazes, the focus is on algorithms to allow the mouse to find the exit as quickly as possible [MB08]. However, in this game, a user constructs a maze, as shown in Figure 1.1, which "Squeaky"[1] needs to escape. He starts at the top left and has to exit through the bottom left. The goal is to keep him inside of the maze for as many turns as possible. A turn consists of him moving from one square to another. Based on the amount of times he has previously visited the horizontal and vertical neighbouring squares he chooses the least visited neighbour.

More specific, we will look at the behaviour and optimization of simple loops and at calculating the amount of steps "Squeaky" needs to escape loop-less mazes. Since loops generally result in him turning around and revisiting squares, this complicates the calculation of mazes that do have loops to a point where better understanding of individual loops is required before these can be calculated. To this end, we made a start analyzing basic loops.

We also focused on simulations of lower dimensional mazes in an attempt to find the optimal solutions where "Squeaky" needs the most steps to exit the maze.

[1] The mouse will be referred to as "he".

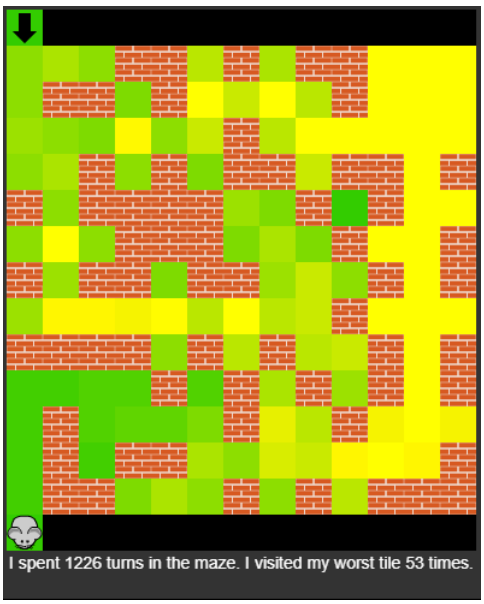



Figure 1.1: Mouse Maze on `www.kongregate.com`.

The game on `www.kongregate.com` shows the amount of times each square is visited with a color scheme. Squares start green and slowly turn yellow as they become more visited. The amount of steps needed is shown as turns spent in the maze.

## 1.1 Thesis Overview

Chapter 2 explains the rules of the game, Chapter 3 shows related work done on Mouse Maze, Chapter 4 explains how mazes are generated to find high-scores and shows the results of these, Chapter 5 contains proof about calculating the scores for mazes without any loops in them and Chapter 6 concludes .

This bachelor thesis is written under supervision of Walter Kosters and Jeannette de Graaf for the bachelor computer science from Leiden Institute of Advanced Computer Science at Leiden University.

# Chapter 2

# Game Rules

MouseMaze is a game on Kongregate [Fra09] created by Tom Fraser, who uses the tag CuriousGaming; see Figure 2.1 for an example game situation.
The goal of the game is to design a maze that locks the mouse, "Squeaky", in the maze for the largest number of moves. The game is usually played on a $13 \times 13$ grid of squares, where every square can either be a wall

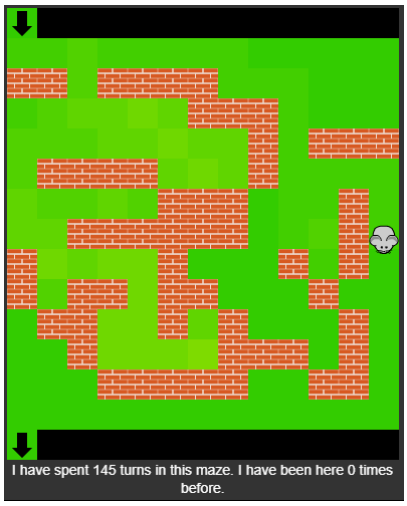



Figure 2.1: Mouse Maze on `www.Kongregate.com`.

or an open space. The player can place the entry square anywhere above the top row and the exit square anywhere below the bottom. In Figure 2.1 the entry is placed top left, and the exit bottom left. For our research, we will always assume that the entry and exit are placed there. The player can place walls to create a maze for Squeaky to walk through. He walks through this maze according to a set of rules. He can move to any direct horizontally or vertically adjacent square that is not a wall, and he selects the square that he walks to based on the number of times he has visited the neighbouring squares. A visit count is used for every square to keep track of the number of times the square has been visited. He will prefer to go to the neighbouring square with the lowest visiting count, however, if there are multiple squares that classify as lowest, he will break the tie by preferring down and right over left and up (in that order).

For Squeaky to be able to reach the exit for any given maze, it only needs to comply with one condition: there has to be at least one path that leads from the start to the exit.

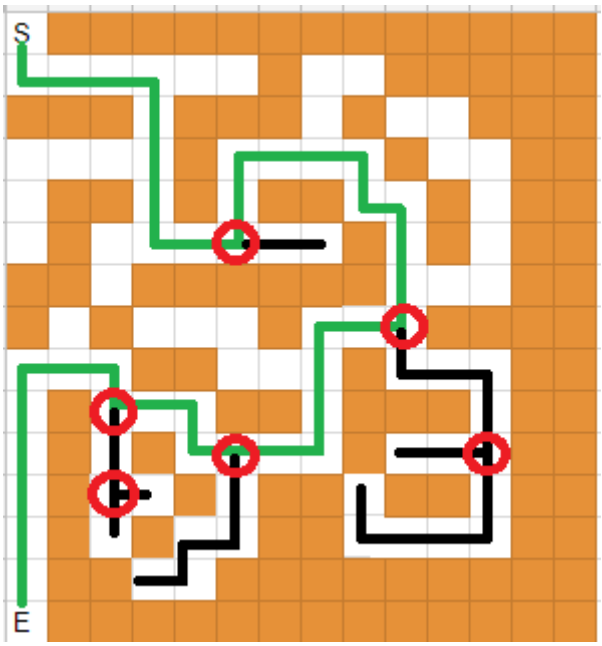



Figure 2.2: An example maze walk.

An example maze is shown in Figure 2.2. The path Squeaky walks is shown in green for the path towards the exit and in black for the side paths he takes. The crossroads where he needs to make a choice on how to progress through the maze are circled in red. He needs 98 steps to exit this maze.

# Chapter 3

# Related Work

Research on the game MouseMaze has been done by Enright and Faben in [EF16] They proved Squeaky is always able to escape given that there is a path leading to the exit by establishing an upper bound based on the following intuition:

> "If Squeaky doesn't escape, then he must enter an infinite loop, but if he enters an infinite loop, then there is a square which is adjacent to that infinite loop which is only visited finitely (perhaps 0) times, but Squeaky would eventually prefer to go there than stay inside the loop, which is a contradiction. "

The upper bound the authors set is the following: $|V(G)| \sum_{i=1}^{diam(G)} \Delta(G)^i$. Here $G$ is a graph representing the maze, $V(G)$ is the collection of vertices in $G$ and $\Delta(G)$ is the maximum degree of any vertex in $V(G)$. $Diam(G)$ is the length of the longest path in $G$, that is, the maximum distance between any two vertices.

However, this upper bound is so much higher than the actual number of steps Squeaky needs in practise to exit the maze that it is more interesting to look at a lower bound: what is the highest achievable score? To get a better understanding of this, the authors have considered different algorithms on lower dimensional mazes and compared the best mazes these algorithms were able to find. The best found algorithm was using brute force to check all feasible mazes. However, the authors were unable to find any maze with a better high-score than the currently known best maze (see Figure 3.1) with dimension $13 \times 13$. They did show that there is no better maze within a Hamming distance of 3 from this best known maze.

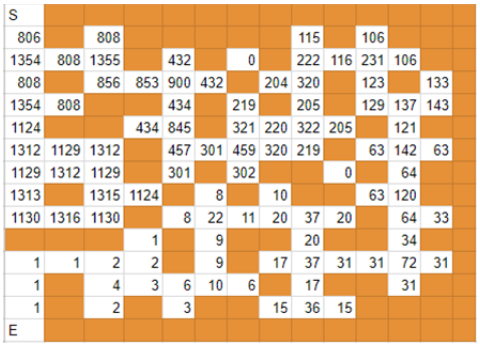



Figure 3.1: The best known maze for $13 \times 13$, showing amount of visits per square. Total steps: $36,314$.

Another issue they examined was what would happen if the mouse is unable to count and would lose count after $k$ visits to a square. These mouses are called $k$-mod mice, and after they count to $k$, the visit count becomes 0 in their mind. With this, constructions can be made to forever trap $k$-mod mice in a maze.

# Chapter 4

# Optimal Mazes

Based on research done by Enright and Faben [EF16], generating all possible mazes to find the optimal maze for lower dimension mazes seemed like the best way to determine which maze is hardest for Squeaky to escape from . In their research they were able to find the best mazes for sizes up to $6 \times 6$, however, they did not put time into optimizing the algorithm, so we tried to see how far we could get by improving the algorithm.

## 4.1 Algorithm

Any board with walls placed on it form a maze, so for any $n \times n$ board, there exist $2^{n^2}$ mazes. From this collection of all mazes, we only have to consider possible mazes. These are defined as mazes with at least one path leading from the entrance to the exit. Any maze that does not contain a path like this can be disregarded as it cannot be the optimal maze. We then propose an algorithm that tests all possible mazes to find the optimal maze, which is the one that requires the most steps from Squeaky before he finds the exit. To implement this, we use a brute force method that generates a mouse run through every possible maze. This is split into different tasks based on the starting position, which allows for multithreading, where every thread starts from a locked starting position and tests every possible maze that can be built from there. These starting positions consist of the initialization of the first two rows, covering every possible wall combination for these rows.

### 4.1.1 Optimization

To optimize the algorithm, the maze is stored as an array of unsigned integers, where each entry contains the bit representation of that row. This also makes it easier to set the starting rows, by just incrementing integers from 1 until $n$, where $n$ represents the dimension of the maze. These integers can easily be used as bit representations of the different starting positions. As example, given $n = 4$: then 0 would be read as four 0 bits (0000) representing no walls on the first row. In the same way, 3 would be read as two 0 bits and two 1 bits (1100) representing two walls at the left side of the row.

Another place where improvement can be gained is determining if a maze has a path leading to the exit. Any maze where there is a wall directly in front of the exit or entrance is immediately discarded, and the same is done if there is a full row of walls anywhere in the maze. On top of this, we can quickly scan the array by using an OR operation on two neighbouring rows and if the result leads to only 1's, we know the maze is blocked without any further testing required. Once a maze is blocked all mazes that can be constructed by adding walls based on that specific maze will be skipped.

There is still probably more optimization possible by increasing detection of mazes that are impossible to be the optimal solution. Another area to look into is the simulation of the mouse running through a maze: if it would be possible to have a formula to calculate this instead of having to fully simulate all the steps, it could speed up the processing of mazes.

One last area of improvement would be rewriting the code to run on GPUs instead of the CPU, for example with CUDA [CUD18], which could lead to many more threads being processed at the same time.

## 4.2 Results

This algorithm was able to find the best possible maze in reasonable time for mazes of sizes up to $7 \times 7$. Table 4.1 shows the number of steps it takes for the mouse to escape these mazes and the total execution time to determine the optimal maze. The algorithm is written in C++ and ran under Windows on an *i*7-6800*k* CPU @3.40GHz with 6 Cores and 12 Logical Processors. The execution time is shown in the following format: "h:m:s:ms".

| Maze dimensions | number of steps | Total execution time | number of mazes tested |
|---|---|---|---|
| $3 \times 3$ | 12 | 0:00:01:004 | 73 |
| $4 \times 4$ | 27 | 0:00:01:006 | 5,712 |
| $5 \times 5$ | 54 | 0:00:02:091 | 490,995 |
| $6 \times 6$ | 117 | 0:01:55:700 | 945,876,388 |
| $7 \times 7$ | 332 | 4:51:28:900 | 156,094,133,213 |

Table 4.1: Optimal Mazes

The best found mazes are shown in Table 4.2– 4.6. The top part of each picture shows the layout of the maze, where *X* represents a wall and . an empty square. Next to it the number of times Squeaky visits every empty square is shown. The algorithm only registers if it finds a bigger maze, so there is a possibility of other mazes existing with the same number of steps required for Squeaky to escape.

| | | | | | |
|---|---|---|---|---|---|
| . | . | . | 1 | 1 | 1 |
| X | . | . | X | 1 | 2 |
| . | . | . | 1 | 2 | 2 |

Table 4.2: Optimal maze for $3 \times 3$: 12 steps.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| . | X | . | . | ‖ | 1 | X | 2 | 2 |
| . | . | . | . | ‖ | 1 | 2 | 2 | 2 |
| X | . | . | . | ‖ | X | 2 | 2 | 2 |
| . | . | . | . | ‖ | 1 | 2 | 2 | 2 |

Table 4.3: Optimal maze for $4 \times 4$: 27 steps.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| . | . | X | . | . | ‖ | 3 | 4 | X | 2 | 2 |
| X | . | . | . | . | ‖ | X | 4 | 3 | 4 | 2 |
| . | . | X | . | . | ‖ | 1 | 3 | X | 3 | 4 |
| X | . | . | X | . | ‖ | X | 2 | 1 | X | 3 |
| . | . | . | . | . | ‖ | 1 | 2 | 3 | 3 | 3 |

Table 4.4: Optimal maze for $5 \times 5$: 54 steps.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| . | . | . | . | . | . | ‖ | 2 | 4 | 3 | 4 | 5 | 4 |
| X | . | X | X | . | . | ‖ | X | 3 | X | X | 6 | 5 |
| . | . | X | . | . | X | ‖ | 2 | 3 | X | 6 | 11 | X |
| . | . | . | X | . | . | ‖ | 2 | 3 | 1 | X | 8 | 10 |
| X | . | . | . | X | . | ‖ | X | 2 | 2 | 1 | X | 8 |
| . | . | . | X | . | . | ‖ | 1 | 2 | 1 | X | 7 | 10 |

Table 4.5: Optimal maze for $6 \times 6$: 117 steps.

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| . | . | . | . | . | . | . | ‖ | 14 | 22 | 14 | 23 | 12 | 23 | 10 |
| . | . | X | . | X | . | X | ‖ | 22 | 15 | X | 12 | X | 11 | X |
| . | . | . | X | . | . | . | ‖ | 15 | 24 | 15 | X | 8 | 22 | 8 |
| X | X | X | . | X | . | X | ‖ | X | X | X | 1 | X | 9 | X |
| . | . | . | . | X | . | X | ‖ | 1 | 1 | 2 | 2 | X | 9 | X |
| . | X | . | . | . | . | . | ‖ | 1 | X | 4 | 3 | 6 | 10 | 6 |
| . | X | . | X | . | X | X | ‖ | 1 | X | 2 | X | 3 | X | X |

Table 4.6: Optimal maze for $7 \times 7$: 332 steps.

Additionally, we have let the algorithm try to find the optimal maze for sizes $8 \times 8$ and $9 \times 9$ (see Table 4.7 and Table 4.8), however there is no guarantee that the best maze the algorithm was able to find is also the optimal maze. Generating all possible mazes was not feasible for these mazes, so more heavy restrictions were placed on generating them. For the $8 \times 8$ maze, the last three rows are set to those from the optimal $6x6$ maze, up until the sixth column, for the last columns every possibility is generated. The $9x9$ maze uses the bottom-left $4 \times 5$ matrix from the optimal $7 \times 7$ maze. This choice was made after recognizing most of the optimal $7 \times 7$ maze was used as basis to find the optimal $13 \times 13$ maze. Since these two mazes are generated on similar restrictions, they look fairly alike, with both of them having a completely empty second row and both attempt to keep "Squeaky" walking around in the bottom-right side of the maze for as many steps as possible.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| . | X | . | X | X | . | . | X | 4 | X | 4 | X | X | 14 | 13 | X |
| . | . | . | . | . | . | . | . | 7 | 5 | 8 | 13 | 12 | 14 | 23 | 14 |
| . | . | X | . | X | X | . | X | 5 | 6 | X | 8 | X | X | 19 | X |
| X | . | . | X | . | . | . | . | X | 5 | 2 | X | 22 | 30 | 23 | 23 |
| . | . | X | . | . | . | . | . | 2 | 3 | X | 22 | 34 | 23 | 30 | 23 |
| . | . | . | X | . | . | . | X | 2 | 3 | 1 | X | 23 | 32 | 24 | X |
| X | . | . | . | X | . | . | . | X | 2 | 2 | 1 | X | 24 | 35 | 24 |
| . | . | . | X | . | . | X | X | 1 | 2 | 1 | X | 24 | 34 | X | X |

Table 4.7: Best found maze for $8 \times 8$: 647 steps.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| . | X | . | X | X | . | . | X | . | 4 | X | 4 | X | X | 10 | 12 | X | 12 |
| . | . | . | . | . | . | . | . | . | 6 | 5 | 8 | 14 | 8 | 15 | 14 | 13 | 20 |
| . | . | X | . | X | . | X | X | . | 5 | 4 | X | 8 | X | 8 | X | X | 21 |
| . | . | X | X | X | X | . | X | . | 5 | 5 | X | X | X | X | 21 | X | 21 |
| X | . | . | X | . | . | . | . | . | X | 5 | 2 | X | 37 | 39 | 38 | 22 | 36 |
| . | . | X | . | . | . | . | X | . | 2 | 3 | X | 37 | 50 | 38 | 33 | X | 21 |
| . | . | . | X | . | . | X | . | X | 2 | 3 | 1 | X | 38 | 44 | X | 41 | X |
| X | . | . | . | X | . | . | . | . | X | 2 | 2 | 1 | X | 44 | 45 | 53 | 43 |
| . | . | . | X | . | . | . | . | . | 1 | 2 | 1 | X | 40 | 49 | 47 | 45 | 49 |

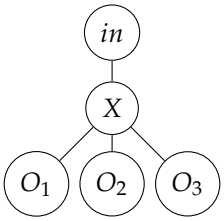
Table 4.8: Best found maze for $9 \times 9$: 1160 steps.

# Chapter 5

# Proving Mouse Behaviour

In this chapter we will provide proof for the number of times any $2 \times 2$ loop has to be entered before Squeaky exits through a designated exit. A $2 \times 2$ loop is a $2 \times 2$ square, with at least two adjacent squares that are both on the main path of the maze. A square is adjacent to the loop if it is a neighbour to any square part of the $2 \times 2$ loop. The main path is the shortest path the mouse eventually takes from the entrance to the exit, stripped from all the side paths. One of these is the entrance into the loop, and another one is the exit on which the main path will continue. Additional adjacent squares can exist, but these are not part of the eventual correct path to leave the maze. In addition to this, we will prove that calculating the number of steps it takes for a mouse to escape a maze without any loops is possible.

Before looking into the proofs, an explanation on the visualization is required. For any square "$X$", with four neighbouring squares, the following representation is used:




Here, the square representing the incoming path is labeled with "$in$" and the other three paths are in order of priority labeled with $O_i$ ($i = 1, 2, 3$). From left to right, this would mean the nodes are ordered in the following way: down, right, left, up. One of these is the incoming path, so that one is left out in the representation from the $O_i$'s. Notice that if there are more or fewer neighbours, the same representation can be used.

## 5.1 Loop Behaviour

We will look at $2 \times 2$ loops, with a designated entrance and exit. If the mouse enters the loop and leaves again through the entrance square, he has "turned around". If the mouse exits the loop through the exit square,

he will not return to the loop as long as he does not encounter any future loops that could make him turn around. To be able to prove that any $2 \times 2$ loop will continue on the main path after at most two visits, the following lemma and theorem apply.

**Lemma 1.** *Before turning around in a* $2 \times 2$ *loop, all squares in the loop are visited at least once.*

*Proof.* Upon entering the loop, all squares will have a visit count of 0. Every square is connected to two squares part of the loop. These are the previous square in the loop, with visit count 1 or higher, and the next square in the loop, with visit count 0. This means the next square in the loop will always be visited instead of directly turning around. Thus, all four squares in the loop will be visited at least once before Squeaky can turn around. □

**Theorem 1.** *To continue on the main path from a* $2 \times 2$ *loop, the loop entrance is to be visited at most twice.*

*Proof.* To exit the loop, we need a visit to the square in the loop leading to the exit where the exit has priority, thus, either all other visiting squares have a visit count of 1 or higher (the exit always has visit count 0), or the exit has a higher priority. Now since Lemma 1 proves that every square in the loop is visited at least once (unless it prioritizes the exit over continuing on the loop on the first visit), when the mouse enters the loop for the second time, the exit will always have a higher priority than continuing with the loop. Thus a single $2 \times 2$ loop never needs more than two visits before the main path is chosen.

An exception could be if the mouse was able to turn around upon entering the first square of the loop. The only way this could happen is if both $X_2$ and $X_3$ (see Figure 5.1) have the same, or higher visit count than the entrance (assuming the entrance would be connected to $X_1$ and the exit to $X_4$). On the second visit, the entrance square has a visit count of 3, which means $X_2$ and $X_3$ would both need to have a visit count of 3. There is only one scenario where this is the case after the first visit, where both $X_2$ and $X_3$ are 4 way splits. However, if both of these are 4-way splits, there is no place to have an entry nor exit at $X_1$ and $X_4$, as that would create an overlapping $2 \times 2$ loop. Since we do not allow cases with overlapping loops, we can conclude that all squares in the $2 \times 2$ loop are visited again on an eventual second visit.

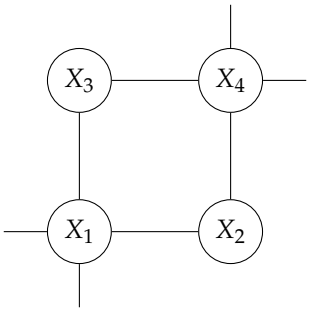



Figure 5.1: $2 \times 2$ loop.

Combined, this means that on any second visit, all squares in the loop will be visited again and when a square

neighbouring the exit is visited on the second entry, it will always prioritize the exit. Thus any $2 \times 2$ loop will at most reverse the mouse once and lead him to the exit on the second visit.

□

## 5.2 Loop-less Mazes

To prove that the algorithm for calculating the number of steps a mouse needs to complete any given maze with no loops (see Figure 2.2) is correct, we need the following statements to be true:

1. The shortest path $P$ from the start of the maze to the exit of the maze takes the lowest priority option on any crossroad it encounters.
2. The number of steps in a path without forks from a crossroad to a corresponding dead end is either $2n$ or $2n + 2$, depending on the priority of the last three squares; here $n$ is the number of squares in the path from the crossroad to the dead end, not including the crossroad itself.
3. If a crossroad not part of main path $P$ is a 4-way split, the lowest priority path will be visited twice.
4. If a path with crossroads, is visited twice, we can recursively calculate the number of steps it costs the mouse to walk through it.

We will start by defining the shortest path for Squeaky from the entrance to the exit of the maze with Theorem 2. Then, using Theorem 3 and 4, we will consider the side paths that Squeaky walks from this shortest path. For these side paths, we will determine the number of steps required to return to the shortest path, based on the length of the side path. This all is then combined to recursively calculate the total number of steps it takes him to leave the maze.

**Theorem 2.** *The shortest path P from the start of the maze to the exit of the maze takes the lowest priority option on any crossroad it encounters.*

*Proof.* Since we know that, on path $P$, once the mouse picks the correct path from any given crossroad, he will never return to that crossroad, a path with low priority can only be visited if all the paths with higher priority do not lead to the exit. Considering that we know the full path, with side paths, that Squeaky walks towards the exit, we can extract the main path $P$ from this by choosing the lowest priority option on every crossroad he encountered. Thus we can conclude that for any crossroad on this main path, the lowest priority path that is visited is always the path that leads to the exit. □

To support further theorems, a definition for the term "having priority" is given as Definition 1.
**Definition 1.** There is priority to square $X$ if the mouse prioritizes going to that square, over any other direction, from his current position.

**Theorem 3.** *The number of steps from a crossroad to a corresponding dead end is either* $2n$ *or* $2n+2$*, depending on the priority of the last three squares.*

*Proof.* Consider any dead end path of length $n$. The first $n-3$ squares will all be visited twice, once on the way to the end and once on the way back.

For the last three squares at the end of the path, there are two options. These options are based on the priorities of the first and last square, seen from the square in between them. If from the middle square the mouse has to visit the last square first based on priority of the three squares, there will be $2n+1$ steps needed. Otherwise, $2n-1$ steps are required.

To illustrate these steps, the following tables show the last 3 squares. In these tables, the numbers represent the number of visits to the corresponding squares. The lowest square is the end of the path. The bracketed numbers show the order in which the mouse walks. Priority is used to show what square he would choose if all neighbouring squares have an equal number of previous visits. The square that would be visited first has priority over the other neighbouring squares.

| | | |
|---|---|---|
| $1_{[1]}$ | 1 | $2_{[7]}$ |
| $1_{[2]}$ | $2_{[4]}$ | $3_{[6]}$ |
| $1_{[3]}$ | $2_{[5]}$ | 2 |

Figure 5.2: Priority to the last square.

| | |
|---|---|
| $1_{[1]}$ | $2_{[5]}$ |
| $1_{[2]}$ | $2_{[4]}$ |
| $1_{[3]}$ | 1 |

Figure 5.3: Priority to the previous square.

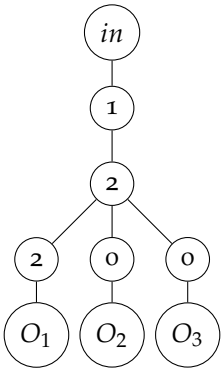
All squares except these last 3 squares are visited 2 times, once on the way towards the dead end and once on the way back to the crossroad. The last 3 squares in total either take $2*3+1=7$ steps or $2*3-1$= 5 steps (see Figure 5.2 and Figure 5.3). So in total we have either $2(n-3)+2*3+1$ or $2(n-3)+2*3-1$. This simplifies to: $2n+1$ or $2n-1$. Now we are only missing the step back from the path onto the crossroad, so the final number of steps is:

$$\begin{cases} 2n+2 & \text{if the last square of the three has priority} \\ 2n & \text{if the first square of the three has priority} \end{cases}$$

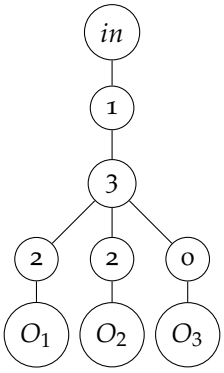
There is a special case with paths of size $n=1$, where the middle square is the crossroad, the previous square is the first square of the path that leads back to the correct path and the last square is the only square in the path with size $n=1$. The formula above is still valid for this case. For the case $n=2$ the crossroad square is used as first square to define which formula to use. □

**Theorem 4.** *If a crossroad not part of the main path P is a 4-way split, the lowest priority path will be visited twice.*

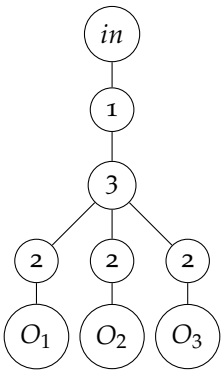
*Proof.* Consider any crossroad with incoming path *in* and three outgoing nodes $O_1, O_2$ and $O_3$. On the first visit to the crossroad, path $O_1$ will be taken. On return, the visit count of the crossroad will be at 2.



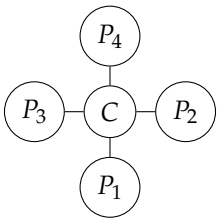

Next $O_2$ is taken, after which the visit count of the crossroad is 3. However, since upon return to the square before the crossroad, both the crossroad and the square back into the path have been visited twice, the path $O_2$ is visited for a second time if going back into the path has priority.




Now when $O_3$ is taken, upon return to the square directly before the crossroad, the crossroad value will still be at 3, however, the path back towards the dead end will have a visit count of 2. This will result in the path being taken a second time.



There are some special cases with lowest priority paths of length $n < 4$ that will be visited twice. For $n \geq 4$, there are patterns, and thus formulas that calculate the number of steps required.




**Lemma 2.** *On path $O_3$ the step back towards the crossroad is always prioritized over going into the path when both have the same visit count.*

*Proof.* Since $O_3$ is the lowest priority path from crossroad $C$, $O_3$ can only be $P_4$ or if $P_4$ is the incoming path, $P_3$. For $P_4$, the step back towards the crossroad is down, which has the highest priority. If $O_3 = P_3$, the step towards the crossroad is right, which has higher priority over left - deeper into the path. The path can't go down here, since no loops ($2 \times 2$ squares) are allowed. Thus, the step towards the crossroad will always be prioritized. □

There is a difference between $O_3$ and $O_2$ being visited for the second time, since the crossroad visit is higher when the mouse is in $O_3$. This comes down to whether there is priority towards the crossroad from the first square in $O_2$, if there is, it will not be visited twice and either $2n$ or $2n + 2$ steps are needed, otherwise they require the same number of steps as if it was $O_3$. We will look into all of theses cases for different length side paths one by one.

**Case 1.** *Length 1*

For length 1, Theorem 3 is used, where the middle square is the crossroad, the previous square is the first square of the path that leads back to the correct path and the last square is the only square in the path. The formula given there is relevant now too; so either $2n$ or $2n + 2$ steps.

**Case 2.** *Length 2*

For length 2, the following situation occurs, with the top square being the crossroad:

| | | | | |
|---|---|---|---|---|
| 3 | 3 | 3 | 3 | 3 |
| 0 | $1_{[1]}$ | $2_{[3]}$ | $3_{[5]}$ | $4_{[7]}$ |
| 0 | $1_{[2]}$ | $2_{[4]}$ | $3_{[6]}$ | 3 |

Since the mouse always has priority on going back into the crossroad, a total of 8 steps is taken.

**Case 3.** *Length 3*

The first special case for length 3 is the case where the last square has lower priority than the step back. This results into the following scenario (10 steps):

| | | | | |
|---|---|---|---|---|
| 3 | 3 | 3 | 3 | $4_{[10]}$ |
| 0 | $1_{[1]}$ | $2_{[5]}$ | 2 | $3_{[9]}$ |
| 0 | $1_{[2]}$ | $2_{[4]}$ | $3_{[6]}$ | $4_{[8]}$ |
| 0 | $1_{[3]}$ | 1 | $2_{[7]}$ | 2 |

If the last square has higher priority, the following situation arises, where, since the step back towards the crossroad has priority, in total 8 steps are needed.

| | | | |
|---|---|---|---|
| 3 | 3 | 3 | 3 |
| 0 | $1_{[1]}$ | 1 | $2_{[7]}$ |
| 0 | $1_{[2]}$ | $2_{[4]}$ | $3_{[6]}$ |
| 0 | $1_{[3]}$ | $2_{[5]}$ | 2 |

**Case 4.** *Length 4*

There are 3 different cases for length 4. After the first visit, there is a split on priority of the last square. The case for a low priority on the last square is shown below.

| | | | |
|---|---|---|---|
| 3 | 3 | 3 | 3 |
| 0 | $1_{[1]}$ | $2_{[7]}$ | 2 |
| 0 | $1_{[2]}$ | $2_{[6]}$ | $3_{[8]}$ |
| 0 | $1_{[3]}$ | $2_{[5]}$ | 2 |
| 0 | $1_{[4]}$ | 1 | 1 |

From here, there are a couple of situations that can occur. Since the mouse is at the middle square with 3 visits and both neighbouring squares have 2 visits, it depends on priority where he'll go. For the simplest case, he would walk back towards the crossroad, which would then lead to another split as both neighbouring squares are then 3. Since going to the crossroad has the highest priority, the mouse will walk back onto it and out of $O_3$ with a total of 10 steps taken.

The other case happens when going further into the path has priority, which leads to the following situation (16 steps).

| | | | |
|---|---|---|---|
| 3 | 3 | 3 | $4_{[16]}$ |
| 2 | 2 | 2 | $3_{[15]}$ |
| $3_{[8]}$ | 3 | 3 | $4_{[14]}$ |
| 2 | $3_{[9]}$ | $4_{[11]}$ | $5_{[13]}$ |
| 1 | $2_{[10]}$ | $3_{[12]}$ | 3 |

The other situation for the first visit is when the last square does have priority, which means that at least part of the path is visited twice. After the 11th step, both the neighbouring crossroad and the step back into $O_3$ have a visit count of 3. However, since there is always priority towards the crossroad, it will step towards it and a total of 12 steps is needed.

| | | | | |
|---|---|---|---|---|
| 3 | 3 | 3 | 3 | $4_{[12]}$ |
| 0 | $1_{[1]}$ | 1 | $2_{[9]}$ | $3_{[11]}$ |
| 0 | $1_{[2]}$ | 1 | $2_{[8]}$ | $3_{[10]}$ |
| 0 | $1_{[3]}$ | $2_{[5]}$ | $3_{[7]}$ | 3 |
| 0 | $1_{[4]}$ | $2_{[6]}$ | 2 | 2 |

So to conclude for length 4, the number of steps required is either $10, 16$ or 12 based on various priorities in $O_3$.

**Case 5.** *Length 5+*

For $n \geq 5$, we can define 5 different functions to calculate the number of steps. Take $m = n - 4$, then after a first visit, the visit values in the path have 2 options, based on priority of the last 3 squares. We use $m$ to represent all the squares between the crossroad and the end of the path. They all get visited they same amount of times.

| previous priority | last priority |
|---|---|
| 3 | 3 |
| 2 | 2 |
| $2m$ | $2m$ |
| 2 | 2 |
| 2 | 3 |
| 1 | 2 |

For previous priority, there is a split depending on the priority after the first step back. In case the step back towards the crossroad is prioritized, $2m + 10$ (equals $2n + 2$) steps are needed.

| | | |
|---|---|---|
| 3 | 3 | 4 |
| 2 | 2 | 3 |
| $2m$ | 3 | 3 |
| 2 | $2m$ | $2m$ |
| 2 | 2 | 2 |
| 1 | 1 | 1 |

If the step deeper into $O_3$ is prioritized from the second square of the path, a total of $4m + 15$ (equals $4n - 1$) steps is needed.

| | | | |
|---|---|---|---|
| 3 | 3 | 3 | 4 |
| 2 | 2 | 2 | 3 |
| $2m$ | $3m$ | $3m$ | $4m$ |
| 2 | 3 | 3 | 4 |
| 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 3 |

For last priority, there's three different cases, starting with the case where after the first step into $O_3$, the step back towards the crossroad has priority. This results in $2m + 12$ (equals $2n + 4$) steps.

| | |
|---|---|
| 3 | 4 |
| 2 | 3 |
| 2 | 3 |
| $2m$ | $2m$ |
| 3 | 3 |
| 2 | 2 |

If there is no priority back towards the crossroad, the path down will be walked and a split will happen at the bottom of $O_3$. The two cases that can happen are shown below, these are based on the priority of the square at place $n - 1$ compared to $n - 3$. If $n - 3$ has priority, a total of $4m + 12$ (equals $4n - 4$) steps is needed, otherwise, $4m + 18$ (equals $4n + 2$) steps are required.

| | | |
|---|---|---|
| 3 | 3 | 4 |
| 2 | 2 | 3 |
| $2m$ | $3m$ | $4m$ |
| 2 | 3 | 4 |
| 3 | 3 | 3 |
| 2 | 2 | 2 |

| | | | |
|---|---|---|---|
| 3 | 3 | 3 | 4 |
| 2 | 2 | 2 | 3 |
| $2m$ | $3m$ | $3m$ | $4m$ |
| 2 | 3 | 3 | 4 |
| 3 | 4 | 5 | 6 |
| 2 | 3 | 4 | 4 |

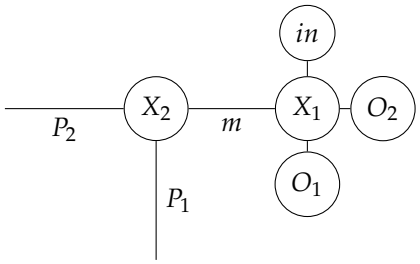
So to summarize, there are 5 different cases and formulas to calculate the number of steps needed for any path where $n > 4$, given that the path is the least priority option in a 4-way split:

$$\begin{cases} 2n+2 & \text{if previous priority and step towards crossroad has priority} \\ 4n-1 & \text{if previous priority and step into } O_3 \text{ has priority} \\ 2n+4 & \text{if last priority and step towards crossroad has priority} \\ 4n-4 & \text{if last priority and step into } O_3 \text{ has priority and square } n-3 \text{ has priority vs } n-1 \\ 4n+2 & \text{if last priority and step into } O_3 \text{ has priority and square } n-1 \text{ has priority vs } n-3 \end{cases}$$

□

**Theorem 5.** *If a path, with crossroads, is visited twice, we can recursively calculate the number of steps it costs the mouse to walk through it.*

*Proof.* All the previous proofs are based on the fact that a path that is visited twice does not have its own splits. If it does, an extension to the previously defined rules is needed. For a path $O_3$ with a split in it, let $m(\geq 1)$ be the distance between the original crossroad and the first split in $O_3$. The splitting square will be named $X_2$ with at least two outgoing paths $P_1$ and $P_2$.




For the second visit in a path $O_3$ with split, the different options are mostly based on the crossroad $X_2$ and its paths are visited or if the value of the crossroad is too high, causing the second visit to not get to the crossroad. For any 4-way split, the value of the crossroad after the first visit is 4. All values of the path between $X_1$ and $X_2$ except the neighbour of $X_1$ will be 3 (this neighbour has value 2). Thus, in this case crossroad $X_2$ will not be visited and the number of steps for the second visit is $2m-2$. This is also the case if $X_2$ is a 3-way split and the two squares on the path between $X_1$ and $X_2$ before $X_2$ both have priority back towards $X_1$.
In case that either of these do not have priority back, both path $i$ and $j$ will be visited twice. The number of steps that this takes can be calculated by using the previous theorem to calculate the number of steps for the second visit for path $P_1$ and path $P_2$ (or recursively use this theorem in case of multiple splits) and then adding those together with the number of steps it takes from $X_1$ to $X_2$. We will write $visit2(i)$ to represent a calculation for the number of steps the second visit takes for path i.
There are two different cases, depending on if the square before $X_2$ on the path between $X_1$ and $X_2$ has priority to $X_2$. If it does, $2m+1$ steps are taken to get to $X_2$ from $X_1$ and back. This would bring the total number of steps for the second visit of $O_3$ to: $visit2(i) + visit2(j) + 2m + 1$ steps. If it does not have priority, a couple more steps are taken to go back and forth on the squares before $X_2$. This adds up to $2m+3$ steps. For this the total number of visits would then be: $visit2(i) + visit2(j) + 2m + 3$ steps. □

To demonstrate the algorithm, we will take a look at an example case. The loop-less maze Squeaky has to escape is shown in Figure 5.4. First we construct the full path Squeaky would walk, which is given by the algorithm that checks if there is any path from the starting point to the exit. Then the main path is extracted from this by picking the lowest priority option on every split in the path.

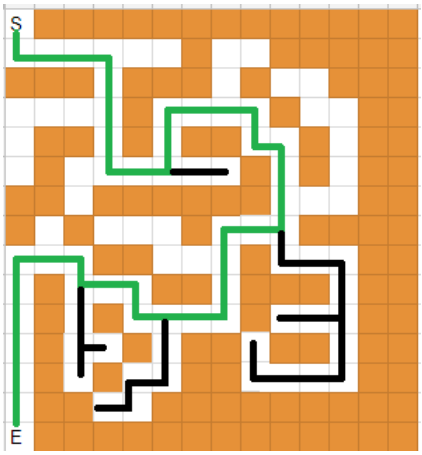



Figure 5.4: The path Squeaky walks through the maze, main path is green.

Then, using the crossroads (see Figure 5.5), we calculate the number of steps it takes to get back to that crossroad and add that all together to the number of steps the main path takes.

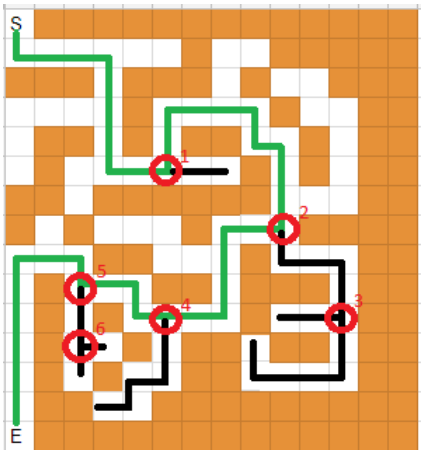



Figure 5.5: Showing the crossroads in the maze.

The number of steps each side path takes is shown in Table 5.1. Crossroad 2 has priority towards the previous square at the outer end and crossroad 3 is visited twice.

| Crossroad | Length ($n$) | Formula used to calculate steps | number of steps |
|---|---|---|---|
| 1 | 2 | $2n+2$ | 6 |
| 2 | 11 | $2n$ | 22 |
| 3 | 2 | $2n+2$ | 6 |
| 4 | 5 | $2n+2$ | 12 |
| 5 | 3 | $2n+2$ | 8 |
| 6 | 1 | $2n+2$ | 4 |

Table 5.1: Calculation for the side paths.

In total these side paths add up to 58 steps, to which the 40 steps of the main path are added for the total of 98 steps that Squeaky takes to leave this maze.

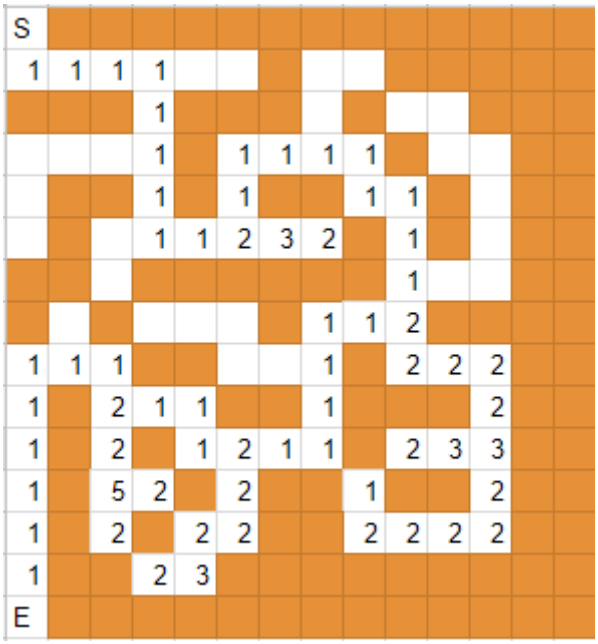



Figure 5.6: Number of visits for each square.

Figure 5.6 shows the number of times he has visited the squares in this maze.

# Chapter 6

# Conclusions

In this thesis we have looked at the game MouseMaze. In this game a player places walls on a $13 \times 13$ grid to construct a maze for the mouse Squeaky to escape. The goal is to design this maze in such a way that he needs as many steps as possible to exit the maze. In particular, mazes have been generated to attempt to find high-scores for lower dimensional mazes and we have looked at the behaviour of $2 \times 2$ loop constructions. Proof has been provided to calculate the amount of steps Squeaky needs to exit any maze with no loops in it.

We were able to generate the optimal maze for any maze up to $7 \times 7$ (which requires 332 steps). Besides these simulations, we have shown that any variant of the $2 \times 2$ loop will at most reverse the mouse once on the first visit and the second visit will lead him to the proper exit of the loop.
The amount of steps Squeaky needs to exit any loop-less mazes can be calculated if the full path he has to walk is given. This reduces the simulation for determining the amount of steps to just checking if a maze is feasible (has a way to the exit), which can then be used for the calculation. Simulating all the steps that Squeaky takes is then no longer required.

## 6.1 Future Work

Further research can be done by looking into different loop structures and trying to better understand those. Once this understanding is there, this could hopefully be used to provide an algorithm that is able to calculate the amount of steps the mouse takes to escape any maze, without restrictions on the amount of loops, which could be used to significantly increase the speed of generating mazes of larger sizes. This would hopefully allow finding those high-scores.

# Bibliography

[CUD18] CUDA, NVIDIA developer. CUDA zone. https://developer.nvidia.com/cuda-zone, 2018. [Online; accessed 12-December-2018].

[EF16] Jessica Enright and John D. Faben. Building a Better Mouse Maze. In *8th International Conference on Fun with Algorithms (FUN 2016)*, volume 49 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:12, 2016.

[Fra09] Tom "CuriousGaming" Fraser. Mouse Maze. https://www.kongregate.com/games/curiousgaming/mouse-maze, 2009. [Online; accessed 12-December-2018].

[MB08] S. Mishra and P. Bande. Maze solving algorithms for micro mouse. In *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*, pages 86–93, Nov 2008.