Practical 4: MDPs and Reinforcement Learning

Writeup due 23:59 on Friday 18 April 2014

There is not a Kaggle component to this practical. However, your main deliverables are still largely the same: a 3-4 page PDF writeup that explains how you tackled the problems. There are two other differences with the previous practical: you are **not** allowed to use off-the-shelf planning/RL software such as PyBrain, and you should turn in all of your code as a zip file or gzipped tar file, with a README that explains how to run it.

CS 181 Students: You will do this assignment in groups of three. You can seek partners via Piazza. Course staff can also help you find partners. Submit one writeup per team by the due date via the iSites dropbox.

CSCI E-181 Students: You will do this assignment on your own. Submit your writeup by the due date via the Extension School iSites dropbox.

Warm-Up: Lawn Darts

Lawn darts was a classic fun family game, that also turned out to be somewhat deadly. The game was played by throwing pointy weighted darts into the air and trying to have them land in a target area. It was kind of like a weaponized version of horseshoes. Like the *Slip-n-Slide* and the *Gilbert U-238 Atomic Energy Lab*, it is a toy that has been banned by the Consumer Product Safety Commission.

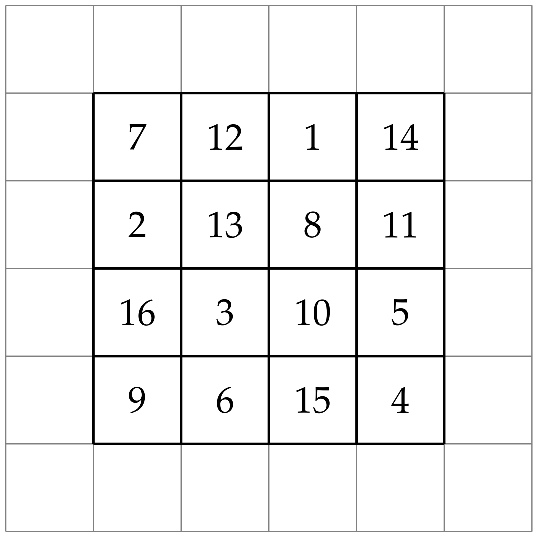
Fortunately, we're going to play it here in simulation. This will be a single-player game. The idea is to throw darts at the grid in Figure 1b until you get **exactly** 101 points. You'll start with zero points, and in each turn you aim a dart at one of the sixteen inner numbered points in the grid. You're not a great aim, so you have a good chance to hit one of the other squares. Whatever square you hit, you add that number of points to your score. If you hit one of the blank squares in the ring, you add zero points. You can't aim for the blank ring. If you get a score of exactly 101, the game ends and you get one unit of reward. If you go over 101, the game ends and you get a negative one unit of reward.

Your objective is to learn a policy to play the game. In this case, you happen to know exactly what your aiming distribution is: you have a 60% chance of hitting the square you aim for, and 10% chance of hitting the square to the north, east, south or west. So, if you aim for the 15, you get 15 points with probability 0.6, 4 points with probability 0.1, 6 points with probability 0.1, 10 points with probability 0.1, and 0 points with probability 0.1. This is a Markov decision process (MDP) in which you know the current state $s \in \{0, 117\}$ (your score), you know the reward function

$$R(s) = \begin{cases} 0 & \text{if } s < 101 \\ 1 & \text{if } s = 101 \\ -1 & \text{if } s > 101 \end{cases}. \tag{1}$$

WARNING:
Pointed, metal lawn darts may cause serious or fatal injury.
Parents are urged to discard or
destroy them immediately.

(a) Lawn Darts



	7	12	1	14	
	2	13	8	11	
	16	3	10	5	
	9	6	15	4	

(b) Point Grid

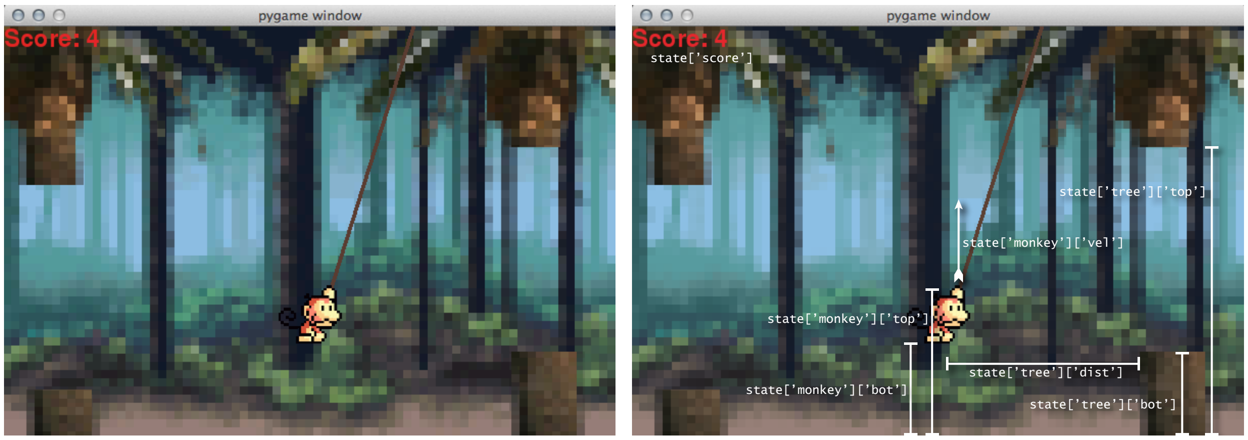
Figure 1: (a) Lawn darts turned out to be a bad idea, except for in simulation. (b) You aim for one of the 16 middle squares. You get zero points if you accidentally hit in the ring outside.

A policy for this game will be a function from the states $s = 0, 1, \ldots, 100$ to 16 possible actions. Your objective is to find such a policy using value iteration or policy iteration. Explain how you set up the problem and why you chose the approach you did. Create a bar graph of the value function. Do you notice any interesting structure in it? Which non-terminal state has the highest value? Why do you think this is?

Swingy Monkey

Earlier this year, the mobile game *Flappy Bird* took the world by storm. Since its discontinuation, iPhones with the game installed have sold for thousands of dollars on eBay. In this part of the practical, you'll be developing a reinforcement learning agent to play a similar game, *Swingy Monkey*. See screenshot in Figure 2a. In this game, you control a monkey that is trying to swing on vines and avoid tree trunks. You can either make him jump to a new vine, or have him swing down on the vine he's currently holding. You get points for successfully passing tree trunks without hitting them, falling off the bottom of the screen, or jumping off the top. There are some sources of randomness: the monkey's jumps are sometimes higher than others, the gaps in the trees vary vertically, and the distances between the trees are different. You can play the game directly by pushing a key on the keyboard to make the monkey jump. However, your objective in this part of the practical will be to build an agent that *learns* to play on its own.

You'll be responsible for implementing two Python functions, `action_callback`



(a) SwingyMonkey Screenshot (b) SwingyMonkey State

Figure 2: (a) Screenshot of the Swingy Monkey game. (b) Interpretations of various pieces of the state dictionary.

and `reward_callback`. The reward callback will tell you what your reward was in the immediately previous time step:

- Reward of $+1$ for passing a tree trunk.
- Reward of -5 for hitting a tree trunk.
- Reward of -10 for falling off the bottom of the screen.
- Reward of -10 for jumping off the top of the screen.
- Reward of 0 otherwise.

The action callback will take in a dictionary that describes the current state of the game and you will use it to return an action in the next time step. This will be a binary action, where 0 means to swing downward and 1 means to jump up. The dictionary you get for the state looks like this:

```
{ 'score': <current score>,
  'tree': { 'dist': <pixels to next tree trunk>,
            'top': <height of top of tree trunk gap>,
            'bot': <height of bottom of tree trunk gap> },
  'monkey': { 'vel': <current monkey y-axis speed>,
              'top': <height of top of monkey>,
              'bot': <height of bottom of monkey> }}
```

All of the units here (except score) will be in screen pixels. Figure 2b shows these graphically. There are multiple challenges here that make this more difficult and realistic than the lawn darts problem. First, the state space is very large – effectively continuous. You'll need to figure out how to handle this. One strategy might be to use some kind of function approximation, such as a neural network, to represent the value function or the Q-function. Another strategy – one that worked well for me – is to discretize the position space into bins. Second, you don't know the dynamics, so you'll need to use a reinforcement learning approach, rather than a standard MDP solving approach. I got a pretty good monkey policy with Q-Learning.

Your task part is to use reinforcement learning to find a policy for the monkey that can navigate the trees. The implementation of the game itself is in file `SwingyMonkey.py`, along with a few files in the `res/` directory. An file called `stub.py` is provided to give you an idea of how you might go about setting up a learner that interacts with the game. I also posted a YouTube video of my Q-Learner at `http://youtu.be/l4QjPr1uCac`. You can see that it figures out a reasonable policy in a few dozen iterations. You should explain how you decided to solve the problem, what decisions you made, and what issues you encountered along the way. As in the other practicals, provide evidence where necessary to explain your decisions. You should not use off-the-shelf RL tools like PyBrain to solve this. Turn in your code.

Questions and Answers

What should I turn in via the dropbox? The main deliverable of this practical is a three-to-four page typewritten document in PDF format that describes the work you did. You will also turn in the code that you write, as a zip file or a gzipped tar file. Concretely, you should turn in via the dropbox:

- A 3-4 page PDF writeup that shows your results and explains your approach to malware classification. Make sure to include the name of the team and the names of all partners.
- A zipped or gzipped folder containing your warm-up code and a README file explaining how to run it.

How will my work be assessed? This practical is intended to be a realistic representation of what it is like to tackle a reinforcement learning problem. As such, there is no single correct answer and you will be expected to think critically about how to solve it, execute and iterate your approach, and describe your solution. The upshot of this open-endedness is that you will have a lot of flexibility in how you tackle the problem. You can focus on methods that we discuss in class, or you can use this as an opportunity to learn about approaches for which we do not have time or scope.

You will be assessed on a scale of 25 points, divided evenly into five categories:

1. **Warm-Up:** In the warmup, you'll be expected to implement a simple algorithm from scratch, run it on some simple data, and turn in your code. You'll be graded on correctness of the implementation.

2. **Effort:** Did you thoughtfully tackle the problem? Did you iterate through methods and ideas to find a solution? Did you explore several methods, perhaps going beyond those we discussed in class? Did you think hard about your approach, or just try random things?

3. **Technical Approach:** Did you make tuning and configuration decisions using quantitative assessment? Did you dive deeply into the methods?

4. **Explanation:** Do you explain not just what you did, but your thought process for your approach? Do you present evidence for your conclusions in the form of figures and tables? Do you provide references to resources your used? Do you clearly explain and label the figures in your report?

5. **Execution:** Did you make an agent that can play the game? Did your methods give reasonable performance?

What language should I code in? You can code the warmup in whatever language you find most productive. You'll need to do the Swingy Monkey problem in Python, however, as that's the language the game is written in.

Can I use {scikit-learn | pylearn | torch | shogun | other ML library}? You can use these tools, but not blindly. You are expected to show a deep understanding of the methods we study in the course, and your writeup will be where you demonstrate this. You should implement the MDP and reinforcement learning parts of this practical yourself.

These practicals do not have conceptual questions. How will I get practice for the midterms? We will provide practice problems and solutions in section. You should work through these to help learn the material and prepare for the exams. They will not be a part of your grade.

Can I have an extension? There are no extensions. Your writeup can be turned in up to a week late for a 50% penalty. There are no exceptions, so plan ahead.

Changelog

This format for assignments is somewhat experimental and so we may need to tweak things slightly over time. In order to be transparent about this, a changelog is provided below.

- **v1.0** – 30 March 2014 at 15:00