CISS450: Artificial Intelligence
Project p01: CISS450 Game AI

Stuff in red is from the website team.

GENERAL INFORMATION

Here's some history ... In Fall 2016, a website was built for experimentation and teaching of game AI. The course is CISS450 at Columbia College. The goal is to allow students to upload python code to play othello against one another. The reason for building the website is because I (Y.Liow) wanted to use a tournament for teaching game AI. At that time I thought I could use a website for that. After searching on the web for 2-3 days, I gave up. I talked to one of my students (since graduated), Seth Kasmann, and he volunteered to search for such a website. After about 1-2 weeks he told me that he was certain that no such website existed. So I decided to create one.

Here are the names of the original team members who built the website with me:

- Frappier, Ryan (graduated, currently at Google)
- Pandey, Ujjwal (graduated, currently at Amazon)
- Woods, Andrew (graduated, currently a PhD student at U of MO)

The URL of the website is currently `http://ciss450othello.pythonanywhere.com`.

Here are videos from the first othello tournament in fall 2016:

- `https://goo.gl/photos/DLpwRSy29HS2RaYD6` part 1 (destruction: 21:56, 33:30)
- `https://goo.gl/photos/R284gUkJtfanFT2c7` part 2 (destruction: 10:05, 13:38)

Unfortunately the original 2016 website was automatically removed due to inactivity. You can view all matches since 2018: `http://ciss450othello.pythonanywhere.com/matches`. (You need to use chrome.)

The long term goal is to add other two-player turn based board games such as chess, go, etc. and also to open the website to other schools and the public.

CISS450 Othello tournament

Game: Othello
Team size: 1 or 2
Due: End of semester
Language: Python (pure), 2.7

Read the othello rules: http://www.hannu.se/games/othello/rules.htm. For our tournament website, we have the following additional rules:

- Board size need not be 8-by-8 (but will be symmetric). You should handle 6-by-6, 8-by-8, 10-by-10, 12-by-12, etc.
- Each match has a maximum total time for each AI program.
- The maximum total time is not fixed.
- Follow standard othello rules (other than non-standard board size).
- A wrong move will end the match with the opponent being the winner.

The actual board sizes and associated times will be announced Mon of the finals week.

Your code is used this way:

- Your code is called to get an action. — (see header for the function below)
  - Return move as [row, col].
  - Return None for no move. You must return None if you cannot make a move.
  - If you make a wrong move, the game ends with you as the loser.
  - If you run out of time, you lose.
- Data provided to your code includes — (see header for the function below)
  - Your color: `'W'` or `'B'`
  - Board state: 2-d array of `'W'`, `'B'`, `' '` (space)
  - Your time left: `int` (in milliseconds)
  - Opponent's time left: int (in milliseconds)
  - Amount of data storage space [?] and code size (in KB). — Not provided; if you guys end up having storage space needs beyond embedding static data into your code directly, we *might* add something, but no promises. So talk to us first before you move to that or just do it for yourself for fun.
- Function header: `get_move(board_size, board_state, turn, time_left, opponent_time_left)`
  - What you upload for an AI must be a single python file
  - You must have a function with the above signature in the file you upload, and you must follow the order of the arguments
  - `board_size` is an integer
  - `board_state` is a 2d array of `'W'`, `'B'`, `' '` (space)

- `turn` is just the character whose turn it is (`'B'` or `'W'`)
- `time_left` and `opponent_time_left` are in milliseconds, integers
- Return the move. The format for the move is `[row, col]` where `row`, `col` are integers or `None` if you cannot make a move. You don't need to print the move – just return the move from this function.

TODO

The following are tasks that you should start working on right away.

- Given a (board) state and a color (`'B'`, `'W'`), compute all actions. Test thoroughly. There's no reason to make any mistakes here – this is a high level class.
- Write a dumb AI program that returns a random (and valid) move from the list of all possible actions.
- Main algorithm: minimax with alpha-beta pruning, ... (See slides and AIMA textbook).
- Read up on Othello strategies (not in the book).

Testing locally

Kevin Weston (graduated, currently at Veterans United) has written a python program (fall 2018) that you can use to test your program locally on your laptop/PC. *This is provided as is.* (For instance there are indentations to fix.) Once you have written a dumb AI that returns a random move, you can run the dumb AI against itself. Once you have written your first AI, test it against the dumb AI. After you have version n, test version n against version $n - 1$. Etc.

```
## Author: Kevin Weston

## This program will simulate a game of othello in the console
## In order to use it, you need to:
## 1. Import code for your AIs.
## 2. You also need to import a function which applies an action to the board state,
##        and replace my function call (main.get_action(...)) within the while loop.
## 3. Your AIs must have a get_move() function. Assign these functions to
##        white_get_move and black_get_move.
## 4. Create an initial board state to be played on

## Replace these imports with the file(s) that contain your ai(s)
import main
import Rando
import MiniMaximus

def get_winner(board_state):
    black_score = 0
    white_score = 0
    for row in board_state:
            for col in row:
            if col == 'W':
                    white_score += 1
            elif col == 'B':
                    black_score += 1
    if black_score > white_score:
            winner = 'B'
    elif white_score > black_score:
            winner = 'W'
    else:
            winner = None
    return (winner, white_score, black_score)


def prepare_next_turn(turn, white_get_move, black_get_move):
    next_turn = 'W' if turn == 'B' else 'B'
    next_move_function = white_get_move if next_turn == 'W' else black_get_move
    return next_turn, next_move_function
```

```
def print_board(board_state):
    for row in board_state:
            print(row)


def simulate_game(board_state,
                        board_size,
                        white_get_move,
                        black_get_move):
    player_blocked = False
    turn = 'B'
    get_move = black_get_move
    print_board(board_state)

    while True:
            ## GET ACTION ##
            next_action = get_move(board_size=board_size,
                               board_state=board_state,
                               turn=turn,
                               time_left=0,
                               opponent_time_left=0)
            print("turn: ", turn, "next action: ", next_action)
            _ = input()

            ## CHECK FOR BLOCKED PLAYER ##
            if next_action is None:
            if player_blocked:
                    print("Both players blocked!")
                    break
            else:
                    player_blocked = True
                    turn, get_move = prepare_next_turn(turn,
                                                   white_get_move,
                                                   black_get_move)
            continue
            else:
            player_blocked = False

            ## APPLY ACTION ##
            ## Replace this function with your own apply function
            board_state = main.apply_action(board_state=board_state,
                                            action=next_action,
                                            turn=turn)
            print_board(board_state)
            turn, get_move = prepare_next_turn(turn,
                                           white_get_move,
                                           black_get_move)

        winner, white_score, black_score = get_winner(board_state)

        print("Winner: ", winner)
        print("White score: ", white_score)
        print("Black score: ", black_score)


if __name__ == "__main__":
    ## Replace with whatever board size you want to run on
    board_state = [[' ', ' ', ' ', ' '],
                          [' ', 'W', 'B', ' '],
                          [' ', 'B', 'W', ' '],
                          [' ', ' ', ' ', ' ']]
    board_size = 4

    ## Give these the get_move functions from whatever ais you want to test
    white_get_move = Rando.get_move
    black_get_move = MiniMaximus.get_move
    simulate_game(board_state, board_size, white_get_move, black_get_move)
```

Using the AI website

When your othello AI is working, it's time to go to the tournament website.

The URL is `https://ciss450othello.pythonanywhere.com/`. If you get an error while using the website, email me (`yliow@ccis.edu`) ASAP.

First thing to do is to

- Create an account for your team.

NOTE: The AI website uses a free account and has limited storage space and CPU cycles. Don't overuse it. You should definitely test your code locally (on your laptop) most of the time. Near the end of the semester, I will upgrade the account to a paid account to get more CPU time.

It's easy to use the website:

- View users: click on "Users" at the top of the main page.
- View matches: click on "Matches" at the top of the main page and then click on the "Match ID" (an integer). Note that you must use the chrome browser.
- Upload an AI: login, click on "Add AI" (top right), and upload your code.
- Create a match: login, click on "Create a match" (top right), complete the form, and submit. Go to the matches page and check on the status of that match.
  - Your match is executed by a separate process (not through the web server) that runs 24/7. Right now this process runs with a pause of 60s-120s. So a 2-minute match might finish, say, in 5 minutes. If the cpu seconds for a day is used up, processes will run much slower and might take longer to complete.
  - NOTE: You should let me (`yliow@ccis.edu`) know if your match does not complete for a very long time say 30 minutes.

Time

Here's an example on timing:

```
import resource

def gettime():
        rs = resource.getrusage(resource.RUSAGE_SELF)
        return rs[0] + rs[1]

t0 = gettime()
print(t0)

for i in range(100000):
        j = i * i * i

t1 = gettime()
print(t1)
print(t1 - t0)

j = 1
for i in range(100000):
        j = i * i * i * i * i

t2 = gettime()
print(t2)
print(t2 - t1)

j = 1
for i in range(100000):
        j = i ** 10

t3 = gettime()
print(t3)
print(t3 - t2)

j = 1
for i in range(100000):
        j = i ** 100

t4 = gettime()
print(t3)
print(t4 - t3)

# nothing
```

```

t5 = gettime()
print(t5)
print(t5 - t4)

print(t5 - t0)
```

Based on the board and your time left, you need to figure out how much time you should allocate for your current get move function call. Remember that if you timeout, you lose the game. Timing is usually pretty accurate, but it's not perfect and exact. So be very careful and conservative with your time.

RUNNER

The match you created at the AI tournament website will usually not be completed. To get the website to finish the match you can run the following in your Fedora virtual machine. You will need to enter your alex user code in the program. You will also need to tell Dr. Liow to add your alex user code to the list of users with permission to execute runner.py.

```
# file: runner.py

#
# your alex user code is in ~/.alex/alex
#
user = 'put-your-alex-user-code-here'

LOOP = True
T = MIN_T = 5 * 60 # once every 5 minutes
dT = 5             # if no jobs found, sleep time T increments by dT.
                   # if a job is found, T resets to MIN_T.
MAX_T = 10 * 60    # max sleep time between runs

import time
import datetime
import urllib.request
import random; random.seed()

start = datetime.datetime.now()

while 1:
    print()
    print("talking to https://ciss450othello.pythonanywhere.com/ ...", flush=True)
    url = 'https://ciss450othello.pythonanywhere.com/s?user=%s' % user
    with urllib.request.urlopen(url) as response:
        now = datetime.datetime.now()
        dt = now - start
        html = response.read().decode().strip()
        if html == 'ERROR':
            print('ERROR')
            break
        else:
            print("now: %s\ndiff: %s\nT: %s\nhtml: %s" % (now, dt, T, html))
            if 'no incomplete matches found' in html:
                #print("no job")
                T += dT
                if T >= MAX_T:
                    T = MAX_T
            else:
                #print("job found")
                T = MIN_T

    if not LOOP:
        print("LOOP is False ... halting ...")
        break

    #==========================================================================
    # Print pause message
    #==========================================================================
    next_T = T + random.randrange(0, 60)
    dt = 1 # Print the pause message once every dt seconds.
           # Increase dt to save on local CPU cycles.
    while 1:
        print(("Ctrl-C to stop ... next run in %s sec        " % next_T) + '\b' * 100,
              flush=True, end='')
        if next_T == 0: break;
        if next_T > dt:
            time.sleep(dt)
            next_T -= dt
        else:
            time.sleep(next_T)
            next_T = 0
    print()
```

Admin note: To add alex usercode for permission for runner, see function `s()` in `/home/ciss450othello`

Dev and admin

https://docs.google.com/document/d/1cgF2YP0RE2GibfVoJeXV3oTY45XGAhFAJAI00wCoed8/edit?usp=sharing