SLITHERIN ROUND

KRISTEN MAHONEY AND KEVIN HARMER

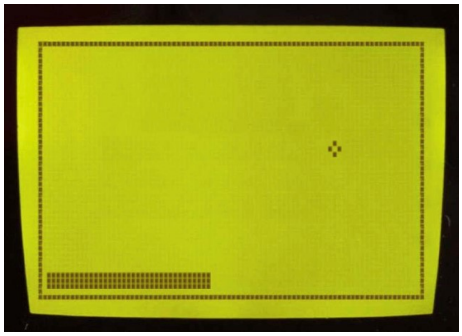
1. Abstract

Video games have been a source of entertainment and challenge for people of all ages. The game Snake that originated on the Nokia cell phone in the 1990s is a game that can capture the attention of players for hours. As the snake grows longer, random food placement and navigating around the field makes it much harder for a player to win. The challenge of winning on such a fairly simple game makes one wonder- is there a guaranteed way to win and have the snake's body fill the entire board every time? The goal of this paper is to determine a method that will win at a game of Snake every time the game is played. By coding our own version of the game, we will analyze the success rate of different snake movement algorithms on various grid sizes to determine if there is a guaranteed method to win. By the end, we will be able to describe which methods are most successful and win the fastest.

2. Introduction

The game Snake, considered the first major mobile game, was created in 1995 to be included in the features of the Nokia 6110 released in 1997. The creator is Taneli Armanto, a Finnish software engineer, who was tasked with creating "some cool little games" on the new phone [4]. He was a fan of board games and Tetris, another early cell phone game. When planning out how the game would work, he took inspiration from theses games. [4]. Constraints of the Nokia phone included a very small screen, a few keys controlling the game, and a small amount of available programming space. Armanto found that a snake was the perfect solution for the game they needed. Other games with a similar concept to Snake had existed before this one, such as the 1976 arcade game Blockade, Nibbler, and Rattler Race, but Snake was the first to be built into a cell phone without downloads, data, or updates needed [4].

Figure 1. Sample Starting Point

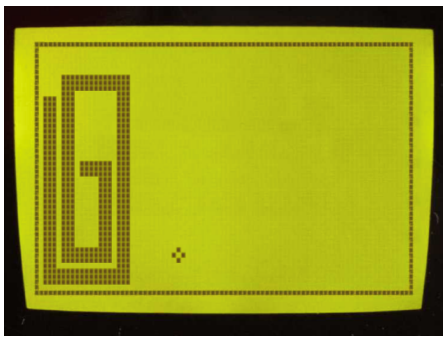


The main goal of the game is pretty direct. Starting as a small snake of one pixel, the player moves the snake around the playing field to eat a little dot of "food." Starting with a length of one, the snake begins its journey from the the bottom left part of the field. With each piece of food the snake eats, its body gets longer by one pixel. The food, additionally, is generated randomly to an open space at the start of the game and directly after, the food is eaten by the snake. This starting setup is shown in Figure 1.

Date: May 7, 2021.

From the starting point, the snake moves automatically in one direction until the players changes the direction. Movement in the game is very simple. There are only four possible directions: up, down, right and left. For the purpose of this project, these movements can be notated in vector form with (0,1) representing one movement upward, (0,-1) one downward, (-1,0) one left, and (1,0) one right. Because you are unable to move backwards, only three possible moves are available at all times. In the game, there is a set timer which gives the player a set amount of time to make their move. If they do not move in the time period, the snake automatically moves in the same direction it had been traveling from the previous direction choice (or initial direction if the game just began). The movement frequency can be at any rate, but most games are played between 1 move every 0.05 seconds to 1 move every 0.7 seconds. The snake, as a result, could move too fast for the player to react. Since the snake is constantly moving, it may end up crashing into its own body or a wall (the edge of the game field). If this happens (like in Figure 2), the player loses and has to start over. The main difficulty, however, does not come from getting to that first food. After the snake eats the food, its length increases, a new dot of food appears, and the process repeats. The player may find the staying alive is much harder when the snake length is five or six times its starting length (also in Figure 2) because it is much harder to avoid the snake's body.

FIGURE 2. Snake wrapping itself



However, if the player avoids hitting the snake's body and the surrounding walls, they have a chance of winning the game. Eventually, the player can fill the entire field with the snake's body. Once this happens, no more food can spawn, and player wins the game.

The major problem is extremely hard to avoid the snake's body or sides of the field later in the game. This paper is to analyze the difficulty and determine if it is possible. Once we found that winning was possible, we expanded the methods to generalized grids and combined existing methods for faster wins.

After describing the existing work in the field, this paper will go into analysis of each method. We begin by discussing the shortest path and analyzing its effectiveness across a 5x7 and 6x6 grid along with the original 20x11. After the shortest path method, we move into Hamiltonian Cycles for the 20x11 and 6x6 grid. After seeing its winning success, we moved into the amount of moves the algorithm takes to search for an improvement. Also, we could not construct a Hamiltonian Cycle for the 5x7 grid, so we analyzed that grid using the Almighty Move. After noticing the difficulty in constructing Hamiltonian cycles for odd by odd grids, we found a conjecture that predicts the existence of Hamiltonian cycles for grids with even dimensions. Moreover, we improved the Almighty Move to produce better winning results along with a similar conjecture for an Almighty Move's existence. Furthermore, we finished with a Sub-Hamiltonian Cycle which outperforms the Hamiltonian on the 20x11 grid.

3. Problem formulation

It takes an extreme amount of precision, skill, and quick thinking to win at snake, but what if there was a way that could always guarantee a win? In our paper, we aim to analyze the game of snake to find an algorithm that wins consistently. Although there are documented ways to win, their successes depend on many of the game characteristics. Our goal is to expand on these methods and determine a method that wins on fields with different dimensions, initial snake sizes, and starting points. We also will evaluate how long the games take depending on the method and fluctuation in dimensions.

This problem is interesting because we are analyzing algorithms to win a computer game that was created before we were born. Although some people have mastered the game, it would be exciting to see if they could compete after changing the game's components. By varying the dimensions and other game characteristics, we can analyze the algorithms and determine how reliant they are on the generalized snake setup.

4. Related work

There is not much research done on the optimization of the game Snake, but there are a few papers that seek to find the best strategy of movement to win at the game. Work by Greg Surma focuses on different path strategies that can be implemented by a player or algorithm using domain specific solvers, which in this case is a bounded rectangle domain [5]. He tested several different methods of movement.

(1) The Shortest Path - Breadth First Search first moves the head of the snake horizontally toward the food until the head is in the same column as the food. It will then move vertically until the snake eats the fruit, and the algorithm then starts over. If the snake cannot move horizontally, it will skip to the vertical portion then repeat the algorithm. This is successful until the body of the snake cuts off the shortest path to the food.

(2) The Shortest Path - Depth First Search explores the areas close to the snake head before making a direct path to the food. Although there is no direct documentation on this, it appears the snake explores the nodes around it before moving a long distance. This does not work as well as the Breadth First Search because the snake tangles with its body much easier after short movements.

(3) The Longest Path takes the longest possible route between the head of the snake's initial position in the graph and where the food is located on the graph. It moves into every other spot before getting the food. The purpose of this method is to get the longest time without retracing steps, but it can lead to the snake running into its body at times, and therefore losing.

(4) The Hamiltonian cycle method has the snake follow a path that hits each pixel in the board exactly once[1]. It leads the snake head to the fruit and the next pixel is where the snake head first started. It therefore follows the same path of movements as the placement of the food changes. It differs from the Longest Path method because the Longest Path does not have a repeated, unchanging path it follows with each new food placement. The Hamiltonian type of cycle is to ensure that the snake is able to avoid hitting itself after eating each piece of food. Essentially the Hamiltonian method ensures an "escape" route for the snake's head where it won't hit its body or a wall while the Longest Path method does not.

(5) The Almighty Move is similar to the Hamiltonian, but with a cycle that hits vertices more than once. It is not as successful as the Hamiltonian due to the extra points which are not hit by the Almighty Move. The snake misses those points and hits its body on the way back to the open spot. Unlike the Hamiltonian, the Almighty Move can be used on all grid sizes, starting points and lengths. The Almighty Move, consequently, serves as a substitute for when no Hamiltonian cycle can be constructed.[3]

Surma found that the Hamiltonian cycle was successful the majority of the time, but the success depends on the starting position and direction of the snake. The dimensions also play a large factor. The Hamiltonian cycle can be carried out in different styles, but is best summarized by covering each spot in the game once before it reaches the food pixel it will eat. His technique gave the snake the rules of the game to follow within the algorithm [5]. Another way to find an algorithm that wins at Snake is to train the snake to learn the rules as it goes. This type of genetic programming is what Tobin Ehlis used to create a winning algorithm.

Ehlis used two different tree-based methods where the tree is the position of the snake head and the leaves of the trees provide possible movement options for the snake (up, down, left, right) [2]. The first focuses on the proximity of the food to the snake and gives the snake the option to move forward, left, or right. The success of this method was determined by the maximum number of food pieces that could be eaten, the number of food pieces actually eaten, and the distance from the snake to the food if no food was eaten. The second method factored in whether an obstacle was located two spaces in front of the snake and determined the success in the same way. [2].

5. Our Method

We will begin by evaluating methods mentioned in the Related Works such as the shortest distance model, the Hamiltonian and the Almighty Move model to see which become statistically successful and where there might be areas for improvement in the approach to winning the game. Due to our limited computer science abilities, we will avoid the methods with artificial intelligence. The methods listed are also significant because they each have components of the new method we are developing, called the Sub-Hamiltonian.

Additionally, we will begin with a 20x11 playing field, the original dimensions of the game, where the snake starts in the bottom left corner with a length of 1. Furthermore, we have developed a code using Python to model the game Snake and simulate our algorithms, and analyzed the data using R programming.

In the simulations, we evaluate the success of each algorithm at any dimension, starting length, and starting location. Specifically, we are looking to find the average score (which is snake length), variation of score, score residuals, and time statistics. From these statistics, especially average scores, we can develop models that can predict the success of each algorithm.

5.1. **Coding the Project.**

Our approach to model snake started with coding the game. We broke the coding into several different parts to efficiently organize the components. This would prove valuable when examining different types of data (algorithm, dimensions, score, average time, etc.).

The first part of the code was to establish the playing field of the game. Specifically, we used a numerical two dimensional array with numerical values representing the game's

characteristics. Three was used for the walls of the field, 1 was used for the snake's body, 2 was used for the food, and 0 was used for the empty spaces of the field. Additionally, we created a food function to randomly place the food on any open spot in the field (numerically 0). The coding to this part was rather trivial, so we will not include it in the paper. We will also wait to show the product of the section (the playing field) until the graphics section when it can be completely understood.

The second part of the code was establishing game logistics. The game logistics are defined as the snake's movement. The basis of the movement function was to translate the snake in any direction (up, right, down, left), just like the game. Embedded in the function is a win condition, lose condition, and food replacement command to direct the user when something happens. Food replacement was a variation of the food function from part 1. The lose condition alerts the user that they hit the body or surrounding wall and resets the field. The win condition occurs when the entire board is the snake without any open space or food. The effect of this function will be shown program section.

The third part of the code was graphics. Although this is the simplest part of the code, it appears consistently throughout the following sections. The premise of the graphics section was to provide a visual of the field described in part 1. We did this through a color map, which is shown in Figure 3.

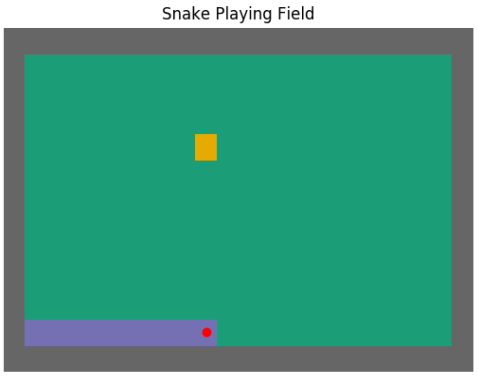


FIGURE 3. This is the starting field with a 20x11 grid and snake length of 7. The snake's body is blue, open space if green, and the food is yellow. The red dot indicates the snake's head.

After we constructed the graphics and game logistics, it was time to develop the actual programming of the code. Part 4 consists of the actual game programming we use to play the game. Due to the length of our complete code, we only show part 4 in the paper, which is located in the Appendix. We show part four in particular because it shows all the main components working together to present the game, including movements, graphics and time restraints. In this part, we can play the game exactly like the phone game. Although this is not the algorithms we are testing, it provides an actual game to run the simulations on, which gets us into part 5.

Part 5 of the code consists of all the simulations. Although this part of the code is most complex, it was relatively easy to create from the basis of the first few parts. For our simulations, we created new script files to run each algorithm. The shortest path method

worked objectively and was very easy to evaluate over different dimensions and starting lengths. The Hamiltonian and Almighty Move, on the other hand, require specific codes from starting conditions, making them much more difficult to evaluate across different parameters. Nonetheless, we ran 10,000 simulations for the Hamiltonian and Shortest Path DFS on 20x11 and 6x6 grids (with different starting points and lengths), then created an Almighty Move and Shortest Path simulation for a 5x7 grid.

6. Preliminary Analysis

First we analyzed the Shortest Path, Hamiltonian, and Almighty Move methods from the Related Works to find which method worked best and where there was room for improvement that our own method of winning is aiming to fix.

6.1. **Shortest Path Simulation.** We will start with the Shortest Path- Depth First Search on a 20x11 grid and a starting length of 1 (and therefore the score equals 1). The snake starts in the bottom left corner of the field. The starting location and starting length had small affects on the ending score, so they will be excluded from the analysis. The figures below show the distribution of the moves and scores for the 10,000 simulations we ran.

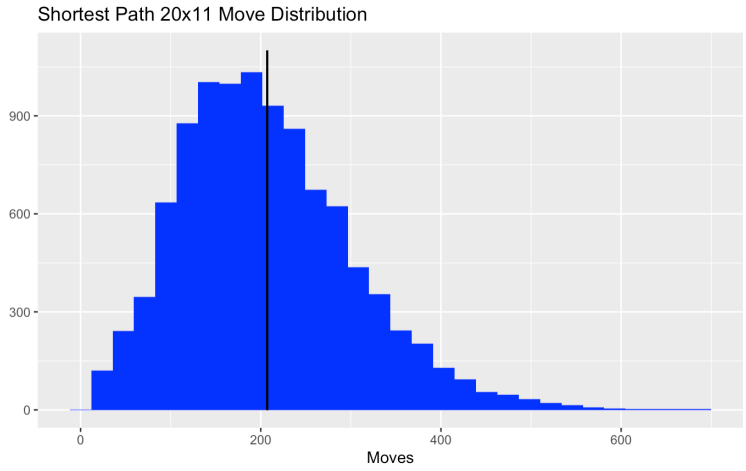


Figure 4. Distribution of Shortest Path Moves on 20x11 Grid

In Figure 4 the moves have a positive skew with a mean of 207 moves and looking at the score distribution, it can be seen that a winning game was never achieved with these parameters. As seen in 5, the maximum score was only 54 out of a possible 220.

6.2. **Hamiltonian Simulation.** We then ran the game using the Hamiltonian method on a 20x11 grid, with a starting snake length of 1 (score = 1). Different starting locations had no effect on the results, while a higher starting length generally corresponded to lower moves, but identical scores. With these specific conditions, we are able to construct the Hamiltonian pattern in Figure 6. The results of the simulations are then shown in Figure 7. The snake again starts in the bottom left corner of the field. We ran 10,000 simulations of these parameters, and every trial produced a winning result of 220. So, for these initial conditions, the Hamiltonian works. In addition to the scores, we can examine the time required to win. The distribution for the number of moves over the 10,000 trials is shown

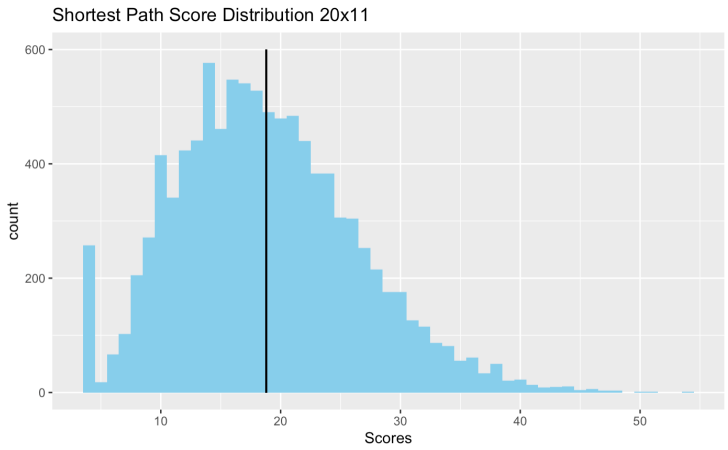


FIGURE 5. Distribution of Shortest Path Scores on 20x11 Grid

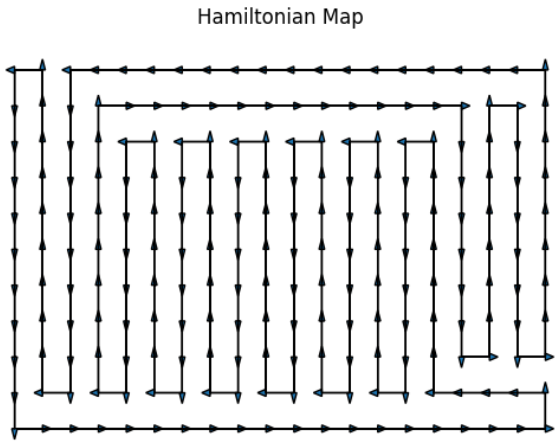


FIGURE 6. General layout of movements for one Hamiltonian Cycle on a 20x11 grid.

by Figure 7. The average number of moves needed to win is 12,155, as seen in the Figure 7. The number of moves is directly proportional to the total time required to win the game using the Hamiltonian Method. If a move occurs every 0.3 seconds (taken from program code; 0.25 time frequency + 0.05 seconds for plot to refresh), it takes around an hour to win using the Hamiltonian with these dimensions! Although it guarantees a win, the time efficiency leaves room for improvement.

6.3. **Sensitivity Analysis.** For both the Shortest Path and Hamiltonian approaches, we changed the field dimensions to be 6x6 and 5x7, and Figure 8 gives an example of what the Hamiltonian method would look like on a smaller grid. We wanted to determine if the combination of odd and even lengths of the board affected the success of the methods. We ran 10,000 simulation for each combination of method and dimension, and the results are summarized in Figure 9. The Almighty Move was implemented as alternative use for the

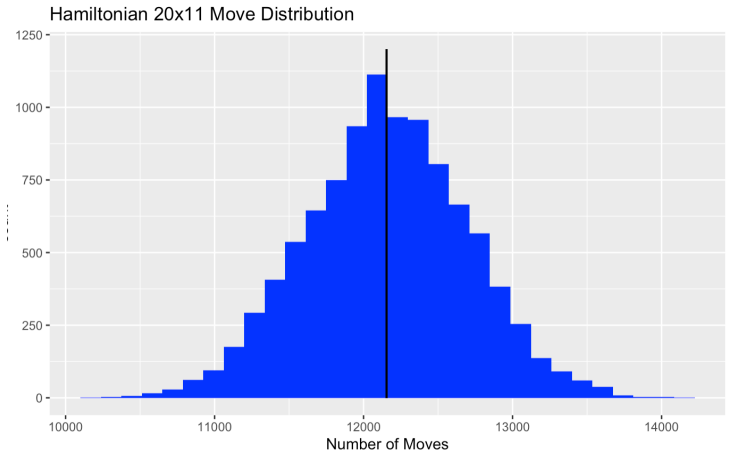


FIGURE 7. Distribution of Moves for Hamiltonian on 20x11

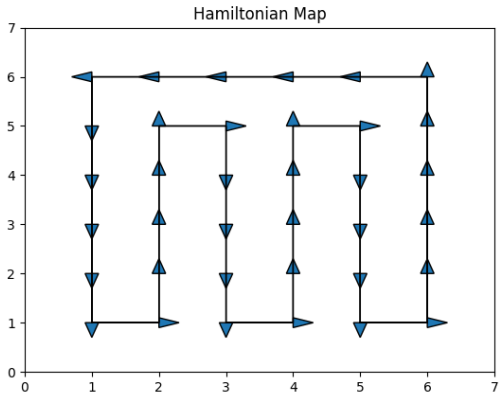


FIGURE 8. Hamiltonian Map on 6x6 Grid. Notice the spiked movements originating at the bottom left corner.

Hamiltonian method on an odd by odd grid, since the Hamiltonian method did not work on an odd by odd grid. This method only saw one winning result in its 10,000 simulations.

As seen in the table, the Hamiltonian method was successful one hundred percent of the time on a 6x6 grid like it was on the 20x11. On the 6x6, the shortest path won 2 out of 10,000 simulations. If we compare these winning results to the Hamiltonian 6x6 results, we see that the Hamiltonian approach took an average of one minute and forty-five seconds to win, while the winning simulations of the shortest path were faster by one minute on the 6x6. The 5x7 dimensions for both movement methods were never successful, and they each have a different reason for why this is. For the shortest path approach, simulations were able to be performed which is why a 0 is shown in the table for it. We could not find a Hamiltonian cycle, however, that worked on the 5x7 grid, which is why its values show N/A and is why we turned to the Almighty Move results.

Dimensions	20x11		6x6		5x7		
Method	**Ham.**	**Short.**	**Ham.**	**Short.**	**Ham.**	**Short.**	**AMB**
Success Rate (out of 10,000)	10,000	0	10,000	2	N/A	0	1
Average Moves to Win	12,155	-	332.3	151.5	N/A	-	456
Average Time to Win	1:00:46	-	0:01:45	0:00:46	N/A	-	0:02:17

FIGURE 9. Table of Sensitivity Analysis Results

7. Our Adjustment of Methods and Analysis of Adjustments

From our preliminary analysis, we were able to develop the following theorems and conjectures.

7.1. **Probability of Moves Theorem.** After evaluating the probability of the Hamiltonian cycle, we wanted to see if there was a general formula to determine the number of moves. Using probability analysis we determined the theoretical average moves and standard deviation, which are shown by the theorem below.

Theorem 7.1. Define $P(m)$ as the probability of eating food after m moves.

$$P(m) = \frac{1}{A-l} \tag{1}$$

Where A is the area an l is the snake's length. For a 20x11 grid with a starting length of 1, the initial probability can be written as

$$P(m) = \frac{1}{(20)(11)-1} = \frac{1}{219} \tag{2}$$

The probability is independent of number of moves. Therefore, the average amount of moves required to get food is calculated by

$$\bar{m} = \frac{1}{A-l}(1\ move) + \frac{1}{A-l}(2\ moves) + ... = \sum_{m=1}^{A-l} \frac{m}{A-l} \tag{3}$$

To find the total number of moves on average, this function needs to be calculated for every snake length. Assuming the player wins, the total amount of moves is

$$\bar{m_{tot}} = \bar{m_{l=1}} + \bar{m_{l=2}} + ... = \sum_{l=1}^{A} \bar{m} = \sum_{l=1}^{A} \sum_{m=1}^{A-l} \frac{m}{A-l} = \frac{1}{4}(A^2 + A) \tag{4}$$

So, for our grid, the Area = 220 and the starting length is 1. Therefore,

$$\bar{m_{tot}} = \frac{1}{4}(220^2 + 220) = 12155\ moves \tag{5}$$

Which matches the experimental average value. The variance can also be calculated by applying theoretical statistics.

$$\sigma^2 = \bar{m^2_{tot}} - \bar{m_{tot}}^2 \tag{6}$$

Although we calculated $\bar{m_{tot}}$ previously, we have to calculate $\bar{m_{tot}}^2$ differently. Because we added up the averages at different lengths, we now have to add up the averages squared at different lengths.

(7)

$$\bar{m_{tot}}^2 = \bar{m_{l=1}}^2 + \bar{m_{l=2}}^2 + ... = \sum_{l=1}^{A} \bar{m}^2 = \sum_{l=1}^{A} \left(\sum_{m=1}^{A-l} \frac{m}{A-l} \right)^2 = \sum_{l=1}^{220} \left(\sum_{m=1}^{220-l} \frac{m}{220-l} \right)^2 = 8.934 \cdot 10^5$$

Now that we have $\bar{m_{tot}}$, we can focus on $\bar{m^2_{tot}}$. We will begin by finding $\bar{m^2_l}$.

$$\bar{m^2_l} = \frac{1}{A-l}(1\ move)^2 + \frac{1}{A-l}(2\ moves)^2 + ... = \sum_{m=1}^{A-l} \frac{m^2}{A-l} \tag{8}$$

Then, like the average, we now add up each length to find $\bar{m^2_{tot}}$

$$\bar{m^2_{tot}} = \bar{m^2_{l=1}} + \bar{m^2_{l=2}} + ... = \sum_{l=1}^{A} \bar{m^2} = \sum_{l=1}^{A} \sum_{m=1}^{A-l} \frac{m^2}{A-l} = \sum_{l=1}^{220} \sum_{m=1}^{220-l} \frac{m^2}{220-l} = 1.187 \cdot 10^6 \tag{9}$$

Now to find our standard deviation, we use equation 6

$$\sigma^2 = \bar{m^2_{tot}} - \bar{m_{tot}}^2 = 1.187 \cdot 10^6 - 8.934 \cdot 10^5 = 2.937 \cdot 10^5 \tag{10}$$

which yields a standard deviation of

$$\sigma = \sqrt{2.937 \cdot 10^5} = 541.98 \tag{11}$$

Which matches the experimental standard deviation

Now that we have found a theoretical average and standard deviation, we create a Gaussian model for the statistical distribution.

$$g(m) = A\mathrm{e}^{-\frac{(m-\bar{m_{tot}})^2}{2\sigma^2}} = (0.0975)\mathrm{e}^{-\frac{(m-12155)^2}{587400}} \tag{12}$$

Where the amplitude, A, was fit to the data. The Gaussian function produces the following probability distribution in comparison to our 20x11 Hamiltonian data (see Figure 10).

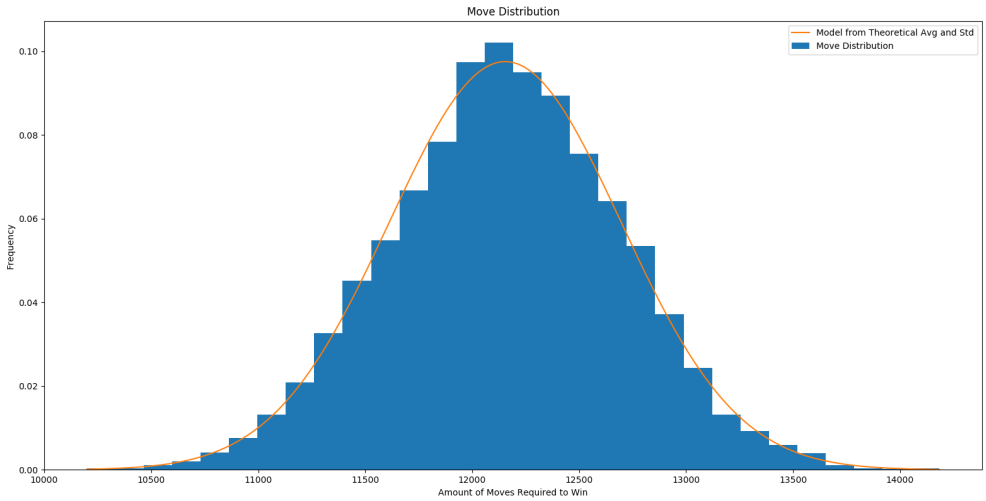


FIGURE 10. Experimental 20x11 Hamiltonian Data with Probability Density Curve

7.2. **Grid Size Conjecture.** Since we were not able to find a Hamiltonian Cycle for 5x7 after finding two successful Hamiltonian Cycles for 20x11 and 6x6 grids, we wanted to investigate if the existence of a Hamiltonian Cycle is reliant on the even/odd behavior of the dimensions. The result was the following conjecture.

Conjecture 1. *Let A be an nxm rectangular lattice with n rows and m columns. If n or m are even, than a Hamiltonian Cycle exists.*

Proof. Let n be even. The vertices of A can be defined by their coordinates (n,m). Suppose the cycle starting point, (1,1) and the initial lattice path:

$$\{(1,1),(2,1),(2,m-1),(3,m-1),(3,1),(4,1)\} \tag{13}$$

The following "spiked" cycle continues for $\frac{n}{2}-1$ times, which hits every vertex area from 2 to $n-1$ and 1 to $m-1$. For a 6x6 grid, there are two cycles $(\frac{6}{2}-1)$ which fill in an area defined by n = 2 through 5 by m = 1 through 5. The cycle concludes with a rotation back to the starting point:

$$\{(n,1),(n,m),(1,m),(1,1)\} \tag{14}$$

Thus completing the Hamiltonian Cycle. Without loss of generality, the same conclusion is made if m is even. □

An example of this method can be seen by Figure 8 on the 6x6 grid.

7.3. **Almighty Move Long.** We compared the two different varieties of the Almighty Move, which were the original Almighty Move method that we call the Almighty Move Branch (Figure 11) and the adjusted Almighty Move Long (Figure 12) option that we came up with ourselves.

As previously mentioned, the Almighty Move was implemented as alternative use for the Hamiltonian method on an odd by odd grid. The paths of the two different varieties of the

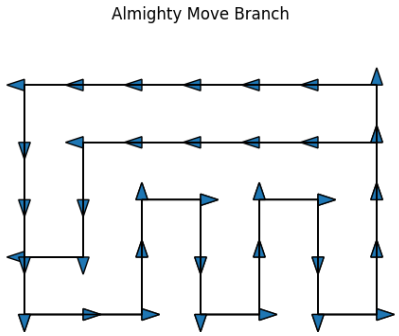


FIGURE 11. Almighty Move Branch Map. Notice there are two overlapping paths beginning at the top right and ending on the bottom left

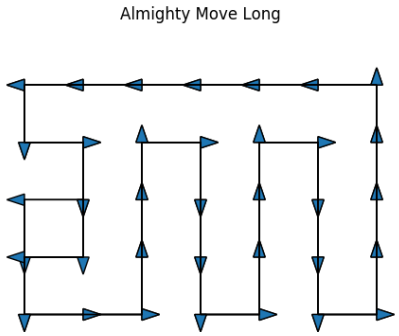


FIGURE 12. Almighty Move Long Map. Notice that there is only one difference in cycles on the bottom left part of the graph

Almighty Move are shown in Figure 11 and Figure 12 options. [h] Our Long method was able to see winning simulations in all 10,000 runs. The average number of moves for the Long Almighty Move was 405.9 and took an average of 2 minutes and 2 seconds to win. This is just 17 seconds faster than the average time of a winning game on a 6x6 grid using the Hamiltonian method, and took 73.6 more moves on average. Due to these statistics which can be seen in Figure 13, we consider the Almighty Move a valid alternative to the Hamiltonian on an odd by odd grid. Additionally, using the same logic for the Hamiltonian Grid Size Conjecture, we were able to formulate one for the Long Method Almighty Move.

Conjecture 2. *Let A be an nxm rectangular lattice with n rows and m columns. If n and m are odd, than an Almighty Move exists.*

Proof. Let n and m be odd. The vertices of A can be defined by their coordinates (n,m). Suppose the cycle starting point, (1,1) and the initial lattice path:

$$\{(1,1),(2,1),(3,1),(3,m-1),(4,m-1),(4,1),(5,1)\} \tag{15}$$

The following "spiked" cycle continues for $\frac{(n-3)}{2}$ times, which hits every vertex area from 3 to $n-1$ and 1 to $m-1$. For a 5x7 grid, there are two cycles $(\frac{6}{2}-1)$ which fill in an area defined by n = 3 through 6 by m = 1 through 4. The cycle concludes with a rotation back to (2,3):

(16)

$$\{(n,1),(n,m),(1,m)\}\, then\, \{(1,m),(1,m-1),(2,m-1),(2,m-2),(1,m-2),...,(2,3)\}$$

At this point, the Almighty Move alternates between two lattice paths

(17) $$\{(2,3),(2,2),(1,2),(1,1)\} \text{ and } \{(2,3),(1,3),(1,2),(1,1)\}$$

The lattice path alternates hitting (2,2) and (1,3) every rotation (which is also seen in our long method), thus completing the Almighty Move. □

An example of this method can be seen by Figure 12 on the 5x7 grid.

Dimensions	**20x11**		**6x6**		**5x7**			
Method	**Ham.**	**Short.**	**Ham.**	**Short.**	**Ham.**	**Short.**	**AMB**	**AML**
Success Rate (out of 10,000)	10,000	0	10,000	2	N/A	0	1	10,000
Average Moves to Win	12,155	-	332.3	151.5	N/A	-	456	405.9
Average Time to Win	1:00:46	-	0:01:45	0:00:46	N/A	-	0:02:17	0:02:02

FIGURE 13. Dimension and Method Analysis Results

7.4. **Sub-Hamiltonian.** The "Sub-Hamiltonian" Method is a variation of the shortest path method and the Hamiltonian Method from experimentation. Specifically, it takes the shortest route component and signals the snake to move to one side of the field. Once the snake enters that side of the field, it enters a Hamiltonian Cycle that occupies half of the field. Once the snake becomes long enough, it drops back into a Hamiltonian Cycle that rotates through the entire grid. To get a visual of the method, it will look similar to Figure 14, where the methods will end bottom left and right points which connect to the bottom transition line. When the food is on the left side, the snake will move through the left Hamiltonian Cycle until the food is generated on the right side. When the snake switches sides, it goes down onto the transition line and begins on the opposite side's Hamiltonian. Once the snake hits a critical length, which we experimentally determined to be 90, the snake breaks down into our 20x11 Hamiltonian Method that we used previously.

Our Sub-Hamiltonian method achieved a winning score for all 10,000 simulations we ran. As seen in Figure 1, our Sub-Hamiltonian method required fewer moves on average to win the game, and it also took less time to do so by cutting about two minutes off of the time. Figure 16 shows the comparison of Hamiltonian and Sub-Hamiltonian move distributions. It can be seen in the figure that the move distributions of both methods follow a normal curve,

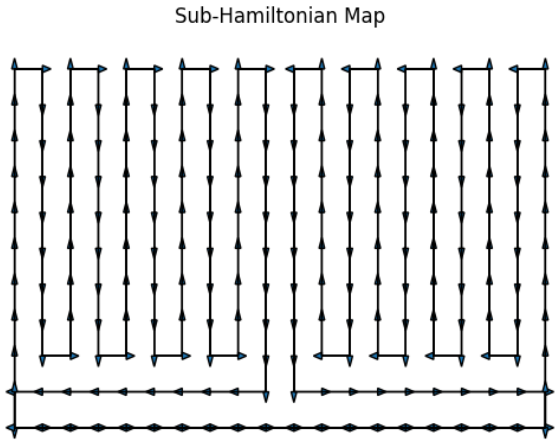


FIGURE 14. Initial Sub-Hamiltonian with transition line on bottom of field

with the average number of moves needed to win being lower for our Sub-Hamiltonian than the Hamiltonian.

Dimensions	**20x11**	
Method	**Sub-Ham.**	**Ham.**
Success Rate (out of 10,000)	10,000	10,000
Average Moves to Win	11,611	12,155
Average Time to Win	0:58:03	1:00:46

FIGURE 15. Sub-Hamiltonian vs Hamiltonian Data

To evaluate the statistical similarity between the averages of the Hamiltonian and Sub-Hamiltonian, we ran a 2 Sample T-test. Before calculations, it is necessary to ensure that the conditions for the test are met.

(1) Independence: Data sets are acquired completely separate from each other
(2) Randomness: Results acquired from a random number generator
(3) Normal Data: Histograms are bell-shaped and data follows Gaussian distribution
(4) Similar Standard Deviations: Sub-Hamiltonian Std = 507, Hamiltonian Std = 540

Now that each condition is met, we are looking to see if the mean of the Sub-Hamiltonian is significantly lower than the Hamiltonian's. Using the data in Table 1, we found that

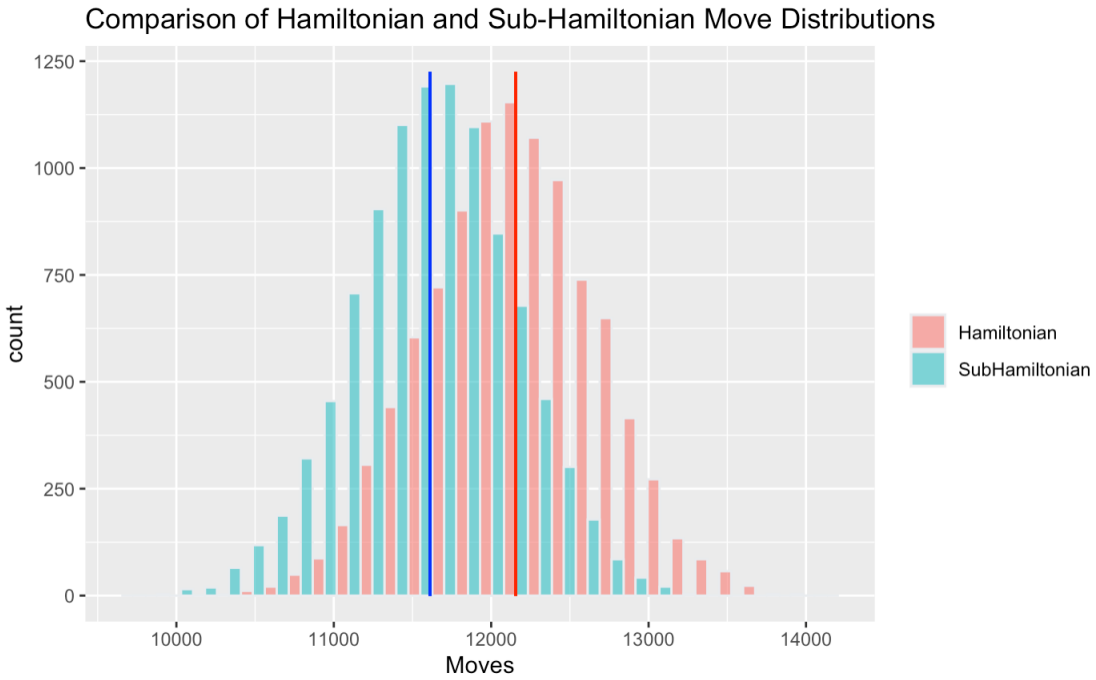


FIGURE 16. Move Distributions for Sub-Hamiltonian and Hamiltonian

Hamiltonian average is significantly higher than the Sub-Hamiltonian. From a t-value of 73.54, the p-value was virtually 0. Outside of statistical terms, this test shows that the Sub-Hamiltonian Method is faster than the Hamiltonian on average.

Method:	Hamiltonian	Sub-Hamiltonian
Sample Average:	12155.4	11610.9
Sample Standard Deviation:	539.62	506.84
Number of Observations:	10000	10000

TABLE 1. Move Statistics on Observed Methods

8. Discussion and Future Work

Although this problem has been around for the last few decades, there is some room for improvement. The major improvement would be an automatic construction of Hamiltonian Cycles. Although our methods were successful, testing and incorporating simulations were difficult and time-consuming. If there was an easier way to construct Hamiltonian cycles, Snake could likely be broken down into a series of Sub-Hamiltonian Cycles, not just half the grid seen in the paper.

Furthermore, these concepts could be expanded using artificial intelligence. Firstly, these methods can compete against AI to see which performances are better. Secondly, AI could learn from the success of these methods to develop its own strengths that were not seen in this paper.

If the constraints on computer science remained the same, on the other hand, the Sub-Hamiltonian could be improved by incorporating reverse direction. Specifically, if the food

is located toward the end of a given Sub-Hamiltonian Cycle, the snake could go through the path in reverse to get to the food earlier. This reduces the time to win by increasing the potential to acquire more food per cycle. These indications can also be experimented on odd grids with the Almighty Moves.

9. Conclusion

Though there are predetermined ways to win at Snake, there are always ways to improve. Finding ways to win were easy. Finding improved ways to win, however, was more difficult. Although Hamiltonian Cycles dominated the playing field, further research showed that they were slower and did not work on all grid sizes. Other methods like the Almighty Move and Shortest Path: Depth First Search were alternatives, but overall could not consistently win.

After some analysis, we found definite ways to determine the existence of a Hamiltonian Cycle based on the grid size. Furthermore, we were able to accurately predict the average number of moves required for any specific Hamiltonian Cycle. Then, using these tools, it was relatively simple applying ways to improve the Hamiltonian. The Sub-Hamiltonian method is just a glimpse of compound Hamiltonian Cycles at work. Although mass coding of these cycles may not be worthwhile, it does raise the question of potential AI servers working to incorporate Hamiltonian Cycles dependent on snake length.

Aside from the Hamiltonian, we were able to construct successful methods for all grids, not just grids with an even dimension. After experimenting with the Almighty Move, we noticed that overlapping two similar cycles would be completely successful as long as the snake followed its tail for most of the lattice path. Overall, we added definite winning algorithms to a complicated phone game that many players continue to struggle with.

10. Acknowledgements

We would like to thank Dr. Peter Muller for his guidance throughout this project. We would also like to thank Dr. Michael Tait for providing us with materials on Graph Theory to better understand the Hamiltonian cycle and how Graph Theory applies to this problem.

References

[1] W. Alhalabi, O. Kitanneh, A. Alharbi, Z. Balfakih, and A. Sarirete, *Efficient solution for finding hamilton cycles in undirected graphs*, SpringerPlus 5, (2016). https://springerplus.springeropen.com/articles/10.1186/s40064-016-2746-8.

[2] T. Ehlis, *Application of genetic programming to the snake game*, Gamedev.Net, (2000).

[3] S. Kong and J. A. Mayans, *Automated snake game solvers via ai search algorithms*. https://cpb-us-e2.wpmucdn.com/sites.uci.edu/dist/5/1894/files/2016/12/AutomatedSnakeGameSolvers.pdf.

[4] Q. Myers, *An oral history of 'snake' on nokia)*. https://melmagazine.com/en-us/story/snake-nokia-6110-oral-history-taneli-armanto. [Online; accessed 03-08-2021].

[5] G. Surma, *Slitherin-solving the classic game of snake with ai (part 1: Domain specific - Shortest,Longestpath, hamiltonian cycle, dnn)*. https://gsurma.medium.com/slitherin-solving-the-classic-game-of-snake-with-ai-part-1-domain-specific-solvers-d1f5a5ccd635. [Online; accessed 02-21-2021].

Appendix A. Coded Snake Program in Python

```
while True:
    #w represents win condition
    if w == 0:
```

```
        plt.figure("SPF") #opens new figure
        plt.pcolormesh(field,cmap=colorset,rasterized=True,
        vmin=0,vmax=3) #sets color map of field
        plt.scatter(body[-1][1]+0.5,body[-1][0]+0.5,
        color = "red") #puts a red dot where the head is
        plt.show("SPF") #shows new plot
        plt.pause(.05) #pause 0.05 seconds to refresh figure
    elif w == 4: #occurs when move goes into snake body or wall
        print("You Lose! Better Luck Next Time")
        break
    elif w == 5: #occurs when entire board is filled
        print("Congradulations! You won!")
        break

    t = .25 #movement frequency (per second)
    now = time.time()
    chn = time.time()
    com = "run"
    while (chn - now) < t: #see move function
        if keyboard.is_pressed("up"):#up movement
            a = 6 #parameterize
        elif keyboard.is_pressed("right"):#right movement
            a = 7 #parameterize
        elif keyboard.is_pressed("down"):#down movement
            a = 8 #parameterize
        elif keyboard.is_pressed("left"):#left movement
            a = 9 #parameterize
        elif keyboard.is_pressed("q"):
            com = "quit"
            break
        else:
            chn = time.time()
            pass
    if com == "quit":
        print("Game Successfully Closed")
        break
    else:
        body,sl,field,w,am = move(a,body,sl,field,am)
    #open in two windows
```

Appendix B. Code for Histograms in R

The following code is the general format for all histograms made for the

various combinations of dimensions and methods analyzed in this project.

Shortest Path Histograms

20x11
Moves
```{r}
short2011m <- read.delim("ShortDFS20x11SimMoves.txt")
summary(short2011m)
short2011m %>%
  rename(moves=X2.570000000000000000e.02) %>%
  ggplot+
  geom_histogram(aes(moves), fill="blue")+
  labs(title="Shortest Path 20x11 Move Distribution", x="Moves")+
  geom_segment(aes(x=207, y=0, xend=207, yend=1100))
```

20x11
Scores
```{r}
short2011s <- read.delim("ShortDFS20x11SimScores.txt")
summary(short2011s)
short2011s %>%
  rename(scores=X2.100000000000000000e.01) %>%
  ggplot+
  geom_histogram(aes(scores), fill="skyblue", bins=55-4)+  # number of bins=
  max score-min score
  labs(title="Shortest Path Score Distribution 20x11", x="Scores")+
  geom_segment(aes(x=18.8, y=0, xend=18.8, yend=600))
```
Joining Hamiltonian and Sub-Hamiltonian Moves Datasets to prepare for coding the double histogram.
```{r}
hamtojoin<-ham2011m %>%
  mutate(id = row_number())#,hammoves=X1.255100000000000000e.04)

subhamtojoin<-sub20x11m %>%
  mutate(id=row_number())#,subhammoves=X1.149800000000000000e.04)

thehams<-hamtojoin %>%
  full_join(subhamtojoin,hamtojoin,by="id") %>%
  rename(Hamiltonian=X1.255100000000000000e.04,
```

```
  SubHamiltonian=X1.149800000000000000e.04)

```
Histogram Comparing Ham 20x11 to Sub Ham 20x11
```{r}
thehams%>%
  select(1,3) %>%
  gather(key=Method, value=Moves) %>%
  ggplot( aes(x=Moves, fill=Method)) +
    geom_histogram(color="#e9ecef", alpha=0.6, position = 'dodge') +
    labs(title="Comparison of Hamiltonian and Sub-Hamiltonian Move
    Distributions", fill="")+
    geom_segment(aes(x=11611, y=0, xend=11611, yend=1225),color="blue")+
    geom_segment(aes(x=12155, y=0, xend=12155, yend=1225), color="red")
```
```
```

Memo

Slitherin Round Blog

With Moves Like These, There's No Way You Can Lose!

Hello Fellow Gamers! Over the past few months, we have been obsessed with the old cell phone game Snake. If you don't remember, it's the one where a snake is moving along a grid, and the goal is to grow the snake's body to fill the screen by eating pieces of food without running into yourself or the wall. It's just so addicting when you think you're on a roll and the snake is getting to be a promising length, but then in a split second you are trapped from your own movements and have no choice but to run into yourself or a wall.

Wouldn't it be great if there was a way that you could always win?

Well, you're in luck, because we have found the answer for you!

After researching different approaches people have taken to solve this problem, we found three approaches to analyze the success rates by running simulations of the game in our own version we coded in Python. We found that one method called a Hamiltonian cycle won the game every time on the original grid size by following a path that hits each pixel on the board once before it ends the cycle by eating the food. It repeats this cycle over and over again until it wins. It is able to win every time because the cycle includes an "escape route" so the snake never hits its own body or the wall. On the original board, this method takes 1 hour and 46 seconds on average to win.

We created our own "Sub-Hamiltonian" method that cuts down on the time and number of moves needed. In this method, we split the board in half, identify which half the fruit is located in, run a Hamiltonian path on just that side, then travel up the middle column and repeat this algorithm again until the snake's body is too long. Then a regular Hamiltonian is followed. Our Sub-Hamiltonian only takes 58 minutes and 3 seconds on average to win.

This works on even by even or odd by even grids. For odd by odd, you can win with our adjusted Almighty Move. It uses Hamiltonian cycles but leaves one pixel out on each run. Once a piece of food is eaten, it switches to a different cycle that leaves a different pixel out but will hit the pixel that was left out in the previous turn. We tested this method on a 5x7, and it won in 2 minutes and 2 seconds on average.

There you have it! You have ways to win on any time of grid! We hope this sparked some 90's/early 2000's nostalgia for you. Give our methods a try and let us know in the comments what you think!