## Artificial Intelligence Course

## Project 1: Search in Pacman

|  |  |  |
| --- | --- | --- |
| First name | Last name | Student number |
| Atte | Jauhiainen | 2433947 |
| Ville | Karsikko | 2463933 |
| Mikko | Rytilahti | 2425560 |

### Comments about the assignment (if you have)

|  |
| --- |
|  |

### Question 1: Finding a Fixed Food Dot using Depth First Search (3 points)

### Explain the data structure that you used and also the way you keep the trace from the starting state to the end state …

### When implementing DFS the obvious choice would be to use a recursive call of DFS to the children nodes of the start node. However, in this implementation we were supposed to use a stack structure when the last node in gets taken out first. The path information from the start state to finish is wrapped in the object that contains both the location of a single point and the instructions how to get to that point from the start.

### Question 2: Breadth First Search (3 points)

### Explain the data structure that you used and also the way you keep the trace from the starting state to the end state …

### Implementation of DFS and BFS are essentially the same. The biggest difference here is that instead of a stack we used a queue as a fringe. The way we kept information about the path was identical as in DFS implementation.

### Question 3: Varying the Cost Function (3 points)

### Explain the data structure that you used and also the way you keep the trace from the starting state to the end state …

### We used PriorityQueue as the data structure since it lets us sort the queue according to cost. Otherwise the implementation is the same as DFS and BFS, just adding the accumulated cost to every node.

### Question 4: A\* search (3 points)

### Explain the data structure that you used and also the way you keep the trace from the starting state to the end state …

### It is the same as UCS, but with distance added to the cost.

### Question 5: Finding All the Corners (3 points)

### Explain the new state representation….

### We added a list of visited corners into the state tuple. Comparing this to the corners list in the problem class, we can check if goal is reached. In getSuccessors() we just check that the successors are not in a wall and again add the corners as a list into every state node.

### Question 6: Corners Problem: Heuristic (3 points)

### Explain your Heuristic function …

### We build a list of unvisited corners, then loop through found corners and calculate the Manhattan distance for each corner, which is a distance between two points in grid environments using only horizontal and vertical movement. Calculated distances will be added to another list, and then we choose the lowest/minimum distance for each corner from that list. We add costs to total\_cost and then continue the loop with selected closest corner as the new current\_state. Finally return the total\_cost which is a sum of all the minimum distances, and use that as the heuristic.

### Question 7: Eating All The Dots (4 points)

### Explain your new state representation and the heuristic function…

### No new state representation except for handling the foodGrid as a list. The function was basically done by dumb luck, just returning the longest distance from current position to a food dot (0 if no food). We used Manhattan distance at first but noticed the mazeDistance()-function and decided to try that. Would have probably gone for something a bit more complicated if the autograder had not given 3 / 4 points on the first try. This heuristic takes ages to run but gets a pretty much optimal path I think.

### Question 8: Suboptimal Search (3 points)

### Explain how did you do the suboptimal search …

### Pretty much just tried how the already implemented search algorithms performed; BFS seemed to do the trick.