Assignment 3 - Tak (Phase I)

Goal: The goal of this assignment is to learn the adversarial search algorithms (minimax and alpha beta pruning), which arise in sequential, deterministic, adversarial situations.

The Game of Tak(n): The game of Tak is played between 2 players (black and white) on an $n \times n$ board with the objective of creating a *road*, which is a line of your own pieces connecting two opposite sides of the board. Each square of the board can contain one or more pieces forming a stack, and each square along the road you build must have a flatstone or capstone of your color on top. The 5x5 version of the game typically has 21 flatstones and 1 capstone per player. To read the full rules of the game, visit `http://cheapass.com/sites/default/files/TakBetaRules3-10-16.pdf` and watch the video at `http://cheapass.com/tak/` to get an idea of the gameplay.

Figure 1: Road win for black (image courtesy cheapass.com)

Algorithm: Implement this game as an instance of minimax search with cutoff and alpha-beta pruning. Learn a rudimentary evaluation function (define features and set weights of features either by hand or by a learning procedure pitting your player against yourself). You are encouraged to gain insight in your player by pitting it against other teams.

What is being provided: Your assignment packet contains code for the game server in python and client code for your player in python that: (1) interacts with the game server, (2) allows you to send moves to the server. You are also provided a GUI which allows you to visualize the proceedings of the game. The code can be found at: `https://github.com/akshaykgupta/Tak-sim`

Interaction with Game Server:

Running the server: The server is run using the command:
python server.py <port no.> <no. of clients>

Running the client: The client is run using the command:
python client.py <server ip> <port no.> <executable>
The executable must be a bash script which runs your code.

Initialisation: After the clients have connected, the server sends to both clients game information as a string in the format: '<player no.> <N> <time limit>'
Player no. is either 1 or 2
N is size of board (5,6,7 etc.)
Time limit is total time allotted to a player for the game in seconds
The three terms are space-separated.

Sending moves to the server:

You must write your move to stdout as a string (described below) followed by a '\n'. Similarly you can read the move of your opponent from stdin. **Note:** You can write debugging/error messages to stderr. Do not write anything except your move to stdout.

Tak has two types of moves, either placing a stone on the board or moving a stack. A square on the board is specified by a lowercase letter (starting from a) followed by a digit (starting from 1) as in chess.

1. If you are placing a stone, first you need to specify what *kind* of stone you are placing. This is specified by the letter F, S or C, indicating flat stone, wall (standing stone), or capstone respectively. This is followed by the *square* on which stone is being placed. Eg:
   (a) Place a flat stone on the square a1 : Fa1
   (b) Place a wall at d3 : Sd3
   (c) Place a capstone at b4 : Cb4
2. If you are moving a stack on the board, first you need to specify the *number* of stones being moved from a stack (eg. 4). Then you specify the *square* from which the stack is being moved (eg. c3). Then you need to specify the *direction* in which the stones picked will move. This is specified by one of the following symbols: < (moving towards the letter a on board), > (the opposite direction), - (moving towards the number 1 on board) or + (the opposite direction). Finally you specify the *number of stones to drop on each square in the given direction.* Eg:
   (a) Move a single stone from a1 to b1 : 1a1>1

(b) Move 4 stones from d3 to d2 : 4d3-4

(c) Move 4 stones from b2, dropping two on b3, one on b4 and one on b5 : 3b2+211

(d) Move 5 stones from e4, dropping two on d4 and three on c4 : 5e4<23

You are also expected to keep track of your opponent's moves and game state on your own. There is a total time assigned to you that gets decremented when your turn is going on which you should aim to stay inside.

Code: Your code must compile and run on our VMs. They run amd64 Linux version Ubuntu 12.04. You are already provided information on the compilers on those VMs. These configurations are similar to GCL machines like 'Todi' (have a RAM of 16 GB). Please supply a compile.sh script for compilation. Also supply a shell script run.sh. This shell script is the 'executable' command line argument to client.py. './run.sh' should run your code.

What to Submit:

1. Submit your code in a .zip file named in the format <EntryNo>.zip. If there are two members in your team it should be called <EntryNo1>_<EntryNo2>.zip. Make sure that when we run "unzip yourfile.zip" the following files are produced in the present working directory:
   compile.sh
   run.sh
   writeup.txt

2. The writeup.txt should have two lines as follows:
   First line should be just a number between 1 and 3. Number 1 means C++. Number 2 means Java and Number 3 means Python.
   Second line should mention names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone else say None.
   After these first two lines you are welcome to write something about your code, though this is not necessary.

Code verification before submission:

Your submission will be auto-graded. This means that it is absolutely essential to make sure that your code follows the input/output specifications of the assignment. Failure to follow any instruction will incur significant penalty. The details of code verification will be shared on Piazza (similar to A1).

Evaluation Criteria:

1. In Phase 1 we will test your code against simple baseline players for Tak(5). The final score is calculated as follows:
   The winning player can win either by building a road (road win) or by owning more squares on the board in case any player runs out of unplayed flatstones (flat win). Some guidebooks mention a third way to win called double win. For our assignment, it will be evaluated just as a road win.

In case of either win, the winning player is first awarded points equal to the number of *unplayed* flatstones of the winning player. In addition to this, if the player scored a road win, it is awarded points equal to the size of the board (25 for 5x5). If the player scored a flat win, it is awarded points equal to the squares owned by that player on the board. The losing player is always awarded points based on the number of squares owned by that player. Each game's scores are normalized between 0 and 1 for computation of the final grade on the assignment.

2. Extra credit may be awarded to standout performers.

3. The Phase 1 and Phase 2 of the project jointly carry weight of two programming assignments. Phase 1 carries only 25% of this weight and Phase 2 carries 75%.

What is allowed? What is not?

1. You may work in teams of two or by yourself. If you work in a team of two then make sure you mention the team details in the write-up. Our recommendation: this will lead to the much more open ended final assignment (phase 2); hence best to work with a partner with whom you communicate well. You will be allowed to use the same partner for the final assignment. Also, you cannot use the partner from previous assignments.

2. You can work in C++, Java or Python. It is recommended that you work in C++.

3. Your code must be your own. You are not to take guidance from any general purpose AI code or problem specific code meant to solve this or related problem.

4. It is preferable to develop your algorithm using your own efforts. However, we will not stop you from google searching.

5. You must not discuss this assignment with anyone outside the class. Make sure you mention the names in your write-up in case you discuss with anyone from within the class outside your team. Please read academic integrity guidelines on the course home page and follow them carefully.

6. You get a zero if your player does not match with the interaction guidelines in this document.

7. We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.