AI Final Project - Snake Game

Ameer Badir - ameer.badir@mail.huji.ac.il
Ameer Ganem - ameer.ganem@mail.huji.ac.il
Lama Tarabeih - lama.tarabeih@mail.huji.ac.il
Omar Issa - omar.issa@mail.huji.ac.il
Siwar Mansour - siwar.mansour@mail.huji.ac.il

Abstract

The main objective of this report is to compare search algorithms in AI - specifically, pertaining to the Snake game. After conducting our experiments, we have concluded that a uniform solution for all comparisons was not found.However, for each game mode we have a different optimal solution.

1. Introduction

Snake is a video game that came out in the late 1970s and became somewhat of a classic. The player controls a long snake-like body made up of blocks, roaming around a plane, picking up food, or fruit, trying to avoid hitting its own body, or the "walls" that surround the gaming area or the obstacles placed in the area. Everytime the snake eats a piece of the food, it grows longer by a block, making the game increasingly more difficult for the player. The player cannot stop the snake from moving while the game is in progress. The player's score is based on the number of fruit the snake has eaten, as in, the final length of the snake. In addition,the snake has four legal moves: up, down, left and right. The player maneuvers the snake by using the arrows on the keyboard

The implementation of the searching algorithms is done to find their performance in the snake game and compare their performance. After conducting a thorough review, we chose some algorithms based on the practical implementation of the AI in the snake game. Breadth-First Search, Best First Search, A* Search, Hamiltonian cycle. We selected these algorithms as they are commonly used searching algorithms and are pathfinding type algorithms.

The game begins with a board of size $n \times n$ blocks and a fruit in a random position on the board containing obstacles generated at random sections on the board, while ensuring that the snake can never be surrounded and can always reach the fruit.

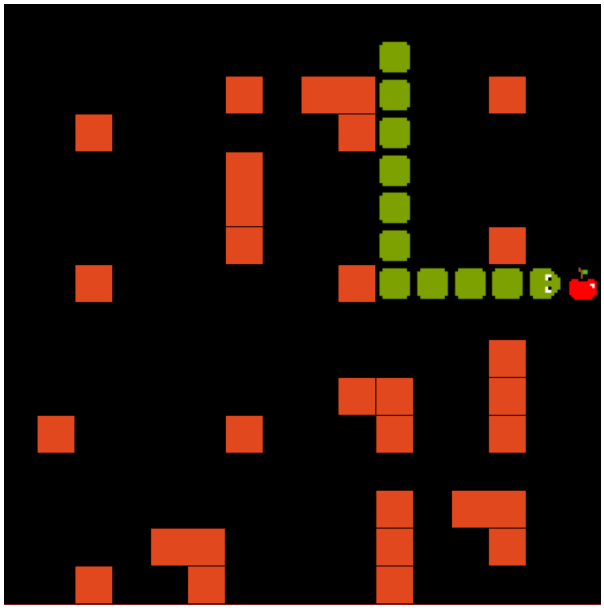

Figure 1 - A picture of the Snake game.

2. Goal of the game

The goal of the game is for the snake to eat as much fruit as possible before the game ends. The score at the end of the game is the length of the snake, or in other words, the number of fruits the snake has eaten.

3. Snake Game As A Graph

A snake game of $n \times n$ can be represented by a graph $G = (V, E)$ which represents the game board. $V = \{v_i\}$ is a set of vertices, each is a block on the board and $E = \{e_{ij} | v_i \text{ is adjacent to } v_j\}$. The snake itself is the path $\{u_1, \ldots, u_k\}$ where u_1 is the head and u_k is the tail. Meaning that we can use search algorithms to find a path from the starting node, the head, to the goal node, the food.

4. Search Methods

There were multiple methods used to create the AI agents that would solve the Snake game:

I. Hamiltonian cycle

A Hamiltonian path is a path in an undirected or directed graph that visits each vertex exactly once. A Hamiltonian cycle is a Hamiltonian path that is a cycle i.e. a closed loop where the first visited node is the last visited node.

In case of the snake game, a Hamiltonian path will visit each block on the board exactly once. This means that it is guaranteed that when the snake uses this path, it will always reach the fruit without hitting itself. To put it differently, using the Hamiltonian cycle in a board with no obstacles will always lead the snake to filling the board and reaching the biggest score it could.

II. Breadth-First Search (BFS)

This search algorithm explores all nodes at the present depth prior to moving on to the nodes at the next depth level. It works with the help of a queue, which is needed to keep track of the child nodes that were encountered but not yet explored. It works based on the principle of the first-in-first-out manner. The time and space complexity of the Breadth-First Search algorithm is $O(b^{d+1})$ where b is the maximum branching factor, d is the depth of the shallowest solution.

III. Greedy Best-First Search (GBFS)

This search algorithm always selects the path which appears best at that moment. It is the combination of the BFS and DFS algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search and a PriorityQueue, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function:

$$f(v) = h(v)$$

IV. A Search*

The A* Search is an informed search algorithm. Starting from a specific starting node, it finds a path to the given goal node having the smallest cost. A* selects the path that minimizes the following equation:

$$f(v) = g(v) + h(v)$$

where $g(v)$ is the exact cost of the path from the start node to v, $h(v)$ is a heuristic function which estimates the cost of the path from v to the goal, and $f(v)$ is the estimated cost our path, from the starting node to the goal that contains v.
We implemented several heuristics and conducted a comparison to estimate which of them give a better solution to the Snake game.

5. Heuristics

We began with the trivial heuristic, ***Manhattan Distance***, where each node contains information on the head position and how far it is from the food. The heuristic function to find the distance and the lowest cost to a node is by using the Manhattan Distance between these two points, name them (x_1, y_1) and (x_2, y_2), and can be calculated using the following equation:

$$d = |x_1 - x_2| + |y_1 - y_2|$$

Having used it and seen that it can be optimized, we decided to look for different ways we can improve our agent. We found some other heuristics we can use to enhance the agent's performance:

I. Squareness
Squareness is an indicator of how the Snake's body is orientated in a square/rectangular manager.

II. Compactness
Compactness is an indicator of how compactly the Snake's body is oriented. It is the number of cases where one body part of the Snake is placed next to another body part of the Snake, without double counting.

III. Connectivity
Connectivity is an indicator of how connected each part of the field is, and whether the Snake is separating one part of the field from another.

IV. Dead End
Dead End is an indicator representing how many spaces are unreachable by the snake based on the current orientation.

We assigned different weights to each one of the heuristics, as shown in the following equation, and used the resulting combination as the heuristic function for the A* search algorithm:

$$Result = w_1 * Manhattan\ Distance + w_2 * Squareness + w_3 * Compactness + w_4 * Dead\ End + w_5 * Connectivity$$

w_i represents the assigned weight for each one of the heuristics. After experimenting with an abundance of different combinations of weights, we found that we got the highest scores when using the following:

$$Result = 1 * Manhattan\ Distance + 4 * Squareness + 4 * Compactness + 3 * Dead\ End + 3 * Connectivity$$

6. Final Results - Combinations of Borders and Obstacles

We ran each one of the search methods on our 16×16 board, and in each case we changed the difficulty level of the board, either adding borders or obstacles to the board, or both.

As the snake only sees its immediate surroundings, knows the general direction to the fruit, and does not have any knowledge about the location of the rest of its body, we wanted to know how well uninformed searches will do while playing the game.

I. No borders and no obstacles

As we've explained previously, while there are no obstacles, the optimal method to work with would be the Hamiltonian cycle, the maximum score (256). The other search methods all averaged out at in the 60s, the best being the A* Search (the heuristic being the Manhattan Distance) with an average score of 65.

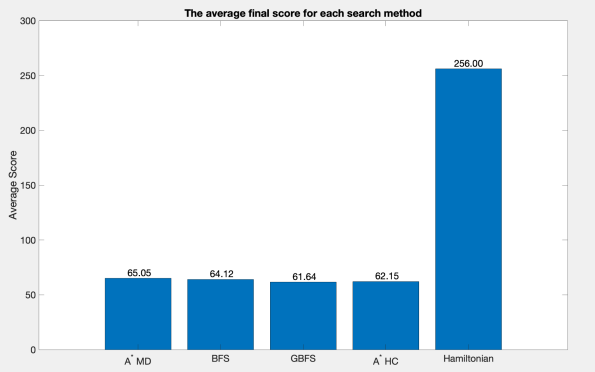

Figure 2 - The average score of different search methods after running the game 1000 times, with no borders and no obstacles.

II. No borders, but there are obstacles

Eliminating the Hamiltonian Path from the group of search methods, as it doesn't function correctly when there are obstacles on the board, as was previously noted, the other search methods all averaged out around the 50s the best being the A* Search (the heuristic being the combination of the different heuristics aforementioned) with an average score of 50. And the runner up being the BFS, averaging out at 49.

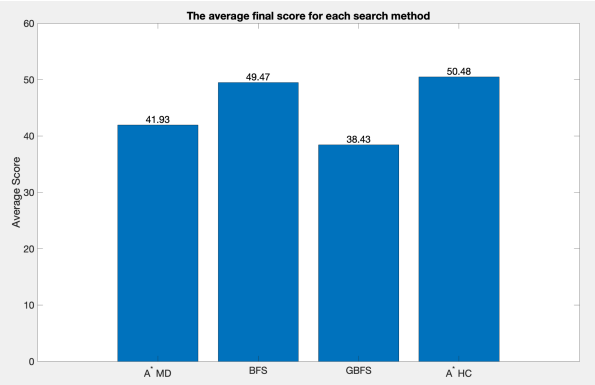

Figure 3 - The average score of different search methods after running the game 1000 times, with no borders but there are obstacles on board.

III. Borders were added, but no obstacles

Back to an obstacle free board, we're back to the Hamiltonian Path, of maximum score (256). The other search methods all averaged out at in the 50s, the best being the GBFS Search (the heuristic being the Manhattan Distance) with an average score of 52, and the runner up being the A* search (the heuristic being the Manhattan Distance) with an average score of 51.

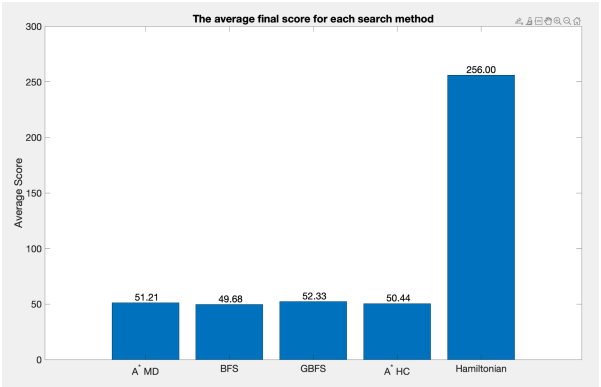

Figure 4 - The average score of different search methods after running the game 1000 times, with borders but no obstacles.

IV. Both borders and obstacles were added

After adding the borders and the obstacles to the board, we found that there was a significant drop in all the average scores reached, with the maximum average score over all the search methods being 32, by the A* search (the heuristic being the combination of the different heuristics aforementioned) and the runner up being again A* search (the heuristic being the Manhattan distance) with an average score of 22. An increase of 10 more fruits eaten, showing that the combination of heuristics enhanced our snake's performance throughout the duration of the game.

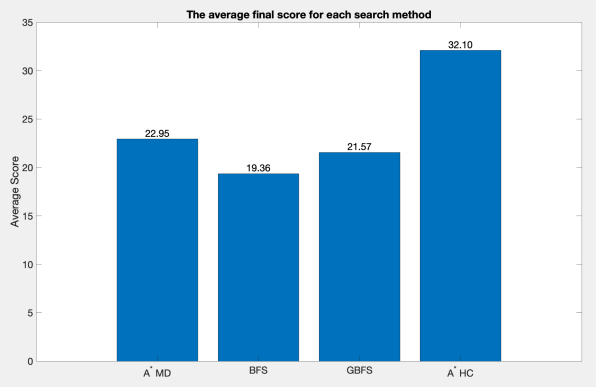

Figure 2 - The average score of different search methods after running the game 1000 times, with borders and obstacles

7. Final Results - Conclusions

I. Hamiltonian Path

On a board with no obstacles, this method achieved perfect results every run, which was expected from what had been explained earlier on.

The main drawback to this approach is that it does not work with obstacles, as there is no guarantee that there exists a Hamiltonian cycle on that board. In this case, the algorithm infinitely searches for a path, to no avail.

As a side note, an interesting realization, the Hamiltonian path approach is not valid when working on odd $n \times n$ boards and using parity we can prove that the snake cannot create a hamiltonian cycle in such a case.

II. Breadth First Search

The BFS algorithm always found the solution if it existed and provided the minimal solution which requires the least number of steps. In 75% of the combinations done, the BFS was 2nd runner up or average, but it had done the worst when we had borders and obstacles in the game.

III. Greedy Best First Search

The Greedy best-first search algorithm always selects the path which appears best at that moment. On average, the GBFS was relatively a good solution, not optimal in any of the modes but wasn't that far off from the rest of the agents, for all modes, it was either average or just below average.

IV. A Search*

We ran the A* algorithm two different times, once with the Manhattan Distance heuristic alone and then with the combination of heuristics. These heuristics always considered the position of the snake's body. Thus, the goal was not only to reach the fruit but to also avoid collision between the head and the body. Most of the time, we were able to achieve better results, a.k.a, higher scores, when utilizing them, and this method worked best when the board had both borders and obstacles.

Hence, if we're playing a board with no obstacles, our safest bet would be the Hamiltonian Path agent. However, if we're working with a board with borders and obstacles, we'd go with our A* search agent when the heuristic is a combination of the above heuristics.

Therefore, our average scores can be put in a graph for every game case:

Search Method	No borders, No obstacles	No borders, obstacles	Borders, No obstacles	Borders, Obstacles
BFS	64	49	49	19
GBFS	61	38	52	21
A* (Manhattan Distance)	65	42	51	22
A* (combination of heuristics)	62	50	50	32
Hamiltonian Path	256	Nil	256	Nil

VI. Conclusions

In this project, we implemented a number of search methods to solve the snake game. We found that path finding algorithms can always be enhanced to get better results, as can be seen from the A* search methods. Moreover, we found that A* search overall works best when working with boards with obstacles.

In addition to that, we've noticed that in general agent performance for different agents when there are no obstacles on board is similar, resulting in almost the same averages. But, when obstacles are added, some do well while others performance significantly worsens.

Thus, when choosing path finding agents for such a game, one must consider the way each algorithm works and decide which would be best for their specific board.

To further enhance the A* search performance, we can create a neural network to find appropriate weights for the combination of heuristics.

A deeper study must be conducted on the creation of an artificial intelligence able to beat the Snake game, and implementation of a deep Q-learning agent or a deep reinforcement learning agent could be of interest to research.

VII. References

[1] Snake (video game genre) - Wikipedia https://en.wikipedia.org/wiki/Snake_(video_game_genre)

[2] Slytherin - Solving the Classic Game of Snake with AI https://gsurma.medium.com/slitherin-solving-the-classic-game-of-snake-with-ai-part-1-domain-specific-solvers-d1f5a5ccd635

[3] Training a Snake Game AI: A Literature Review https://towardsdatascience.com/training-a-snake-game-ai-a-literature-review-1cdddcd1862f

[4] Greedy Best First Search in Automated Snake Game Solvers - Muhammad Daru Darmakusuma https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Makalah/stima2020k3-032.pdf

[5] Game AI: Snake Game - Hangry blog https://han-gyeol.github.io/fun/2017/07/08/Game-AI-Snake-Game/

[6] Snake: Hamiltonian Cycle - Nigel Chin https://kychin.netlify.app/snake-blog/hamiltonian-cycle/

[7] Nokia 6110 Part 3 – Algorithms - JOHN TAPSELL https://johnflux.com/2015/05/02/nokia-6110-part-3-algorithms/

[8] How to implement Snake Game in Python - Wajiha Urooj https://www.edureka.co/blog/snake-game-with-pygame/