Project #3: Multi-Agent Reinforcement Learning

Due Date: 05/09/2022, Monday, 10 pm

This is your last project for this course — that is, we will not have the final course project any more. This project will be more open-ended than project 1 and 2, but we have tried to provide as much guidance as possible so hopefully you have a clear path about what to do. Note that, you will work with the same group members, as you did for project 1 and 2.

Project Overview. This project is about multi-agent reinforcement learning for playing entertainment games. The goal is to design an RL computer agent that can compete with humans like you or me. In this project, you will pick a simple entertainment game of your own interest (e.g., battleship games, one-card poker, etc.), implement and train *one RL agent* to play this game, and demonstrate your RL agent by playing with a human (e.g., yourself).

Deliverables. Like project 1 and 2, you will need to submit a similar **report** to: (a) explain your project results; (b) explain your implemented methods; (c) report the performance of your RL agent. However, besides these, you are additionally asked to submit your **source code** and a **recorded video (up to 10 mins)** which explains the rule of your game and demonstrates a play between you and your RL agent.
(To record the video, you can simply play with the game on your computer, and record the play.)

Task I: Read and Try

This is a simple warm-up task. In the project folder, we have already provided the source code for an RL agent that can successfully play the tic-tac-toe game on a 3×3 board. The RL agent uses the the simple ϵ -Greedy RL algorithm, which is the simplest possible (and also super intuitive) RL algorithm. We will cover this algorithm in class, but we have also included a description of this algorithm in the project folder.
The purpose of this Task I is simply to get your familiar with the implementation of RL agent and then play with it on your computer. Specifically, you are asked to do the following:

- Read the source code in detail, and understand the structure of the code, what modules will you need to implement, and how they are implemented in this example code.
- Tried this code in your computer, and see what output it generates. As a suggestion, you can try your agent using 1000 epochs (each epoch = a sample play of the game), and see how strong an RL agent you can train with 1000 epochs.
- You play the tic-tac-toe with trained RL agent for 10 rounds (you will need to understand the code in order to see how you can play with the RL agent). Repeat this with three different RL agents, which are trained with 1000 epochs, 5000 epochs, 10000 epochs respectively. Report how many times you won and how many ties you reach in each situation.

Task II: Do it by Yourself

This is the main part — here you will need to implement a similar project like the tic-tac-toe RL agent we provided. In this task II, you can pick an *arbitrary entertainment game* of your interest, except that it cannot be tic-tac-toe.
Some example games include

1. Checkers
2. Battleship game (you can simplify it to, e.g., 6×6 board with one 2-cell ship for each player)
3. Black jack (can have < 54 cards if you want to simplify it)
4. Any fun poker games (e.g., one-card poker http://www.cs.cmu.edu/ ggordon/poker/ or texas hold'em).
5. ...

We want to emphasize that you are free to pick any game of your interest (and this is the best time to try it out with an algorithm!). Moreover, you can make modifications to either simplify or complicate the game, all up to your wish.
You are asked to implement *an RL agent* to play this game of your choice. In this task, you will need to think about how to formulate your game as an RL problem (e.g., what is a state, what is state transition, how to model state value, etc.), how to implement an RL algorithm, and how to make your agent play with a real human. You should be able to see how the tic-tac-toe example code did these.
The grading will take into account how complicated your game is and how well your RL agent performs.