CSC3206 ARTIFICIAL INTELLIGENCE ASSIGNMENT 1

Group Name	HACK O’ HOLICS

<u>Group Members</u>

YEOW WEI ZEN	19079128
HEW CHEN YANG	19071695
KARISHMA RAVEENDRAN	18038810
YAP CIA ING	19075720

1.0 Introduction

Aim : To create agents to play the snake game using informed and uninformed search algorithms to find the optimum algorithm .

What to be solved: To search for the shortest path to get to the food and avoid hitting the wall and biting itself. (Extra: It gets longer along the way)

Implementation: Breadth First Search and A* Search

2.0 Breadth-First Search (Uninformed Search)

The first agent was created using Breadth-First Search (Uninformed search algorithm).

2.1 Definition

2.1.1 Theory

The technique functions by going through or exploring the deepest node before proceeding to the adjacent node. It works by exploring through the search tree by expanding the nodes in the first-level and then expands in the second level and reaches the goal. So it is called the level by level traversal method. In the method, all solutions for each node are discovered. This ensures the optimal solution. The Breadth-First Search operates a queue data structure to keep the values. Though there are cycles in the graph, it uses an array to store the passed by vertices in the search tree. It works in the principle of the FIFO(first in first out).

2.1.2 Applying the theory into our scenario

The algorithm begins at the root node which is the head of the snake and transverses through all neighbor nodes which are the positions on the grid at the present depth, before going deeper. It concludes when it reaches the food. Shortest Path BFS performs optimally until snake’s length interrupts its shortest path to the food.

2.2 Advantages and disadvantages

2.2.1 Advantages

- BFS will provide a solution if any solution exists.
- Ensures a path to achieve the goal node in the search tree.
- The least number of steps is needed to achieve the goal node in the search tree.

2.2.2 Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

2.3 Why we chose BFS in comparison to other algorithms

- In comparison to DFS (Depth First Search), BFS(Breadth First Search) uses the FIFO(First In First Out) method to queue whereas DFS uses LIFO (Last In Last Out) method to stack. This difference causes DFS to be less suitable because if termination does not occur when detecting a cycle, the algorithm will be infinite when finding a deeper neighbor node.

2.4 Approach of algorithm towards the problem (How do we get the game to work?)

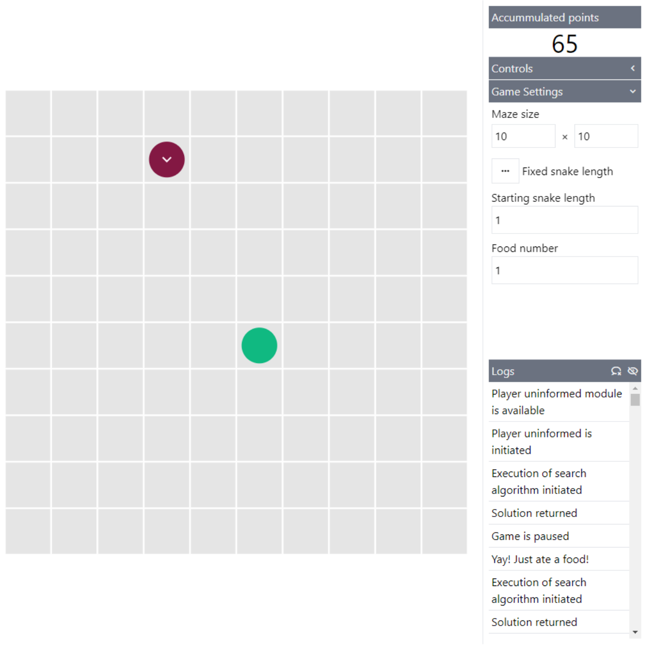
By using the Breadth-First Search approach in this game, the snake moves through the adjacent coordinates instead of the deepest coordinates of the game. The algorithm uses the queue data structure and adjoins every adjacent nodes recursively and searches the trail of the food coordinates in the snake game. By using the trail, the snake achieves the food coordinates in the game. The snake does not visit the coordinates again until it has achieved the food coordinates in the game.

Working steps of the Breadth-First Search algorithm is shown below:

- Step 1: In the first step, the starting coordinates of the snake are pushed on the queue and the starting coordinates are set as the explored coordinates
- Step 2: Next, dequeue the coordinates in the queue step by step, and all unexplored adjacent coordinates are pushed on the queue and are set to explored coordinates.
- Step 3 : The second step will be repeated continuously until the food coordinates are explored.
- Step 4: Lastly, if the food coordinates are explored, it stops and traces the path then returns it.

2.5 Testing and Results

2.5.1 Requirement 1: One food at any time; Non-increasing snake length; Snake length of one; reaching at least 15 points [Working]



Accummulated points

65

Controls

Game Settings

Maze size

10 × 10

Fixed snake length

Starting snake length

1

Food number

1

Logs

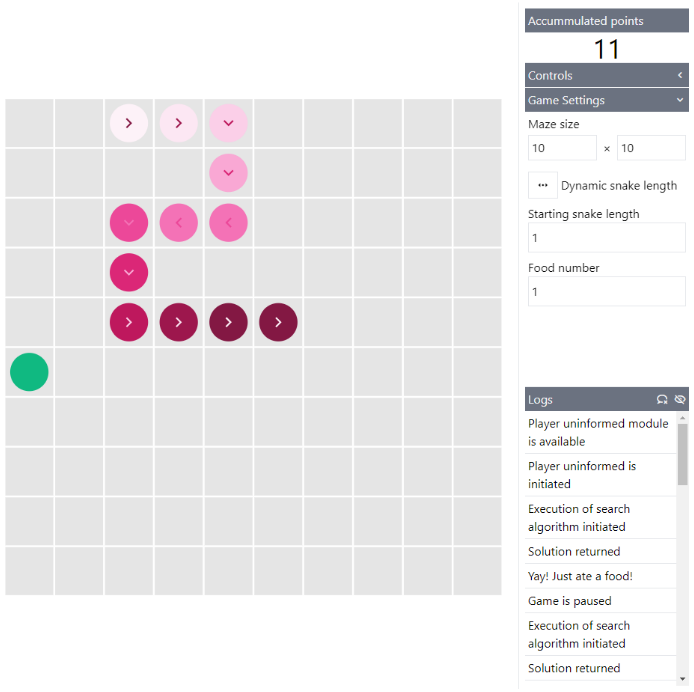
```
Player uninformed module is available
Player uninformed is initiated
Execution of search algorithm initiated
Solution returned
Game is paused
Yay! Just ate a food!
Execution of search algorithm initiated
Solution returned
```

The game settings were configured to the following list in order to fulfill the requirement:

- Maze size of 10 x 10
- Fixed snake length
- Starting snake length of 1, and
- Food number of 1

Based on the above settings, due to the fact that the snake has a fixed length and there is no external way of it resulting in hitting the wall (due to the dynamic maze size we feed to our algorithm) or eating itself and resulting to death, it ultimately means the game will enter an infinity loop until the user pauses the game manually.

2.5.2 Requirement 2: One food at any time; Increasing snake length with food; Starting snake length of one; reaching at least 10 points



Accumulated points

11

Controls

Game Settings

Maze size

10 × 10

Dynamic snake length

Starting snake length

1

Food number

1

Logs

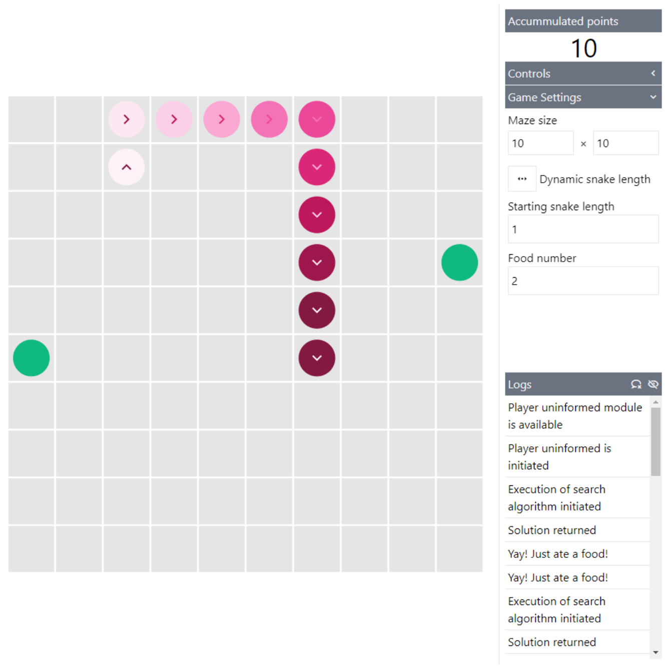
```
Player uninformed module is available
Player uninformed is initiated
Execution of search algorithm initiated
Solution returned
Yay! Just ate a food!
Game is paused
Execution of search algorithm initiated
Solution returned
```

The game settings were configured to the following list in order to fulfill the requirement:

- Maze size of 10 x 10
- Dynamic snake length
- Starting snake length of 1, and
- Food number of 1

In this setting, due to the fact that the snake has a dynamic snake length, meaning the snake length will increase whenever it consumes the food, it ultimately means the snake has the possibility of biting itself. However, by declaring a function of letting the snake not return to the previous path it has taken, we ultimately increase the probability of it not eating itself up before the 10th round. Although the food locations were inconsistent throughout every gameplay, an average score between 10 - 14 was achieved in a testing of 10 rounds.

2.5.3 Requirement 3: Two food generated when no food is available on the maze; Increasing snake length with food; Starting snake length of one; reaching at least 10 points



Accummulated points

10

Controls

Game Settings

Maze size

10 × 10

Dynamic snake length

Starting snake length

1

Food number

2

Logs

```
Player uninformed module is available
Player uninformed is initiated
Execution of search algorithm initiated
Solution returned
Yay! Just ate a food!
Yay! Just ate a food!
Execution of search algorithm initiated
Solution returned
```

The game settings were configured to the following list in order to fulfill the requirement:

- Maze size of 10 x 10
- Dynamic snake length
- Starting snake length of 1, and
- Food number of 2

In this setting, due to the fact that the snake has a dynamic snake length, meaning the snake length will increase whenever it consumes the food, it ultimately means the snake has the possibility of biting itself. However, by declaring a function of letting the snake not return to the previous path it has taken, we ultimately increase the probability of it not eating itself up before the 10th round. Although the food locations were inconsistent throughout every gameplay, an average score between 8 - 11 was achieved in a testing of 10 rounds.

3.0 A* (Informed Search)

The second agent was created using A* (Informed search algorithm).

3.1 Definition

3.1.1 Theory

A* Search algorithm combines both the best features of Uniform Cost Search and pure Heuristic Search to find an optimal solution and completeness of the path and optimal efficiency in the state space search tree. To reach the goal node, A* Search algorithm maintains the tree of nodes and extends the path from by using the cost estimation during each iteration.The A* Search algorithm uses the formulae for the cost estimation which selects the minimize path in the state space search tree is f(n) = g(n) + h(n) where g(n) cost of the path from the source node to n, h(n) is the estimated heuristic cost from the target node to n, f(n) is the total optimal cost of a path going through node n [7]. The A* Search algorithm always takes the least cost path from the beginning to the destination node if the heuristic function in the algorithm is admissible. The priority queue data structure is used by the A* Search. The lower f values are deleted and the neighbouring nodes of f(n) and g(n) values are updated accordingly during each step. When the f(n) reaches its least value than any node in the queue, the iteration ends.

what is the actual heuristic function?

3.1.2 Applying the theory into our scenario

A∗ Search achieves the snake in eating the food but later on the snake is only able to move forward until there are no more moves available. This occurs because of the property of A∗ Search that it only evaluated in the current situation and how to achieve the goal more orderly regardless of possible effects after.

3.2 Advantages and disadvantages

3.2.1 Advantages

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.
- It uses a linear data structure to store the f(n) values.

3.2.2 Disadvantages:

- It does not always produce the shortest path as it is mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

3.3 Why we choose A* in comparison to other algorithms

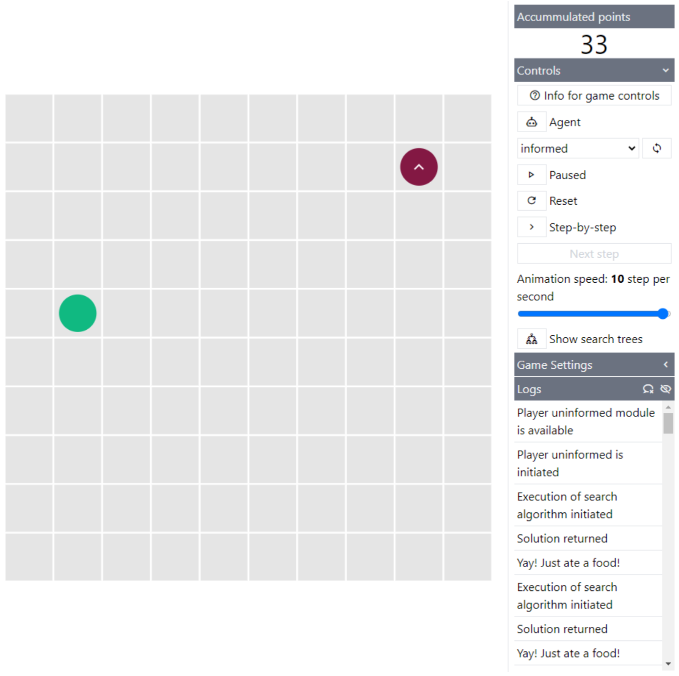
- We evaluated that the function of the A* Search algorithm was better when compared with other informed algorithms in the game. The reason for this is because Manhattan distance and heuristic cost are the search algorithm used which drives the AI to take the shortest path and guides the AI to proceed fewer nodes to achieve the goal . A* Search algorithm also uses the least amount of time to execute. While a Human Agent uses the random path, So the path to reach the goal is unpredictable in the game. We concluded that the performance of the algorithms was better than the human agent in the game.

3.4 Approach of algorithm towards the problem (How do we get this game to work?)

- Step1: Place the starting node in the OPEN list.
- Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
- Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
- Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
- Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
- Step 6: Return to Step 2.

3.5 Testing and Results

3.5.1 Requirement 1: One food at any time; Non-increasing snake length; Snake length of one; reaching at least 15 points [Working]



Animation speed: **10** step per second

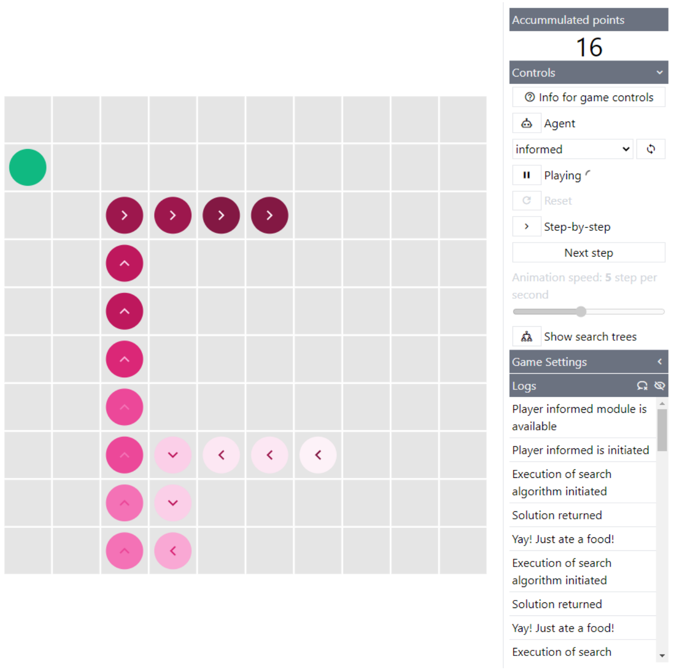
```
Player uninformed module is available
Player uninformed is initiated
Execution of search algorithm initiated
Solution returned
Yay! Just ate a food!
Execution of search algorithm initiated
Solution returned
Yay! Just ate a food!
```

The game settings were configured to the following list in order to fulfill the requirement:

- Maze size of 10 x 10
- Fixed snake length
- Starting snake length of 1, and
- Food number of 1

Based on the above settings, due to the fact that the snake has a fixed length and there is no external way of it resulting in hitting the wall (due to the dynamic maze size we feed to our algorithm) or eating itself and resulting to death, it ultimately means the game will enter an infinity loop until the user pauses the game manually.

3.5.2 Requirement 2: One food at any time; Increasing snake length with food; Starting snake length of one; reaching at least 10 points



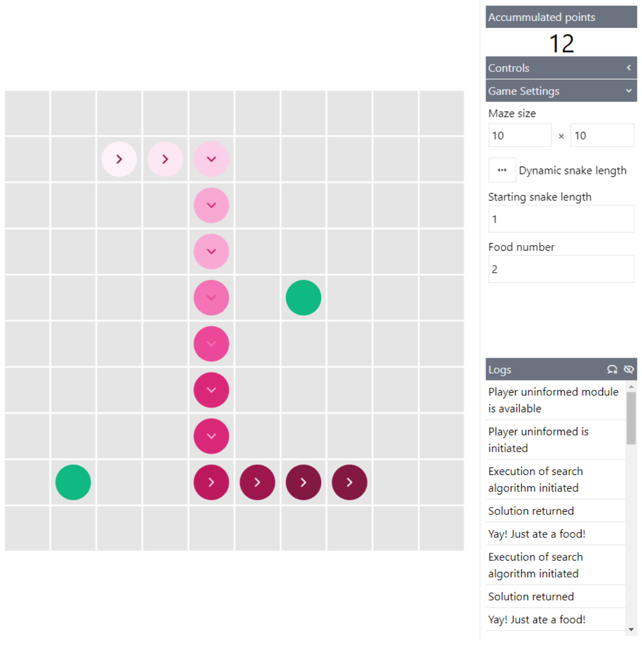
```
Player informed module is available
Player informed is initiated
Execution of search algorithm initiated
Solution returned
Yay! Just ate a food!
Execution of search algorithm initiated
Solution returned
Yay! Just ate a food!
Execution of search
```

The game settings were configured to the following list in order to fulfill the requirement:

- Maze size of 10 x 10
- Dynamic snake length
- Starting snake length of 1, and
- Food number of 1

In this setting, due to the fact that the snake has a dynamic snake length, meaning the snake length will increase whenever it consumes the food, it ultimately means the snake has the possibility of biting itself. However, by declaring a function of letting the snake not return to the previous path it has taken, we ultimately increase the probability of it not eating itself up before the 10th round. Although the food locations were inconsistent throughout every gameplay, an average score between 15 - 18 was achieved in a testing of 10 rounds.

3.5.3 Requirement 3: Two food generated when no food is available on the maze; Increasing snake length with food; Starting snake length of one; reaching at least 10 points



Accummulated points
12
Controls
Game Settings
Maze size
10 × 10
Dynamic snake length
Starting snake length
1
Food number
2

Logs

```
Player uninformed module is available
Player uninformed is initiated
Execution of search algorithm initiated
Solution returned
Yay! Just ate a food!
Execution of search algorithm initiated
Solution returned
Yay! Just ate a food!
```

The game settings were configured to the following list in order to fulfill the requirement:

- Maze size of 10 x 10
- Dynamic snake length
- Starting snake length of 1, and
- Food number of 2

In this setting, due to the fact that the snake has a dynamic snake length, meaning the snake length will increase whenever it consumes the food, it ultimately means the snake has the possibility of biting itself. However, by declaring a function of letting the snake not return to the previous path it has taken, we ultimately increase the probability of it not eating itself up before the 10th round. Although the food locations were inconsistent throughout every gameplay, an average score between 10 - 12 was achieved in a testing of 10 rounds.

4.0 Conclusion (Discussion / Analysis)

To sum up, A* is assured to search for the most optimal path if it exists. In the snake game, A* Search uses the Manhattan distance as a heuristic. In comparison with BFS (Breadth First Search), it both can find the optimal path. A* expands less nodes when compared to BFS as using heuristic information which only expands nodes that is most likely to lead to an optimal path. Despite outperforming BFS as according to the testing and results covered throughout this report, A* Search may lead to a dead end if the snake is long enough.