



CSC3206 ARTIFICIAL INTELLIGENCE ASSIGNMENT 1

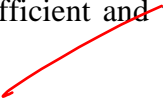
Group Name: **Smart AI**

Group Members

Theysigan	17072182
Raguram	18068718
Puvanesh	16029423
Muhammad Mahad	19053131

Problem Description

This is a snake game which needs to be solved by creating agents/players to play the game using search algorithms. We are going to be using 2 players, 1 player will be using uninformed search algorithm and another player will be using informed search algorithm. First of all, searching is a method of finding a sequence of steps needed to solve any sort of problem. In addition, an uninformed search algorithm is basically a searching technique that has no additional source of information on getting the fastest route or more like what is the distance from the current state to the goal state. Whereas, an informed search algorithm is a technique which has additional information like knowing the estimated distance from current state to the goal state. Uninformed search algorithm has no knowledge, very low on efficiency, costs high, slow in speed on finding the goal state. But, an informed search is the vice versa of uninformed search because uses knowledge to find the goal state, highly efficient and very fast which consumes less time and cost



Explanation of Problem ~~Formulation~~

The following problem is solved using uninformed and informed search algorithms to capture and collect points. For the respective uninformed search algorithm, breadth first search is implemented in this case. Breadth first search is an algorithm used for traversing search trees and data structures. It begins at the root node and traverses across neighbour nodes traversing to the next depth level. It uses the opposite strategy of depth-first search, which instead explores the node branch as far as possible before being forced to backtrack and expand other nodes. It explores and traverses search tree by expanding nodes in the first level and then expanding it in the second level until the objective is reached, which is also known as traversal technique. This guarantees the most effective solution by finding out all of the nodes for all solution. Queue data structure is utilized to store values. The main advantage of breadth first search is that it guarantees path to reach objective node.

Greedy best-first search algorithm always chooses the path which appears best. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function for the determination of distance between the nodes. Heuristic search iteratively

calculates the distance between pair of nodes in state space tree. This function evaluates the next favourable move or path in order to win a game. Algorithm also uses dual lists which is the open and closed list. The open list are visited in the state space when an open node is discovered the least shortest path then is moved to the open node linked with it. The main disadvantage is that it is not optimal.

Search Algorithm Implementation

The breadth first search algorithm was implemented for the agent using the uninformed search while the greedy best first search algorithm was used for the agent using the informed search.

Uninformed Search

For the implementation of breadth first search, the run function will be called hence the maze size, initial location of snake and the location of the food will be obtained. An empty search tree and solution list will also be created while directions will be defined as possible movements for the snake.

A visited matrix will be defined as a 2d list which represents the unvisited cells as false and the visited cells as true hence the source node will be marked true. The current ID variable will be initialized as 1 and the current expansion sequence variable will be initialized as 1. The first entry will be appended into the search tree with id = current ID, state = source node, expansion sequence = current expansion sequence, children = [], actions = [], remove = false, parent = None.

Index variable will be initialized as 0 and a loop of each entry of the search tree will begin whereby:

If the current entry of the search tree has a removed = true value, the current entry will be skipped to the next one

The current node from the search tree will be obtained and parent ID variable will be initialized as the id of the current search tree entry.

If the current node is the destination, then:

While this parent node index is not None,

The parent node will be obtained from the parent index of the search tree

If statements were used to append the location as solution based on the location of the parent node.

directly starting the code in sentences doesn't mean explaining

The current node will be set to be the parent node and parent index will be updated using search tree parent.

Initialize the childs list as the 4 possible adjacent cell of the current node and loop through each child:

If the current child is within the same maze:

It will be added into the list of children of the parent entry and hence, current ID and seq will be incremented. A new entry in the search tree will be appended with id = current ID, state = current child, expansion sequence = seq, children = [], actions = [], remove = false, parent = None

The parent entry in search tree will be updated with the current child and its direction.

If the child is already visited, expansion wont happen and hence decrement the seq and update the current child entry in search tree with expansion sequence = -1 and remove = true

Else, mark the current child as visited.

Then the solution list and search tree will be returned as the results.

Informed Search

For the implementation of the Greedy best first search, a Heuristics function was used to come up with a heuristics to follow in order to use this specific algorithm.

The Heuristics function will be called and it'll take the maze size as well as the location of the food in it's input arguments. It then will define a heuristics matrix as a 2d list of zeros as well as a visited matrix as a 2d list of false zeros. A queue will be initialized an the location of the food will be inserted. Now the location of the food will be marked as visited.

The queue will be looped through:

The current node from the front of the queue will be popped from the queue. Childs list will be initialized as the 4 possible adjacent cells of the current node.

Each child will be looped through:

If statement was used to check if the current child is within the maze and not visited before, if so, the cost of visiting the current child will be updated as (cost of visiting its parent) +1 . The child will then be appended into the que and will be marked as visited.

The calculated heuristics will be returned as result.

What's the actual heuristic function you use?

The run function will be called and maze size, location of food and snake's initial location will be obtained. An empty search tree and solution list will be created. The Heuristics function will then be called and it will return the cost to reach from any cell in the maze to the destination cell. Then the directions will be defined as possible movements for the snake.

A 2d list will be defined as a visited matrix which represent the unvisited cells of the maze as false and visited as true. The source node will then be marked as visited and the Current ID variable as well as the current expansion sequence variable will be initialized with the value 1

The first entry will be appended into the search tree with id = current ID, state = source node, expansion sequence = current expansion sequence, children = [], actions = [], remove = false, parent = None

A heap will store the cost and index of the nodes in ascending order. So the first element in the heap will always have the minimum cost to reach the destination. Hence, at all given times, only the first entry of the heap will be used as the current node.

An infinite loop will begin:

The first entry from the heap will be obtained as the current entry

If the current entry of search tree has the variable removed to be true, this entry will be skipped and the next entry will be taken.

The current node from the search tree will be obtained and parent ID will be initialized as the id of the current search tree entry.

If the current node is the destination, then:

The parent node will be obtained from the parent index of the search tree

If statements were used to append the location as solution based on the location of the parent node.

The current node will be set to be the parent node and the parent index will be updated using search tree parent.

The solution list will be reversed to get the direction from source to destination and break from the loop

Childs list will then be initialized as the 4 possible adjacent cell of the current node and each child will be looped through:

An if statement is used to check if the current child is within the maze and if so, it will be added to the list of children of the parent entry and hence, Current ID and sequence will be incremented. A new entry will be appended in the search tree with id = current ID, state = current child, expansion sequence = seq, children = [], actions = [], remove = false, parent = None

The parent entry will be updated in search tree with the current child and its direction

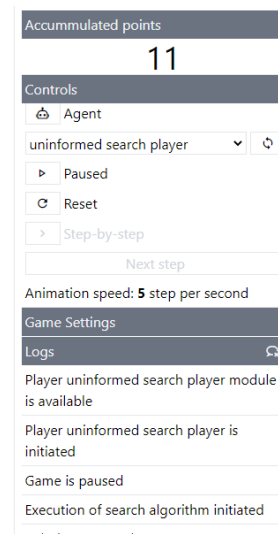
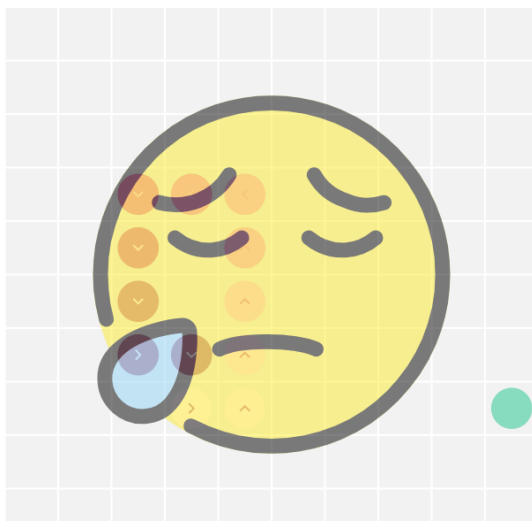
If the current child has been visited before:

No expansion will happen again and hence, sequence will be decremented and the current child entry will be updated in the search tree with expansion sequence = -1 and remove = true

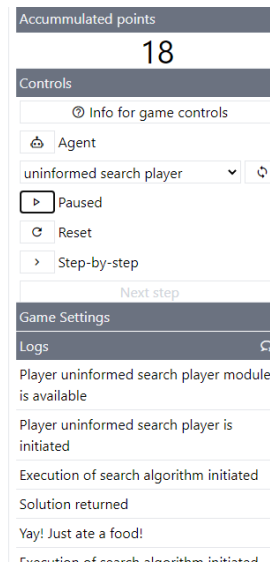
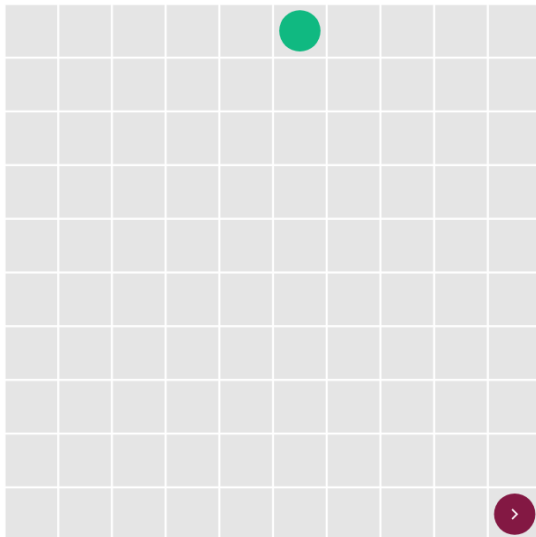
Else, the current child will be marked as visited and the current child will be pushed to the heap.

The solution list and search tree will then be returned as result.

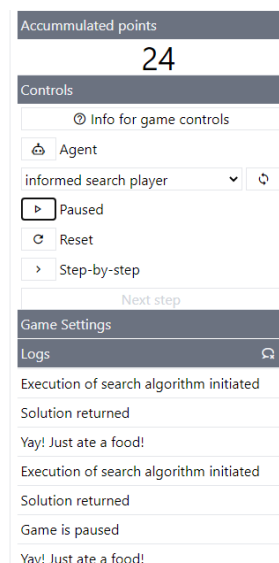
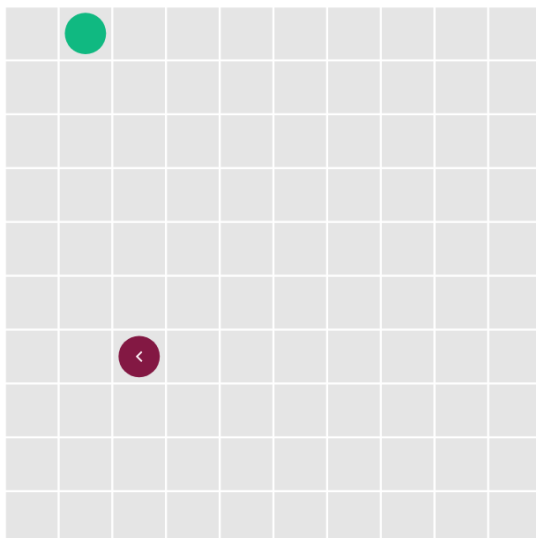
Results



This is a screen capture of the attempt for running the snake game with the agent using the uninformed search method and having a dynamic snake length. The snake has hit itself after collecting 11 points and this is one of the drawbacks of the algorithm as in the uninformed search, the algorithm does not necessarily take the shortest route to the goal and hence might expand one of the tiles that contains its own body on.



This is a screen capture of the attempt for running the snake game with the agent using the uninformed search method and a fixed snake length. A point tally of 18 was achieved and the snake did not manage to hit the walls and did not hit itself as the length is fixed.



This is a screen capture of the attempt for running the snake game with the agent using the informed search method. A point tally of 24 was easily achieved as this algorithm takes into account a certain heuristic function to calculate the path length to the goal. Hence moves made by the snake were always the cheapest moves according to the heap that was set to hold each possible child nodes in an ascending order of path cost.

Performance of the Agents

In this section we will be evaluating the performances of the two uninformed and informed agents/players in the game.

The search algorithm used for uninformed agents/players is the breadth first search algorithm. In breadth first search algorithm, all adjacent nodes are visited in a level before moving to a different level and each node is enqueued at least once during the traversal. Firstly, in an event of a worst-case scenario where the head of the snake and the food is at opposite corners of the game board the algorithm will traverse through all the nodes of the graph/tree until it finds the destination. Secondly, breadth first search algorithm does not have or need a weighted graph/tree to traverse so the algorithm will travel through all the nodes without knowing the most time/cost effective route to take. For this reason, breadth first search is a very costly and time consuming algorithm. However, breadth first search algorithm is better than depth first search algorithm because it is a complete algorithm. Breadth first search algorithm always finds the solution and is not trapped in searching useless paths or loops.

The search algorithm used for informed agent/player is the greedy best-first search algorithm. In greedy best-first search algorithm, the algorithm expands the node which is closest to the goal node and the closest cost is determined by setting up a heuristic function that makes a weighted graph/tree to estimate cost. Firstly, in an event of a worst-case scenario where the head of the snake and the food is at opposite corners of the game board the algorithm will not have to travel through all the nodes instead the algorithm expands the nodes that are closest to the goal node using the weighted graph/tree generated by the heuristic function. For these reasons greedy best-first search is more time and cost efficient than breadth first search and depth first search. However, greedy best-first search is not an optimal algorithm. The performance of the algorithm depends on how well the heuristic function is designed. In certain cases, the heuristic function can generate wrong results which can cause it to behave like an unguided depth first search and get trapped in loops.

When we compare the performances of the two algorithm, we see that the time complexity for breadth first search is $O(n^2)$ and the time complexity for greedy best-first search is $O(n * \log(n))$. Due to this we can determine that greedy best-first search is a better performing algorithm than breadth first search. Both the agents will take similar time to reach the goal for a small board size. However, if we increase the board size, an uninformed agent will take more time than an informed agent to reach the goal.