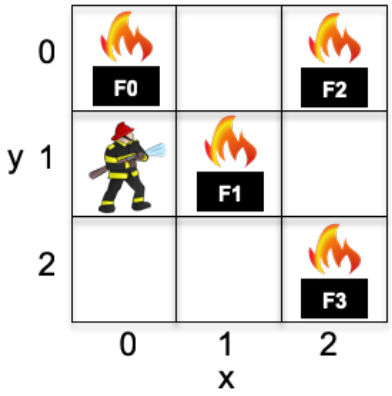
Assignment #3: Robotic Wildfire Suppression
(Markov Decision Processes)
CSCI 364 Fall 2021 Oberlin College
Due: Monday December 6 at 11:59 PM



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 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 an MDP that models the problem, given at the end of this document for anyone interested in seeing the details. I've encoded all of the states, actions, transitions, and rewards for you in a text file called `wildfire.mdp` that is in your GitHub repository.

Gitting Started

You can get started on the assignment by following this link:

https://classroom.github.com/a/idpIZpop

MDP File

Recall that every MDP has four parts: a set of states S, a set of actions A, a state transition function $T(s_t, a, s_{t+1}) = P(s_{t+1}|s_t, a)$, and a reward function $R(s_t, a, s_{t+1})$. The MDP file format encodes these details as follows:

- Under the `States` heading is a list of all states in the environment, one state per line. Each state is recorded in the format:

  `UniqueID,Label`

  where `UniqueID` is a unique number identifier for each state (ranging from 0-2303) and `Label` is a textual description of the state, here the values of each of the individual state variables (x, y, F0, F1, F2, F3) [described at the end of this document]. For purposes of this assignment, `Label` is only included to help you debug; otherwise, you will always refer to a state by its `UniqueID`.

- Under the `Actions` heading is a list of all actions that the agent can take in the environment, one action per line. Similar to states, each action is recorded in the format:

  `UniqueID,Label`

  where `UniqueID` is a unique number identifier for each action (ranging from 0-4) and `Label` is a textual description of the action ("Extinguish", "Up", "Down", "Left", "Right"). As with states, the `Label` is only there to help you debug; otherwise, you will always refer to an action by its `UniqueID`.

- Under the `State Transitions` heading is a list of all state transitions that do not have 0% probability. Each state transition is recorded in the format:

  `state,action,next_state,probability`

  where `state` represents the unique identifier of the current state s_t, `action` represents the unique identifier of the chosen action a, `next_state` represents the unique identifier of the next state s_{t+1}, and `probability` represents the transition probability.

  Of note: any state transition $T(s_t, a, s_{t+1})$ not listed in the MDP file has 0% probability.

- Under the `Rewards` heading is a list of all reward combinations that do not have 0 value. Each reward is recorded in the format:

  `state,action,next_state,reward`

  where `state` represents the unique identifier of the current state s_t, `action` represents the unique identifier of the chosen action a, `next_state` represents the unique identifier of the next state s_{t+1}, and `probability` represents the transition probability.

  Of note: any reward $R(s_t, a, s_{t+1})$ not listed in the MDP file has 0 value.

Programming Assignment

Your assignment is to write a program in **Python or Java** that:

1. Parses a MDP file according to the above format into states, actions, state transitions, and rewards
2. Performs the Value Iteration algorithm to create the appropriate Q and V tables, then determines an appropriate policy from the resulting Q table. **You should use $\epsilon = 0.1$** in Value Iteration, whereas γ will be specified by the user (see below).
3. Saves the policy to a file so that it can later be used in a simulation of the environment.

Your program 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. 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 second 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 initially contain the aforementioned `wildfire.mdp` file.

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

```
python3 mdp_planner.py <mdpFilename> <gammaValue> <policyFilename>
```

or

```
java MDPPlanner <mdpFilename> <gammaValue> <policyFilename>
```

where `<mdpFilename>` is the path to the MDP file, `<gammaValue>` is a value to use for γ in the Value Iteration algorithm, and `<mdpFilename>` is the name of the file where your policy will be saved.

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 provided `MDPSimulator` Java program to evaluate the quality of the plans generated by your program. To run the program, you should use:

```
java MDPSimulator <mdpFilename> <policyFilename> <startState>
```

where `<mdpFilename>` and `<policyFilename>` are defined above and `<startState>` is unique identifier of the state where you want to start your simulations.

The simulator will 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 100,000 runs, then print out to the average cumulative rewards (i.e., utility) earned by the agent across those 100,000 simulation runs.

In these experiments, we will 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 different policies for the values of 0.1, 0.3, 0.5, 0.7, 0.9 for γ using your program, then and use the simulator to determine the average utility earned for the different policies. For the `<startState>`, you should use 938, which is the state `0,0,2,2,2,2` where the agent starts in the top left corner (x = 0, y = 0) and all fires are at intensity level 2.

With the results of your simulations, you should create a line chart plotting the average utility earned in the simulator for each γ value (so that γ is on the x-axis and average utility is on the y-axis). You can use any program you want to create these line charts; some options include Excel, R, and Python with matplotlib. Please save the line chart as an image file and upload it to your GitHub repository.

Within a README file, you should include:

1. A paragraph discussing how γ impacted the average utility earned by the agent – do you see any trends? What do you think these trends imply, and why did they occur?
2. 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?
3. A short paragraph describing your experience during the assignment (what did you enjoy, what was difficult, etc.)
4. An estimation of how much time you spent on the assignment, and
5. 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.

Grading Rubric

Your solution and README will be graded based on the following rubric:

Program flow: /10 points
Parsing the MDP file into a MDP model: /20 points
Value Iteration implementation: /30 points
Correct policy file: /5 points
Program efficiency: /5 points
Line chart from your experiments: /10 points
Answers to README questions: /15 points
Appropriate code documentation: /5 points

By appropriate code documentation, I mean including a header comment at the top of each file explaining what the file provides, as well as at least one comment per function explaining the purpose of the function.

MDP Formulation

States S: the states of the environment (x, y, F0, F1, F2, F3) keep track of:

(elements 0-1) the x and y coordinates of the robots, as measured by its GPS sensors
(elements 2-5) 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:

$$R(s_t, a, s_{t+1}) = 10 * NoFire - 10 * BurnedOut + E$$

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

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