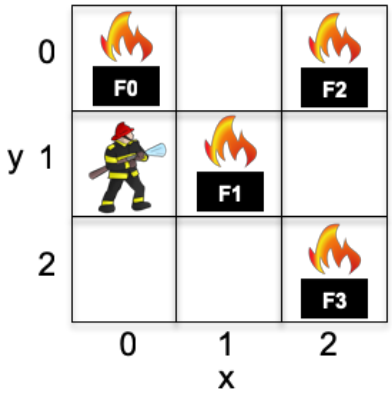
Assignment #4: Robotic Wildfire Suppression (Markov Decision Processes)

CSCI 364 Spring 2019 Oberlin College

Due: Wednesday April 10 at 1:30 PM



Background

You've recently been hired by a robotics company that is developing autonomous robots whose purpose is to fight wildfires, offering 24/7 availability to offer greater responsiveness to reduce dangers and protect property, as well as alleviate the need for people to put themselves at risk as firefighters. Engineers in the company have nearly perfected the mechanics of the robot – these state-of-the-art machines feature GPS sensors, satellite uplinks to infrared sensors that reveal the locations and intensities of all nearby fires, unlimited suppressant resources, and off-road capabilities for moving anywhere in a forest.

Your task is to help develop the artificial intelligence that will guide the robot's decision making. In particular, you are responsible for implementing the offline planning that will create policies (i.e., plans) that the robots will follow when acting in the world, with the objective of monitoring for fires, then putting them out as quickly as possible. Based on what you learned at Oberlin, you know that a Markov Decision Process is a good model for the agent's reasoning – the environment is stochastic (fires appear and change in intensity with some random effects) and non-episodic (the robots are tasked with continually operating 24/7).

After talking with domain experts, you've developed the following MDP to model the problem:

States S: the states of the environment (x, y, F0, F1, F2, F3) record (1) the x and (2) y coordinates of the robots, as measured by its GPS sensors, and (3-6) the intensities of the four nearby fires that the agent could fight. The x and y coordinates are each integers in the range [0, 2] and each fire's intensities are also integers in the range [0, 3] where 0 represents no fire and 3 represents the location is burned up (whereas 1-2 represent active fires of increasing intensity).

Actions A: a robot can take five actions: Extinguish to use its suppressant resources on the current location where it is standing, as well as Up, Down, Left, and Right to move in each cardinal direction.

Transitions T(s_t, a, s_{t+1}): the robot's state factors change as follows. Changes in its location are deterministic, whereas changes to fire intensities are stochastic.

Movement Changes

- If the agent moves Up, its y coordinate decreases by 1 (unless it is already at the top where y == 0) and its x coordinate does not change.
- If the agent moves Down, its y coordinate increases by 1 (unless it is already at the bottom where y == 2) and its x coordinate does not change.
- If the agent moves Left, its x coordinate decreases by 1 (unless it is already at the left where x == 0) and its y coordinate does not change.
- If the agent moves Right, its x coordinate increases by 1 (unless it is already at the right where x == 2) and its y coordinate does not change.
- If the agent chooses the Extinguish action, its location does not change

Fire Intensity Changes

- If an agent uses an Extinguish action on an active fire (a location with an intensity of 1 or 2), then the fires intensity decreases by 1 with probability 80% and stays the same with probability 20%
- If the agent uses an Extinguish action on a non-active fire (either there is no fire since intensity is 0 or the fire is burned out with an intensity of 3), then the fire does not change
- Any fire for which the agent does not use an Extinguish action on its turn behaves as follows:
  - If the fire is not active because its intensity is 0, there is a 5% change that the intensity increases to 1, else it stays 0 with probability 95%
  - If the fire is not active because it is burned out (with an intensity of 3), it stays burned out
  - If the fire is active (intensity 1 or 2), its intensity increases by 1 with 10% probability, else it stays the same with 90% probability

Reward R: agents earn rewards for each state based on its action in the following way:

$$R(s,a) = 10 * NoFire - 10 * BurnedOut + E$$

where $NoFire$ is the number of fire locations in state s with intensity 0, $BurnedOut$ is the number of fire locations in state s with intensity 3, and

$$E = \begin{cases} 0 & \text{if } a = \text{Up, Down, Left, or Right} \\ 5 & if\ a = \text{Extinguish and the agent is standing on a fire with intenisty 1 or 2} \\ -10 & \text{otherwise} \end{cases}$$

Gitting Started

You can get started on the assignment by following this link:
https://classroom.github.com/a/MygUBRGI

Programming Assignment

Your assignment is to write a program in **Python or Java** that implements the above details into an MDP, performs Value Iteration to create the appropriate Q and V tables, then determines an appropriate policy from the resulting Q table. I have started the GitHub repository with a file called states.csv that provides a full list of the states in the environment, so you do not have to generate those yourself. You ***do*** need to implement the state transition and reward functions.

Inside the states.csv file, each state is represented by a line where the first number the state's unique identifier, and the following six numbers represent the values of the six variables that make up a state: the agent's x and y coordinates and the intensities of the fires in the four fire locations.

After your program determines a policy for the agent, it should save the policy to a file in the following format. Each line should represent a single state and the prescribed action for that state. The first value in a line is the state's unique identifier, followed by a comma, followed by the action's unique identifier (0 = Extinguish, 1 = Up, 2 = Down, 3 = Left, 4 = Right). Since there are 2304 states in this problem, your policy file should have 2304 lines. For example, the first few lines might be:

```
0,4
1,4
2,4
3,4
4,4
```

Here, the policy says the agent should move to the Right in the first five states.

Similar to the third homework (Sudoku), there is no code base to start with. Instead, you are free to develop your program and represent your data structures however you wish. Also, **you are allowed to work in groups of two students** (only one needs to submit the solution on GitHub). Your GitHub repository will only initially contain the aforementioned states.csv file.

Please call your program either WildfirePlanner.py (in Python) or WildfirePlanner.java (in Java), so that it is executed with either:

```
python3 WildfirePlanner.py <gammaValue> <epsilonValue>
```

or

```
java WildfirePlanner  <gammaValue> <epsilonValue>
```

where `<gammaValue>` is a value to use for γ and `<epsilonValue>` is a value to use for ϵ in the Value Iteration algorithm.

You should not import any modules or libraries not already built into Python and Java.

Experiments

After you have your program implemented, you should use the WildfireSimulator.class Java program to evaluate the quality of the plans generated by your program. I will upload this program to Blackboard for you to download (under "Slides"). To run the program, you should use:

```
java WildfireSimulator <PathToPolicyFile>
```

The simulator will then upload your policy into an agent and ask the agent to interact with the world for 50 sequential actions, moving around and hopefully putting out fires. The simulator will automatically repeat this process for 100 runs, then print out to the screen a list of 100 floats representing the cumulative rewards earned by the agent (1 float per run) and 100 integers representing how many fires the agent put out (again 1 integer per run).

To investigate how the value of γ affects the agent's ability to reason about future situations and thus the overall quality of its plans, you should create multiple policies with different values of γ using the `<gammaValue>` command line parameter (for your experiments, please use a fixed value of 0.1 for `<epsilonValue>`). You should use the values of 0, 0.1, 0.3, 0.5, 0.7, 0.9 for γ, then create two line charts: (1) the first plotting the average cumulative reward earned in the simulator for each γ value (so that γ is on the x-axis and average cumulative reward is on the y-axis), and (2) the second plotting the average number of fires put out in the simulator for each γ value (so that again γ is on the x-axis and the average number of fires put out is on the y-axis).

Within a README file, you should include:

1. A paragraph discussing how γ impacted the cumulative rewards earned by the agent – do you see any trends? What do you think these trends imply, and why did they occur?
2. A paragraph discussing how γ impacted the average number of fires put out by the agent – do you see any trends? What do you think these trends imply, and why did they occur?
3. A paragraph comparing the results of your two line charts. Did you observe the same trends for the two different performance measures or were they different? Why do you think this happened?
4. A couple paragraphs documenting how you designed and implemented the MDP in your program? What design options did you consider, and how did you decide on this implementation?
5. A short paragraph describing your experience during the assignment (what did you enjoy, what was difficult, etc.)
6. An estimation of how much time you spent on the assignment, and
7. An affirmation that you adhered to the honor code

Please remember to commit your solution to your repository on GitHub. You do not need to wait until you are done with the assignment; it is good practice to do so not only after each coding session, but maybe after hitting important milestones or solving bugs during a coding session. ***Make sure to document your code***, explaining how you implemented the MDP as a data structure, as well as the algorithms used to solve the MDP.

Honor Code

Each student is to complete this assignment with no more than one partner. However, students are encouraged to collaborate with one another to discuss the abstract design and processes of their implementations. For example, please feel free to discuss the pseudocode for the Value Iteration algorithm to help each other work through issues understanding exactly how the algorithm works. At the same, no code can be shared between groups, nor can groups look at each other's code.