CSC3206 Artificial Intelligence Assignment Report

Uninformed and Informed Search Algorithm in Snake Game

by

Hay Su Sim	18112342
Gan Chu Heng	18111773
Moo Junn Liang	18027623
Loh Wil Ken	19031301

0 Table of Contents

1 Introduction

The report is presented as an assignment for CSC3206 Artificial Intelligence course. In the report, we implement two different AI search algorithms in the agents to automatically play a snake game, which are Breadth-First Search (BFS) algorithm that is an uninformed search and Greedy Best-First Search (GBFS) that is an informed search. At the end of this report, we will show the difference between these two algorithms in terms of their performance.

The snake game has two modes, which is the human mode and agent mode. In the human mode, the player can use left, right, up and down arrow keys to control the direction and movement of the snake so that the snake moves according to the direction it is heading. In the agent mode, the player can select which agent to run the game. The player can click the play button or keyboard shortcut P to run and stop the game. To reset the game, the player can click the reset button or keyboard shortcut R. The user can choose to run the game through step-by-step or automatic progression. The player also can control the moving speed in human mode and animation speed in agent mode. To show the search tree for each food eaten by the snake, the user can click the show search tree button. The goal of the game is to eat as many foods as possible without hitting the wall of the maze or the snake biting its own tail. When this happens, the game will accumulate the foods eaten as the accumulated points.

In the report, the project methodology and the algorithm implementation for BFS and GBFS algorithms is written and explained in section 2, the results of running the snake game through agent mode with the agent implemented with BFS and GBFS algorithms is tabulated in section 3, the discussion and analysis about BFS and GBFS algorithms and their performance is written and explained in section 4, and the conclusion for the project is written in section 5.

1.1 Problem Description

In this section, we explain about the functionalities of the snake game.

In the simplest way, the concept of the snake game is a snake moving in a maze trying to eat as much food as possible without hitting the wall of the maze or biting its own tail. The food in the maze will be generated at random position once it has been eaten. However, in the user interaction (UI) of the game, the user can change the number of foods generated according to the requirements. Besides, there is a configuration button where the snake can have a static or dynamic snake length in the game. This simply means that the length of the snake can be static which the length of the snake will not increase upon eating the food, or dynamic which the length of the snake will increase by one every time it eats a food. The starting snake length can also be configured according to user needs. The maze size can be configured in the UI but the default maze size is set to 10X10. The game ends when the snake hits the wall of the maze or bites its own tail. Finally, the total food eaten by the snake will be accumulated and calculated as the final score.

Nevertheless, the snake can move step-by-step or automatically with the use of search algorithm that we have implemented in the agents. The possible directions that the snake can move are north "n", south, "s", west "w" and east "e" depending on the current direction of the snake.

The purpose of this assignment is to create two agents that one implements uninformed search algorithm and the other implements informed search algorithm to run the snake game, where the snake will eat the food generated randomly in the maze. There are three requirements that we are expected to achieve in the assignment:

1. One food at any time; Non-increasing snake length; Snake length of one; Reaching at least 15 points.
2. One food at any time; Increasing snake length with food; Starting snake length of one; Reaching at least 10 points.
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.

We are required to achieve at least one of these three criteria; however, we shall attempt to achieve all the criteria given. Therefore, our goal in this assignment is to make the snake to eat as much food as possible.

1.2 Problem Formulation

1.2.1 Initial State

The initial state of the snake would be In([0, 5]). However, depending on the starting snake length, the snake length can be more than one coordinate.

1.2.2 Actions

The actions available to the snake would be different depends on the snake location and direction. For example, if the snake is in coordinate [2, 2] facing the direction North (N), the snake's possible movements would be to the North, East and West. The snake is unable to head South with the function defined in Diagram 1. This is because the program prevents the snake from going to the opposite direction which is not valid in the snake game. Same goes for the snake location that is at the side of the maze and at the corner of the maze. In the search algorithm, it will decide which action to be taken for the snake to move a step as the outcome. For example, the action of snake going from coordinate [2, 2] to [2, 1] can be described as Go([2, 1]). The applicable action of snake going from coordinate [2, 2], that is In([2, 2]) to the adjacent coordinates with the snake facing the direction North (N) can be described as {Go([2, 1]), Go([1, 2]), Go([2, 3])}.

```
def expandNode(self, node, node_num):
    children = []
    directions = ['n','s','w','e']
    nswe = [-1, 1, -1, 1]
    pos = [1, 1, 0, 0]
    if node.direction == 'n':
        directions.pop(1)
        nswe.pop(1)
        pos.pop(1)
    elif node.direction == 's':
        directions.pop(0)
        nswe.pop(0)
        pos.pop(0)
    elif node.direction == 'w':
        directions.pop(3)
        nswe.pop(3)
        pos.pop(3)
    elif node.direction == 'e':
        directions.pop(2)
        nswe.pop(2)
        pos.pop(2)
```

Diagram 1: Code of Defined Function to Control the Action of the Snake

1.2.3 Transition Model

The transition model applied in the algorithms would be the use of function to return the coordinates of the snake when it goes from the current coordinates to the adjacent coordinates. An example of transition model is Result(In([0, 5]), ~~Go([1, 5])~~) = In([1, 5]).

Go East

1.2.4 Goal Test

In our case, the goal test would be the food coordinates, that is {In([food_locations])}.

1.2.5 Path Cost

In our case, the path cost would be the number of steps for the snake to reach the food location, assuming that each number of steps = 1. The path cost would be sum(step_cost).

1.2.6 State Space

The state space in our program would be the possible ways for a node to generate its children. To generate the state space, the algorithm will take into consideration each coordinate of the snake. Depending on the location and direction of the snake, there are different ways a snake coordinates can possibly generate its children. In the program, we have set a condition where it would not consider the coordinate that is outside the maze. The check condition is shown in Diagram 2.

```
for child in list(children):
    if (child.state[0] < 0 or child.state[0] >= self.setup["maze_size"][0] or
    child.state[1] < 0 or child.state[1] >= self.setup["maze_size"][1]):
        children.remove(child)
```

Diagram 2: Code of Check Condition to Remove the Coordinates Outside the Maze

1.2.7 Solution

The solution would be the sequence of actions which is in the form of directions from the initial node, that is the snake locations to the goal node, that is the food locations. An example of solution is [‘s’, ‘w’, ‘w’], which means south → west → west. It simply means that it takes three steps for the snake to eat the food.

2 Methodology

2.1 Breadth-First Search (BFS)

Breadth-First Search (BFS) is one of the uninformed search algorithms. Uninformed search is also known as blind search which is a search technique without additional information about the states beyond that provided in the problem statement. BFS algorithm works by traversing the tree in breadthwise thus exploring the neighbour nodes in the same level before moving towards the next level neighbour nodes. As its name suggests, it starts to expand and explore the tree from the root node in the first level, then it is followed by the neighbour nodes in the second level and so on. Generally, the pattern of tree traversal is performed as horizontally then vertically. BFS algorithm applies a goal test at each process of traversing the node. The algorithm of traversing the tree is ended when the goal is found. The advantage of BFS algorithm is that it guarantees the completeness of the solution as all the solutions of each node are explored, and the optimality of the solution when the path cost is uniform or not in decreasing function of the depth of the node. This can be further explained as the BFS algorithm works in the principle of First in First Out (FIFO), so the goal node found is always the shallowest goal node. The disadvantage of BFS algorithm is that the solution is not necessarily be optimal in certain conditions where the path cost is considered.

In the snake game, the BFS algorithm follows the same principles as mentioned above. Starting from the root node, which is the current snake location, it expands and traverses the adjacent nodes before moving towards the nodes in the next level. The algorithm will end expanding and traversing when the goal node, which is the food location is found. Because of the BFS algorithm always finds the shallowest node in tree traversal, it returns path with the shortest number of steps for the snake to move towards the food location. A frontier list is defined as the queue data structure to store the adjacent nodes recursively and follows the FIFO manner for the node expansion. The explored list is defined to store the expanded nodes. The solution list is defined to store the directions of the snake to move along the coordinates until it reaches the food coordinates in the game.

Breadth-First Search (BFS) algorithm

Step 1: Append the initial node into the frontier list. Note that the first node in the frontier always acts as the current node for expansion.

Step 2: Expand the current node to generate the children of the current node.

Step 3: Append the child node into the frontier list. Delete the current node from the frontier list and append it into the explored list.

Step 4: If the child node is not in the frontier list and explored list, apply a goal test at the child node to determine whether it has the same coordinate with the food coordinates. If the child node is not the goal, then append the child node. The duplicated child node is skipped.

Step 5: Repeat Step 2 to 4 until the goal is reached. The solution is returned by backtracking the node's directions from the goal node (food location) to the initial node (snake location).

2.2 Greedy Best-First Search (GBFS)

Greedy Best-First Search (GBFS) algorithm is one of the Heuristic Search algorithms that works under the principle of goal node exploration according to a specific rule. It uses the heuristic function for the specific rule estimation in the problem domain. In the snake game, the heuristic function is used to estimate the distance from the current node to the goal. It is the combination of Breadth-First Search (BFS) and Depth-First Search (DFS) algorithm that uses the priority queue to maintain the distance between the node and the goal in ascending order. The node which is closest to the goal node is expanded as the algorithm always selects the path which appears best. GBFS algorithm applies a goal test at the expansion of the node. The advantage of GBFS algorithm is that it can switch between BFS and DFS as it uses both BFS and DFS techniques in expanding the nodes. The disadvantage is that it is neither complete as it can get stuck in loops nor optimal as it does not guarantee to render the lowest cost solution.

GBFS algorithm expands the node that is closest to the goal. The distance from the node to the goal is estimated by using an evaluation function f(n) = h(n) where the heuristic function h(n) calculates the Manhattan distance between the adjacent coordinates of snake coordinates to the food coordinates in the game. Based on the least f(n) value, the snake moves along the coordinates until it reaches the food coordinates in the game. GBFS algorithm finds the optimal path for the snake to reach the food coordinates in the game. The frontier list works as a priority queue to store and sort the nodes from lowest to highest evaluation so that the nodes with lowest evaluation will be expanded first. The duplicated nodes are removed from the frontier list. The explored list stores the expanded nodes. The solution list stores the direction of the snake to move along the coordinates until it reaches the food coordinates in the game.

Greedy Best-First Search (GBFS) algorithm

Step 1: Append the initial node into the frontier list. Note that the first node in the frontier always acts as the current node for expansion.

Step 2: Apply a goal test at the expansion of the current node to determine whether the current node has the same coordinate with the food coordinates.

Step 3: If the current node is not the goal, expand the current node to generate the children of the current node.

Step 4: Append the child node into the frontier list. Delete the current node from the frontier list and append it into the explored list.

Step 5: If the child node is not in the frontier list and explored list, calculate the Manhattan distance between each of the child node coordinates and the food coordinates. Sort the frontier list from lowest to highest value of Manhattan distance. The first node in the frontier list acts as the current node.

Step 6: Repeat Step 2 to 5 until the goal is reached. The solution is returned by backtracking the node's directions from the goal node (food location) to the initial node (snake location).

3 Results

3.1 Experiment 1

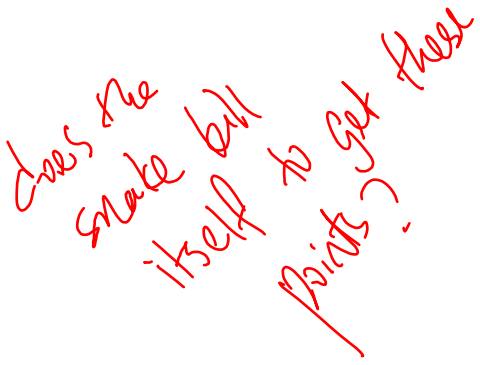
The experiment was conducted to test the performance of Breadth-First Search (BFS) and Greedy Best-First Search (GBFS) algorithms. BFS algorithm is implemented in the agent Uninformed Search Player and GBFS algorithm is implemented in the agent Informed Search Player. It was done by running the game ten times for each agent with the maze size of 10 X 10. One food is generated at any time and the snake length is dynamic with the starting snake length of one. The performance of the algorithms is evaluated to determine whether the snake game reaches at least 10 points.

Game	Points
1	15
2	14
3	12
4	8
5	10
6	14
7	11
8	9
9	17
10	13

Table 3.1: Results of Points of Uninformed Search Player in One Food Snake Game

Game	Points
1	14
2	12
3	11
4	10
5	12
6	16
7	9
8	11
9	13
10	15

Table 3.2: Results of Points of Informed Search Player in One Food Snake Game



3.2 Experiment 2

The experiment was conducted using the same agents with the same algorithms. It was done by running the game ten times for each agent with the maze size of 10 X 10. Two foods are generated when no food is available on the maze and the snake length is dynamic with the starting snake length of one. The performance of the algorithms is evaluated to determine whether the snake game reaches at least 10 points.

Game	Points
1	8
2	12
3	16
4	13
5	12
6	10
7	13
8	11
9	15
10	10

Table 3.3: Results of Points of Uninformed Search Player in Two Foods Snake Game

Game	Points
1	11
2	11
3	17
4	14
5	8
6	10
7	9
8	12
9	12
10	15

Table 3.4: Results of Points of Informed Search Player in Two Foods Snake Game

4 Discussion and Analysis

4.1 Analysis of Results

The performance of the algorithms for each agent is evaluated by using the formula $performance = \frac{round\ of\ games\ that\ reaches\ at\ least\ 10\ points}{total\ round\ of\ games} * 100$, as the performance value of 0 means that the algorithm has a poor performance and the performance value of 100 means that the algorithm has a good performance. From Table 3.1, there are 8 out of 10 games that reaches at least 10 points, and the performance is $\frac{8}{10} * 100 = 80$. From Table 3.2, there are 9 out of 10 games that reaches at least 10 points, and the performance is $\frac{9}{10} * 100 = 90$. From Table 3.3, there are 9 out of 10 games that reaches at least 10 points, and the performance is $\frac{9}{10} * 100 = 90$. From Table 3.4, there are 8 out of 10 games that reaches at least 10 points, and the performance is $\frac{8}{10} * 100 = 80$. This proves that the implementation of BFS and GBFS in the agent has a success rate of 80 to 90 percent to reach at least 10 points in one round of snake game.

4.2 Discussion

Breadth-First Search (BFS) and Greedy Best-First Search (GBFS) algorithms are implemented in the agents with the condition of the game ends when the snake bites itself. Since this is the only condition to end the game, the requirement of the project that one food is generated at any time and the snake length is static with the starting snake length of one to reach at least 15 points definitely can be achieved as the snake is impossible to bite itself with a snake length of one.

The second requirement of the project is that one food is generated at any time and the snake length is dynamic with the starting snake length of one to reach at least 10 points. The third requirement of the project is that two foods are generated when no food is available on the maze and the snake length is dynamic with the starting snake length of one to reach at least 10 points. Basically, the second and third requirements have the same condition that the agents are implemented with BFS and GBFS algorithm, and the game ends when the snake bites itself. Two experiments are carried out to evaluate the performance of the algorithms, and it has been observed that not every round of games can reach 10 points. This is due to the agents do not implement a list to store all the coordinates of the snake body for each node expansion, and there is a possibility for the current node to have the same coordinates with the snake body that causes the game to end. Hence, it is suggested to implement a list to store the coordinates of the snake body for each node expansion so that the algorithm can suggest another path to reach the food in order to prevent the snake to bite itself and causes the game to end.

BFS algorithm is algorithm that works by traversing the tree in breadthwise where it expands the adjacent nodes in the same level before moving towards the next level. Thus, BFS algorithm ensures the shortest number of steps from the initial node to the goal node. In the snake game, BFS algorithm is implemented by using a First in First Out (FIFO) queue data structure to act as the frontier to store the nodes for expansion. The FIFO approach illustrates that the adjacent nodes are prior to be expanded before the next level nodes. Hence, it ensures that the shallowest solution is returned.

In term of completeness, BFS algorithm is not complete in tree search because it will lead to infinite looping at the repeated path due to its FIFO approach. Meanwhile, it might be complete in graph search if the goal node is at some finite depth. In the snake game, the graph search is applied to remove the nodes that create redundant paths at the generation of the nodes. Since the goal node (food location) is in finite depth, BFS algorithm has present in completeness that it is able to return the shallowest solution which is path with the shortest number of steps. In term of optimality, BFS algorithm does return the shallowest solution but it is not necessarily to be optimal and depends on the certain conditions. The conditions include the path cost is uniform or non-decreasing function of the depth of the node, if true, then yes for optimality, else no. In the snake game, as the path cost of all the nodes are uniform, the optimality of the solution is yes.

GBFS algorithm uses the concept of heuristic function and a priority queue. The heuristic function is used for cost estimation between the adjacent nodes of the current nodes to the goal node. In the snake game, the heuristic function is the Manhattan distance between the adjacent coordinates of snake coordinates to the food coordinates. The priority queue is used to sort the nodes with increasing order of cost. In the snake game, the priority queue is the frontier list that stores the child nodes with ascending order of evaluation. This makes the algorithm efficient in searching.

In term of completeness, GBFS algorithm is not complete in tree search as it will generate loopy path and can easily get stuck in loops. However, it is complete for graph search with finite spaces. In the snake game, graph search is applied to remove the nodes that creates redundant paths at the expansion of node. Hence, the answer is yes for the completeness of GBFS algorithm in the snake game as it guarantees to find the solution, which is the path to reach the food location. In term of optimality, GBFS algorithm is not optimal for both tree search and graph search as it does not guarantee to render the lowest cost solution. Hence, the answer is no for the optimality of GBFS algorithm in the snake game as the solution is not necessarily the shortest path to reach the food location.

5 Conclusion

In this report, we have presented two search algorithms in the snake game. We have analysed both algorithms and conducted experiments to study their performance.

Both algorithms have their own uniqueness in running the snake game. In the snake game, BFS is more optimal compared to GBFS as the shallowest goal node in the snake game is considered the shortest path because the path cost in this case is all uniform. Similarly, both algorithms can achieve completeness given that the state spaces are finite in the Snake Game. Even though both algorithms have their own steps to achieve the goal, but they can achieve similar results which is the success rate of 80 to 90 percent to reach at least 10 points for each game round. However, we anticipate the total points of the game will depend on the size of the maze as a bigger maze will more likely to accumulate more points. Performance of the snake game can be improved in our program by implementing more checking conditions to prevent the snake from hitting a wall or biting its own tail.