ARTIFICIAL INTELLIGENCE
CSC3206

ASSIGNMENT 1

NAME	STUDENT ID	Programme
TEOH SHI HONG	19072735	BSc(Hons) Computer Science
LAU WEI XUAN	19081199	BSc(Hons) Computer Science
SENG WEN LONG	20076105	BSc(Hons) Computer Science
RUSHIL BHATT	18043299	BSc(Hons) Computer Science

Lecturer: Dr Richard Wong Teck Ken

Due Date: 21th May 2021

I hereby state that this material is original and a result of my own effort. Any material copied from other sources will be given their due respect in the form of referencing and bibliography

Marks Award

_____

Introduction

Snake game is a classic computer action game, in which the player can control the snake's direction to eat the food and the length will increase depending on both conditions such as fixed or dynamic snake length. In this project, we have also developed two agents that have the objective in the game to find a path that leads the snake to the food (without hitting the screen border or biting itself).The snake should be able to find new paths whenever a new fruit pops after the previous one is consumed. Both agents address different search algorithms to find the fruit in the search space.If at any point of the game the snake hits an obstacle, like screen border or its own body, the game will be stopped immediately.

Challenges and Problem Solving

One of the challenges that we solved was the directions of the snake. We used coordinates to determine the direction of the snake moved which are north, west, east and south.

The most dominant step in the problem solving of our snake game is based on the following six elements:

- State: It describes the location of snake and food
- Initial State: Start from snake head as initial state
- Actions: Actions of snake either north, west, east and south
- Transition Model: It returns the resulting as per the given state and actions
- Goal test: It identifies whether snake have reached the food
- Path cost: It is determined by different search algorithms

Problem formulation

We assist the agent to find the food according to the two search algorithms which are uninformed search and informed search. For uninformed search, we used Breadth First Search (BFS), whereas for informed search we used Best First Search. Breadth First Search is a common search algorithm that radiates outwards in all four directions. It will evaluate all of a given node's neighbours before moving outwards.However, Best First Search, also known as greedy algorithm takes the least number of steps to obtain the minimal solution. Best First Search extends the node that is nearer to the goal state. Basically, we used the Manhattan distance to define how close the snake head is to the food. Every iteration of the algorithm runs until the path to reach the food is searched.

We have decided to implement these algorithms as follows:

Uninformed Agent

Here the Snake doesn't know the location of the Food. We use a BFS/DFS (in our case its – BFS). Data Structure used is a Queue. The head of the snake is pushed to the Queue and all the possible 4 directions are explored in uninformed search (as the snake doesn't know the location of food). After finding the location of the food we backtrack to the head and store the path. Now the snake moves on this path.

Informed Agent

Here, the Snake knows the location of Food beforehand. We have used the Best first Search algorithm with the Heuristic -> Minimum Remaining Manhattan Distance from the food.The head of the snake is pushed to the Queue. In this case not all the cells are explored, the neighboring cell which has the optimal Heuristic Value (here – min Manhattan distance from Food) is explored first.The formula used in Manhattan Distance is as follows:

Manhattan Distance – (Hx – Fx) + (Hy – Fy)

where H – location of HEAD of snake, F – location of FOOD

After finding the location of the food we backtrack to the head and store the path. Now the snake moves on this path.

Movement of Snake

After the path if found out from the head to the food. The snake moves as follows-Add the current path cell as the new head and remove the tail cell. In this manner the snake will move forward.This scheme can also be viewed as FILO (First In Last Out).

Explaining the algorithm

these are not algorithm

For our project, we apply the common algorithm in our project such as for, while and other statement. The following that will explaining the detail of algorithm and explaining their purpose.

```
def updateMaze(self, snake_locations, food):
    self.maze = [[ 0 for j in range(self.M) ] for i in range(self.M)]
    self.maze[food[1]][food[0]] = 'F'
    for s in snake_locations:
        self.maze[s[1]][s[0]] = 'S'
```

this is a function, not an algorithm

(Figure 1)

The purpose of this algorithm (Figure 1) is to update the maze and marks the current location of snake body and food. For example, the current location that consist of food inside the current node will call 'F' to state the status and snake body use 'S' to represent it. In the main search function, we are using the For loop for food the def updateMaze was used.

```
def moveSnake(self, snake_locations, path):

    for p in path:
        hx,hy = snake_locations[0][1], snake_locations[0][0]
        if p=='s':
            nx, ny = hx+1, hy
```

(Figure 2)

The implementation of this function (Figure 2) is to make the snake move forward after getting the desired path.

```
for p in path:
    hx,hy = snake_locations[0][1], snake_locations[0][0]
    if p=='s':
        nx, ny = hx+1, hy


        snake_locations.pop(-1)

        |
        snake_locations.insert(0,[ny,nx])

    elif p=='n':
        nx, ny = hx-1, hy
        snake_locations.pop(-1)
        snake_locations.insert(0,[ny,nx])
    elif p=='e':
        nx, ny = hx, hy+1
        snake_locations.pop(-1)
        snake_locations.insert(0,[ny,nx])
    elif p=='w':
        nx, ny = hx, hy-1
        snake_locations.pop(-1)
        snake_locations.insert(0,[ny,nx])
return snake_locations
```

(Figure 3)

In this loop (Figure 3), snake head will make a move and new head will be generated in the new node so that the tail of snake will need to chop off. In this case, if the snake moves to the one of the directions, then the algorithm will use if else to judge whether it moves and implements the insert and pop to the snake.

As above we apply the same concept in these search algorithms. After that, there are two different approaches that we apply in these search algorithms and their implementation are different from each other. They are breadth first search algorithm for uninformed search and best first search algorithm for informed search.

Implementation of Informed Search

The informed search that we assign is best first search (Greedy). The algorithm for implementing Best First Search uses the Heuristic which is the minimum remaining Manhattan distance from food eg.[|x2 -x1|+|y2-y1|]. First, we create a visitor array that can state the node that is visited. After that, we calculate the Manhattan distance from the food by this algorithm.

```
distance = abs(snake[0][0]-food[0]) +abs(snake[0][1]-food[1])
q = [ Node(snake[0],curDir, newNode, distance) ]
heap.heapify(q)
```

(Figure 4)

In this algorithm (Figure 4), we used the Manhattan distance as an approach for this algorithm. We enqueuing in the Q and the Q will be Min-Heap since that we need to always process the node which can find the nearest path to the food. If the snake finds the food, we store the path in the finalPath and start to find the next food that is generated in the maze. In the final part, we can calculate the Manhattan distance and push it to the Q means Min-Heap

```
distance = abs(newPoint[0]-food[0]) +abs(newPoint[1]-food[1])
heap.heappush(q, Node( newPoint, d[2], newNode, distance))
```

(Figure 5)

So that, we have generated the path from the snake head to the food and move forward the snake over that path.

Implementation of Uninformed Search

The uninformed search that we use is breadth first search algorithm. The method of implementing this search algorithm is using a queue as a list that keeps track of the node that is currently in the queue.

```
q = [ [snake[0],curDir, newNode] ]
```

(Figure 6)

This line will represent the enqueuing in the ‘q’ when start the Breadth First Search. In other words, it will take the front item of the queue and add it to the visited list.

```
for d in dirs:
    nx, ny = f[0][1] + d[0], f[0][0] + d[1]

    if 0 <= nx < self.N and 0 <= ny < self.M and (vis[nx][ny] == 0 or vis[nx][ny]=='F'):


        f[2]['actions'].append(d[2])
        exp += 1


        f[2]['children'].append(id)

        newNode = {
                "id":id,
                "state":""+str(ny)+","+str(nx),
                "expansionSequence": exp,
                "children":[],
                "actions":[],
                "removed":False,
                "parent":f[2]['id'],
                "path":f[2]['path'] + d[2]
            }
```

(Figure 7)

After that, the function work in this part. If the cell is valid and there is not contain food or visited, then the action and children will be updated and generate the information of search tree.

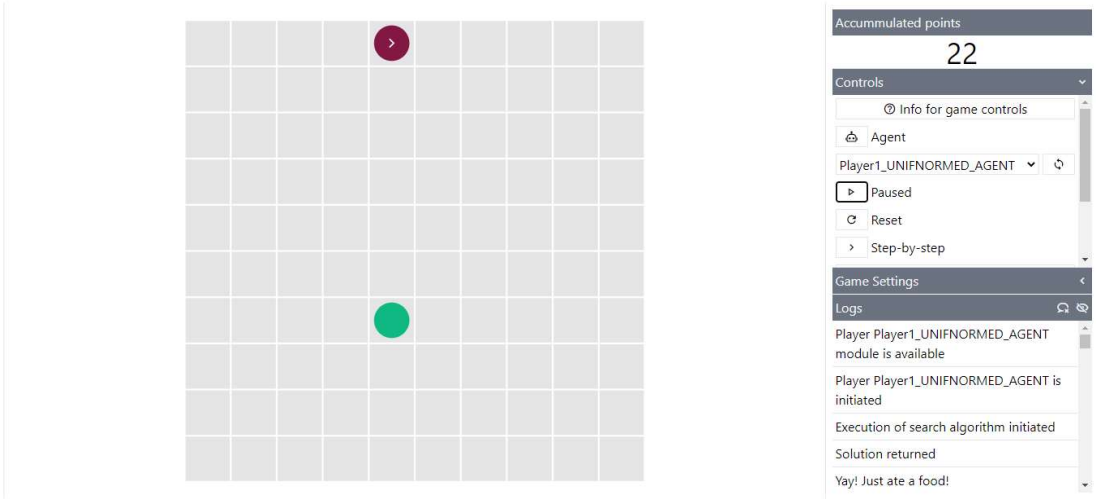
In this search algorithm, the purpose we mark the visited and not visited is to avoid the cycle and redundant move.

Results

We were able to achieve all the three challenges for both uninformed and informed search.The results of the searches can be seen below.

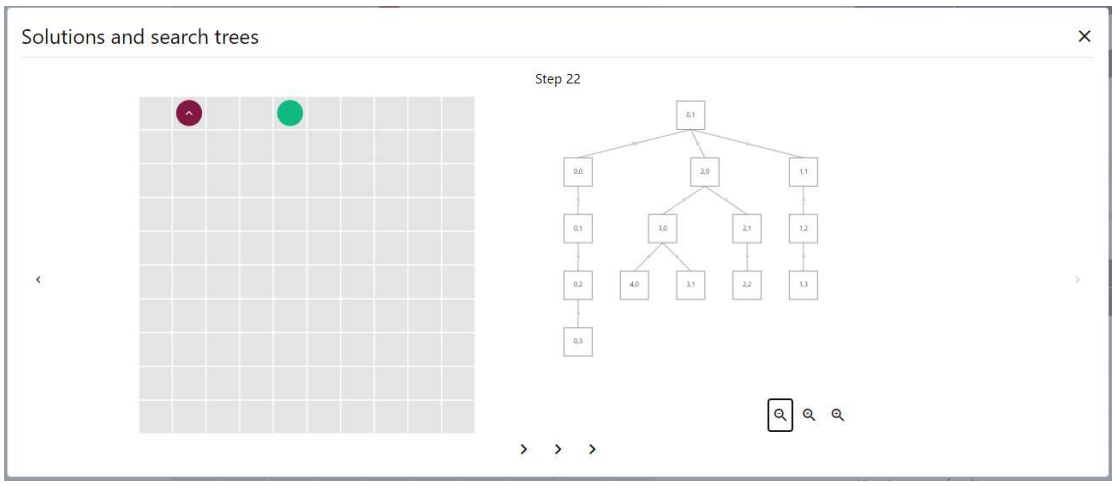
Uninformed Search Algorithm

a)One food at any time; Non-increasing snake length; Snake length of one; reaching at least 15 points.



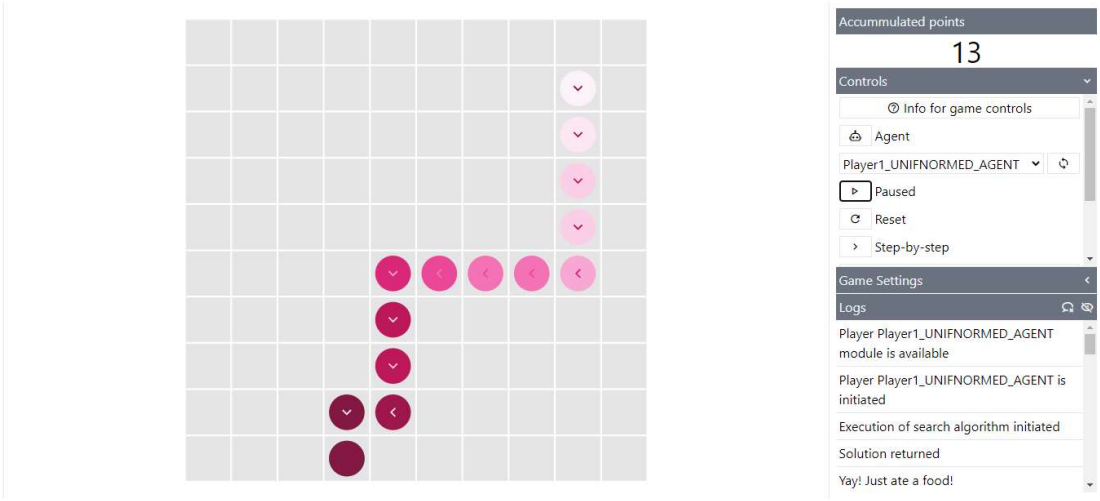
(Figure 8)

Solutions and search trees shown as below:



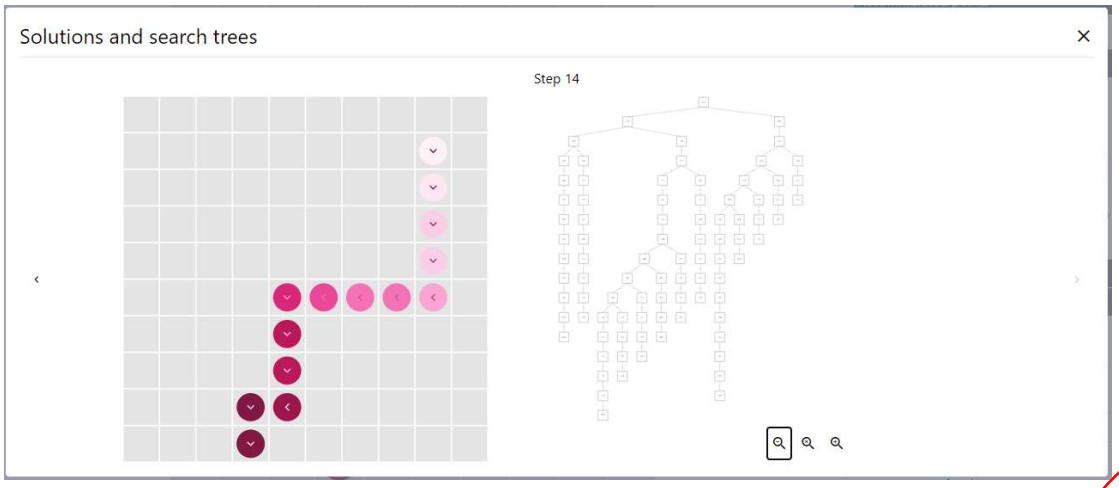
(Figure 9)

b) One food at any time; Increasing snake length with food; Starting snake length of one; reaching at least 10 points.



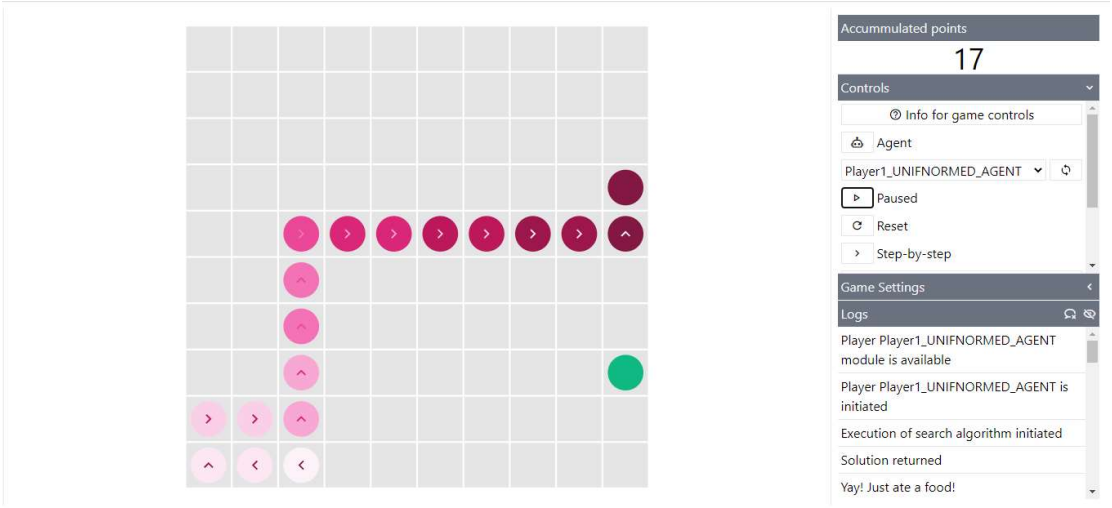
(Figure 10)

Solutions and search trees shown as below:



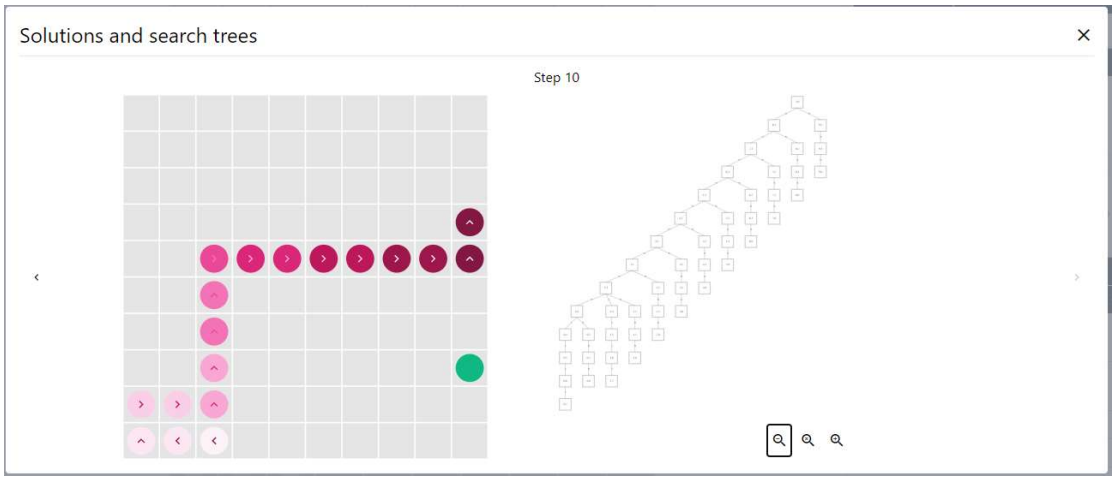
(Figure 11)

c)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.



(Figure 12)

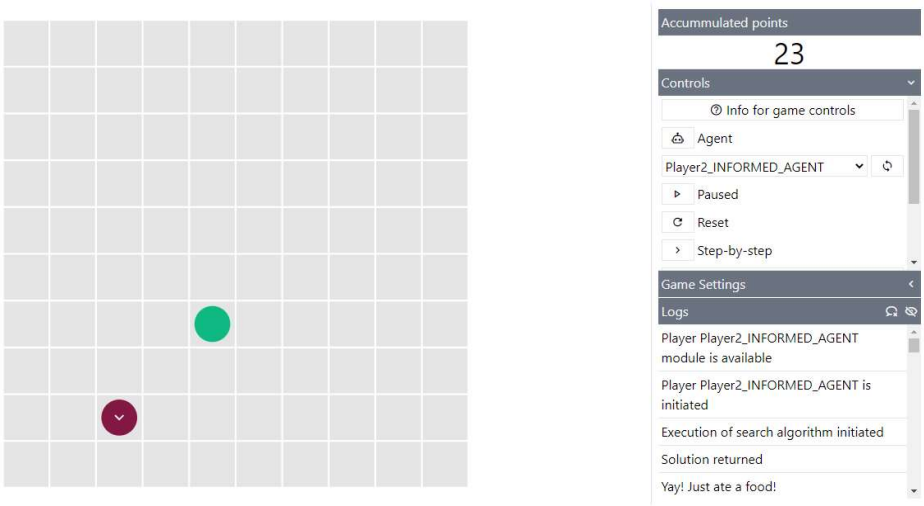
Solutions and search trees shown as below:



(Figure 13)

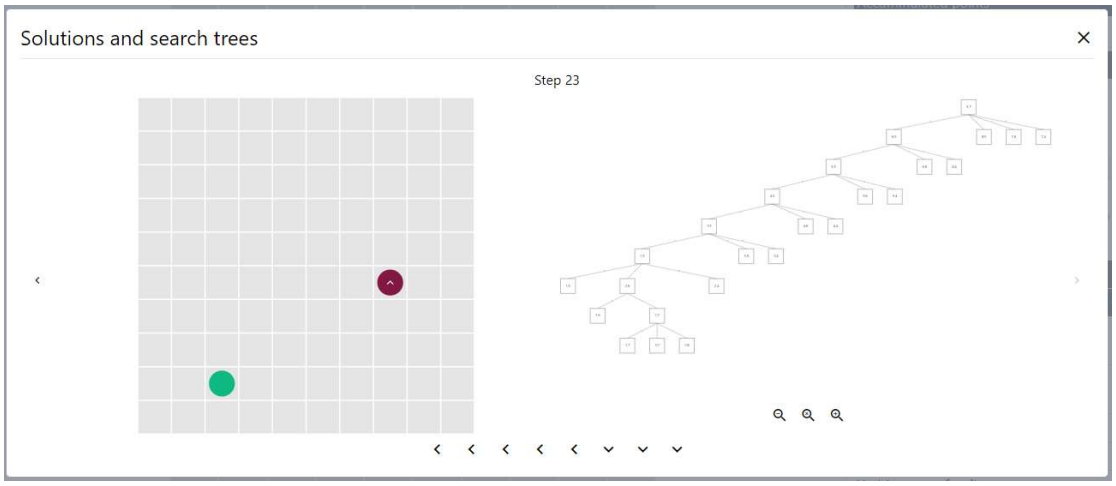
Informed Search Algorithm

a)One food at any time; Non-increasing snake length; Snake length of one; reaching at least 15 points.



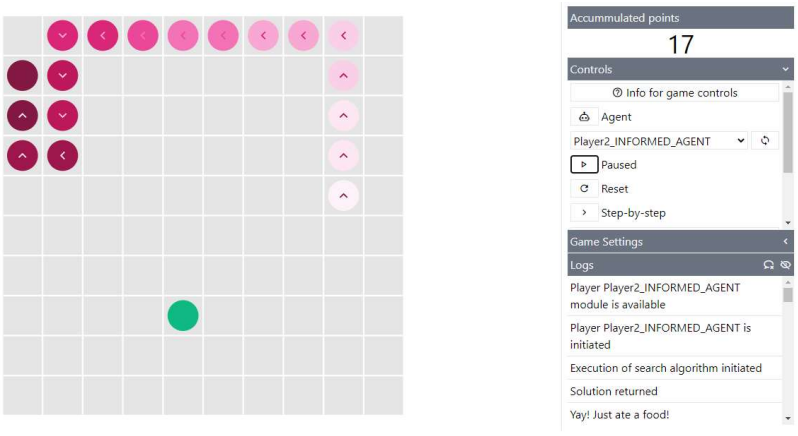
(Figure 14)

Solutions and search trees shown as below:



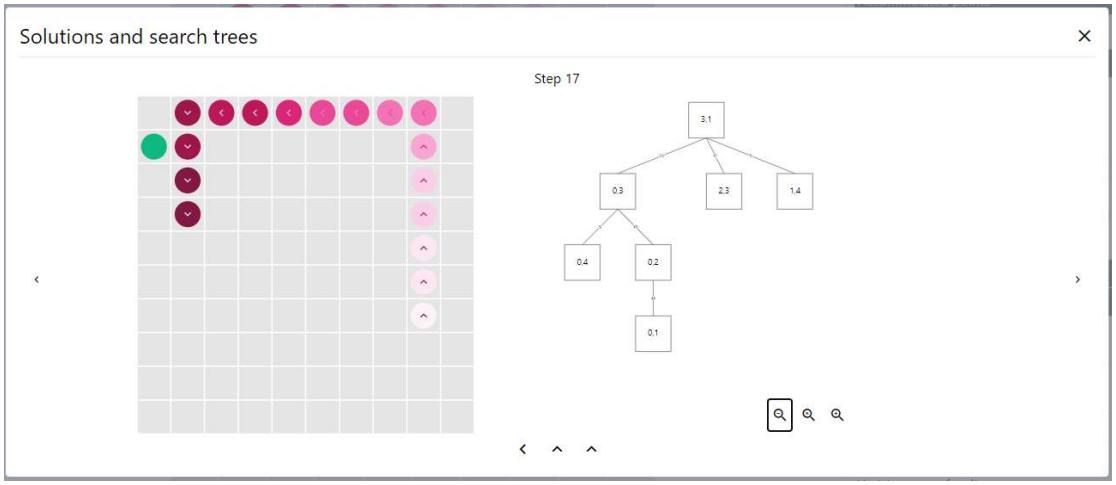
(Figure 15)

b)One food at any time; Increasing snake length with food; Starting snake length of one; reaching at least 10 points.



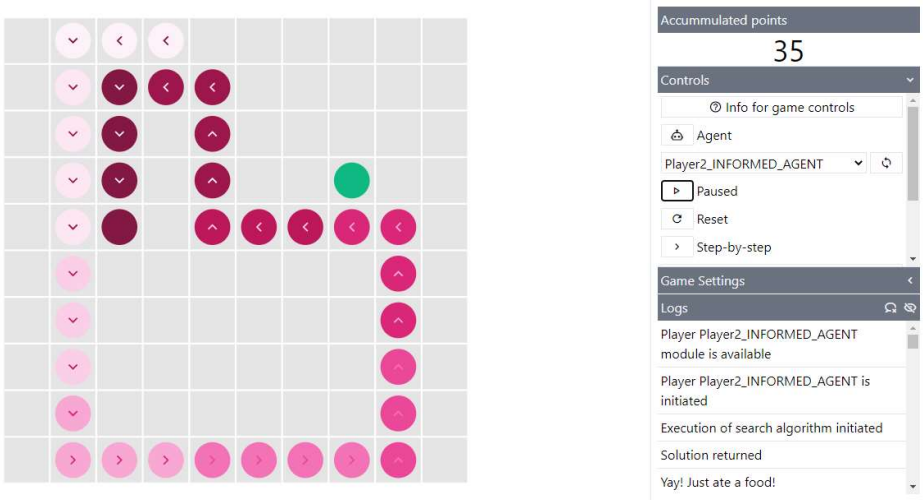
(Figure 16)

Solutions and search trees shown as below:



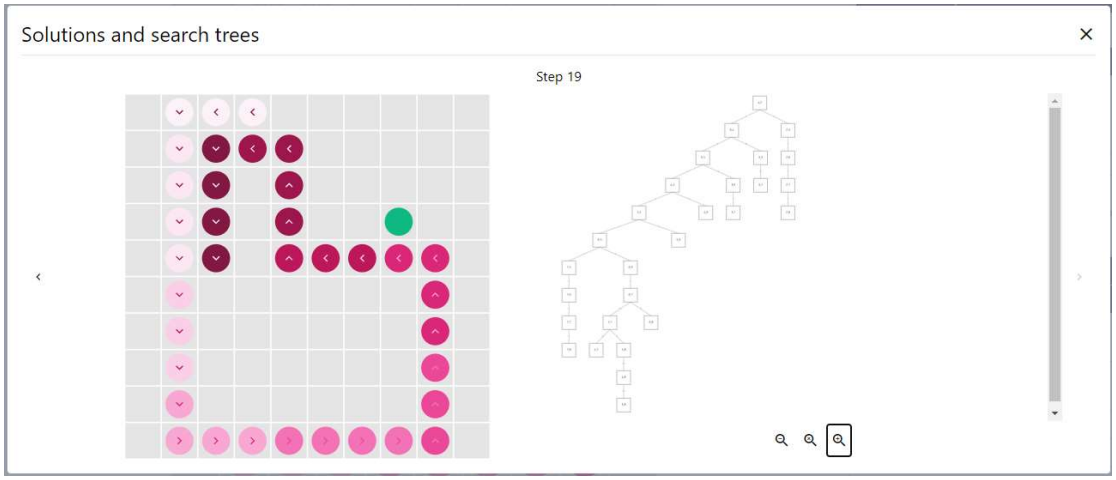
(Figure 17)

c)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.



(Figure 18)

Solutions and search trees shown as below:



(Figure 19)

Efficiency of the algorithms

In order to measure the performance of the algorithms,we measured the number of points that the snake is able to achieve in 30 seconds for both static and dynamic snake lengths.We carried out 4 trials and then calculated the average amount of points the snake is able to achieve for both the algorithms.The table showing the data is as follows.

Trials	1		2		3		4	
Snake length	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static
Breadth-first search	17 points	26 points	19 points	23 points	22 points	24 points	18 points	21 points
Greedy-best first search	22 points	20 points	19 points	23 points	20 points	24 points	19 points	22 points

i)The average points for breadth first search for dynamic length is 19 points while the average points for static length is 24 points.

ii)The average points for greedy best first search for dynamic length is 20 points while the average points for static length is 23 points.

Discussion

i)From the average points obtained,it is seen that the snake is able to achieve more points for static length as compared to dynamic length for both the algorithms.This is because the paths become more and more convoluted when the snake size increased in length.The snake's body became an obstacle for the algorithm to path around.This has resulted the snake to achieve lower number of points in a fixed amount of time in dynamic snake length.

ii)Based on the average points,it can be seen that breadth first search for dynamic snake length has a lower number of points achieved compared to average points for dynamic snake length in greedy best first search.This means that the greedy best first search is more efficient compared to breadth first search for dynamic snake length.

iii) Based on the average points,it can be seen that breadth first search for static snake length has a higher number of points achieved compared to average points for static snake length in greedy best first search.This means that the breadth first search is more efficient than greedy best first search for static snake length.

iv)The efficiency of algorithm here is measured by the amount of points the snake scores in a fixed amount of time,30 seconds.

References

1. JavaTPoint. (n.d.). *Uninformed Search Algorithms*. www.javatpoint.com. https://www.javatpoint.com/ai-uninformed-search-algorithms.
2. Darmakusuma, M. D. (2019/2020). Greedy Best First Search in Automated Snake Game Solvers. *Makalah IF2211 Strategi Algoritma,* 1-6. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Makalah/stima2020k3-032.pdf
3. Admin. (2020, November 10). *Problem-solving in Artificial Intelligence*. Tutorial And Example. https://www.tutorialandexample.com/problem-solving-in-artificial-intelligence/#:~:text=According%20to%20computer%20science%2C%20a,focuses%20on%20satisfying%20the%20goal
4. Tutorials. (2019, Jan 1). *Snake AI in Pygame.* https://pythonspot.com/snake-ai-in-pygame/