Programming Assignment 1 Instructions

Organization

Programming Assignment 1 Instructions

Introduction & Assignment

You are lost but fortunately, you have an A* framework that you can use to find your way home.

In this assignment, you will devise an algorithm for plotting a course home, minimizing the total cost of the path and the amount of time spent searching.

The World

In this assignment, you'll first work with the Map class, a class encapsulating a two-dimensional world layered on top of a rectangular grid. Each point in the world has a height, represented by an integer value between 0 and 255. You can move to any of the eight squares adjacent to your own location (e.g. the four cardinal directions and the diagonals). As you would expect, the cost to traverse between tiles is dependent on the differences in height between the tiles. Below are different const functions for you to experiment with.

The Map class also keeps track of which tiles your algorithm visits as it looks for an optimal path home. Part of your grade on this assignment will be determined by how many tiles your algorithm visited. For example, if two different algorithms each yield the optimal path, but one accomplishes this task and only considers 10% of the number of squares as the other, the algorithm that visited fewer squares would be considered superior to the other. Interestingly, many of the better search algorithms that visit fewer squares also run in considerably less time, so you'll have a dual incentive to keep your search space small. Please note that this is still secondary to finding the shortest path.

Coding Instructions

Your assignment is to create an implementation of the AIModule interface that computes a path from the start location to the end location while minimizing the total search space. Once you've written this function, you can run your program through command line arguments, those being:

- w (int) - width of the map (default 500)
- l (int) - length of the map (default 500)
- seed (int) - seed to set for random terrain generation (default None, set by random module)
- filename (string) - filename for .npy file to load for map
- cost (string) - cost function. Values [‘exp’, ‘div’] accepted. Default ‘exp’.
- AI (string) - Name of AIModule to use. Values [‘AStarExp’, ‘AStarDiv’, ‘AStartMSH’, ‘Dijkstra’] accepted. Default ‘exp’.

The program can be executed using the above command-line instructions. For example, running the following command:

Programming Assignment 1 Instructions

```
>> python Main.py -seed 0 -cost ‘div’ -AI ‘div’
```

Will create a random terrain map of size 500x500 using the division cost function (see below) and use the AStarDiv cost function. While the terrain map is random, you can get the exact same terrain map whenever you use the seed 0. Note that length and width are not set and their default values are used.

To help test your implementation, we've provided a working Dijkstra's algorithm AI class called DijkstraAI. As you've learned in class, Dijkstra's algorithm always yields the optimal path, so you can compare your own AI against the DijkstraAI module to see if your path is indeed optimal. Note that when we run your program for grading purposes, we will be using a fresh copy of the provided starter code. Please do not modify the starter code. Otherwise, you may be under the impression that your code is working properly but it will not work properly on our machines.

Submission Instructions

Please submit the following:

AIModule.py - modified python file to include your code.

Assignment1.pdf - Your answers to the written questions.

As individual files. Please do not include any starter code or any other files.

Programming Assignment 1 Instructions

Questions

Cost Functions

Exponential Cost Function: 2^{h1-h0}
Division Cost Function: $h_2/(h_1+1)$

Question 1 (12pts):

For each cost function above **for the chess movement** (all eight neighbors), do the following:

a) Create an admissible heuristic, document the exact form of the heuristic and **prove/show** it is admissable. (6 points per cost function)

Question 2 (6pts):

a) You will now implement your own version of A*. Look at the StupidAI class to get an idea of how to search the state space. (4 points)

b) To implement the heuristic, write valid python code in a separate function (1 point for each):

Question 3 (10 pts):

Try out your heuristic functions with the appropriate cost function on 500x500 maps with random seeds 1, 2, 3, 4 and 5.
Use the following command to run this part of the assignment: "python Main.py -cost c -AI a -seed x" with c being either ‘exp’ or ‘div’ and a being the appropriate AI module, and x being the seeds listed above.
Submission Requirements: For each execution record the cost of the shortest path and the number of nodes expanded as per the output of the program.

1 points per cost function for getting the shortest path. Non-optimal paths will get (shortest path cost) / (your path cost)

For all students who get the shortest path: we will rank your performance and you shall receive bonus marks of:
5*(Number of Qualified Students + 1 - Your Rank) / (Number of Qualified Students) for each cost function.

Question 4 (5pts):

Programming Assignment 1 Instructions

This is a much larger grid and hence you will have to more cleverly implement your algorithm. Modify both your A* algorithm and admissible heuristic so as to find the optimal path in this new environment
in the least possible time.   You only need to consider the chess style movement (all 8 neighbors) and the "New height divided by old height" cost function. You can use any valid technique to improve the performance of the algorithm. By this, I mean any clever modification of the base algorithm (such as waypoints) but NOT say writing the A* algorithm in assembler or memorizing paths etc. If you have any doubts about the validity of your approach then first consult me.

Use the following command to run this part of the assignment: "python Main.py AStarMSH -load msh.npy".

Programming Assignment 1 Instructions

Programming API

Classes

Point (Object)

Description

The point class exists to store information about any given x and y coordinates on the map.

Variables

x (int) - X position on the map

y (int) - Y position on the map

comparator (float) - Single floating point value associated with a point. Used for comparison operations

Methods

__init__(int posx, int posy) - Initialises Point object with coordinates specified above. Comparator initialized as infinity.

< (Point other) -> bool - Compares comparators between itself and another point. Returns True if its own comparator is strictly less than other's, else False.

> (Point other) -> bool - Compares comparators between itself and another point. Returns True if its own comparator is strictly greater than other's, else False.

== (Point other) -> bool - Compares x and y coordinates of itself and point other. Returns True if both coordinates are the same, else False.

Map (Object)

Description

Keeps information about the height of all points the AI module will navigate. Maintains information about explored nodes, movement cost function, and path cost.

Variables

length (int) - Length of the map; x coordinate moves through length.

width (int) - Width of the map; y coordinate moves through width.

seed (int) - Sets seed for random terrain generation via Perlin noise.

explored (list of Points) - List of points accessed from map.

explored_lookup (dict String -> Bool) - Lookup table for if a particular point has already been added to the explored list. Key string in the format "<x>,<y>". Initialized for all points to False.

start (Point) - Start Point, <0,0> (Northwest corner).

end (Point) - End Point, <length-1,width-1> (Southeast corner).

costfunction (function(int h0, int h1)) - lambda function to calculate cost for traversing from one height to another in a single step.

Methods

__init__(int length, int width, int seed=None, String filename=None, String costfunction='exp') - Creates map object of size <length,width> from .npy file if filename provided or randomly through Perlin noise otherwise. Seed set if provided, random otherwise.

generateTerrain(filename = None) -> None - Creates a member variable map (numpy array) from .npy file if filename is provided, otherwise randomly via Perlin noise algorithm.

calculatePathCost(<Points> path) -> int - Calculates the cost from a particular path from the start to the end node. If the first node in the path is not the start, the final node is not the goal, or if any consecutive points are not adjacent, return infinity. Otherwise, returns the cost for traversing the path.

validTile(int x, int y) -> Bool - Returns True if point is within the map, else False.

getTile(int x, int y) -> int - Returns height value of point

isAdjacent(Point p1, Point p2) -> Bool - Returns True is p1 is adjacent (chessboard) to p2.

getNeighbors(Point p) -> <Points> - Returns a list of all points of distance (chessboard) one from point p.

getEndPoint() -> Point - Returns goal Point.

getStartPoint(int x, int y) -> Point - Returns start Point.

getHeight(int x, int y) -> int - Returns height of the map.

generateImage(<Point> path) -> None - Creates and displays image of the map. Path is highlighted in blue, explored nodes are highlighted in red.

AIModule (Interface)

Description

Interface that all AI Modules inherent from.

Methods

createPath(Map map_) -> <Points> - Creates a path from start to goal points on the map

StupidAI (AIModule)

Description

Sample AI Module to provide a template for your implementations of AStar.

Programming Assignment 1 Instructions

Methods

createPath(Map map_) -> <Points> - Creates a path from start to goal points on the map by simply moving across the X-axis until aligned with goal and then the Y-axis until aligned with the goal. Does not consider the difficulty of any move.