CSC3206 Artificial Intelligence

Assignment 1

Group Members:

1. Kok Shi Qi 19049808
2. Lim Pei Ni 19066273
3. Tang Wen Yi 17102617
4. Lan Yee Faye 18123729

Group Name:
Waiting for naptime

Table of Content

I. Introduction

In the late 20th century, the snake game makes its first appearance in the history of the gaming sector and becomes one of the most popular mobile games until now. In a snake game, the player controls a line which is known as the snake. The goal of the snake is to consume food that appears in the playground. When food is eaten by the snake, it will grow a unit longer and the food will then be regenerated in different coordinates on the playground. During the process, the player also needs to avoid two obstacles: the snake body and the wall. The snake cannot hit on the wall or collide with its own body, else it will die and the game will be over.

However, as each food eaten makes the snake longer, the game becomes progressively more difficult because the player needs to avoid more obstacles. Then, it comes to the thought of letting the snake moves automatically without human controls. Can an automated player identify the path to reach food and collect it like a human player? To validate the query, it needs to work closely with the application of artificial intelligence in snake games.

The history of artificial intelligence begins in the year 1943. Until now, artificial intelligence has slowly and certainly become a part of human life and changes the way of viewing the world. Artificial intelligence has significant use in many sectors, including healthcare, business, automotive, and games. In artificial intelligence, there is one important area that can be applied in snake games, which is known as Search.

Search refers to the process of looking for a sequence of actions that reached the goal. It works with the information provided to formulate the solution. In a snake game, the usage of search algorithm is possible to find the shortest and the most optimal path to reach the food, which in this case, is the goal of each game round.

This report summarizes the process of creating two automated players in the snake game using two different search algorithms – Breadth-First Search for uninformed search algorithm and Greedy Best-First Search for informed search algorithm. The sections included in the report are: Problem Formulation, Methodology, Result and Discussion, and lastly, Conclusion.

1.1 Objectives

The objectives is to create two agents with one using uninformed search algorithm and another using informed search algorithm. The two agents are also required to solve the challenges below:

1. One food will be generated at any time on the game field, the snake can reach at least 15 points with non-increasing snake length.
2. One food will be generated at any time on the game field, the snake can reach at least 10 points with increasing snake length.
3. Two food will be generated when all the food on the game field is consumed, the snake can reach at least 10 points with increasing snake length

**A food will contribute one point and increase the snake length by 1.*

Besides, the programs should not throw errors while running and should be ended without exceptions.

II. Problem Formulation

Since search is the process of looking for a sequence of actions that reaches the goal, it is important to determine the goal, actions and states in the snake game problem:

Goal: The food locations provided by the front end before each run.
Actions: The cardinal directions, which are north (n), east (e), south (s) and west (w).
State: The location of the node on the game field that is in the form of coordinate [X, Y], where X refers to the column and Y refers to the row.

Before applying the search algorithms and developing search programs, the problem of snake game should also be defined. Formally, there are five main components in a problem: initial state, actions, transition model, goal test and path cost.

Components	Description	Example
Initial state	State where the snake starts at and is provided in the problem definition as the first state in the snake locations.	In[0,5]
Actions	Movement of snake to north (n), east (e), south (s) or west (w).	The applicable actions from the state [0,5] are {MoveNorth([0,4]), MoveEast([1,5]), MoveSouth([0,6])}
Transition model	Description of the result after each action, represented as Result(state, actions)	Result(In[0,5], MoveEast([1,5])) = In[1,5]
Goal test	Test that determines if a given state is a goal state	If food location is [1,6]: The goal is In[1,6]
Path cost	Summation of the step cost to reach the goal state from the initial state	If the solution for [0,5] to [1,6] = [0,5], [1,5], [1,6]: The path cost = 2

Table 1: Main components in problem

2.1 Problem Definition

2.1.1 Input

In this program, the agent will receive two information: setup information and problem information. The next section will be describing these two inputs.

1. Setup Information

Setup information comes at the beginning of the game from the game settings, which includes the maze size and the dynamicity of snake length.

- maze_size: Displayed in the form of the number of rows then followed by the number of columns
- static_snake_length: Defines if the snake length is fixed.

2. Problem Information

At the beginning of each run, the player also receives problem information. Problem information contains the current snake location, the current direction and the food locations.

- snake_locations: The coordinates of each node in the snake body.
- current_direction: The current direction of the head of the snake.
- food_locations: The coordinates of each food.

2.1.2 Output

The output of each run is an array that contains the solution and the search tree. Based on the returned result, the snake will move according to the actions stored in the array, and a search tree diagram will be generated by the search tree returned.

- solution: Defines the series of actions that the snake should take for reaching the food.
- search_tree: Defines the search tree diagram that includes nodes existing in the algorithm.

The search tree is for visualization, it will include all nodes expanded. The removed nodes which depicted with red cross, are either redundant nodes or updated snake body after moves.

III. Methodology

3.1 Uninformed and informed Search

As the objectives of this program is to develop two agents with uninformed and informed search, the following section will give a brief explanation to these two search algorithms and justify the chosen search algorithms.

Uninformed search is also known as blind search. Uninformed search program will generate the solution solely based on the received information with no additional information beyond the definition of the problem. Consequently, uninformed search might be less efficient in terms of execution time and cost.

Informed search, on the other hand, uses knowledge beyond problem definition, strategies derived from previous experience, and estimation based on common knowledge. It utilizes a heuristic function to produce additional information on the goal to enlarge the efficiency.

With various algorithms being categorized into two, *Breadth-First Search (BFS)* is chosen for the solution of uninformed search, whereas *Greedy Best-First Search* is the solution of informed search.

3.1.1 Justification – Breadth-First Search

Breadth-First Search (BFS) first explores the search tree horizontally followed by vertically, that is, BFS will only explore the next depth after all nodes at current depth are explored. BFS practices First-In-First-Out (FIFO), where the first element generated will be the first element explored.

Comparing the Depth-First Search (DFS) algorithm to BFS algorithm, BFS is more efficient in this case. Due to DFS will search vertically then horizontally in the search tree, it will explore a direction as deep as possible until it finds the goal. In most of the cases, the time complexity and space complexity are than BFS.

Comparing the Uniform-Cost Search (UCS) algorithm to BFS algorithm, in this problem, the UCS algorithm is similar to BFS. This is because, the costs from each node to its adjacent nodes are all equal to 1. Every child at the same level has the same cost (cost from the initial

node to the current node). Therefore, with or without sorting the expanded node based on its cost, it will still practice First-In-First-Out queue instead of priority queue, which is identical to BFS. As the calculation of path cost is less important in this case, BFS is considered as a better solution for a snake game.

3.1.2 Justification – Greedy Best-First search

Greedy Best-First search, as known as Greedy Search, utilizes the estimated path cost retrieved from the heuristic function to find the solution. There is another informed search algorithm named A* Search, which utilizes the actual path cost and estimated path cost to obtain the solution. The main difference between Greedy Search and A* Search is the heuristic function.

Greedy Search:

$$f(n) = h(n)$$

A* Search:

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

- $f(n)$: Estimated cheapest path cost from the initial node to the goal node.
- $g(n)$: Actual path cost from the initial node to the node n.
- $h(n)$: Estimated path cost from the node n to the goal node.

An experiment was conducted to inspect the performance of these two algorithms. The given problem definition is: a 7-by-7 maze, the length of snake is 8, the initial state is [0,3], the goal state is [3,6]

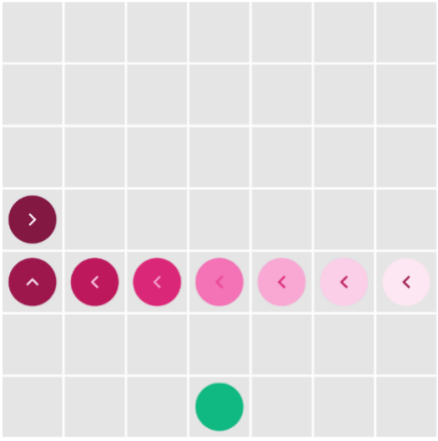

Figure 1: The problem definition of the experiment

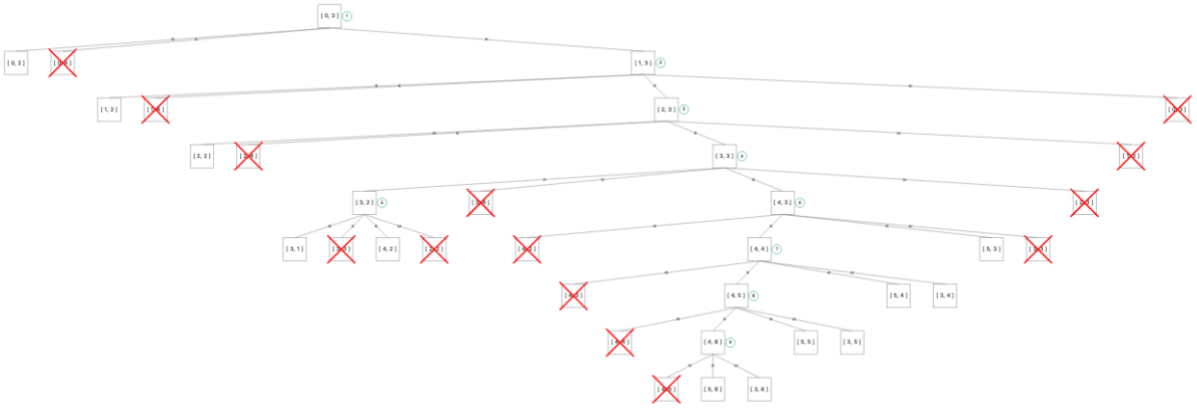

Figure 2: The search tree of Greedy Search

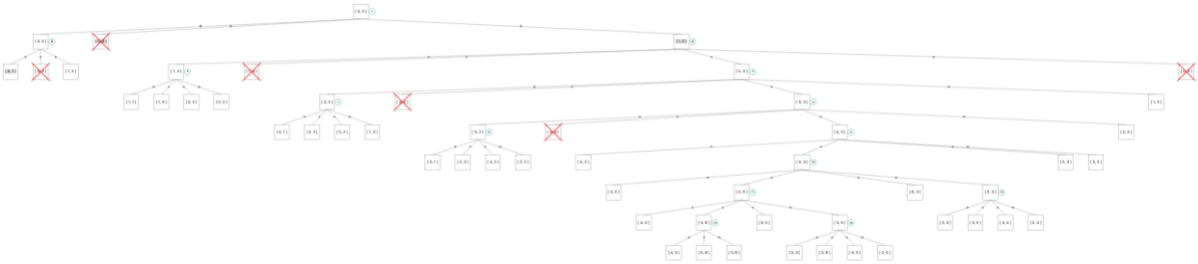

Figure 3: The search tree of A Search*

The solution given by these two algorithms is the same, which is [e, e, e, e, s, s, s, w]. However, the storage complexity and time complexity are different.

	Number of generated nodes (Storage Complexity)	**Number of explored nodes (Time Complexity)**
Greedy Search	35	9
A* Search	54	14

Table 2: Storage complexity and time complexity of Greedy Search and A Search*

Despite the fact that A* search includes actual path cost in estimation and it provides an optimal solution, its time complexity and space complexity are way higher than Greedy Search. From here, it can be concluded that Greedy Best First Search will be a relatively better solution for this problem.

Terminology

As the following section will illustrate the complex logic behind the two algorithms developed, this section will introduce the terminologies used throughout the whole process:

- Frontier: An array contains node objects that are waiting to be explored.
- Explored: An array contains all explored node objects.
- Children: An array of node objects which are the children of a specific node.

3.2 Uninformed Search

3.2.1 Algorithm

Agent: Breadth-First Search

In this section, the methodology of Breadth-First Search (BFS) will be discussed. A flowchart of BFS is depicted below:

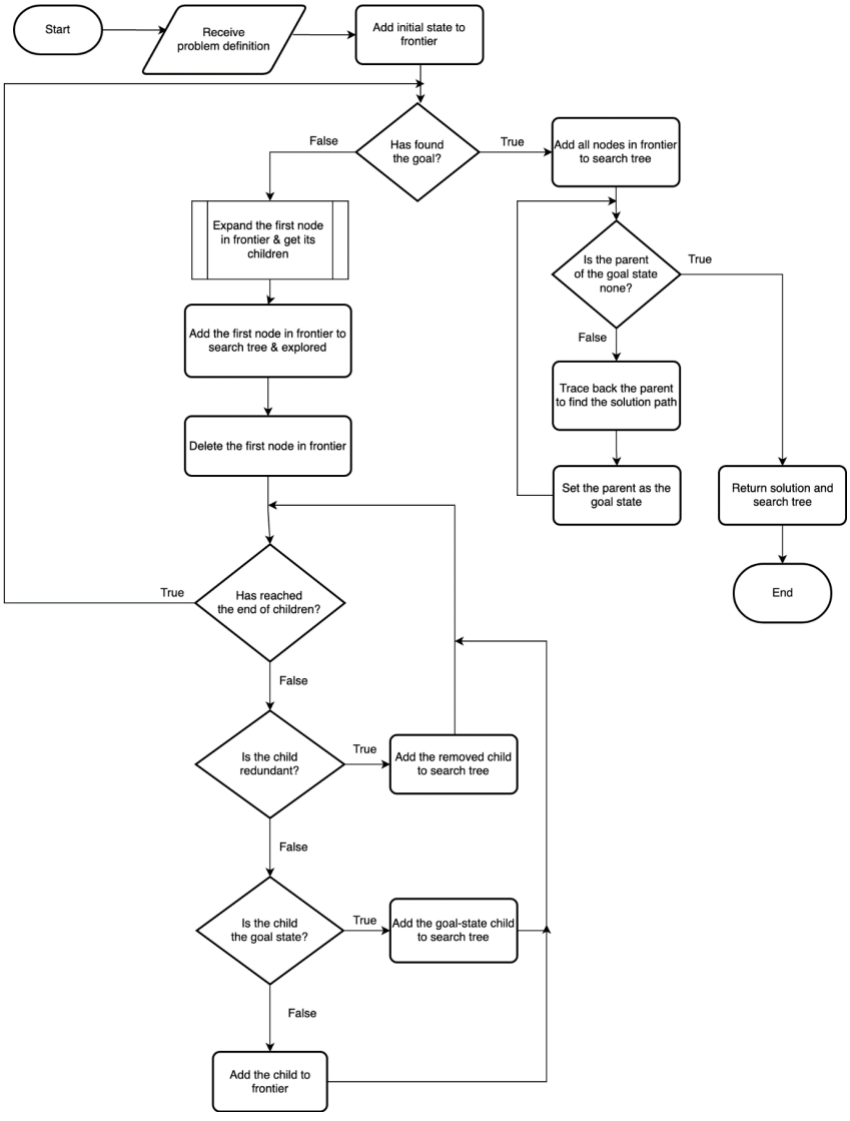

Figure 4: Flowchart of Breadth-First Search

The detailed steps are explained in the following section:

1. **Receive problem definition**

   Before each run, problem must be provided for generating solution. This consists of snake locations, current direction, and food locations. (Refer to Section II. Problem Formulation)

2. **Add the initial node to frontier**

   Firstly, the first node, which is the initial node, is added to the frontier as it would be the first node to be expanded.

3. **Expand the first node in frontier to get its children and directions**

   A function 'expandChildrenDirection' is created to get child nodes of a node. In this function, maze size and the current node state are utilized for the expansion. The current node can take 4 actions, namely "n", "s", "e" or "w" to its children. If the coordinates of a child node exceed the range of the maze, the child node is invalid. Calculation of child coordinates is (X, Y is the coordinates of the current node):

   - n = [X, Y – 1]
   - e = [X + 1, Y]
   - s = [X, Y + 1]
   - w = [X - 1, Y]

   Besides, this function also checks if the expanded child is overlapping with the snake's body after moves. If the child is in the body, the child will be updated as removed node. Finally, the function will return all children of the current node.

4. **Update the first node in frontier and delete it**

   After expanding the node, the algorithm will update the expansion sequence of the node and append its tree node to the search tree. A tree node is generated using the method 'getTreeNode' in the Node class by returning the attribute value of the node object. Finally, the first node will be appended to the array explored, then be deleted from frontier.

5. **Filter out redundant and loopy path**

   Each child node will be checked if it is redundant. If it is not, it will proceed to step 6. Else, it will be added to the search tree as a removed node. In this way, the redundant path can be filtered out to prevent the same node from being expanded multiple times. Once all children have been checked, the algorithm will loop back to step 3.

6. **Carry out goal test**

   If a child is not redundant, it will undergo a goal test. It is because BFS emphasizes the execution of the goal test during node generation. Hence, the goal test would be applied to non-redundant children. After the goal node is found, it will be appended to the search tree and the algorithm will move to the step 7. If the node is not the goal node, it would be added to the frontier list and the process will back to the step 5 until all children are checked.

7. **Add the tree node of the nodes in frontier to search tree**

   This step will append all unexpanded nodes to the search tree. The purpose of this step is to complete the search tree without leaving out any nodes.

8. **Trace back the goal's parent to retrieve the solution path**

   When the goal node is explored, the goal node would be set as a goalie to obtain the complete path from the initial node to the goal node. From the goalie, the algorithm can obtain its parent's ID. Afterwards, the algorithm will loop through the explored list to find the parent node by matching the ID. When the parent node is found, the corresponding action can be retrieved by using the index of the child node in the parent's children property. Finally, the action will be added to the solution. The algorithm will repeat this step until the root node is reached.

Following the process above, the algorithm will be able to return the desired solution path from the initial state to the goal state, as well as the search tree.

3.3 Informed Search

Agent: Greedy Best-First Search

The following section will be illustrating algorithm logic of the agent that uses Greedy Best-First Search. As the heuristic function used is the major concern in informed search, the next part will first explain the heuristic function, then only continue to the logical explanation.

3.3.1 Heuristic Function

As informed search needs extra information on the goal state, the algorithm will be using a heuristic function to calculate the extra information needed.

$$f(n) = h(n)$$

- $f(n)$: Estimated cheapest path cost from the initial node to the goal node through node n.
- $h(n)$: Estimated path cost from the node n to the goal node.

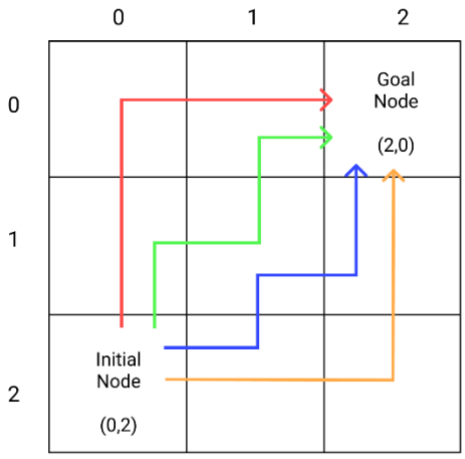

Figure 5: Experiment on the calculation of estimated cost

From the experiment in Figure 3, there are total four possible solutions to reach the goal node departing from the initial node.

- S1: N, N, E, E (red)
- S2: N, E, N, E (green)
- S3: E, N, E, N (blue)
- S4: E, E, N, N (orange)

Either of these four solutions takes 4 steps to reach the goal node. Besides, the below shows the calculation using coordinates to obtain the path cost:

$$|2-0|+|0-2|=4$$

so this is h(n) ?

In conclusion, the sum of the difference between X coordinates from initial and goal node, and the difference between Y coordinates from initial and goal node will be the cheapest path cost in the best-case scenario.

From the experiment above, $h(n)$ is designed as the function *'calEstimatedCost'*, executing the logic of summing the differences between coordinates. The pseudocode is shown as:

```
Receiving 2 sets of coordinates for node A and node B:
    Calculate the absolute value of the subtraction of x coordinate from node A and B
    Calculate the absolute value of the subtraction of y coordinate from node A and B
    Add the 2 answers from above 2 steps to obtain the estimated cost
```

Figure 6: Pseudocode of the function 'calEstimatedCost' [h(n)]

$f(n)$, this function is implemented in the algorithm using the function 'updateEstimatedCost', which updates the estimated cost calculated from $h(n)$, to the attribute 'estimated_cost' for every node expanded.

3.3.2 Algorithm

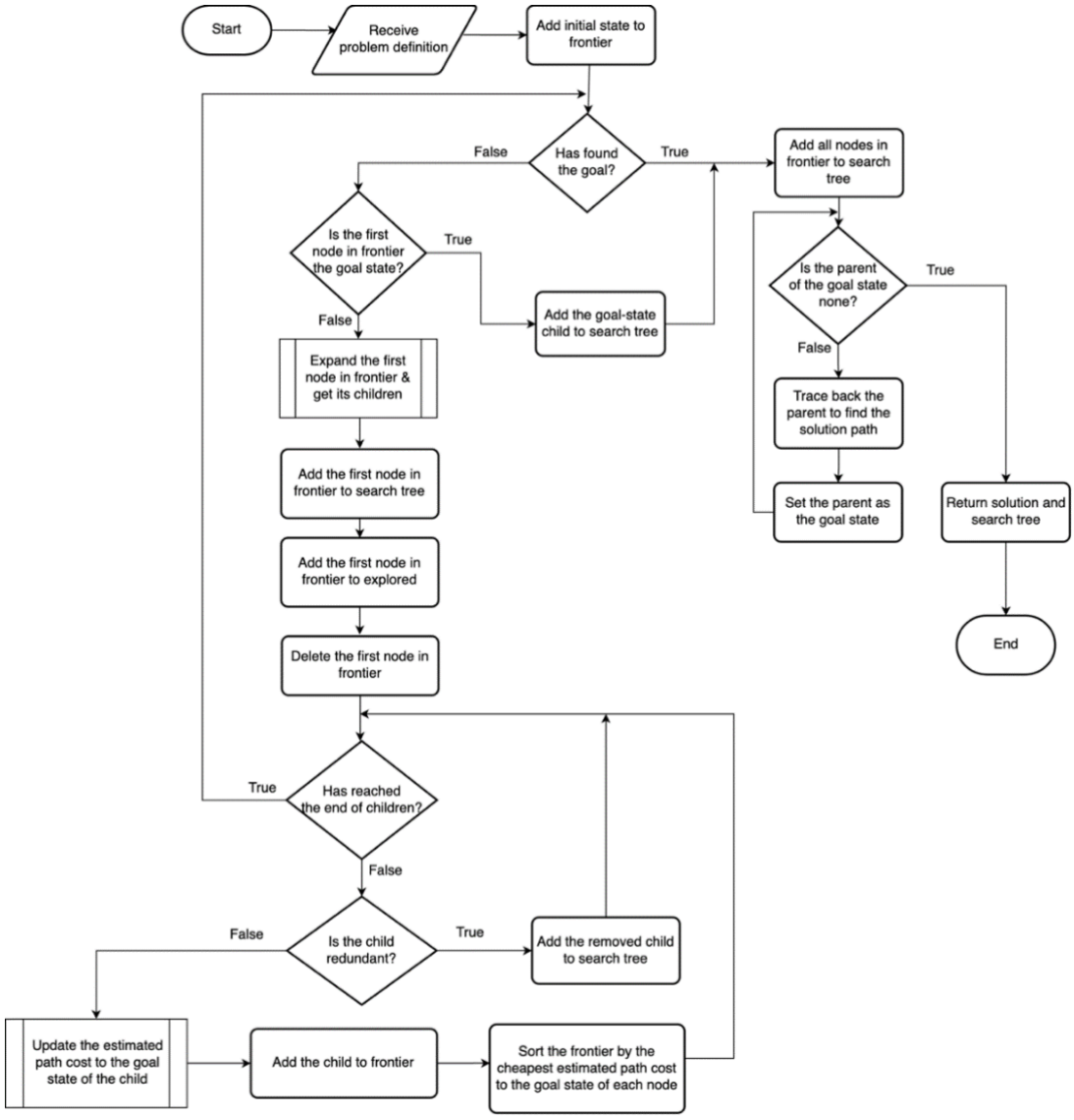

Figure 7: Flowchart of Greedy Search

Compared to Breadth-First Search, the flow is similar but with some different portions. Figure 2 describes the conceptual flow by hiding detailed and complicated logic behind sub-functions. Hence, the detailed steps are listed below:

1. **Receive problem definition**

   This step is identical to the process in BFS.

2. **Sort the goal states**

   To enlarge the efficiency when two or more goal state are given, the algorithm will sort the goal states by the lowest estimated cost from the initial state, which is the head of the snake, to the goal state itself. The purpose of this step is to manipulate the agent to reach the nearest goal state.

   * *Goal states are sorted based on the estimated cost obtained from the heuristic function 'calEstimatedCost'.*

3. **Add the initial node to frontier**

   As the array frontier is used to store the nodes that are going to be explored, the initial node would be the first node that being expanded.

4. **Goal test on the first node in frontier**

   Differing from Breadth-First Search, the goal test is implemented before the expansion of a node. If the node is not the first goal node (the goal node that costs the least estimated path cost), the algorithm will proceed to the next step. On the other hand, if the node is the goal node, the algorithm will append the tree node of the goal node to the search tree then proceed to step 8.

5. **Expand the first node in frontier and get its children**

   The expansion of Greedy Search is using the same function 'expandChildrenDirection' in BFS. It will return all valid children of a nodes.

6. **Update the first node in frontier and delete it**

   After expanding the node, the algorithm will update the expansion sequence of the node, append its tree node to the search tree and the array explored, then lastly delete it from frontier.

7. **Update the children's information**

   For every child obtain from the step 5, if the child is not redundant, the algorithm will update its 'estimated_cost'. Note that the estimated cost is calculated using the heuristic function 'calEstimatedCost'.

   Additionally, the algorithm will add the child to frontier and sort frontier by the lowest estimated path cost. Nonetheless, if the child is redundant, the child will not be added to frontier but appended to search tree as a removed node. If all the children have been amended, the algorithm will loop back to step 4.

8. **Add the tree node of the nodes in frontier to search tree**

   This step will update all unexpanded nodes to the search tree. The purpose of this step is to complete the search tree without leaving out any nodes.

9. **Trace back the goal's parent to retrieve the solution path**

   This step is identical to the process in BFS, which will loop through the array explored to obtain the solution path.

3.4 Difficulty

There are 2 difficulties faced throughout the development process:

1. **Avoidance of biting snake body**

With increasing snake length, the solution path might include the snake body. Hence the snake will bite itself and the game will be over. The solution of this difficulty is to conduct a verification during the expansion of children.

When expanding a node, the algorithm will check if the states of its children are equal to the coordinates of the snake's body. If a child overlaps with the snake body, the child will not be added to frontier.

2. Update of snake location after moves

In the situation that the snake's body is lengthy, the snake would not be able to reach the food because the given initial snake's body "blocks" the solution path. However, this can be solved by updating the snake body after taking a move.

To make the algorithm more computationally intelligent, the algorithm is designed to calculate the estimated snake locations after taking moves. The estimation formula is used for calculating the location of tail (last node in snake's body), which take **the total length of the given body – the step from the initial node to the current node**. Therefore, the estimated snake body will be from the head of the given snake body to the location of the calculated tail.

This solution impressively enlarges the effectiveness and efficiency of two agents.

IV. Result and Discussion

The following section will discuss about the result of the Breadth First Search and Greedy Search algorithms and further analyze the performance of the two algorithms.

4.1 Results

Based on the 3 challenges given, the performance was measured by conducting 10 tests with the maze size of 10 x 10 for each challenge. However, since the snake length in challenge 1 is fixed, the game will be infinite ideally. Therefore, it is assumed that if an algorithm can achieve 50 points in challenge 1, the test run will be terminated.

4.1.1 Breadth-First Search

From the 10 game plays, this algorithm is able to archive 50 points per game for challenge 1. The details of the performance in the challenge 2 and 3 are stated in the following table:

B = bite itself *H = Hit the wall*

Breadth-First Search				
Test	**Challenge 2**		**Challenge 3**	
	Point	**End Case**	**Point**	**End case**
1	29	B	33	B
2	40	B	29	H
3	25	H	40	B
4	42	H	20	B
5	34	B	29	B
6	32	B	29	B
7	32	B	33	B
8	37	B	51	H
9	21	H	35	H
10	29	H	29	B
Total	321		328	
Average	**32.1**		**32.8**	

Table 3: Result of Breadth-First Search

The average point of the challenge 2 is 32.1, where its highest point is 42 and the average of the challenge 3 is 32.8, where its highest point is 51.

4.1.2 Greedy Best-First Search

Again, the Greedy Search algorithm is also capable of scoring 50 points per game in 10 tests for challenge 1. The table below is the performance of this algorithm in challenge 2 and 3.

B = Bite itself *H = Hit the wall*

Greedy Search				
Test	**Challenge 2**		**Challenge 3**	
	Point	**End Case**	**Point**	**End Case**
1	34	H	27	H
2	17	B	40	B
3	38	B	41	H
4	53	B	40	B
5	37	B	41	H
6	30	B	33	H
7	24	B	33	B
8	45	H	29	H
9	35	H	45	H
10	19	B	28	B
Total	332		357	
Average	**33.2**		**35.7**	

Table 4: Result of Greedy Search

The Greedy Search algorithm scored an average 33.2 points per game in challenge 2, where its highest point is 53. Whereas it scored an average 35.7 points per game in challenge 3, although its highest point is lower than challenge 2, which is 45.

4.2 Discussion

Based on the result collected, it showed that both of the algorithms are capable of solving the challenges with reaching at least 10-15 points per game. There is no significant difference while testing the algorithms between challenge 2 and challenge 3. From the average result retrieved, it was observed that Greedy Search was performing marginally better than Breadth-First Search in the snake games. Also, the end case of each test suggested that both programs will end the game without throwing any errors that will affect the execution of algorithm and able to achieve the objectives of the programs.

4.2.1 Completeness and optimality

In terms of completeness, both algorithm able to return a solution with the goal state being reachable in the search space. Since graph search is applied in both algorithms, the redundant nodes are removed and the search space is finite. Therefore, both algorithms are complete. In terms of optimality, Breadth-First search is optimal since the path cost is a non-decreasing function of the depth of the node and all actions in the snake game have the same cost. On the other hand, Greedy Search is not optimal. This is because the algorithm proceeds without considering the known cost. In certain situation, the path cost of solution provided by Greedy Search is not the lowest. For example, the algorithm might proceed to a route that has the lowest estimated cost but full of obstacles, which eventually causing the snake to take a longer route in avoiding the obstacles.

4.2.2 Storage complexity and time complexity

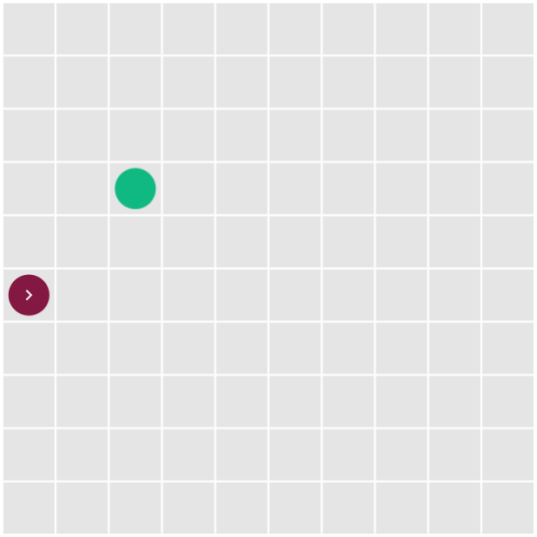

Figure 8: Example of game field

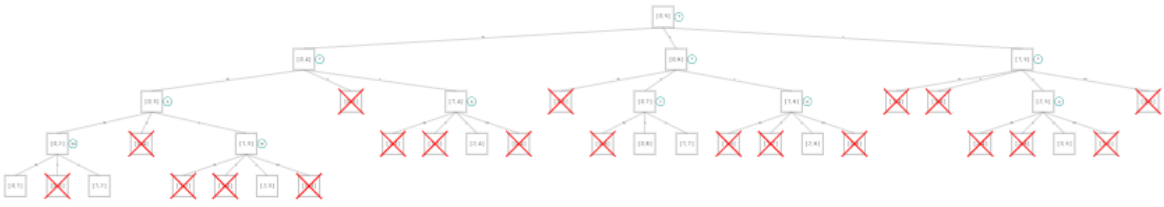

Figure 9: Search tree of Breadth-First search

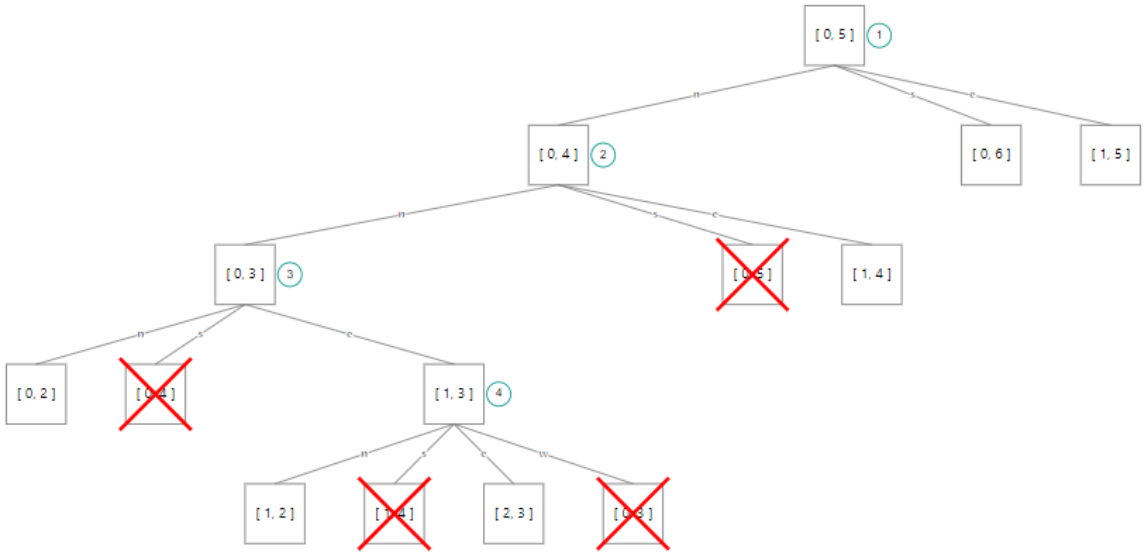

Figure 10: Search tree of Greedy Search

	Number of generated nodes (Storage Complexity)	**Number of explored nodes (Time Complexity)**
BFS	39	11
Greedy Search	14	4

Table 5: Storage complexity and time complexity of BFS and Greedy Search

Given the same problem information in Figure 8, it is observed that the storage complexity and time complexity of Breadth-First search is relatively higher compared to Greedy Search (Table 5). This is because Breadth-First search explored more nodes while proceeding to the goal state, and therefore, storing more information. On the other hand, Greedy Search that utilizes the heuristic function is able to achieve the same result as Breadth-First Search with lesser nodes being generated and explored. It proved that the usage of heuristic function can greatly reduce the storage complexity and time complexity of a search algorithm.

4.2.3 End case

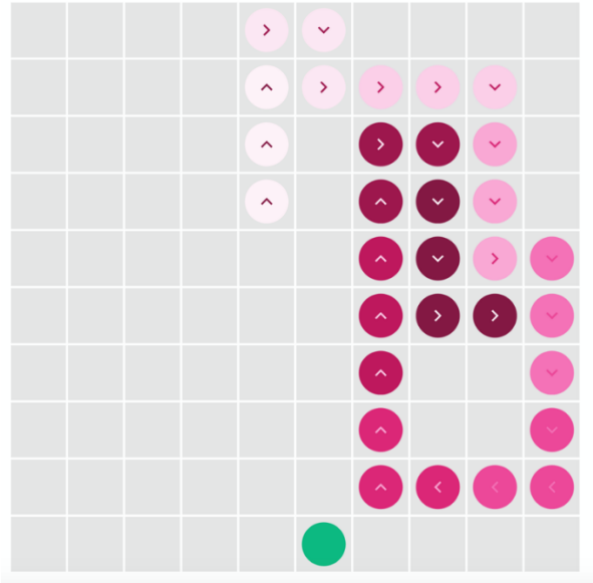

Figure 11: Example of end case

In the scenario where there are genuinely no possible solutions, in which the 4 cardinal directions are occupied by snake's body, the algorithm will return the solution from the initial node to the last explored node. When there is no action can be taken, the algorithm will return the 'current_direction' received from the problem definition, letting the snake to bite itself or hit the wall. This is the reason why the game is always ended in either case and no error will be thrown during the game process. The end case example is shown in Figure 11.

V. Conclusion

The application of artificial intelligence is significant in many industries, including the game industry. In this snake game, two algorithms have been successfully applied to achieve the automation of players, which are Breadth-First Search and Greedy Best-First Search.

In short, the applied search algorithms are manged to solve the three challenges. With increasing snake length regardless of the number of foods generated on the game field in each run, Breadth-First Search program can obtain the average point of 32.1 – 32.8 and for Greedy Search algorithm, the average point of 33.2 – 35.7. Both algorithms are able to avoid the problem of biting snake bodies by checking the coordinates of expanded nodes and update the snake location after moves by estimating the current snake body based on the steps taken. However, there is also a weakness that remains unsolved in the programs. For example, while proceeding to the current goal state, the snake might enter a situation that surrounds itself with its body and unable to escape from it in the next run. Although forward checking is found as the solution for this issue, it is not applied in the current development process due to its complexity. For future development, it is suggested that forward checking to be included in the application of search algorithm.