Final Project

Rationally Automating The Snake Game

Yahya Alhinai
alhin010@umn.edu
CSCI4511W
December 18, 2019

Abstract

This paper is presenting several algorithms that are intended to automate the snake game. The goal of comparing several algorithms is to find the best way pros and cons for each algorithm and ultimately finding an optimal or near optimal in every stage of of the snake game. In order to find the best path to take in each stage the snake is in, an efficient and fast search techniques needed to be formulated and developed specifically to solve such a problem where it is so much time and resources in order to investigate every possible path in each state which might be impossible for sequential processing computer if it's given a large snake with high resolution. This paper presenting the best algorithm that automates the snake game to take rational actions in every stage with the least amount of memory usage as well as the fastest way to get to an optimal solution. The investigation of such an algorithm that works perfectly on snake game have led to A* search, CSP search algorithm, and Hamilton algorithm to be implemented which are presented in this paper.

1 Introduction

Snake is one of the most common games in the current ear. The inflation of electronic devices has made this game popular as it doesn't consume a lot of computation power. Now as we enter the golden age of AI, Snake game develops as a fundamental guide for programmers to understand the inner game theory as several algorithms can be fully implemented on the game.

Snake game is a one player game who controls the head of the snake to move up, down, left, and right. Trying to collect as many points as possible on an NxM grid. The game carries on until the player hits the borders of the game or itself. The ultimate goal is to have the sake be as tall in length as possible.

In the traditional snake game, a real person player has to direct the snake and that comes with problems like human's response time and the non-optimal path they take. However, This paper is automating the snake game to take rational decision in every stage focusing mainly on algorithm like A* search and CSP search that works perfectly in a well defined heuristic environment. In the following paper, we will analyze some reasonable algorithm and why A* search and CSP search would be the best option for all.

It is worth mentioning that it is theoretically possible to investigate each possible path in every stage to find the optibmal path but, it might be impossible for sequential processing computer if it's given a large snake environment with a high resolution. For instance, if the Snake game is 300x300 grid, it would take over 140 CPU-years [3]. The implication of Snake game is useful in several real-life applications.

2 Literature Review

2.1 A* Search Literature Review

In the paper “Near Optimal Hierarchical Path-Finding”, the authors have developed several methods that overcomes the restrictions A* search faces [1]. Those restrictions are mainly seen when the problem gets large and as the size of search space increases, the number of paths that could be taken increase exponentially. Therefore, path-finding can result in serious performance bottlenecks. This paper presented Hierarchical Path-Finding A* (HPA*) which reduces the possible paths taken by disregarding the distance that is not near optimal distances. Near optimal distance is defined by the user where it shows in the paper that "HPA* is shown to be up to 10 times faster, while finding paths that are within 1% of optimal". Leading for this algorithm to work under constraints of limited memory and CPU resources.

The paper “Path planning with modified A star algorithm” introduces several modifications and improvements of A star algorithm [4]. These modifications are focused primarily on path optimality and the time it takes to find a path. Even though the paper has specified those improvements to work on mobile robot based on a grid map, the methodology can be implemented for similar problems such as the snake game. Several interesting findings have been articulated in the paper. First, it was shown that the fastest algorithm to find a path is JPS A* and it was shown that it works the fastest in various environments. Though, the resulted path would be less optimal than the original A* algorithm would have produced. Another finding that was noted in the paper is the best algorithm to find the optimal or near optimal path is Basic Theta* even in environments with low symmetry. The cost of Basic Theta* finding an optimal path comes with time insufficiency in comparison to other A* algorithms family.

2.2 Multi-agent Path Finding Literature Review

Multi-agent path finding (MAPF) is an important computational algorithm that is introduced by the authors Adi Botea¸ Pavel Surynek [2]. The main purpose of MAPF algorithm is to navigate agent from its location to their target location while avoiding collisions and deadlocks. This is methods and strategies shown in this paper are helpful for the project because in the snake game not only you need to get from one location to another, but also you have to avoid collisions with the tail of the snake as a rule for the game. Fundamentally, MAPF is an improvement over multi-agent path finding focuses on undirected graphs. MAPF works on directed graphs which is ideal in my case as snake game is consider to be semi- directed graphs.

2.3 evolutionary algorithm Literature Review

Several papers have approached this problem by using an evolutionary algorithm (EA) [9]. In the paper "Snake Game AI", the authors have proposed the idea about dynamically adjusting the weights according to the length of the snake. As the snake get taller the collision probability increases; therefore, the weight has to change to tune in performance. Since the algorithm presented is an evolutionary-base algorithm, the paper has identified good crossover operator to make generation-to-generation evolution the best it could that I might use in my project.

2.4 Genetic Algorithm Literature Review

Genetic algorithm (GA) is a possible candidate that I will consider to solve the snake game [8]. The basic functionally of genetic algorithm can be sum in two operations. The first process is the selection of individuals to be moved to the next generation. The second process is manipulation the selected individuals to form the next generation by crossover and mutation techniques As the authors indicates in the paper “the selection mechanism determines which individuals are chosen for mating (reproduction) and how many offspring each selected individual produces” [8]. This means that the main principle of this algorithm is the better an individual; the higher is its chance of being parent. Making the produce of each new generation be better than its successor. In this algorithm, it’s important to balance between exploration and exploration within the mechanism of the selection. In solving the snake game, The GA algorithm start set initial random cities and then evaluate how fit each city current set of cities. Followed by selecting parent cities for next generation and cross over these parents' cities to create new set of cities. Evaluate the fitness of search city and then decide if it is the optimal or near optimal path. If it is then terminating the algorithm and if it's not, then keep producing new generation of cities. The downside of GA algorithm is that it takes a lot of time to reach a to a good-enough result. As well as it does not produce an optimal solution for the snake game because the fact it's randomness nature in moving from on generation to another.

3 Related Work

3.1 Routing Algorithm

The paper "A force-directed maze router" investigate a new routing algorithm [6]. The algorithm is utilizes force-directed placement and maze searching techniques in order to full functions. The algorithm is optimized to for maze routing in the since that it is able to route around complex layouts with various obstructions.The algorithm presented is an improvement of maze algorithm which requires a large memory consumption. This algorithm uses less memory. This is done by investigating only the agent part of the maze. The part of algorithm that optimized routing might be used in the snake game if time allows.

3.2 Lin-kernighan Algorithm

Another possible candidate to solve snake game is lin-kernighan algorithm (LH). This algorithm starts by having a set of cites described in paper [5] by a tour. LH algorithm refining the path that links cities by swapping pairs of sub-tours of cities to make a new tour and thus getting closer to the optimal solution each time. It works by switching two or three cities to make the tour shorter. With each step, LH decides how many paths between cities need to be switched to find a shorter tour. It keeps refining and switching between two/three cites until there is no further improvement that can be made to any of the cities. By then, it can be concluded that LH algorithm have reached an optimal solution. LH algorithm is the best out of the three algorithms presented in this paper. Because the fact it's relatively fast in reaching to a result. Also, usually the results produced by this algorithm are usually optimal. The only downside is that it requires some sort of symmetry to the problem.

3.3 Buffalo Optimization Algorithm

The last algorithm presented in solving snake game is African Buffalo Optimization Algorithm (ABO). First of all, the authors of this paper [7] have chosen to represent population-based stochastic optimization technique as African buffalos as the ABO algorithm was inspiration from their behavior. ABO start choosing a start city for each of the buffalos and randomly locate them in those cities. Then it Updates the buffalo fitness by updating their locations as they follow the current best path with the shortest heuristic value. The determine the local minimal path and compare it to the overall minimal path. Using heuristic to construct new path by adding cities that the buffalos have not visited. Repeats this process until the local path cannot be minimized further. Once reaching the minimal local path the algorithm spits out the results. The only downside for ABO algorithm is that it takes an exponential time to reach to the solution. This is due to the repetitiveness of updating the path for each node each time something changes.

4 Problem Approach

Search algorithms are required so that the snake find an optimal path from the start point to the end goal. The following subsections are presenting the best algorithm found in the literature review section. First, t map of the Snake game had to be represented heuristically in order for heuristic algorithms to work the best.

4.1 Heuristically Representation

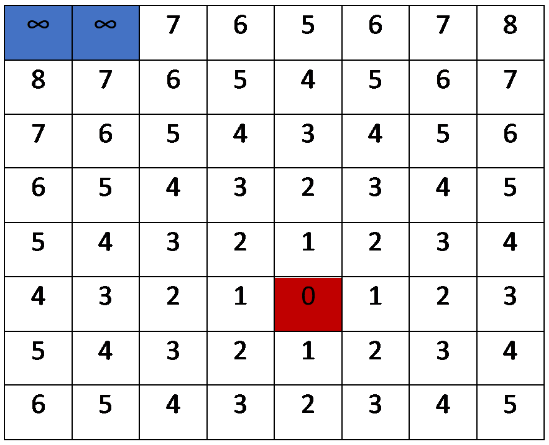
In order to make the environment of the Snake game heuristically represented, it was based on how far the any spot on the map from the Snake food.In order to make the environment of the Snake game heuristically represented, it was based on how far the any spot on the map from the Snake food.

The measurement of the heuristic value is made by the following equations:

$$\text{heuristic value[row, column]} = \\ \text{abs(row - FoodRow) + abs(column - FoodCol)}$$

this equation can be interpreted as the difference between the row of location in respect to the row of the goal state summed with the difference between the column of location in respect to the column of the end goal. This process should be running through all the locations in the game environment and the heuristic values that are produced should be consistent. Those values got to be updated in each stage.

The following figure shows a state example of the a Snake environment after being represented heuristically. The blue color indicates the location of the snake has value approaching infinity. The read location is the end goal that we try to read and the White space represents a possible movement.



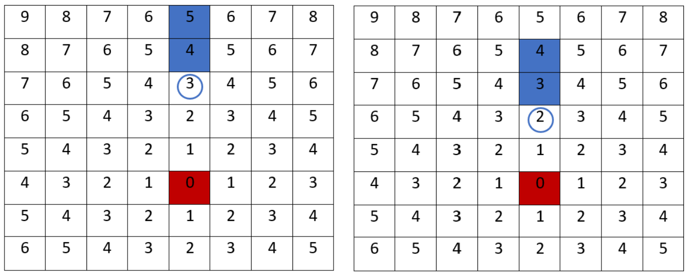
∞	∞	7	6	5	6	7	8
8	7	6	5	4	5	6	7
7	6	5	4	3	4	5	6
6	5	4	3	2	3	4	5
5	4	3	2	1	2	3	4
4	3	2	1	0	1	2	3
5	4	3	2	1	2	3	4
6	5	4	3	2	3	4	5

Fig. 1. Heuristically Representation of Snake Game

4.2 A* Search Algorithm

The A* Search algorithm is an algorithm which can determine the shortest path between two nodes in a graph. IN every state, there are up-to 3 options to take. Using A* algorithm, the path taken in each state has to be the path with the least heuristic value. If there were more than one options with the least heuristic value the algorithm chooses the one it checked first.

Figure 2 shows the direction the snake took while on doing A* search.



9	8	7	6	5	6	7	8
8	7	6	5	4	5	6	7
7	6	5	4	3	4	5	6
6	5	4	3	2	3	4	5
5	4	3	2	1	2	3	4
4	3	2	1	0	1	2	3
5	4	3	2	1	2	3	4
6	5	4	3	2	3	4	5

9	8	7	6	5	6	7	8
8	7	6	5	4	5	6	7
7	6	5	4	3	4	5	6
6	5	4	3	2	3	4	5
5	4	3	2	1	2	3	4
4	3	2	1	0	1	2	3
5	4	3	2	1	2	3	4
6	5	4	3	2	3	4	5

Fig. 2. taking a path using A* algorithm

The problem that night be faced using this algorithm is that it doesn't account for the movements after reaching the goal. The snake body might be locked to itself.

4.3 CSP Search Algorithm

This algorithm is an improvement from A* star algorithm because it always gets the shortest possible path. Also, instead of deciding what movements to take in every new state like A* search, it queues all the movements necessary to get from the initial stage to the goal stage as in first-in first-out bases. This is after going through CSP tree and taking the shortest path using the least to most heuristic value until it reaches the goal stage. The path it returns is always the optimal path. When it find a goal state, we can back track via the parent to get the sequence. The following figure is an example of a path that CSP algorithm produces even before the initial movement is initiated.

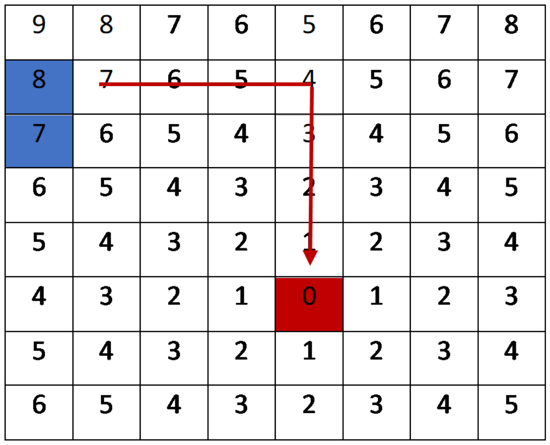


Fig. 3. path produced by CSP algorithm

Even though CSP algorithm is the best candidate to find the optimal path in a sufficient amount of time, The same problem faced in A* algorithm is present here which is CSP algorithm is that it doesn't account for the movements after reaching the goal. The snake body might be locked to itself.

4.4 Hamilton Algorithm

Although Hamilton algorithm is not the best algorithm to find and optimal path in a short amount of time, it solves the problem A* and CSP algorithm that are dealing with which doesn't account for the movements after reaching the goal. Figure 4 shows a sample path.

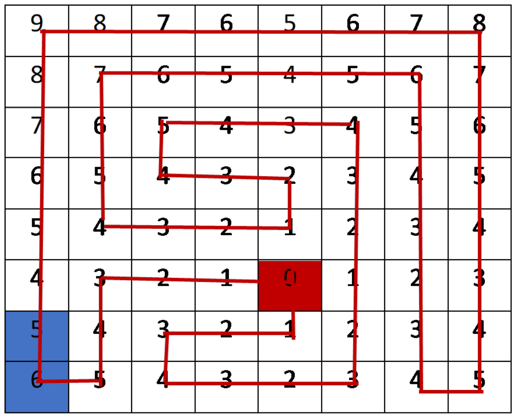


Fig. 4. Hamilton algorithm produced path

Hamilton algorithm requires the path to start and end in the same position as well as it covers all locations.

This Algorithm might be further refined by making the snake jumps to the path that leads to the goal state faster based on the length of the snake. This is done by utilizing the well defended heretic environment. The following figure is example of a refined Hamilton algorithm

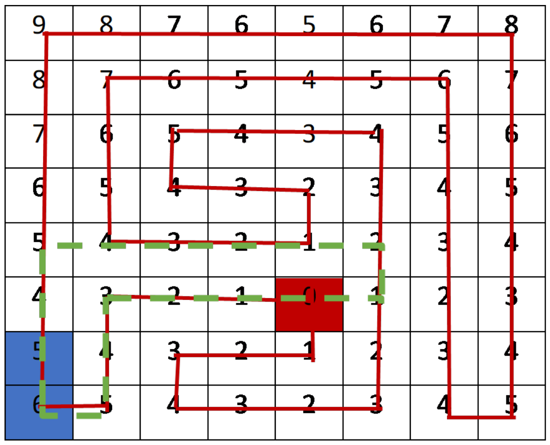


Fig. 5. refined Hamilton algorithm produced path

The refined Hamilton algorithm will be conducted instead of the original Hamilton algorithm for the tests because it produces better results.

5 Experiment Design and Results

Several test have been conducted to the algorithm presented above. The Snake game environment was scale down from the usual size to be set to 10X10 grid to be easier to measure and repeat. Using the program Processing for easy make user interface.

The following Figure is the starting point of Snake game represented by Processing. All algorithms were tested and refined to work at Processing. Each algorithms had 10 trails and the average as taken afterwords.

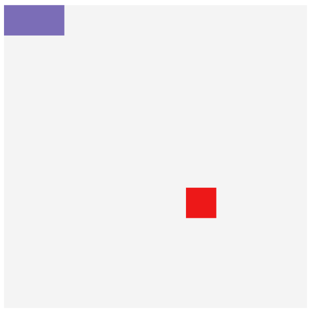


Fig. 6. starting point of Snake game represented by Processing

5.1 A* Search Algorithm

A* Search Algorithm	
Average Length	39.42% map filled
Average Steps	436.8 steps

5.2 CSP Search Algorithm

CSP Search Algorithm	
Average Length	71.07% map filled
Average Steps	335.2 steps

5.3 Refined Hamilton Algorithm

Refined Hamilton Algorithm	
Average Length	98.71% map filled
Average Steps	639.7 steps

6 Result Analysis

After running through a lot of testing and examination towards the three algorithm, The best algorithms to solve a snake game is certain. There were two algorithms preferred to solve such a puzzle.

If the goal of the game is to reach the end goal with the least amount of steps then CSP algorithm is the best candidate to do so. CSP algorithm always takes the most optimal path from the initial place to the end goal. It does so by mapping the path before it initiates its first movement. Starting by having 3 options to choose from. The algorithm calls a function to choose the movement with the lease heuristic value out of the 3 options and the function call itself recursively with the updated position. Whenever the function decides a movement it is saved into a queue that servers on bases of first-in first-out. The function keeps calling itself until it reaches the end goal. If the function happen to hit the boarder of the game or the snake body, the function backtrack one step backward to check the other options and proceeds to call itself recursively until it reaches the end goal. This is the most effective algorithm that finds the optimal path with the least amount of computations possible. The same thing could be achieved by checking every possible combination and decide to choose the most efficient one of them, but this could required up-to 10000 times of computation power as well as it would take longer to reach the end goal.

If the goal, on the other hand, is to have the length of the snake get as long as it can be then the best algorithm found to achieve this goal was Hamilton algorithm. Unlike the A* and CSP algorithm, Hamilton maps out all the locations in the environment. That is to say, it does not only finds a path to the goal, but also it decides beforehand to where it goes after reaching its goal. This feature would make the snake live longer and therefore having greater length because it would not have the problem of locking itself after reaching its goal which was the main problem faced with the other two tested algorithm. There is one downside for Hamilton algorithm which is taking a non-optimal path to reach the end goal. This problem was solved by having a refined Hamilton algorithm to make it faster reaching the goal. This means that if it was was next to adjacent path that reaches to goal faster and it the length of the snake allows it, it will skip the rest of its path to hup into the path that is closer to the end goal. This refined makes the steps needed to reach the end goal significantly less than the original Hamilton algorithm. This is being said, the algorithm is still behind A* and CSP algorithm in terms of steps needed to reach the goal in favor of surviving longer.

7 Conclusion

In this paper, several different algorithms that have the potential to solve the Snake game were investigated. The three algorithms that ended up in being implemented and tested thoroughly were A* algorithm, CSP algorithm, and Hamilton algorithm. These three were chosen on the bases of being the top most effective that works perfect in an environment like the Snake game.

This paper has shown that A* algorithm algorithm will have the fastest reaction because of the reduction of the time for each decision making and it was faster out of the three on making a direct decision. However, it does not produce an optimal solution path.

Unilike A* algorithm, CSP always produces an optimal path from the initial state to the goal state. Also, CSP is every efficient as it requires the minimum computation power to reach the goal. CSP only problem is being locked up after reaching the end goal because the algorithm does not account for the movements after reaching the end goal.

This is where the final algorithm, Hamilton algorithm, comes to place to solve this problem. Hamilton algorithm maps all locations in the map so they it connects all of them as well as it starts from and end from the same location. By doing so, Hamilton algorithm guarantees not being locked up once it reaches the goal.

A future work will be aimed to combine the three presented algorithms if possible as every algorithm excels in an aspect that the other two are having non-sufficient performance on.

References

[1] A. Botea, M. Müller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of game development*, 1(1):7–28, 2004.

[2] A. Botea and P. Surynek. Multi-agent path finding on strongly biconnected digraphs. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[3] K. K. Burusco Goñi. *Methodological approach to conformational search a study case: cyclodextrins*. Universitat Autònoma de Barcelona,, 2009.

[4] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica. Path planning with modified a star algorithm for a mobile robot. *Procedia Engineering*, 96:59–69, 2014.

[5] K. Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[6] F. Mo, A. Tabbara, and R. K. Brayton. A force-directed maze router. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 404–407. IEEE Press, 2001.

[7] J. B. Odili and M. N. Mohmad Kahar. Solving the traveling salesman's problem using the african buffalo optimization. *Computational intelligence and neuroscience*, 2016:3, 2016.

[8] N. M. Razali, J. Geraghty, et al. Genetic algorithm performance with different selection strategies in solving tsp. In *Proceedings of the world congress on engineering*, volume 2, pages 1–6. International Association of Engineers Hong Kong, 2011.

[9] J.-F. Yeh, P.-H. Su, S.-H. Huang, and T.-C. Chiang. Snake game ai: Movement rating functions and evolutionary algorithm-based optimization. In *2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 256–261. IEEE, 2016.