Dungeon is a game that was custom designed for application of AI strategies for educational purposes.

The Classical setting of this game is that , the player (a treasure hunter ) navigates through a dark cave filled with lions and traps to find the gold hidden within. The only perception that the player receives at each position he reaches is whether the position he is standing at is roared or stinky or clean or shiny. These serve as indicators of adjacent positions for the player

Indicators :

Roar : indicates presence of lion in one of the adjacent positions

Stinky : indicates presence of trap in one of the adjacent positions

Clean : indicates that all adjacent positions are safe

Shiny : indicates presence of gold in one of the adjacent positions

((Note : Please don’t tab out of the window when the UI for the game is running))

Project 1 : Search

Overview :

In this project your goal is to implement search strategies such as Breadth first search, depth first search, A\* search etc. to the agent of the game to help it navigate through the mazes in the game.

Files to read through once :

game.py

Search.py

Files to edit :

Search.py

Priority queue data structure from util.py

Question 1 :

Breadth first search strategy explores shallowest nodes first. It uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

The Breadth first search function is left empty in Search.py . Implement Breadth first search in Search.py to help the agent navigate through the shortest path through the maze to the treasure.

Write your implementation in bfs function specified in Search.py

Use these three commands to see the Gameplay :

python dungeon.py -p agent -method bfs -graphics True -world smallmaze.txt

python dungeon.py -p agent -method bfs -graphics True -world Medmaze.txt

python dungeon.py -p agent -method bfs -graphics True -world Bigmaze.txt



Question 2 :

Depth first search strategy starts at the root (initial position )and examines each branch as far as possible before backtracking

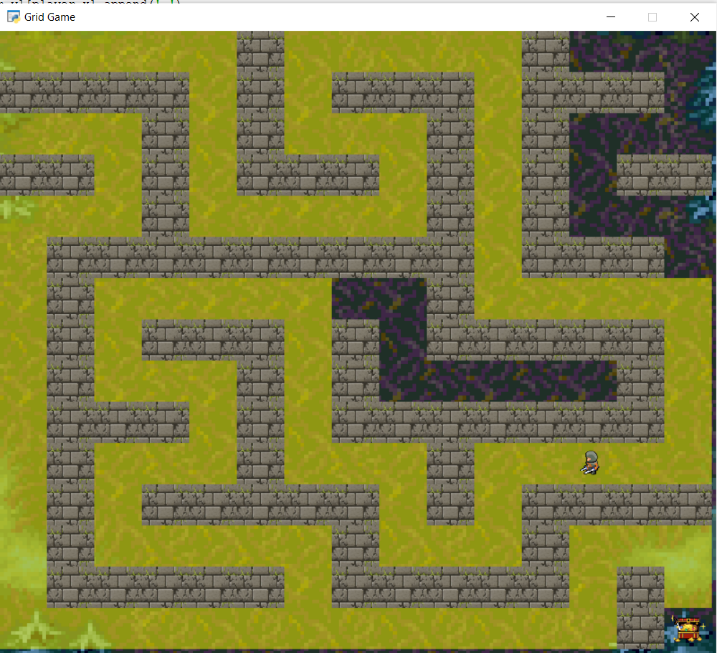
Implement Depth first search in Search.py to help the agent navigate through the maze to the treasure.

Write your implementation in dfs function specified in Search.py

python dungeon.py -p agent -method dfs -graphics True -world smallmaze.txt

python dungeon.py -p agent -method dfs -graphics True -world Medmaze.txt

python dungeon.py -p agent -method dfs -graphics True -world Bigmaze.txt



Question 3:

Uniform cost search strategy uses a priority queue data structure that uses the cost of attaining the next node as the value for ordering the nodes in the queue. This strategy always selects to the position with the least cost as successor.

Implement Uniform cost search in Search.py to help the agent navigate through the maze to the treasure. Write your implementation in ucs function specified in Search.py

python dungeon.py -p agent -method ucs -graphics True -world smallmaze.txt

python dungeon.py -p agent -method ucs -graphics True -world Medmaze.txt

python dungeon.py -p agent -method ucs -graphics True -world Bigmaze.txt



Question 4:

A star search strategy uses a priority queue that uses the specified heuristic of the next state as the value for ordering the nodes in the queue. This strategy always pops out the state with the least heuristic value as successor.

A\* takes heuristic function also as an argument. The outline of heuristic function is specified in Search.py (takes two inputs : the state and search problem). Implement Uniform cost search in Search.py to help the agent navigate through the maze to the treasure.

Write your implementation in astar function specified in Search.py

python dungeon.py -p agent -method astar -graphics True -world smallmaze.txt

python dungeon.py -p agent -method astar -graphics True -world Medmaze.txt

python dungeon.py -p agent -method astar -graphics True -world Bigmaze.txt

