Building a Better Mouse Maze

Jessica Enright[1] and John D. Faben[2]

1 **University of Stirling, Stirling, UK**
`jae@cs.stir.ac.uk`
2 **Glasgow, United Kingdom**
`jdfaben@gmail.com`

Abstract

Mouse Maze is a Flash game about Squeaky, a mouse who has to navigate a subset of the grid using a simple deterministic rule, which naturally generalises to a game on arbitrary graphs with some interesting chaotic dynamics. We present the results of some evolutionary algorithms which generate graphs which effectively trap Squeaky in the maze for long periods of time, and some theoretical results on how long he can be trapped. We then discuss what would happen to Squeaky if he couldn't count, and present some open problems in the area.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, G.2.1 Combinatorics

Keywords and phrases graph evolutionary genetic algorithm traversal

Digital Object Identifier 10.4230/LIPIcs.FUN.2016.15

1 Introduction

There is a mouse (named Squeaky) who is trapped in a maze. He follows a very simple deterministic rule to decide which square to visit next – always choosing the least visited available neighbour of his current square, and breaking ties by preferring to go down and left over right and up (in that order). This seemingly simple rule leads Squeaky to behave in unpredictable ways, and results in an interesting game designing mazes to trap Squeaky for as long as possible.

In this paper, we first define the natural generalisation of the Mouse Maze game to a problem on general graphs, and then we explore upper and lower bounds on the question of how long Squeaky can be trapped in the maze. We get upper bounds from theoretical considerations, and establish lower bounds using genetic algorithms to search possible maze configurations. We then consider the question of whether Squeaky could still escape if he were not so good at counting, and finally end with some open questions.

1.1 Some history

The implementation of Mouse Maze that sparked our interest is a Flash game hosted on Kongregate[1]. It was written by Tom Fraser, who is Kongregate user CuriousGaming, and is based on the game Micro Mouse, which was written by Jeremy Hammett for PCW Games for the Commodore 64 in 1984 [3].

The game was introduced to the mathematics department at Queen Mary University of London in the summer of 2009 by Michael Brough. He pioneered the approach of using

[1] `http://www.kongregate.com/games/CuriousGaming/mouse-maze`

Figure 1 Squeaky has finished navigating a maze with a single block. The brick walls show squares that Squeaky is blocked from visiting, warmer colours are squares he has visited more often.

genetic algorithms to generate mazes which take a long time for the mouse to navigate. This was taken up by a small group of researchers, including Andrew Drizen, who has held the high score on Kongregate since then. An informal proof of the folklore result in Theorem 7 was known in 2009, but to our knowledge no-one has previously demonstrated an explicit finite upper bound for how long it can take Squeaky to escape a maze.

2 What is mouse maze?

2.1 The original game

To quote the instructions for the original game:

> Draw a maze. Release the mouse. Confound it for as long as you can. Squeaky follows a primitive algorithm; he always chooses the square he has visited least. Squeaky prefers down and right. Squeaky doesn't like ups and lefts.

Drawing a maze is accomplished by removing cells from the 13x13 grid, preventing Squeaky from visiting those cells. This can lead to some surprisingly erratic behaviour from the mouse – for example, on the blank grid from the original game, he will take just 14 moves to exit. However, if just one square is blocked off from his route, it will take him 180 turns to escape, as shown in Figure 1 (we use a slightly different counting convention to the Flash game, which gets 182 turns for this maze).

2.2 Generalised mouse maze

We define a mouse maze problem as follows. All graphs $G = (V, E)$ will be labelled, undirected, simple and loopless.

▶ **Definition 1.** A **maze** $M = (G, s, f, r)$ is tuple consisting of a graph G along with two vertices $\{s, f\} \in v(G)$ where s is the start vertex and f is the exit vertex, and a function for each vertex $v \in V(G), r_v : N(v) \to \mathbb{N}$, which gives Squeaky's preference order on the neighbours of v. We say a maze is **feasible** if s is in the same connected component of G as f.

▶ **Definition 2.** We say that Squeaky **runs** a maze M by executing the following algorithm.

```
v ← s                                   ▷ v is Squeaky's current position, he starts at s
for x ∈ V(G) do
    visits_0(x) ← 0                     ▷ Initially, he has visited each vertex 0 times
end for
t ← 1
while v ≠ f do                          ▷ keep going until he escapes
    for x ∈ V(G) \ {v} do
        visits_t(x) ← visits_{t−1}(x)
    end for
    visits_t(v) ← visits_{t−1}(v) + 1
    next ← ∞
    x ← None                            ▷ x will be the next vertex Squeaky visits
    for w ∈ N(v) do                     ▷ consider these from most preferred to least preferred
        if visits_t(w) < next then
            x ← w
            next ← visits(x)
        end if
    end for
    v ← x                               ▷ update Squeaky's position
    t ← t + 1
end while
```

▶ **Definition 3.** For any maze $M = (G, s, f, r)$, for any vertex $v \in V(G)$, we define $visits(v)$ to be the number of times Squeaky visits v if he starts at s and runs through the maze until he gets to f. While Squeaky is running through the maze, we define $visits_t(v)$ to be the number of times Squeaky has visited v after t steps.

▶ **Definition 4.** For each maze M, $visits(M)$ is defined to be $\sum_{v \in V(G)} visits(v)$

▶ **Example 5.** The original game is played on a subgraph of the 13x13 grid graph, with r_v being such that Squeaky prefers to go down, right, left then up in that order, where possible. Getting a high score is the problem of choosing a subgraph H of the grid such that $visits(H)$ is maximal.

3 Squeaky can always escape

3.1 An upper bound exists

The astute reader may have noticed that we haven't actually established that the function *visits* is well-defined. Here we show that it is. In particular, if M is feasible, then Squeaky will eventually visit f.

The intuition for this is simple – 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 many (perhaps 0) times, but Squeaky would eventually prefer to go there than stay inside the loop, which is a contradiction. We formalise this intuition below, and give a finite upper bound on $visits(M)$ based on the degree structure of G. We will use the following concepts:

- The **distance** between any two vertices $v, w \in G$, denoted $d_G(v, w)$, is the length of the shortest path in G which goes from v to w.

- The **diameter** of a graph G, denoted $diam(G)$, is the longest shortest path in G, that is, the maximum distance between any two vertices in G.
- The **neighbourhood** of a vertex in $V(G)$, denoted $N_G(v)$, or more commonly $N(v)$, if G is implicit from the context, is the set of vertices $\{u \in V(G) | (u, v) \in E(G)\}$.
- The **degree** of a vertex $v \in V(G)$, denoted $deg(v)$, is the number of neighbours of v in G, or $|N_G(v)|$
- The **maximum degree** of a graph G, denoted $\Delta(G)$, is the maximum degree of any vertex in $V(G)$.

We are now ready to prove that Squeaky will always escape any feasible maze (thus justifying our choice of the word 'feasible'). We use the following technical lemma.

▶ **Lemma 6.** *If Squeaky leaves a vertex v for the k^{th} time at iteration t, the following holds,*

$$\forall w \in N(v),\ visits_t(w) \geq \lfloor \frac{k}{deg(v)} \rfloor .$$

Proof. If $k < deg(v)$, this is clearly true, as $\frac{k}{deg(v)} < 1$, so the RHS of the above expression is equal to 0. If $k = deg(v)$, the claim is that every neighbour of v has been visited at least once. But this must be true, as Squeaky has left v $deg(v)$ times, so if there is a neighbour of v which has been visited 0 times, say w, then on one of those occasions Squeaky chose to visit a neighbour of v which had already been visited once in preference to w, but this is a contradiction.

The expression on the right only increases when $deg(v)$ divides k, so if it is true for all multiples of $deg(v)$ then it must also be true for all other k. Assume that it is true for all $j < k$, and that k is a multiple of $deg(v)$, *i.e.* $\exists q \in \mathbb{N}$ such that $k = q * deg(v)$. Now, by induction, every neighbour of v had been visited at least $(q-1)$ times when Squeaky had left v $(q-1)deg(v)$ times. But Squeaky has made $deg(v)$ more visits to v since then, and has chosen neighbours to visit exactly $deg(v)$ more times. Assume that there is still some vertex $w \in N(v)$ which only been visited $q-1$ times, then as in the previous paragraph, Squeaky has at some point chosen to visit a neighbour of v which had already been visited q times in preference to w, which is a contradiction. ◀

This lemma will form the basis for both of the next two proofs. First, we prove that Squeaky will eventually escape the maze, then we will give a bound on how long it takes him. What the lemma essentially says is that Squeaky can never visit a vertex much more often than he visits all of its neighbours. Using this idea, we show that if Squeaky visits any vertex more than a certain number of times, then he must also eventually visit f, since there is a chain of neighbours which connects f to this vertex.

▶ **Theorem 7.** *For any maze $M = (G, s, f, r)$, if M is feasible, then when Squeaky runs through M, he will eventually reach f.*

Proof. In the below, we assume that $\Delta(G) > 1$ if not, G consists of a collection of isolated edges and vertices, and the result is trivial.

If Squeaky never visits f, then eventually there is some vertex v which is visited $\Delta(G)^{2diam(G)}$ times. We will show that if this is the case, then Squeaky has in fact visited f, a contradiction. Let t be the time step immediately after Squeaky leaves v this many times.

First, consider neighbours of v. By Lemma 6,

$$\forall w \in N(v) visits_t(w) \geq \lfloor \frac{\Delta(G)^{2diam(G)}}{deg(v)} \rfloor$$

and since $deg(v) \leq \Delta(G)$,

$$visits_t(w) \geq \Delta(G)^{2diam(G)-1}.$$

But then it is certainly the case that Squeaky has left w at least $\Delta(G)^{2(diam(G)-1)}$ times, since $\Delta(G) > 1$, so this is smaller than the RHS of the above expression. By a second application of the Lemma, if we let z be a vertex at distance 2 from v, then z is adjacent to some vertex at distance 1 from v, and we have,

$$visits_t(z) \geq \Delta(G)^{2(diam(G)-1)-1}$$

and in particular, Squeaky has left z at least $\Delta(G)^{2(diam(G)-2)}$ times, since $\Delta(G) > 1$, so this is less than $\Delta(G)^{2(diam(G)-1)-1)}$.

By repeated application of the lemma, we get that for any vertex at distance k from v,

$$visits_t(x) \geq \Delta(G)^{2(diam(G)-k)-1}.$$

But by definition $d_G(v, f) < diam(G)$, so in particular,

$$visits_t(f) \geq \Delta(G)^{2(diam(G)-diam(G))-1} \geq 1$$

and Squeaky has visited every vertex in G. ◀

3.2 A finite upper bound

Note that in above proof it seems that $\Delta(G)^{2diam(G)}$ is in some sense bigger than the number of visits we needed to ensure that Squeaky visited every vertex. Something closer to $\Delta(G)^{diam(G)}$ should suffice. Using the fact that we know Squeaky will eventually escape, we prove an upper bound on how long it takes him which is indeed of the order $\Delta(G)^{diam(G)}$. Note that we use the result of Theorem 7 in the very first step of this proof, where we assume that Squeaky eventually visits f.

▶ **Theorem 8.** *The total number of turns Squeaky spends in any maze M on a graph G before he visits every vertex is bounded above by,*

$$|V(G)| \sum_{i=1}^{diam(G)} \Delta(G)^i.$$

Proof. Since Squeaky stops when he visits f for the first time, we know that $visits(f) = 1$. We will show that no vertex in $V(G)$ was visited more than $\sum_{i=1}^{diam(G)} \Delta(G)^i$ times.

First consider vertices in $v \in N(f)$. We claim that $visits(v) \leq \Delta(G)$. The logic here is similar to that in the proof of Lemma 6. Assume that Squeaky did visit v more than $deg(v)$ times, and consider the situation just after he left v for the $deg(v)^{th}$ time, at time t. At this time, $visits_t(f)$ was 0, by assumption (since Squeaky got back to v again, in order for $visits(v) > deg(v)$), but then at some time t_1 in those $deg(v)$ occasions he must have chosen to visit a vertex $w \in N(v)$ with $visits_{t_1}(w) \geq 1$ in preference to visiting f, but this is a contradiction, so $visits(v) \leq deg(v) \leq \Delta(G)$.

Now we consider the vertices at distance 2 from f. Let w be such a vertex. By Lemma 6,

applying the result in the previous paragraph, we have,

$$\Delta(G) \geq \lfloor \frac{visits(w)}{deg(w)} \rfloor \tag{1}$$

$$\geq \lfloor \frac{visits(w)}{\Delta(G)} \rfloor \tag{2}$$

$$\Delta(G) + 1 > \frac{visits(w)}{\Delta(G)} \tag{3}$$

$$\Delta(G)(\Delta(G) + 1) > visits(w) \tag{4}$$

Now assume for all $j < k$, and all vertices $x \in V(G)$ such that $d(x, f) \leq j$ we have $\sum_{i=1}^{j} \Delta(G)^i > visits(x)$. We will show that for all vertices y at distance k from f, we have $\sum_{i=1}^{k} \Delta(G) > visits(y)$. Any vertex at distance k from f is adjacent to some vertex at distance $k-1$ from f, let x be such a vertex, then by our induction hypothesis, we have, $\sum_{i=1}^{k-1} \Delta(G)^i > visits(x)$. Let t be the last time that Squeaky left y, then $visits(y) = visits_t(y)$, but by Lemma 6, we have $visits_t(x) \geq \lfloor \frac{visits_t(y)}{deg(y)} \rfloor$, and since $visits(x) \geq visits_t(x)$, combining these inequalities gives:

$$\sum_{i=1}^{k-1} \Delta(G)^i > \lfloor \frac{visits(y)}{deg(y)} \rfloor \tag{5}$$

$$\geq \lfloor \frac{visits(y)}{\Delta(G)} \rfloor \tag{6}$$

$$\sum_{i=1}^{k-1} \Delta(G)^i + 1 > \frac{visits(y)}{\Delta(G)} \tag{7}$$

$$\Delta(G)(\sum_{i=1}^{k-1} \Delta(G)^i + 1) > visits(y) \tag{8}$$

$$\sum_{i=1}^{k} \Delta(G)^i > visits(y) \tag{9}$$

But the farthest any vertex can be from f in G is $diam(G)$, so for every vertex $v \in V(G)$, we have

$$\sum_{i=1}^{diam(G)} \Delta(G)^i > visits(v) \tag{10}$$

And since $visits(M)$ is the sum over all vertices in $V(G)$ of $visits(v)$, and every vertex satisfies the above inequality,

$$|V(G)| \sum_{i=1}^{diam(G)} \Delta(G)^i > visits(M) \tag{11}$$

◀

4 Lower bounds: the search for a high score

The ultimate question in mouse maze is: what is the highest achievable score? That is, given certain constraints on (G, s, f, r), what is the maximum value of $visits(M)$ where M is a maze on (G, s, f, r). The original game constrains G to be a subgraph of the 13x13 grid, with

s being in the top row, f being in the bottom row, and r being a preference for down, right, left, up in that order. In the following we restrict our attention to games on the square grid with this preference function. We present a brief summary of some types of algorithm that have been used in the search for better mouse mazes, with a discussion of the effectiveness of these algorithms on smaller grids, where the exact solution can be computed. Finally we present the best known maze for the original problem, which was found using a genetic algorithm and some local optimisation.

4.1 Flipping squares in a grid

Since we will only be considering mazes where G is a subset of some 2-dimensional grid graph, in the following it will often be easier to refer to 'square of the grid' rather than 'vertices of G'. We will also talk about 'flipping' squares.

▶ **Definition 9.** A maze M is obtained from another maze M_0 by **flipping** square (i, j) if M_0 and M_1 are identical except for the fact that the square of the grid graph corresponding to the $(i, j)^{th}$ coordinate is either in $V(M_0)$ and not in $V(M_1)$ or vice versa. We say that M_0 and M_1 are **neighbours**. We will also define the Hamming distance between two mazes which are subgraphs of a given grid. This is the number of vertices that need to be flipped in one to reach the other. This corresponds exactly to the Hamming distance between the natural matrix representation of subgraphs of the grid.

4.2 Some algorithms

We present below five types of algorithm which we have used to generate mazes to trap Squeaky. For a summary of these techniques, the reader is referred to [4].

In **random search**, we simply generate feasible mazes at random and evaluate them. In practice, we limit random search to generating mazes by considering each grid square independently and deciding whether to include it with a fixed probability p. This can be used as a baseline for other methods.

In **hill-climbing**, we pick a maze at random (using the same method for picking random mazes we used in random search), evaluate that maze, then evaluate all of its neighbours. If any neighbour has a higher value for the *visits* function, we then evaluate all of that maze's neighbours, and so on until there are no further improvements in the evaluation function to be found.

In **simulated annealing** we again start with a random maze, M. We pick one of its neighbours, N uniformly at random, and evaluate $visits(N)$ (we keep picking until N is feasible). If $visits(N) > visits(M)$, then we move to N and recurse. If $visits(N) < visits(M)$, then we still move to N with some probability dependent on the difference, which reduces as the algorithm runs (a process known as cooling). This allows simulated annealing to escape local maxima, in contrast to hill-climbing.

In an **evolutionary search**, we pick a random subset of the search space – that is a random group of mazes, and we evaluate all of them. We then take the top n of these to be 'parents' for the next generation, and mutate these parents slightly to produce a new generation. In particular, to generate a member of the new generation, we pick a member of the list of parents at random, we then decide for each vertex independently whether to flip this with a fixed probability, and the resultant maze is placed into the next generation.

In a **genetic search**, more than one parent is combined to produce each child. There are a variety of ways in which this combination can be achieved. We discuss these a little more in Section 4.4, but the simplest, and the on which has in practice proved most effective

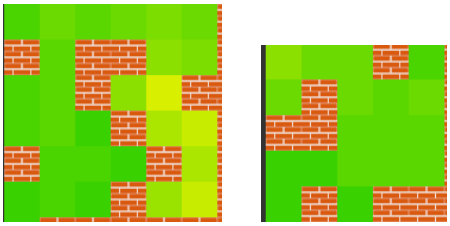

Figure 2 The best possible mazes on the 6x6 and 5x5 grids, which take 115 turns and 54 turns to navigate respectively. Squeaky starts in the top left and exits in the bottom left. The brick walls show squares that Squeaky is blocked from visiting, lighter colours are squares he has visited more often.

is **uniform crossover**. We evaluate each member of the current population, and take the top n to be parents. We then select two parents uniformly at random (so both parents can be the same). For each vertex in the grid, we pick a parent at random and include that vertex in the child if it is included in that parent. We may then apply some mutation to the children, in the same way that we did for the evolutionary algorithm.

4.3 Some smaller mazes

For small grids, it is possible to completely enumerate all mazes (possibly with some restrictions on s and f). We present the optimal mazes for a 5 by 5 and 6 by 6 grids in Figure 2. These were obtained by a simple exhaustive search on all possible feasible mazes. The last of these took approximately 60 days of parallel computation, so extending this analysis using the same method to a 7 by 7 grid is prohibitive, as there are approximately 10,000 times as many feasible 7 by 7 grids as feasible 6 by 6 grids. 7 by 7 may be achievable with more aggressive pruning of the search space to identify feasible mazes, and faster maze evaluation implementations. Finding optimal mazes for 4 by 4 and 3 by 3 grids can be done by hand, and is left as an exercise for the interested reader.

We ran each of the algorithms described in Section 4.2 to search for mazes on the 6 by 6 grid until they had evaluated 1 million mazes each. Monte Carlo simulation shows that of the 2^{34} possible subgraphs of the 6x6 grid including the start and the finish vertex, approximately 4% are feasible, so each of these algorithms has searched around 0.01% of the total search space – much more than we could reasonably hope to achieve for a 13 by 13 grid. The table below presents the results of these searches. Note that the best possible score, represented by the grid in Figure 2 is 115.

Search method	Highest *visits*	Comments
Random Search	67	We used p=0.25, the fraction of blocked squares in the optimal 6x6 grid. We searched until 1 million feasible mazes were evaluated.
Hill Climbing	87	We started with a random maze with $p = 0.25$. The search was restarted whenever a local optimum was reached, and continued until 1 million mazes had been evaluated.
Simulated Annealing	109	We evaluated 1 million mazes, using a simple linear cooling function.
Evolutionary Algorithm	109	Using a mutation rate of 0.06, and picking 10 parents from each generation of 100 mazes, we ran 10,000 iterations, evaluating 1 million mazes.
Genetic Algorithm	111	Using a mutation rate of 0.06 and uniform crossover, picking 10 parents from each of 100 mazes, we ran 10,000 iterations, evaluating 1 million mazes.

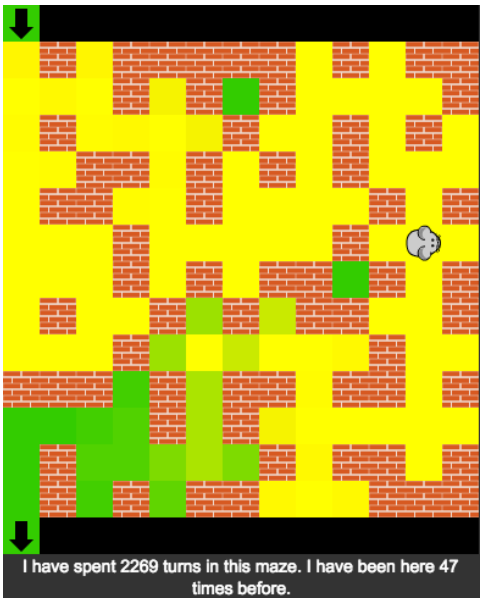

Figure 3 The best known maze, which takes Squeaky 36314 turns to navigate, shown as Squeaky is running through it. Squeaky starts in the top left and exits in the bottom left. The brick walls show squares that Squeaky is blocked from visiting, lighter colours are squares he has visited more often.

4.4 The Genetic Algorithm

The results in Section 4.3 are only suggestive, but they do indicate that an evolutionary/-genetic approach may be suitable for finding good mazes to trap Squeaky. This is backed up by previous experience. Indeed, M_{34188}, which has held the high score on Kongregate since 2009 was generated by Andrew Drizen using a simple genetic algorithm with uniform crossover [2]. Despite having some success on the 6 by 6 grid, simulated annealing has proved ineffective on 13 by 13 grids. After searching 10 million mazes, we failed to find one with a visits number exceeding 2000.

It may seem that the uniform crossover operator doesn't do a good job of preserving the structure in the parents, which is usually a feature of good crossover operators for evolutionary algorithms [4]. However, various crossover operators which do seem more likely to have this property were tried, including point crossover (choosing everything above and to the left of a random point from one parent, and everything else from the other); multi-point crossover (essentially point crossover, but with multiple points); line crossover (choosing each row or column of the offspring to come entirely from one or other parent) and an operator in which we chose 2x2 grids from the parents at random. All of these were abandoned as they didn't produce any promising solutions (defined as >10,000 visits) after evaluation of over 10 million mazes in each case. In contrast, we were able to produce a large number of mazes with $>$ 20,000 visits using uniform crossover.

4.5 The Best Known Maze

In our search for better solutions, we were able to find several mazes which exceeded 25,000 visits, but we have failed to identify a more promising area of the search space than that identified by Andrew Drizen in 2009. However, we have managed by locally searching the

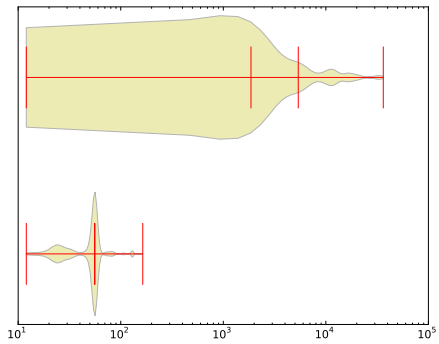

Figure 4 A violin plot, on a log scale, showing the distribution of the visit numbers of the neighbours of M_{36314} (top) compared to a similar plot for a random maze feasible with the same number of blocked squares (bottom).

area around that maze to produce a small improvement on the lower bound – we present M_{36314}, depicted in Figure 3, with $visits(M_{36314}) = 36314$.

M_{36314} is very close to the previous high scoring maze (Hamming distance = 10). It was found by the authors using that maze as a starting point. We used a simple evolutionary algorithm starting from a population consisting entirely of that maze with a very low mutation rate and no crossover, an approach that had proved successful in exploiting the results of genetic algorithms in our own searches. This produced a maze which took Squeaky 36118 turns to escape. We then did an exhaustive search of the local neighbourhood of this maze – searching all mazes within Hamming distance 3, and successfully located M_{36314}. There is no maze within Hamming distance 3 of M_{36314} with a higher number of visits than 36314.

4.6 Local Search is limited

The chart in Figure 4 summarises $visits(M)$ for all feasible mazes which are within a Hamming distance of 2 of the best maze found so far, M_{36314}, compared with the mazes which are within Hamming distance 2 of a random feasible subgraph of the grid with the same number of vertices.

Note that while several mazes close to M_{36314} do have high visit numbers, there are also mazes which are directly adjacent to our optimal maze which have a tiny number of visits – in particular, there is one maze N such that $visits(N) = 14$, which is the minimum possible for a maze on the 13 by 13 grid, but N is adjacent to M_{36314} (N is the maze obtained by removing the brick which blocks Squeaky from heading straight to the exit). We can contrast this with the neighbours of a randomly chosen feasible maze with the same number of vertices removed from the grid as M_{36314}. This maze only had a visit number of 56. Some of its neighbours have visit numbers higher than this, but none exceed 200.

5 Mice who cannot count properly

We've shown that Squeaky can escape any maze, even on an arbitrary graph, but to do so, he might need to count to a very large number. Typical mice probably cannot count that high, though there is some evidence that at least some mice can count to 40 [1]. We therefore consider k-modular mice which count mod k. Apart from their counting, these mice have

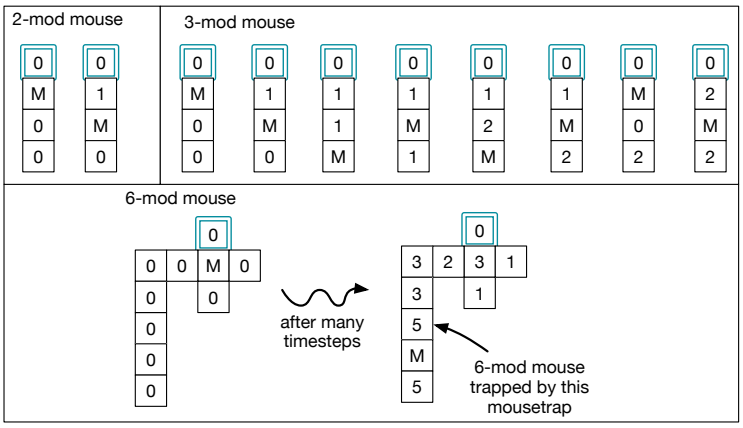

Figure 5 Initial maze states and eventual resulting mousetraps for a 2-mod mouse (top left), a 3-mod mouse (top right), and a 6-mod mouse (bottom). All intermediate timesteps are shown for the 2- and 3-mode mice, for both of which the mousetrap vertex set consists only of second vertex from the top. In all three the exit is the double-outlined box, the M is the mouse position, and the numbers indicate number of visits at a node.

the same directional preferences as Squeaky. We show that, unlike Squeaky, these types of mice can be trapped in a maze – counting is an important skill! We give examples of some small mazes that can trap k-mod mice for small values of k.

In our constructions, we use the idea of a *mousetrap*. A set of vertices V_c is a mousetrap of a k-mod mouse if at time t:

- for every vertex $v \in V_c$, $visit_t(v) = k - 1$
- in $G[V \backslash V_c])$, the mouse is not in the same component as the exit
- at every vertex $u \in N(V_c)$ that the mouse can reach without passing through a member of V_c, the mouse directionally prefers some neighbour of u that is not in V_c over its neighbours that are in V_c.

▶ **Lemma 10.** *If at some time t there is a mousetrap for a k-mod mouse, the mouse will never escape.*

Proof. It suffices to show that the mouse will never visit a vertex of the mousetrap, as all paths from the mouse to the exit go through the mousetrap. We proceed by contradiction. Consider the first time t' after t at which the mouse visits a member v of V_c. Then at $t' - 1$, the mouse was at a neighbour u of v that was reachable from the mouse's position at t without passing through V_c. By the definition of a mousetrap, we know that the mouse directionally prefers some neighbour $w \in N(v) \backslash V_c$ to v. Because $visits_{t'}(v)$ and $visits_{t'-1}(v)$ are the maximum possible value, and v is not directionally preferred, the mouse should not visit v at t', a contradiction. ◀

We can trap 2-mod mice and 3-mod mice the same single path, as in Figure 5. Note that the construction for a 3-mod mousetrap depends on the fact that $2 + 1 < 2 \pmod 3$, so that the visit number of a vertex can go down). We have investigated similar maze constructions for small values of k, up to $k = 6$ (Figure 5) We can more easily trap a general k-mod mouse in a graph that is not a subgraph of the grid.

▶ **Lemma 11.** *We can trap a k-mod mouse in a maze for which the graph consists of a tree with a single vertex of degree greater than two, and that vertex has degree $k + 1$.*

Proof. We define maze (G, s, f, r) as follows: Let G be a tree with a single vertex s of degree greater than two. Vertex s has k neighbours $v_1...v_k$, and one of those neighbours, v_{k-1}, has a neighbour w. All other neighbours $v_1...v_{k-2}$, and v_k are leaves. We need only define our directional preference function r for non-leaves (because at leaves Squeaky has no choice of where to go, so his preferences don't matter). The only non-leaves in this graph are s and v_{k-1}. First, we give the preference function at s: r_s is a function from $v_1...v_k$ to $\mathbb{N}$, where $r_s(v_i) = i, 1 \leq i \leq k$. The directional preference function $r_{v_{k-1}}$ maps from $\{s, w\}$, with $r_{v_{k-1}}(s) = 2$ and $r_{v_{k-1}}(w) = 1$ – that is, Squeaky prefers w to s at v_{k-1}. Finally, let f be v_k.

So, informally, Squeaky is in a star with a short path in place of one of the leaves, where he starts at the central vertex. The exit is in his least-preferred direction from the central vertex, and the short path is in his second least-favourite direction. Once he is on the short path, he prefers going away from the central vertex.

It remains to show that he will become trapped. First, observe that Squeaky will not visit the exit until he has visited all other neighbours of s at least once. In fact, Squeaky, starting at s, will visit v_1, then return to s, then v_2, then return to s, and so forth. So, when Squeaky is visiting v_i, where $i \leq k-1$ for the first time, he has already visited each of v_j, where $j < i$ exactly once: and has visited (and left) s exactly i times. Then when Squeaky visits v_{k-1}, $visits(s) = k-1$ (the maximum possible value), all paths from v_{k-1} to f pass through s, and Squeaky directionally prefers w to s. Therefore $\{s\}$ is a mousetrap at this timestep. ◀

6 Open questions

There is one obvious question which we leave open: what is the high score? We provide the maze with the highest known number of visits, but there is still a lot of room between this and the upper bound in Theorem 8. One could also consider the crossed paths version of the game – rather than maximising the amount of time Squeaky spends in the maze, maximise the number of times he visits a single vertex.

Other interesting questions include the generation of new mazes for Mouse Maze 2. This is a variant of the original game in which edges, rather than vertices are deleted from the 13x13 grid. In general the related question would be: how do we trap Squeaky for as long as possible in a maze given certain constraints on the graph, preference function and start and finish vertices?

Acknowledgements. We want to thank Andrew Drizen for generously sharing the maze which had led to the previous high score with us, Michael Brough for bringing the game to our attention all those years ago, and Tom Fraser for his implementation, which made it available for all of us to play.

References

1 Bilgehan Çavdaroğlu and Fuat Balcı. Mice can count and optimize count-based decisions. *Psychonomic Bulletin & Review*, pages 1–6, 2015. doi:10.3758/s13423-015-0957-6.

2 Andrew Drizen. private communication, 2016.

3 Jane Green, editor. *Personal Computer World: Best of PCW Software for the Commodore 64*. Century Communications, 1984.

4 Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2000.