B551 Assignment 2: Games

Fall 2016
Due: Sunday October 16, 11:59PM
(You may submit up to 48 hours late for a 10% penalty.)

This assignment will give you practice with adversarial and stochastic games.

You'll work in a group of 2-4 people for this assignment; we've already assigned you to a group (see details below) based on the input you provided with your A1 team feedback. We tried to accomodate as many of your requests as possible, but we could not satisfy all of them. You should only submit **one** copy of the assignment for your team, through GitHub, as described below. All people on the team will receive the same grade on the assignment, except in unusual circumstances; we will collect feedback about how well your team functioned in order to detect these circumstances. Please read the instructions below carefully; we cannot accept any submissions that do not follow the instructions given here. Most importantly: please **start early,** and ask questions on Piazza or in office hours.

The following problems require you to write programs in Python. You may import standard Python modules for routines not related to AI, such as basic sorting algorithms and basic data structures like queues. You must write all of the rest of the code yourself. If you have any questions about this policy, please ask us. We recommend using the CS Linux machines (e.g. `burrow.soic.indiana.edu`). You may use another development platform (e.g. Windows), but make sure that your code works on the CS Linux machines before submission.

For each programming problem, please include a detailed comments section at the top of your code that describes: (1) a description of how you formulated the search problem, including precisely defining the state space, the successor function, the edge weights, and any heuristics you designed; (2) a brief description of how your search algorithm works; (3) a discussion of any problems you faced, any assumptions, simplifications, and/or design decisions you made; and (4) answers to any questions asked below in the assignment.

Academic integrity. You and your teammates may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials (e.g. in source code comments). However, the work and code that you and your partners submit must be your group's own work, which your group personally designed and wrote. You may not share written answers or code with any other students except your own partner, nor may you possess code written by another student who is not your partner, either in whole or in part, regardless of format.

Part 0: Getting started

You can find your assigned teammate by logging into IU Github, at `http://github.iu.edu/`. In the upper left hand corner of the screen, you should see a pull-down menu. Select `cs-b551-fa2016`. Then in the yellow box to the right, you should see a repository called *userid1-userid2-userid3*-a2, where the other user IDs correspond to your teammate(s).

To get started, clone the github repository:

`git clone https://github.iu.edu/cs-b551-fa2016/`*your-repo-name*`-a2`

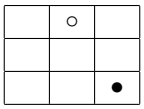
where *your-repo-name* is the one you found on the GitHub website above.

Part 1: n-k-coh-coh

n-k-coh-coh is a popular childhood game in a certain rural midwestern town that requires just a board consisting of a grid of $n \times n$ squares and some white and black marbles. Initially the board starts empty

and all marbles are in a pile beside the board. Player 1 picks up a white marble and places it in any square of the board. Player 2 then picks up a black marble from the pile, and places it in any open square (i.e. any square except the one selected by Player 1). Play continues back and forth, with Player 1 always using white marbles and Player 2 always using black. A player *loses* the game as soon as they place a marble such that there is a continuous line of k marbles of his or her color in the same row, column, or diagonal of the board. (For example, note that 3-3-coh-coh is nearly the same as tic-tac-toe, except that players are trying to avoid completing a row, column or diagonal instead of trying to complete one.)

Your task is to write a Python program that plays n-k-coh-coh well. Your program should accept a command line argument that gives the current state of the board as a string of `w`'s, `b`'s, and `.`'s, which indicate which squares are filled with a white, black, or no marble, respectively, in row-major order. For example, if $n = 3$ and the state of the board is:



then the encoding of the state would be:

```
.w.....b
```

More precisely, your program will be called with four command line parameters: (1) the value of n, (2) the value of k, (3) the state of the board, encoded as above, and (4) a time limit in seconds. Your program should then decide a recommended move given the current board state, and display the new state of the board after making that move, *within the number of seconds specified.* Displaying multiple lines of output is fine as long as the last line has the recommended board state. For example, a sample run of your program might look like:

```
[djcran@macbook]$ python nkcohcoh.py 3 3 .w.....b 5
Thinking! Please wait...

Hmm, I'd recommend putting your marble at row 2, column 1.
New board:
.w.w....b
```

The tournament. To make things more interesting, we will hold a competition among all submitted solutions. We will not reveal ahead of time the values of n, k, or the time limit that we will enforce, but we plan to hold multiple tournaments with relatively low and high values of n and k and the time limit. While the majority of your grade will be on correctness, programming style, quality of answers given in comments, etc., a small portion may be based on how well your code performs in the tournaments, with particularly well-performing programs eligible for prizes including extra credit points.

Note: Your code most conform with the interface standards mentioned above! The last line of the output *must* be the new board in the format given, without any extra characters or empty lines. Also, note that your program cannot assume that the game will be run in sequence from start to end; given a current board position on the command line, your code must find a recommended next best move. Your program can write files to disk to preserve state between runs, but should correctly handle the case when a new board state is presented to your program that is unrelated to the last state it saw.

Part 2: Tetris

Tetris is one of the most popular games of all time, having generated over $1 billion dollars in sales across dozens of devices and gaming platforms. You're probably familiar with the game. It starts off with a blank

board. One by one, a random piece (consisting of 4 blocks arranged in different shapes) falls from the top of the board to the bottom. The player can change the shape by rotating it or moving it left or right, in order to control where it lands when it hits the bottom of the board. It stops whenever it hits the ground or another fallen piece. If the piece completes an entire row, the row disappears and the player receives a point. The goal is for the player to score as many points before the board fills up.

We've prepared a simple implementation of Tetris that you can play from the command line! To try it, run:

```
python ./tetris.py human animated
```

To play, use the `b` and `m` keys to move the falling piece left and right, the `n` key to rotate, and the space bar to cause the piece to fall. (This works on the SOIC Linux machines and should work on Mac OS command line. It may or may not work on Windows or other platforms because of the way it handles keyboard input. If it doesn't work for you, you can use a simpler but less-fun interface by running `python ./tetris.py human simple`.)

Your goal is to write a computer player for this game that scores as high as possible. We've provided a really simple automatic player that can be run using the command line options `computer animated` and `computer simple`. The code for this is in `tetris.py`, whereas the other python files contain the "back end" code that runs the game. You should not modify those other source code files, since that would not be fair (it would be like changing the rules of the game!) and may prevent us from grading your submission accurately.

We'd recommend starting with the simple version as opposed to the animated version. In the simple version, your program issues a sequence of commands which are then executed immediately before the piece begins to fall. This is simpler but prevents your program from making complicated moves (like moving left and right as the piece falls to try to wedge a piece into an awkward spot in the board). Then, consider the animated version, which also introduces an implicit time limit (since your decisions have to be made before the piece reaches the bottom of the board.)

This version of Tetris has one twist which you may want to use to get as high a score as possible: the falling pieces are not chosen uniformly at random but based on some distribution which your program is not allowed to know ahead of time. However, it may be able to estimate the distribution as it plays, which may let it make better decisions over time.

The tournament. To make things more interesting, we will hold a competition among all submitted solutions to see whose solution can score the highest across several different games. While the majority of your grade will be on correctness, programming style, quality of answers given in comments, etc., a small portion may be based on how well your code performs in the tournaments, with particularly well-performing programs eligible for prizes including extra credit points.

Extra credit. If you have time, create an *adversarial* version of Tetris that tries to "play against" the human player by choosing falling pieces in an attempt to cause them to lose, given their current board position and any other information you want to use. This mode should be selected if the user runs `python tetris.py human animated adversarial`.

What to turn in

Turn in the files required above by simply putting the finished version (of the code with comments and PDF file for the first question) on GitHub (remember to `add`, `commit`, `push`) — we'll grade whatever version you've put there as of 11:59PM on the due date. To make sure that the latest version of your work has been accepted by GitHub, you can log into the github.iu.edu website and browse the code online.