1. PathFinding.js demo

Go visit the webpage at http://qiao.github.io/PathFinding.js/visual/. Read the directions on the upper left, play around with the page for a bit, and then hide the directions (for screenshots later). Now answer the questions below, submitting your typewritten answers and screenshots in a single printed document.

a. Keep the default start and goal state. Run A* search (use the Euclidean heuristic). Then, run Dijkstra’s algorithm (you should allow diagonal movement) and compare the results. Explain how the two algorithms differ in their “exploration” of open space. Which one seems more “focused” to you, and why does it behave that way?

b. Move the start and goal further away from each other, and add some small, scattered obstacles in between, as shown below:

![PathFinding.js demo](image1.jpg)

Run A* to see how it copes with these obstacles, and take a screenshot. How badly do the obstacles affect the search?

c. Next, fill in around some of the obstacles near the start position to make a kind of “cup” shape, like this:

![PathFinding.js demo](image2.jpg)

Rerun A*, and take a screenshot. In what sense does this cup constitute a local minimum? Explain why A* must “fill the cup” before successfully finding a solution.
d. Return to the “scattered” obstacles layout, and play with the Weight setting in the A* tab. This essentially applies a scalar multiple to the heuristic $h(x)$. Try large weights (much bigger than one) and small weights (very close to zero). Compare each one to a weight of 1.0. Which one (big or small) results in a sub-optimal path? Why? Provide a screenshot of both the optimal path and a sub-optimal one. Why would you ever use re-weighting? What do you gain, if optimality is lost?

2. Search in a twisty maze

Imagine that you created a map like this, with the start and goal placed at the entrance and exit of the maze (also imagine that the robot is not allowed to go around the outside of the maze).

Would A* significantly outperform Dijkstra’s in this scenario? Why or why not?

3. Other search methods

Do a web search to research one of the IDA*, Best-First Search or Jump Point Search methods implemented on the PathFinding.js webpage. Write a paragraph summarizing what it does, how it relates to the search methods we’ve discussed in class, and who discovered/proposed it, if you can find that out. Make sure to cite your sources.
4. Footstep planning for LittleDog

Skim through the paper “An Optimization Approach to Rough Terrain Locomotion”, online at http://repository.cmu.edu/cgi/viewcontent.cgi?article=1735&context=robotics. Then answer the following questions:

a. The *Dubins car* heuristic in section III-A refers to modeling the quadruped as a car with some well-defined minimum turning radius that may drive forward or turn, but not reverse. Why might that be a better-fitting heuristic than Euclidean distance “as the crow flies” from the start to the goal? What characteristics do the car and the quadruped share in common?

b. Why does computing the “pose certificate” discussed in sections III-C and III-D help to prevent things like knees colliding with the terrain during locomotion?

c. Heuristic inflation, as described in section III-E, is exactly the same as the Weighted A* search on the PathFinding.js webpage. Essentially, what ARA* does is to re-run weighted A* with iteratively decreasing weights until the planner has run out of time. Why might this be a better approach than committing ahead of time to a single weight and running A* once?