AI Assignment 1

Pathfinding

Script Features and Explanation

The script can be run directly with python:
“python robotplanner.py <args>”

Alternatively the tests specified on the assignment homepage can be run automatically with:
“./testrunner”
or
“Python testrunner.py”

```
usage: robotplanner.py [-h] [--graphics]
                       filepath origin_x origin_y destination_x destination_y

Uses A* algorithm to find a path from origin to destination in the given environment.

positional arguments:
  filepath         The path to the input file
  origin_x         The origin x coordinate as an integer
  origin_y         The origin y coordinate as an integer
  destination_x    The destination x coordinate as an integer
  destination_y    The destination y coordinate as an integer

optional arguments:
  -h, --help       show this help message and exit
  --graphics, -g   Enable graphical output.
```

In graphical output mode, the script will display the path from origin to destination in the environment with ascii graphics.

The script will inform the user if the specified filepath does not exist or if the starting or ending coordinates are out of the world and will exit with code 1.

The script will inform the user if the origin or destination points are on a non-empty space and will exit with code 1.

The script will inform the user if there is no possible path from origin to destination and will exit with code 1.

A* Algorithm Explanation

The A* algorithm is a modified version of Djikstra's algorithm to include a heuristic function. It is an example of a greedy algorithm that always arrives at the global minimum (optimum solution). This gives it generally better performance than Djikstra's algorithm with the same level of accuracy.

The heuristic algorithm chosen calculates a 'cartesian distance' which is simply the sum of the differences in x values of the two points and the differences in y values of the two points.
(abs(point1.x - point2.x) + abs(point1.y - point2.y))

The implementation in the code creates a priority queue which stores the points currently being 'looked at' or 'followed.' In addition, a dictionary of the parent cells is kept as well as a dictionary of the costs of the calculated routes.
The 'looking at' queue starts with the origin point.
For each loop of the calculation, the algorithm takes the highest priority point (the one with the lowest value for the sum of its cost and cartesian distance to the destination) and loops through its empty unvisited neighbour cells and adds them to the 'looking at' queue with their priority (the sum of their cost and distance to the end point). The parent dictionary is updated with their parent.

The path is then calculated by going through the parent dictionary starting with the destination point - a route has been calculated meaning the destination point should have parents going all the way back to the origin point. Once this 'backwards path' has been calculated, it is simply reversed and then returned.

The program will then convert this calculated path to a list of 'direction commands' or graphical output (if the -g/--graphics switch has been used).