Frontier Navigation

Welcome to the Frontier Navigation family! We're glad to have someone with AI experience on board. As you know, our landmark product, the Skamania Hiker's GPS, has been a commercial success, and our customer base is waiting excitedly for the Skamania II which will be cheaper and more powerful than the original model. One innovation that will set us apart from the competition is our off-trail pathfinding capabilities that can find the most hiker-friendly path between two points regardless of the terrain. Unfortunately, at the moment our pathfinding algorithms have either been too slow or return suboptimal paths. With the launch of the Skamania II only ten days away, we could really use your insights on this feature. We are hoping that your team can integrate your pathfinding algorithm into our existing code and present your findings by next week.

We are looking forward to seeing your work.

—----------
Sincerely,

Janetta Torres
VP of Operations
Frontier Navigation Incorporated

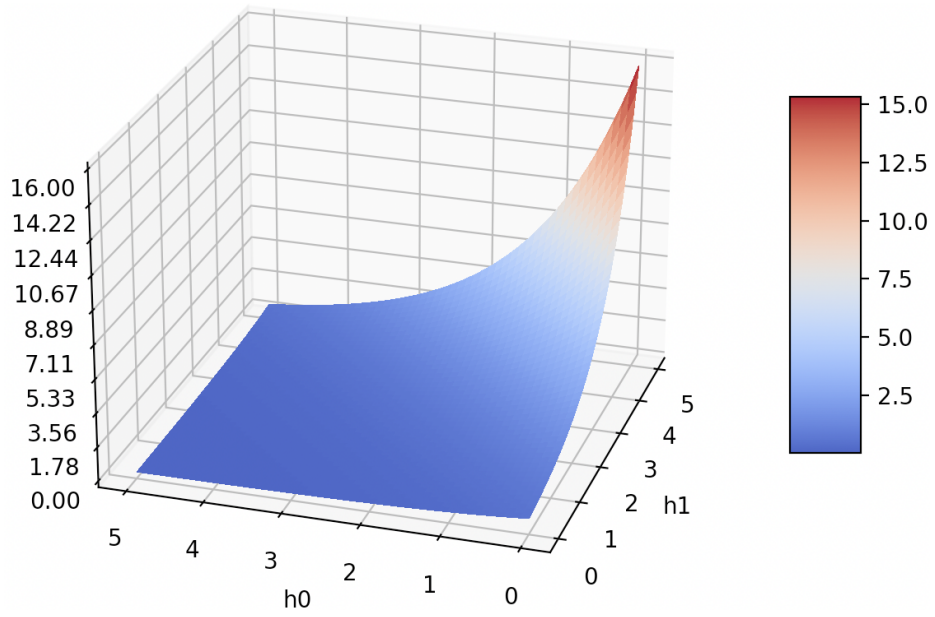
Project_requirements.pdf

Skamania II - Off-trail Pathfinding Requirements

The Skamania II GPS's off-trail pathfinding feature needs to find the most hiker-friendly trail for an arbitrary terrain. The terrain is represented as a 3D surface discretized into uniformly spaced tiles. The hiker can move with chessboard motion across the XY plane of this surface with the "cost" per step being defined by the change in height for each step:

$\text{cost}(h_0, h_1) = e^{h_1 - h_0} = e^{\Delta h}$

Where h_0 is the height of the current title and h_1 is the height of the tile to be moved to. The cost surface can be visualized with the following surface:



The total path cost is defined as the sum of all of the individual step costs.

We are hoping to implement the A* algorithm for this task, but our team lacks the expertise to code this and design admissible heuristics to ensure that the optimal path is found. As such, the project has the following components:

Designing Admissible Heuristics

Assuming the cost function **for chess movement** (all eight neighbors), create an admissible heuristic, define it as a mathematical equation, and **prove/show** it is admissible.

Implement The A* Algorithm

a) You will now implement your own version of A*. We have provided you with a sample agent to demonstrate the inheritance process and show you how paths should be returned.

b) Implement your heuristic into code.

Initial Testing

We have prepared a series of randomly generated 500x500 test maps to ensure that your code is working properly. In your report, please record your output for seeds 1-5 (inclusive), and make sure the path cost is the same as the optimal path cost we have already calculated for you.

Seed 1 - 241.55253710831192
Seed 2 - 238.5060683830129
Seed 3 - 236.667690049357
Seed 4 - 422.01798243905006
Seed 5 - 254.34464507852198

Note that while failing to meet these costs would demonstrate a problem with your code, meeting these costs does not necessarily mean the heuristics are admissible - perhaps there is an issue with your heuristic which did not arise in testing. Since this project assumes many floating-point calculations, your results for this part of the assignment can be within a margin of 0.3.

Use the following command to run this part of the assignment: "python Main.py -AI a -seed x" with “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.

Climbing Mt. St. Helens - A-Star Variants

As part of our coming demonstration, we will have a hiker use the Skamania II to climb to the top of Mt. St. Helens, located here in Skamania County. It is crucial that during this demonstration, hikers can get the optimal route quickly, so we want you to modify your A-Star approach from above to use an A-Star variant of your choosing. As long as you can demonstrate that this variant poses some benefit (eg more memory efficiency, faster) compared to the vanilla A* algorithm you implemented previously, that’s fine. You are free to use any variant you chose, and recommend the following resource. The optimal path cost for this route is [find].

Presentation

Your team will present your findings to a senior member of the project (your TA). You will reserve a 10-minute timeslot for your presentation, in which 8 of these minutes will be for presentation, and 2 will be reserved for Q&A. **These are strict time limits** - we need to move through all groups in a timely manner, so be sure to rehearse. In your presentation, you must go over the following points:

- Why you believe that this algorithm and approach is appropriate for this setting
- Your heuristic and proof of admissibility
- A demonstration of your program running start to finish for at least one of the random seeds
- The results for each of the random seeds
- A description of the AStar variant you used for the MSH problem, specifically:
    - What the variant is
    - Why it is useful
    - How you know it will always produce the optimal path cost result

The presentation should be divided amongst team members in a relatively egalitarian manner. Teams should strive to present their work in a 50-50 split and ensure that they do not extend beyond a 60-40 split in terms of both time and content.

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.

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.