Project Statement - TAK

Goal: The goal of this Project 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)

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

There is a total time assigned to you that gets decremented when your turn is going on which you should aim to stay inside.

-