AI Assignment:
Snake Game using Search Algorithms and their comparisons.

Outline

Assigned Task

Every group must provide implementation of at least three agents for playing the game including A* based agent and a Greedy Best First search based agent.

Our group included 3 members

1. **L21-5695** Muhammad Ahmad implemented **A* Search Algorithm**
2. **L21-6225** Muhammad Abdullah **Greedy Best-First Search Algorithm**
3. **L21-5666** Farheen Akmal **Uniform Cost Search Algorithm**

Source Code

Github: https://github.com/avcton/snake-ai

Search Algorithms Used

A* Search:

- **Implementation Details:** Uses a priority queue (**heapdict**) based on the sum of the cost to reach a node and a heuristic estimate to prioritise node expansion. Utilises the Manhattan distance as the heuristic function to estimate the cost from a node to the goal.
- **Working:** Begins with the source node and explores nodes with the lowest estimated total cost first. Expands nodes by generating possible moves, evaluating their costs, and updating the priority queue accordingly. Continues until the goal node is reached.

Greedy Best-First Search:

- **Implementation Details:** Also employs a priority queue (**heapdict**) but prioritises nodes solely based on their heuristic values.
- **Working:** Starts from the source node and expands nodes with the lowest heuristic values, aiming to move closer to the goal quickly. Expands nodes by evaluating potential moves, updating the priority queue based on their heuristic values. Stops when the goal node is reached or no more nodes are left to explore.

Uniform Cost Search:

- **Implementation Details:** Utilises a priority queue (**heapq**) based solely on the actual path cost from the initial state to prioritise node expansion.
- **Working:** Begins with the source node and expands nodes based on their path costs from the initial state. Consider all possible moves from each node, evaluating their costs and updating the priority queue accordingly. Continues until the goal node is reached or no more nodes are left to explore.

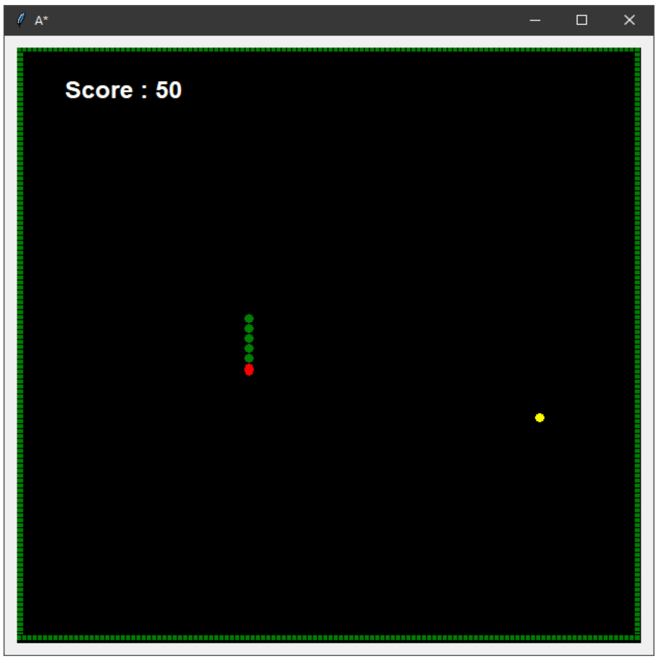
Additional Features Implemented (For Bonus):

1. Integrated Multi-processing for parallel processing of 3 different algorithms.
2. Handled the body increasing scenario where the snake's body increases after eating food.
3. Each algorithm caters the body movement for each step it takes and thus knows where the body will move if it takes a step. This introduces efficiency in the algorithms as they can make moves really close to the body and compute the efficient path even if it's close to the body.

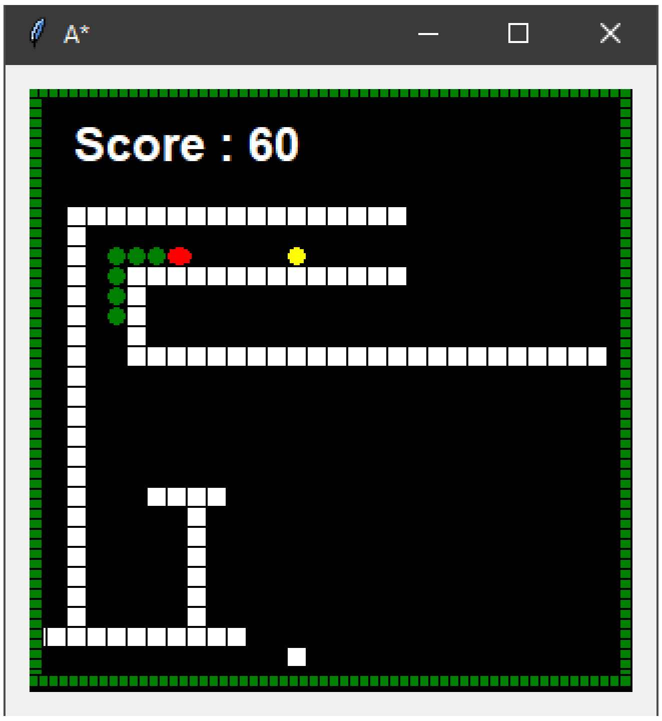
Test Mazes:

We used three different types of mazes for testing the efficiency and effectiveness of our search algorithms and we also ran the snakes for infinitely to see which runs the longest among all.

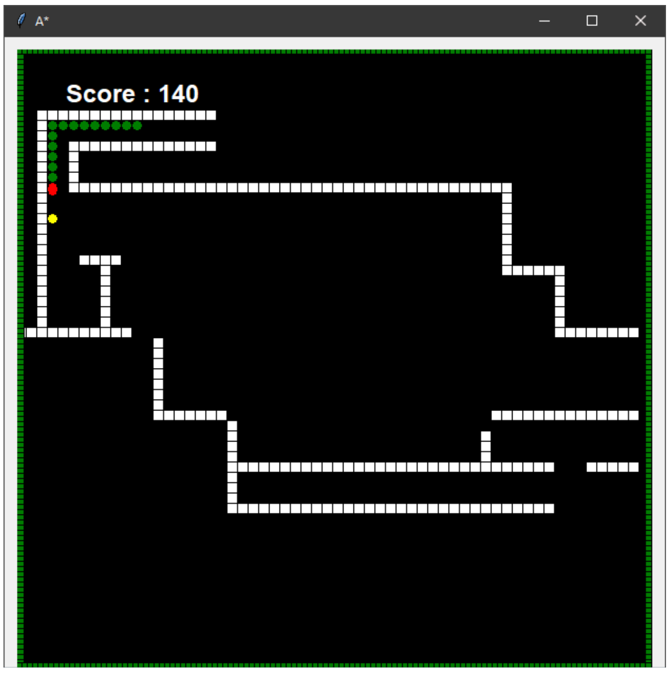
No Hurdles Maze:



Small Grid 30*30 Hurdle Maze:



Standard 60*60 Hurdle Maze:



Results/Comparison:

No Hurdle Maze: (Target 500 score)

Algorithm	**Time Taken in minutes**	**Positions**
A*	1:33.17	1st (winner)
GBFS	1:33.79	2nd
UCS	1:35.73	3rd

Small Grid 30*30 Hurdle Maze: (Target 300 score)

Algorithm	**Time Taken in seconds**	**Positions**
A*	37.41	1st (winner)
GBFS	38.58	2nd
UCS	38.61	3rd

Standard 60*60 Hurdle Maze: (Target 500 score)

Algorithm	**Time Taken in minutes**	**Positions**
A*	2:44.53	1st (winner)
GBFS	3:13.83	3rd
UCS	2:48:54	2nd

Standard 60*60 Hurdle Maze: Last to stop wins (infinite)

Algorithm	**Time Taken in minutes**	**Snake Length**	**Positions**
A*	8:56.44	111	1st (winner)
GBFS	7:34.10	91	2nd
UCS	3:36.17	58	3rd

Conclusion

To sum up all the experiments, we can observe the behaviour of each search algorithm by some variables.

Time: With respect to the time A* Algorithm worked well among other two algorithms in No hurdle maze, in small grid and also in standard maze too. However GBFS also worked fine in no hurdle and in small grid but it took more time in standard grid then other two algorithms. UCS performed better than GBFS in standard grid and took less time to reach target score.

Efficient Path Handling: The most efficient and less time consuming algorithm was A* which was handling all the possible scenarios for finding the shortest path. UCS lacked efficiency in some scenarios because it had no heuristic approach. However in terms of efficiency GBFS also performed nearly equal to A*.

Infinite Run: We ran all three algorithms to check which one can achieve maximum snake length, irrespective of the time taken so as expected A* won(**1st position**) because it ended up having 111 length then GBFS(**2nd position**) ended up having 91 length and the last(**3rd position**) was UCS which was only 58 in length.