Homework 3

CS 4710

March 26, 2018

Instructions

- Submit one file per group. In that file, define the following constants relative to your group:

```
TEAM_NAME = "mighty_ducks" #Pick a team name
MEMBERS = ["mrf9n","jr5cf","kc4bf"] #Include a list of your members' UVA IDs
```

- Be absolutely sure your code works against the interfaces described. Your single file submission must be importable as a library in a fresh Python3 interpreter without issue.

Trial 1: Chicken

Chicken is a game where two players compete heads' up. The game models as if both players are driving cars head-on at each other. The only decision is at what time you will swerve (if at all). The player who swerves first suffers a minor loss, granting a victory to the opponent. If neither player swerves in time and they crash, both suffer a crushing defeat. If both swerve at exactly the same time, they tie.

	Swerve in time	Don't swerve in time
Swerve in time	0, 0	-1, +1
Don't swerve in time	+1, -1	-10, -10

When a round begins, my program will call a function called **get_move(state)** in your file. An example of a value of state and a rough interface for get_move appear below. This is meant to take in the current "state of the game" and return what move your player will choose. Players have thirty seconds per round to submit a move.

Each team will be assigned a code. You must report that code back with your move so it can be matched properly. You will be able to see your opponent's name, but not their code (so you can't simply move on their behalf).

This game is scored **cumulatively**. The points from each round played against each opponent will be **summed up** across all opponents to create the rankings.

Each round a random value will be rolled which is called "reaction time". If your bot has chosen a move that is less than the reaction time, your bot may not have an opportunity to swerve in time even if it intends to. For example: if you choose to swerve with 0.01 seconds left, and the reaction time is 0.05, then you will **collide** if your opponent hasn't swerved by 0.05.

You can use **load_data() and save_data(info)** to record what you know and reload it whenever your program is called for a move. **To use load and save properly, the "info" field should be in JSON format.** For example:

```
info = {
   "past-reaction-times": [],
   "player-behavior": {
      "player1": [],
      "player2": []
   }
}

save_data(info)
loaded_info = load_data() #Would be silly to load just after saving, just including this
    as example of call
```

Below is an example of a state object and a get_move interface.

```
state = {
   "team-code": "eef8976e",
   "game": "chicken",
   "opponent-name": "mighty_ducks"
}

def get_move(state):
   # Your code can be called from here however you like
   # You are allowed the use of load_data() and save_data(info)
   info = load_data()
   # But you must return a valid move that looks like the following:
   return {
      "move": 0.25, # Number of seconds at which we swerve
      "team-code": "eef8976e" # Must match the code received in the state object
   }
```

Trial 2: Connect More

“Connect Four” is a game using the following style board:

Typically the game is played on a 7 by 6 board as shown above. Players alternate, each placing a single disk into a single column at a time. This fills one of the holes on the board as the disk slips down to the lowest unoccupied hole.

The goal is to get four of your color in a row, either cardinally or diagonally. The first player to do this wins and the game halts. **These games will be played best-of-9 where players get +1 for winning a best-of-9, and 0 for losing.**

This isn’t just Connect Four, though. First of all, I will choose an arbitrary number of columns at the beginning of play. You may use load_data and save_data in this game. Also, **there is no height limit**. In other words, players could conceivably create an arbitrarily high stack in this version of the game. **I will also choose the number which you must connect, which might be different than four.** The first to connect the specified number wins. Players have one minute a piece to complete a round – if the clock runs out, you lose. The “your-token” field in the state will let you know how to read the information contained within the board.

```
state = {
   "team-code": "eef8976e",
   "game": "connect_more",
   "opponent-name": "mighty_ducks",
   "columns": 6,
   "connect_n": 5,
   "your-token": "R",
   "board": [
      ["R","Y"],
      ["R"],
      [],
      ["R",],
      ["Y","Y"],
      [],
   ]
}

def get_move(state):
   # Your code can be called from here however you like
   # You are allowed the use of load_data() and save_data(info)
   info = load_data()
   # But you must return a valid move that looks like the following:
   return {
      "move": 2, # Column in which you will move (create mark "your-token")
      "team-code": "eef8976e" # Must match the code received in the state object
   }
```

GOOD LUCK!!!