Project 2: Drones
Computer Science 4013/5013: Artificial Intelligence
Due 11:59 PM February 28, 2018

Introduction

As with Project 1 for drones, this project will mirror the project 2 for Spacewar/Spacesettlers. The main focus of this project is to implement and learn A* search. In your case, you will be implementing A* in simulation and then having the drones fly the A* paths in the real-world.

As with spacesettlers, your environment is continuous. In the real world, the obstacles are moving about the environment and you will not be able to use straightforward A* search. To deal with the continuous environment, you should implement ONE of the following changes to A*.

- **Gridded A*:** Break the environment up into n grid squares (where you determine the grid spacing). Create a graph based on these grid squares where adjacent grid squares are connected unless one of them contains an obstacle or a ship. Implement A* search to traverse the graph and find a path from the starting state to the goal beacon. You must design your own admissible heuristic (more detail on this below).

- **Hierarchical gridded A*:** The idea behind this approach is similar to the gridded A* except that the grid spacing is not fixed. Instead, the agent searches at a very coarse granularity (large grid squares) and finds a path in this easier space. The agent then refines its path by making finer grid squares along the previous path. This process is refined until the agent has a reasonable path from the start state to the goal state. As with the previous approach, this approach will require you to design your own admissible heuristic.

- **Roadmap A*:** This approach is based on roadmap navigation. Instead of making grid squares of your entire environment, sample n points from the space and draw a straight line between each set of points. If the line connecting the two points does NOT intersect an obstacle or ship, connect the two points in your search graph. Implement A* and search on this graph. As with the previous approaches, this approach will require you to design your own admissible heuristic.

Any one of these approaches will allow you to create a navigation plan in a continuous environment using A*. However, none of these approaches will allow you to deal with the dynamics of the environment. This is inherently a tricky problem and the most straightforward way to deal with the dynamics is to replan frequently. For this project, you should keep in mind that replanning at every time step will be too slow to be feasible. You should replan dynamically when something in the world has changed that affects your existing plan. When the agent is not planning, it should execute its plan from the previous planning period. You may choose other approaches to replanning but you can NOT replan at each step.

One big difference between this project and spacesettlers: the spacesettlers project has the map build in. For this project, I have uploaded a quick GUI to Canvas that will allow you to draw a map of the room and click on the start and goal states. The GUI is called DroneGUI.py. It will create a numpy map of the room where 0 means a square is empty, 1 means it has an obstacle, 2 means it is a start, and 3 means it is a goal state. To create a set of obstacles and goals, first specify the room size and then click button 1 for obstacles, button 2 for start states, and button 3 for goal states. The File menu will allow you to either load an existing map (from the first window) or save the current map (from the map window).

Extra credit (super bonus amazing extra credit) can be found by building this map automatically using Visual SLAM. Even more amazing extra credit can be had by keeping the map up-to-date while flying using Visual SLAM. Because Visual SLAM is a HARD algorithm, you do NOT need to implement it yourself. You can find a python package for it and incorporate that into your code. You still need to write A* yourself but not Visual SLAM.

Note that this project leaves a LOT of room for creativity. The success at making your agent move around the world effectively will pay off in all of your future projects. You can be creative in how you draw your graph and how you navigate the graph efficiently so long as you are doing A* search at the heart of it!

Also keep in mind that the A* should be written in a modular fashion so you can use it for many high-level actions. Because A* is so critical for all of the students, it is required for both CS 4013 and 5013 students. The CS 5013 students are required to implement an additional search method and to compare the results of using that method to A*. This method can be informed or uninformed.

Extra-credit opportunities

To keep grades within a fair range, all grades will be capped at 110 points. This means that no project can receive a grade higher than 110.

Wanted dead or alive: bugs or new features

The pyparrot library is brand new and likely has some bugs to work out. We want to know about them and fix them! If you find a bug, you can receive extra credit according to the following scale:

- **1 or 2 points:** General bugs are likely given how new this code is. Finding a bug and reporting it can get you 1 point. Fixing the bug and giving us the fix (you can't check it in directly but you can give it to us in the bug report) can get you two points. Both bug and fix must be verified for any extra credit to be awarded. Bug reports should be verbose and state exactly what happened. Simply stating "it broke" will not count. Feel free to report them on GitHub using issues.

- **3-10 points:** If you a particularly clever way to implement a feature that is not currently implemented, please submit a pull request with your code. If I accept it, you will gain extra credit worth 3-10 points, depending on the feature and the code you submit (one line quick fixes are worth less than more complicated features).

- **10+ points:** Definite extra credit: using Visual SLAM to build the map automatically. Super bonus extra credit: Keeping the map up to date using Visual SLAM. Note: you can use an existing visual SLAM library. You just need to incorporate it into your project. Your focus is on A* code but this would be an amazing accomplishment and worthy of both extra credit and major demos.

Wanted (alive please): creative individuals

Creativity is highly encouraged! To make this real, there are up to 10 points of extra-credit available for creative solutions. Some ideas here include real-time planning where you plan in the background while executing your current best plan (see the discussion of real-time A* in the book) or other forms of A* search that deal with continuous and dynamic environments. If you choose to implement anything that you consider creative, please do the following:

- Document it in your writeup! I can't give extra credit unless I know you did something extra.

- Your navigation search must still have A* at the heart of it. If you are unsure if your search qualifies, come talk to me.

- Your high level behavior heuristic must still have local search at the heart of it. If you are unsure if your search qualifies, come talk to me.
- Remember that by being creative I am referring to the algorithm and *not* to the ability to creatively download code. **All project code must be written exclusively by your group except for Visual SLAM, which I am explicitly giving you permission to download. You are also welcome to use any functions in the opencv libraries.**

Those pesky details

1. Update your code from project 1. You can update your code at the command line with "git pull". I have released fixes for the codebase since project 1.
2. All students: Create a drone program that makes moves using A* as described above and in environments described through the GUI.
3. CS 5013 students: Create an alternative search method for your ships navigation. You may implement one of BFS, DFS, Hill climbing, or Greedy-best-first search. Collect at least 20 examples from your drone using your chosen alternative method and another 20 from your drone using your A* implementation. Put a graph in your writeup comparing the two implementations in terms of how they accomplish the goal of navigation.
4. ONE member of your team should submit your final code to canvas, along with your writeup.
5. Bring your drone to class on the day listed on the schedule for project 1 demos. Be prepared to demo!

Point distribution

- CS 4013/5013: Astar
  - 35 points for correctly implementing A*. A correct drone will have an admissible heuristic and will move to the target efficiently and avoid obstacles. Full credit requires FULL code documentation explaining what each function does. The top of each python file should also contain a 2-3 sentence description of everything that python file does.

  - 30 points if there is only one minor mistake. An example of a minor mistake would be having off-by one errors (where you miss a search node). You can also lose 5 points for not documenting your code well (but at least somewhat).
  - 25 points if there are several minor mistakes or if your code is missing a lot of documentation.
  - 20 points if you have one major mistake. An example of a major mistake would be failing to mark a grid square as occupied (which causes you to run into obstacles, ships, or bullets) or failing to correctly connect your graph.
  - 15 if there are several major mistakes.
  - 10 points if you implement a search other than A* that at least moves the agent around the environment in an intelligent manner.
  - 5 points for an agent that at least does something other than random movements.

- Heuristic
  - 10 points for designing, correctly implementing, and documenting an admissible and consistent heuristic. This will be graded as follows:
  - 8 points for one minor mistake. An example would be correctly designing an admissible heuristic but failing to implement it in an admissible way or making a minor coding error or failing to document your heuristic.
  - 5 points for several minor mistakes or one major mistake. An example of a major mistake would be designing a heuristic that is not admissible (but correctly implementing it)

- CS 5013 students only: second search method
  - 20 points for a fully implemented and correct second search method from BFS, DFS, hill climbing or GBFS. The method must be well documented and data must be collected from at least 20 examples of using this search and 20 examples of using A*. A graph of the comparison must appear in your writeup along with a discussion of how the methods compare and why one is preferred over the other.
  - 15 points for implementing the search but it is lacking documentation or has minor errors or the graph is missing in the writeup
  - 10 points for implementing the search correctly but failing to test it and discuss it in the writeup

  - 5 points for major errors or no documentation and testing

- 10 points for correctly dealing with the dynamics of the environment.
  - 10 points for dynamically replanning as needed in an efficient manner. This CANNOT be replanning every time step.
  - 5 points if you replan when you achieve your target only
  - 0 points if you never replan

- 10 points: We will randomly choose from one of the following good coding practices to grade for these 10 points. Note that this will be included on every project. Are your files well commented? Are your variable names descriptive (or are they all i, j, and k)? Do you make good use of classes and methods or is the entire project in one big flat file? This will be graded as follows:
  - 10 points for well commented code, descriptive variables names or making good use of classes and methods
  - 7 points if you have partially commented code, semi-descriptive variable names, or partial use of classes and methods
  - 0 points if you have no comments in your code, variables are obscurely named, or all your code is in a single flat method (not sure you can do that with A* anyway!)

- 15 points for your writeup. CS5013 students have additional requirements of documenting their other search method, described above. **Your writeup is limited to 2 pages maximum. Any writeup over 2 pages will be automatically given a 0.**
  - 8 points: Describe what form of A* you implemented, and how you dealt with the dynamics of the environment
  - 3 points: Describe your heuristic and why it is admissible and consistent
  - 1 points: Why did you choose the cooperative or competitive path? A sentence or two is sufficient here.
  - 5013 only (points above but the description goes here): Include a graph of your alternative search method compared to A* and explain how and why they differ.

- 3 points: Briefly describe how you worked together as a group or describe why you chose to work alone. In the case of groups, full points requires full collaboration and not just splitting the project into parts. In the case of singletons, full points requires a reasonable justification ("I didn't know anyone else" is not a sufficient justification. "I work full-time in OKC and commute in just for this class" is a great one.)
- **Your writeup is limited to 2 pages maximum. Any writeup over 2 pages will be automatically given a 0.**