An AI Implementation of the Classic Snake Game

Sydney Cholewinski & Eric Jordan II

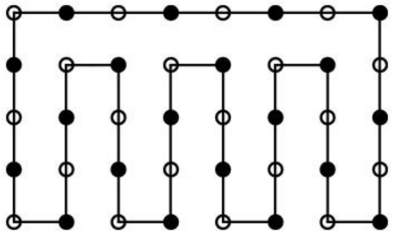
Snake is a common simple game in which the player maneuvers a line that grows in length, with the line itself being the main obstacle. We developed a project that uses different search methods to lead the snake to food and complete the game. Our goal was to compare two different methods, the Hamilton method and the greedy method, to see which one is better in the end. The Hamilton method takes extra steps which in turn uses more time, but it always beats the game. We developed a greedy algorithm to compare and see if the extra time and steps of the Hamilton method are worth it.

In the game, the snake moves inside a 2-dimensional map. At each time step, the snake must either move forward, turn left, or turn right. The game requires that the snake cannot stop moving. The game will randomly generate and place one piece of food on the game map whenever there is not already food on the map. When the snake moves through a piece of food, the snake eats the food and the length of the snake grows by one. The goal is to eat as many pieces of food as possible until the length of the snake fills the game board. The game ends if the snake collides into itself or any of the walls. The game map is set to be 12 units tall and 12 units wide. The snake begins at four units long at the top left corner, heading right. The snake can eat up to 140 pieces of food until it fills up the map.

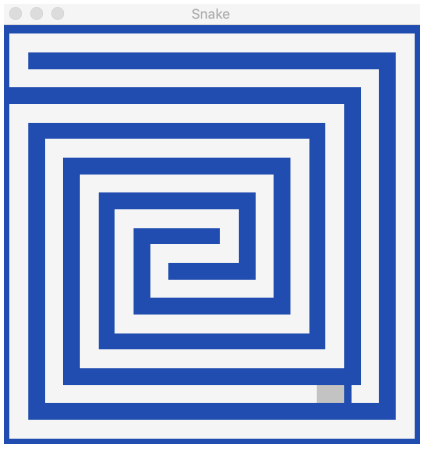
Both the Hamilton and Greedy solver are built using information from a base solver. The base solver finds the shortest path and the longest path from the snake's head to other points on the game map. Specifically, it finds the longest path to the tail, and the shortest path to the food. It does not directly decide the next moving direction of the snake, but it helps the main solvers in

their final decision process. Breadth-First Search is used to find the shortest path. In order to prevent scattered empty spots on the map, the line traveled by the snake needs to be as straight as possible. To implement this, in each iteration of the snake the adjacent point in the last traveled direction will be traveled first. In simpler terms, it is what gives the snake a zig zag shape. The shortest path is used to find the shortest path to the food which is the main information the Greedy solver uses to play the game but it is also used in the Hamilton solver. In order to find the longest path to the tail the path solver uses a heuristic algorithm to find suboptimal solutions. It might seem counterintuitive to find a suboptimal route, but it is all part of the Hamilton method. Suppose we want to find the longest path from point A to point B on a four-by-four game map. The solver first finds the shortest path between the two points and then continues to extend each pair of path pieces until no extensions can be found.

The Hamiltonian path is a route that visits all the spots on the grid exactly once. It is a cyclic path meaning all nodes are visited once and the starting point and the endpoint are the same. To implement this for the snake game, we divide the playable space in a grid and then keep traversing through the map on a hamiltonian cycle. This means the snake would never hit itself and it would eventually get all the food until the snake is the length of the map which is 140 in this case. The Hamilton Solver builds a Hamiltonian cycle on the playing map. It then directs the snake to eat food along the path. To reduce the average steps the snake takes to success, it enables the snake to take shortcuts anywhere along the map if possible. This Hamiltonian solver depends on



A Hamiltonian cycle on a rectangular grid



Completed Snake game using the Hamilton Solver

the path solver to find the shortest path to the food which is how it knows when to take a shortcut and the longest path to the tail which is part of how it completes the Hamilton cycle. The time complexity of this solution is less than ideal but it solves the game every time with no wasted space.

The Greedy solver relies on the shortest path to the food which comes from the base solver. As previously stated, it uses breadth-first search algorithm to find the shortest path. This algorithm thinks one move at a time and only considers moving the snake to the position that appears to be closest to the goal which is the food. The use of this algorithm is what gives the snake a zig zag shape because it is constantly turning in the adjacent direction of the last traveled direction. This method has almost guaranteed that the snake will be able to optimally eat at least the first eight pieces of food. The one-step horizon is what makes it fast but it also makes it easy to get stuck and leave open spots on the map. The snake only looks for the next step that seems to be best or closest to the food without even considering the location of the tail. It is easy for the snake to run itself once it gets longer. Once the snake has grown eight units, this solution is no longer optimal.

After multiple trials of running both solutions, we can conclude that the greedy solver is faster. We can also conclude that it never fully wins the game. Based on our collected data, the greedy method takes between 180 seconds and 195 seconds with between four and nine wasted spaces. The Hamilton method takes between 210 seconds and 238 seconds with no wasted spaces, or a completely filled map, every time. It is between a 30-45 second difference time wise but in terms of completely the game it is either always or never. We decide that Hamilton method is worth the extra time if you want to win every time because it is not even that much of a time

time difference. The snake game itself does not have any real-world applications but the tradeoff of time and accuracy does which is demonstrated in our two methods. This tradeoff can be seen in applications like encryption. One encryption method may be faster but less secure and another may take longer but is much more difficult to decrypt. The user must decide which factor is most important in order to choose a method. For further work, A* search could be implemented as it is an improved version of breadth-first search to see if would compare better to the Hamilton solution. We could search for an algorithm that would accurately solve the snake game but faster than the Hamilton method.

References

Melissa DeLeon. 2000. A Study of Sufficient Conditions for Hamiltonian Cycles. (2000).

Retrieved 2020

Shu Kong. 2016. Automated Snake Game Solvers via AI Search Algorithms. (2016). Retrieved

2020 from http://sites.uci.edu/joana1/files/2016/12/AutomatedSnakeGameSolvers.pdf

Muhammad Darmakusuma. 2019. Greedy Best First Search in Automated Snake Game Solvers. (2019).

Greg Surma. 2019. Slitherin - Solving the Classic Game of Snake with AI. (January 2019). Retrieved November 16, 2020 from https://towardsdatascience.com/slitherin-solving-the-classic-game-of-snake-with-ai-part-1-domain-specific-solvers-d1f5a5ccd635

Marzio Biasi. 2016. The Complexity of Snake. *Department of Mathematics and Computer Science, TU Eindhoven* (2016).

Nigel Chin. 2020. Snake: Hamiltonian Cycle. (June 2020). Retrieved November 16, 2020 from https://kychin.netlify.app/snake-blog/hamiltonian-cycle/

Naga Appaji. 2020. Comparison of Searching Algorithms in AI Against Human Agent in Snake Game. *Faculty of Computing, Blekinge Institute of Technology* (2020).

K. Seeja. 2018. HybridHAM: A Novel Hybrid Heuristic for Finding Hamiltonian Cycle. (2018).

Siddharth Gupta. 2020. The Parameterized Complexity of Motion Planning for Snake-Like Robots. (2020).