CS 331: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

ASSIGNMENT 2:

Total Marks: 100

Instructor: Dr. Mian Muhammad Awais

TAs: Habiba Farrukh, Aiza Anjum Chaudhry, Fatima Tariq, Hamza Mahmood and Maryam Khalid

Submission Deadline: October 04, 2015 11:59 PM

Instructions and Submission Details

This assignment can be done in groups of 2. However, the evaluation of the assignment will be on an individual basis. Some important points to notice:

All the submitted codes will be tested against each other and the solutions present online for plagiarism and in case of slightest doubt the concerned case will be forwarded to the Disciplinary Committee.

- There are 4 main parts that you will have to implement in the **gridnavigator.py file**
- Make sure that you have properly commented your code
- You should submit gridnavigator.py only. Zip it in a folder. Name the folder "CS331-Assignment2-Roll_number1-Roll_number2". Example: CS331-Assignment2-17100302-16100222.zip
- Submit this folder on LMS BEFORE the deadline. **No late submissions will be accepted after the third late day.**
- In case your LMS isn't working, email your code to the TA's. Only timely submissions will be considered.
- **Make sure that you contribute equally to the assignment**. Each student will be graded individually even though the assignment is in pairs. Vivas will be taken thoroughly to assess the contribution of each member in a group.
- You should not modify any other files in the game engine.
- DO NOT upload the entire game engine.

Pathfinding [100 Marks]

One of the main uses of artificial intelligence in games is to perform *path planning*, the search for a sequence of movements through the virtual environment that gets an agent from one location to another without running into any obstacles.

In this assignment you get the opportunity of implementing 4 primary path finding algorithms that you have studied in class: Breadth First Search (BFS), Depth First Search (DFS), A* heuristic search and the Best First Search.

1) Breadth First Search (BFS) [25 marks]

Breadth First Search (BFS) searches breadth-wise in the problem space. Breadth-First search is like traversing a tree where each node is a state which may a be a potential candidate for solution. BFS is widely used because of the following advantages

1. Breadth first search will never get trapped exploring the useless path forever.
2. If there is a solution, BFS will definitely find it out.
3. If there is more than one solution then BFS can find the minimal one that requires less number of steps.

You are to write the code in the **implementBFS** function located in the **gridnavigator.py file.**

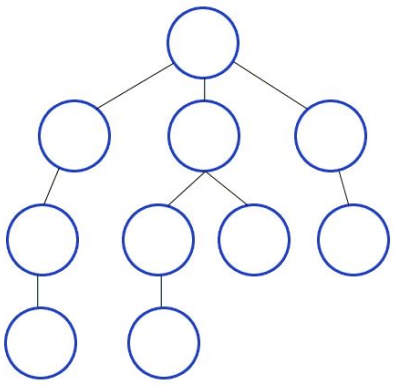

Fig 1.1 How the Breadth First Search Works

2) Depth First Search (DFS) [25 marks]

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root(selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

You are to write the code in the **implementDFS** function located in the **gridnavigator.py file.**

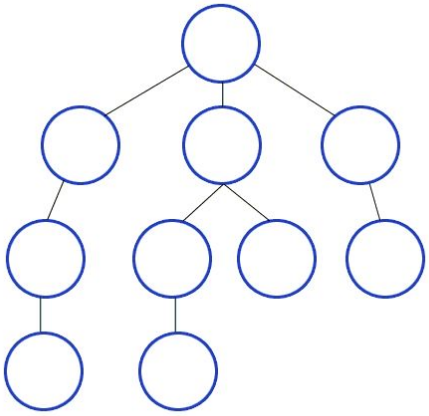

Fig 1.2 How the Depth First Search Works

3) Best First Search [25 marks]

You are required to implement the iterative Best First Search algorithm in the **implementBestFirst** function located in the gridnavigator.py file.

4) A* Search [25 marks]

Another commonly used pathfinding algorithms for computer games is A*. With an admissible heuristic, A* provides nice guarantees about optimality---it will find the shortest path---and keep the amount of unnecessary search to a minimum.

You are to write the code in the **implementAstar** function located in the gridnavigator.py file.
Note: There will be bonus marks for an efficient and effective heuristic function.

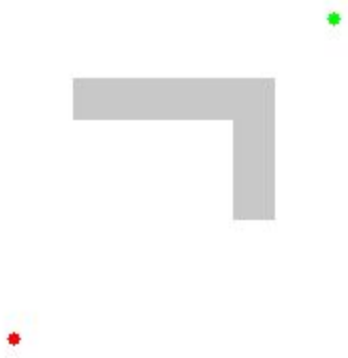

Fig 1.3 How A* Search Works

Your job is to implement these path finding algorithms such that the agent uses the chosen algorithm to find its **path** to the destination. The destination is the grid point that you want to go to i.e the point on the grid that you click using your mouse.

Please do not click inside the obstacles and in the white regions surrounding them if you want your agent to behave normally.

The path that you find as well as the grid cells that your algorithm explores should be correct and in accordance with the algorithm.

What you need to know

This assignment is slightly different from the previous one in the following ways:

1. The **mycreategrid**(*world, cellsize*) function has been implemented for you already since not all of you were able to complete it in the last assignment.

2. Because the agent gets stuck if it collides with an obstacle, collisions have been removed for convenience.
However, your code must be such that the agent avoids all obstacles i.e it does not go through them. If it does, you need to recheck your implementation.

3. **In each of the path finding functions, the "path" variable is being returned.** This is essentially a list consisting of all the nodes(in the appropriate order) that make up the path that your algorithm finds to move the agent from the starting point to the destination.

4. The agent CANNOT move diagonally in your implementations. This means that there are **four direct neighbours of a cell**: the right grid point, the left grid point, the grid point on top of it and the grid point below it.

5. You should not be concerned about the various grid navigator classes and you should **NOT** change them. Just implement your algorithms in the required functions.

Some Useful Functions and Classes:

getCellSuccessors(*cell, grid, dimensions, last* = None):
This function will help you find the neighbours of the current point/grid cell. Here the cell argument is the current point. The grid can be accessed inside the implementation

functions using self.grid. Also, dimensions = self.dimensions. You can see how the function has been used in assignment 1 for further clarity.

Stack & Queue:
These classes have been included in the util.py file. Therefore, these can be used in your implementation.

You can also implement any helper class or functions inside the gridnavigator.py file

Other than this, please print out values to check your logic whenever you have to debug. **Remember, printing is your best friend.**

Testing Your Code:
You can test your implementations by running the following files respectively

1. **runAstarNavigator.py**
2. **runBFSNavigator.py**
3. **runDFSNavigator.py**
4. **runBestFirstNavigator.py**

The following link contains videos on what the expected output should and should not look like. The motion might seem slow because of the screen recording software but you will get an idea of what to expect.

- **Wrong/No implementation of algorithm. Agent moving diagonally and also going through obstacles.**

https://drive.google.com/file/d/0B86ZgEWjN5ufRnAyZjdIMnpldVU/view?usp=sharing

- **Correct implementation. Agent avoiding obstacles when going to destination. Note that when the agent moves correctly i.e when it follows a path you can see a X mark on each of the grid cells that it goes to. However the agent shows random behavior at the end when the destination is an obstacle so do not click inside an obstacle or on the white regions surrounding it.**

https://drive.google.com/file/d/0B86ZgEWjN5ufbjl6VXdOdGp2S1E/view?usp=sharing

Furthermore, you can use the following link for help with some of the algorithms that you are required to implement. Please note that the agent is NOT supposed to move diagonally in your implementations.

https://qiao.github.io/PathFinding.js/visual/

Hints:

You can check if a point is traversable or not by checking if its value in the grid has been set to True or False. Do not allow a non traversable point to be a part of your path so that your agent avoids all obstacleS when it moves.