Lab 14: Go, Speed Racer!

You should work with your same partner from last week.

You can use any computer that runs soar.

- **Athena machine:** Do `athrun 6.01 update` and `add -f 6.01`.
- **Lab laptop:** Do `athrun 6.01 update`.
- **Personal laptop:** Download design lab 14 zip file from course web page.

When doing Bayes Map (last part of DL13)

- Read the documentation about `util.make2DArrayFill` for a handy way to make the initial grid of state estimators.
- When the sonar reading is greater than the maximum good value, you should mark the cells along the first part of the ray (up to `sonarMax`) clear.
- You might want to use this code in your definition of `squareColor`, where `p` is the probability you assign to that square being occupied.

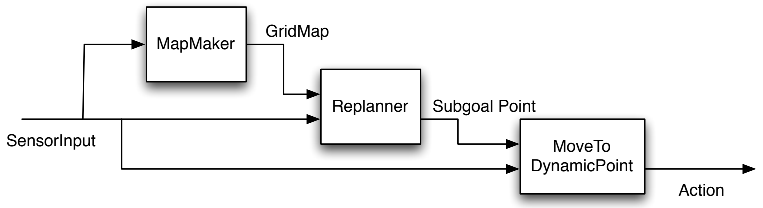
```
if self.robotCanOccupy((xIndex,yIndex)):
    return colors.probToMapColor(p, colors.greenHue)
else:
    return colors.probToMapColor(util.clip(0.1 + p, 0, 1),
                                 colors.redHue)
```

  Squares will be shown with a green hue if we think the robot can inhabit them, and with a red hue if we think the robot cannot inhabit them. So, the light pink squares are actually believed to be free of obstacles, but are too close to obstacles to be habitable by the robot.

Thus far, we worked on speeding up planning time by using a heuristic. And our robots can avoid obstacles by building a map and planning paths to avoid them. Now, we're going to work on making our robots move more quickly through the world. Your job during this lab is to speed up your robot as much as possible; at the end, we'll have a race.

Athena is MIT's UNIX-based computing environment. OCW does not provide access to it.

We will use the same basic architecture as in the last labs:



This time, however, we will concentrate on the `Replanner` and `MoveToDynamicPoint` modules. You will use your implementations of `Mapmaker` and `BayesGridMap` from lab 13.

Set up:

1. Copy your `GridDynamics` class from the tutor or your `lab13/swLab/plannerStandaloneSkeleton.py` file into `lab14/designLab/gridDynamicsSkeleton.py`.
2. Copy `lab13/designLab/bayesMapSkeleton.py` and `lab13/designLab/mapMakerSkeleton.py` into the `lab14/designLab` directory.

The `mapAndRaceBrain.py` is currently set up to work in `raceWorld.py`: select that as the simulated world, and run the brain. To change the world you're working in, change the `useWorld` line in the brain (and remember to change the simulated world, as well).

When you run using `mapAndRaceBrain.py`, you'll notice that, when the robot reaches its goal, it stops and prints out something like

```
Total steps: 320
Elapsed time in seconds: 209.554840088
```

That's the number of soar primitive steps it took to execute your plan, and the amount of elapsed time it took. These numbers will be your 'score'. Note that we are aiming for low scores!

You can debug on your own laptop or an athena machine, but scores will only be considered official if they are run on a lab laptop.

Check Yourself 1. Run your robot through `raceWorld` and see what score you get. Write this down, because it's your baseline for improvement.

Improving performance

You will notice that there are several things that slow your robot down as it executes its plans:

- Each individual step, from grid cell to grid cell, is controlled by a proportional controller in `move.MoveToFixedPoint`. The controller has to slow down to carefully hit each subgoal.
- Rotations take a long time.

Below are some possible strategies for addressing these problems. You don't need to do any or all of these. But, by the end of the week, you should aim to have selected and implemented at least two ideas, either from this list or of your own devising. If you pick one of your own (which we encourage!), talk to a staff member.

You can speed up the robot by producing a plan that requires less stopping and/or less turning. Implement these by editing your `GridDynamics` class or the `ReplannerWithDynamicMap` class in `replannerRace.py` (read that code carefully).

1. Increase the set of actions, to include moves that are more than one square away. You can use the procedure `util.lineIndicesConservative((ix1, iy1), (ix2, iy2))` to get a list of the grid cells that the robot would have to traverse if it starts at `(ix1, iy1)` and ends at `(ix2, iy2)`. This list of grid cells is conservative because it doesn't cut any corners.
2. Plan with the original set of actions, but then post-process the plan to make it more efficient. If the plan asks the robot to make several moves along a straight line, you can safely remove the intermediate subgoals, until the location where the robot finally has to turn.
3. Augment the space in which you are planning to include the robot's heading. Add an additional penalty for actions that cause the robot to rotate. Experiment with the penalty to improve your score.

You can also speed up the execution of the paths by changing the gains and tolerances in the `move.MoveToDynamicPoint` behavior (in `move.py`). Read the code in the file to understand what these parameters mean, and then consider adjusting them to improve the robot's behavior. But be sure you do not cause crashes into obstacles! You can edit these lines of code in `mapAndRaceBrain.py`.

```
move.MoveToFixedPoint.forwardGain = 1.0
move.MoveToFixedPoint.rotationGain = 1.0
move.MoveToFixedPoint.angleEps = 0.05
```

You are not allowed to change the `maxVel` parameter.

Checkoff 1. Discuss your progress on improving the robot's performance with a staff member.

Checkoff 2. Discuss your progress on improving the robot's performance with a staff member.
Post your best scores in `raceWorld` and `lizWorld` on the board.

On Thursday

The teams with fastest robots in simulation will face off on real robots.