CSC3206 Artificial Intelligence 202104 Assignment 1 Report

Group Members:	Leo Paul Larkin	18088971
	Thevendren Paranthaman	18067074
	Ng Jun Ren	17098302
	Fun Kok Fui	18047829

Problem Description:

Given a snake game, write two program codes that each represents informed search and uninformed search respectively for the snake to reach the food and be able to perform repetition to continue eating the snake foods. A search tree is generated based on the algorithms created that is used to show how the programs searched for the path. The condition for the snake could be non-increasing length which is also known as static snake length or increasing snake length which is known as dynamic snake length. ✓

Explanation of problem formulation:

The problem for this program is that how can the snake reach and eat the food in the snake game by using different search algorithms? The search algorithms can be classified into two main categories, which are known as informed search and uninformed search. There are more than one searching algorithms in each of the categories. Therefore, it is important to decide which searching algorithms are the best to be used. Other than searching for a path to eat the food, there are several other problems that need to be considered. First, the maze size has a limited size, the program will need to ensure that the snake is not going to hit the maze border. The program will also need to make sure that the first direction of the solution will not be the opposite direction of the current direction as it will not allow the snake to move. Moreover, it is necessary to check for the frontier and visited lists to ensure that the program does not execute redundant steps which reduce the performance of the program.

In this program, the snake location will be the initial state for the search problem while the food location will be the goal state. The action that can be taken by the snake is moving to its adjacent squares in the maze, which are the top, bottom, left and right side of the snake location. The path cost can be measured by calculating the steps needed for the snake to reach the food where one unit of cost is assumed when the snake moves into each adjacent square.

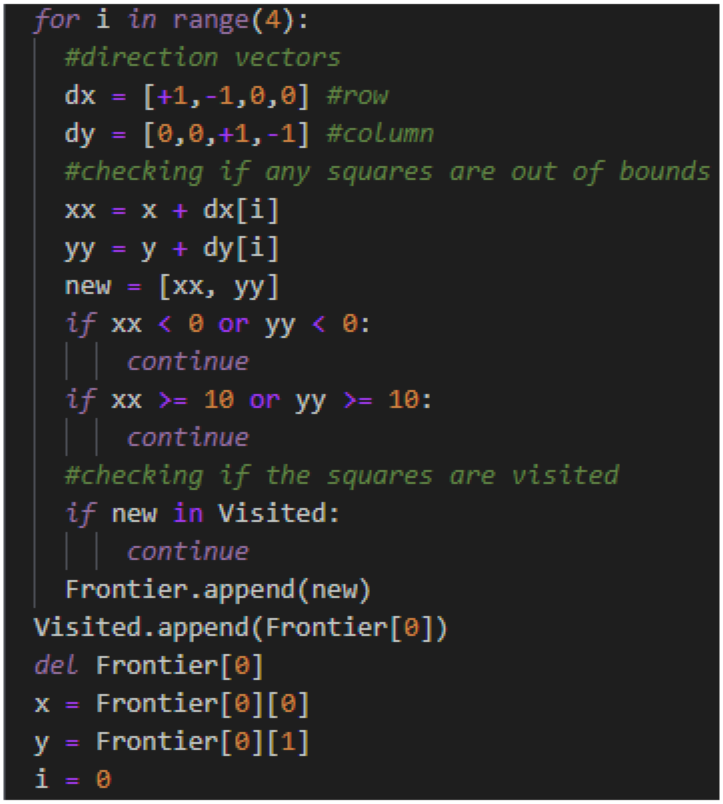
Search Algorithm Implementation:

Uninformed search: Depth first search

Explanation:

For the uninformed search algorithm, depth first search (dfs) was chosen over breadth first search (bfs) because it will consume very less memory space in finding the goal. It will reach the goal node in a shorter time period than bfs if it travels in the right path. As depth first search might find the goal on its first cycle (if the goal was on the same row or the same column as the initial snake location), it would also take much longer if the goal is further away from the initial state and in the different row or column as the initial snake location.

The dfs algorithm in the explore function uses direction vectors to expand and check the next squares by adding and subtracting on the x and y axis by 1, the values are then tested for being in bounds of the grid size and then checked if they have been visited before being appended to the Frontier list.



```
for i in range(4):
    #direction vectors
    dx = [+1,-1,0,0] #row
    dy = [0,0,+1,-1] #column
    #checking if any squares are out of bounds
    xx = x + dx[i]
    yy = y + dy[i]
    new = [xx, yy]
    if xx < 0 or yy < 0:
        continue
    if xx >= 10 or yy >= 10:
        continue
    #checking if the squares are visited
    if new in Visited:
        continue
    Frontier.append(new)
Visited.append(Frontier[0])
del Frontier[0]
x = Frontier[0][0]
y = Frontier[0][1]
i = 0
```

In the run function of the algorithm, the current location is obtained and put into the Frontier. A while loop is used to loop the search algorithm until the goal state is found, this is confirmed by checking the location of the Frontiers against the food locations.

Finally, the path and directions must be found to feed the solutions list. In order to achieve it, a function is created that takes the differences in x and y between the food location and initial location. The x and y values are tested if they are greater or less than zero to determine if the snake has to move north, south, east or west.

Informed search: Greedy best first search

Explanation:

This algorithm is based on both the depth first search algorithm and breadth first search algorithm. It carries over the path finding function from the dfs algorithm, bfs algorithm and their expansion algorithm. This means that it can switch between dfs and bfs and this makes it a more efficient algorithm.

The main difference is that at the end of expanding and checking adjacent squares the same way as in the bfs or dfs algorithm, the difference of each square from the goal location is taken as an estimate and the squares are sorted into ~~their~~ lowest estimated cost to the goal. This square is then appended to the Frontier. The checking mechanism if the Frontier is the goal is the same as bfs or dfs.

Result of Algorithm:

The algorithms for depth first search and greedy best first search are implemented in the program. However, no result is produced from both algorithms as there is an infinite loop that causes no definite solution being generated.

You could pause at any point to obtain the result. Result is not just when it stops but also about what you get in the process of running it.

Evaluation of Player/Agent performance:

Depth First Search:

Depth first search (dfs) was chosen over breadth first search (bfs) due to it being easier to implement. However bfs would have been better due to it having a more consistent time in finding the goal. As depth first search might find the goal on its first cycle (if the goal was on the same row or the same column as the initial snake location), it would also take much longer if the goal is further away from the initial state and in the different row or column as the initial snake location.

Greedy Best First Search:

Greedy best first search is a variant of best first search. Compared to its other variant, A* algorithm, it has the limitation of loopy paths which may lead to no-end solution.

Discussion:

Although the algorithms did not work on the server, there will be some screenshots depicting the algorithms being used in a test environment with some test values used to test the operation of the algorithm without the use of dictionaries.

would be good to evaluate what's the issue. Is your code not formatted correctly?

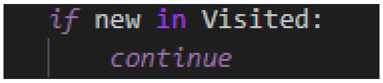
Below is a copy of the results using the initial snake location of 0,5 and the food location of 1,4 in the dfs algorithm:

```
['e', 's', 's'] [{}, {'id': 1, 'state': [0, 5], 'expansionsequence': 1, 'children': [], 'actions': [['e', 's', 's']], 'removed': False, 'parent': None}, {'id': 1, 'state': [0, 5], 'expansionsequence': 1, 'children': [], 'actions': [['e', 's', 's']], 'removed': False, 'parent': None}]
```

Problems faced and their Solutions:

This section will discuss the problems faced after the initial design of the algorithms. This section will be split into two parts, one for the depth first search algorithm and one for the greedy best first search algorithm.

For breadth first search the first problem encountered was in expanding the squares and validation of grid squares. Particularly ensuring already visited values were not fed back to the Frontier list. Initially using a nested if statement in a for loop with the continue keyword being used to skip values that were invalid, the for loop was removed and the if statement changed to search the visited squares.



```
if new in Visited:
    continue
```

Another issue that depth first search presented was the creation of the solution that would accurately represent how the search traversed the grid. Initially the solution was created by subtracting the location of the food from the current location then finding the directions in two straight lines from the current location to the food. This was unable to be solved.

Generating the search tree was an issue that was discovered late in the creation of the algorithm, in the next section the use of lists instead of nodes and the issues they presented is further expanded on. Using lists to keep track of the parents and child of each square proved to be unachievable with the time left.

Greedy best first search generally used the same algorithm as dfs but had the addition of sorting the expanded squares by distance, which was calculated by total x and y to the food, before appending it into the frontier list. Which implies that this algorithm had the same issues faced by the depth first search algorithm. This method didn't produce the expected results, and was not solved.

Limitations of Algorithms:

The major limitation of both algorithms used was the employment of lists to keep track of all the processes instead of nodes. Nodes, although more complicated to implement, would have enabled the algorithms to be more efficient and easier to work with when implementing any additional functions.

Lists have caused a lot of issues and complications when carrying out the different functions of the algorithms, mainly when generating the search trees, obtaining the values such as parents and children in particular.

Despite initially writing the functions for exploring the squares and finding the solution, these functions were later added into the run function due to not being able to call the functions. This makes the program very stiff as any changes that need to be made would have to be made to the main run function.

you are talking about the limitation of your coding style, not the limitation of the algorithm.

Possible Solutions to unsolved issues and Limitations:

Starting the design and programming earlier to provide more time to learn the best approach to the algorithms. Furthermore learning how to use nodes for this assignment would have overcome a lot of these issues that using lists created.

Reference

Breadth First Search grid shortest path | Graph Theory - YouTube. (n.d.). Retrieved May 21, 2021, from https://www.youtube.com/watch?v=KiCBXu4P-2Y