Artificial Intelligence

ASSIGNMENT 4

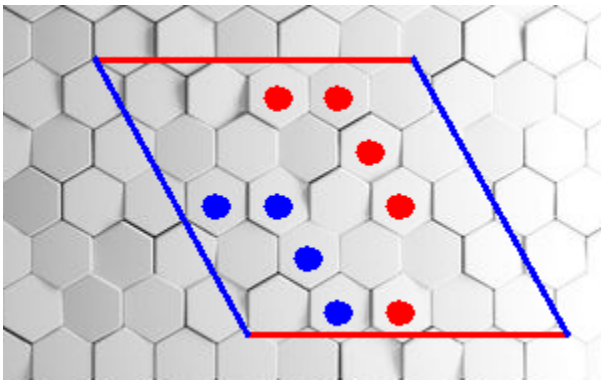
Due Date: 19 May 2021

QUESTION 1

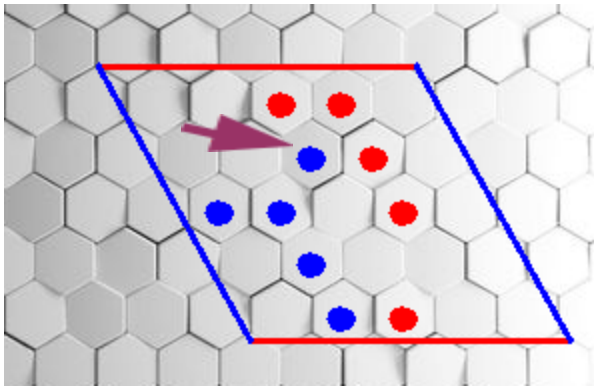
Consider the programs in the *Files* folder. The file *RunDemo.bat* will open a two player human Hex board which you can use to familiarize yourself with the game (no time limits are placed on moves in this example). Note that the red player always plays first.

The file *RunServer.bat* will open a server application which will wait for two client players to connect via sockets. A dialog box provides options for just a single game or a tournament. A tournament will consist of 8 games to be played on four board sizes starting at the board size you enter (two games on each board size). The server application will write the current board state to the client and then read a move from the client. Responses must occur within 20 seconds (or whatever the response time is that you selected) otherwise the game will be forfeit. The following communication protocol is used between server and client (all messages are in the form of strings):

1. After connection occurs the client writes a player name to the server. Eg.: "Bob"
2. The server responds with a board size, colour of the player (1 for red, 2 for blue), and maximum response time in seconds. The values are separated by commas, e.g.: "8,2,20" means that an 8 by 8 board will be used, the player is playing as blue, and moves must be made within 20 seconds.
3. The server will prompt a player to make a move by sending the current board state. This will be a *n*x*n* list of board positions, one row at a time (*n* is the board size). Empty tiles are represented by "0" while red and blue plays are represented by "1" and "2" respectively. E.g. the string "0,0,1,1,0,0,0,0,1,0,2,2,0,1,0,0,2,0,0,0,0,2,1,0,0" encodes the following board:



4. The client must respond with a move within the time limit. The response is two integers, the column and row value where the move must be made, separated by a comma. E.g. the response "2,1" will represent the following move:



5. The only other communication from the server the notification of the game ending when one of the players have won. This will take the form of either a "1" if red won or a "2" if blue won.

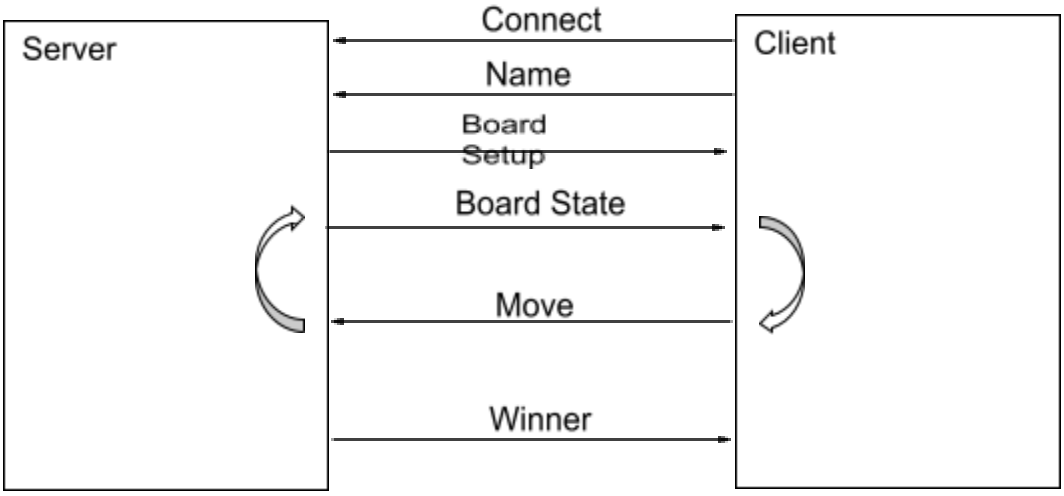
The server thus sends three types of messages:

1. The board setup which is sent in response to the player name from the client, e.g. "8,2,20". This is sent only once.
2. The board state which prompts the player to make a move, e.g. "0,0,1,1,0,0,0,0,1,0,2,2,0,1,0,0,2,0,0,0,0,2,1,0,0". This type of message will typically be sent several times during the course of a game.
3. The winner at the end of the game, e.g. "2". This is sent only once.

The client player sends two types of messages:

1. The name of the player, e.g. "Bob". This is sent only once.
2. A move in response to a board state message from the server, e.g. "4,2". This is sent several times during the course of the game.

The following diagram shows the client-server interaction:



The server will record the result of each tournament in a text file *GameResults.txt*. It is important for each player to have a unique name so that the server can distinguish between adversaries (currently, this is achieved by appending to date and time to the name of the random player when it is sent to the server).

The *ClientPlayer* folder contains code for the client player. Dialog boxes allows the user to select where the server is running (if not local then the IP address of the computer must be entered), whether a single game will be played or a tournament, and the type of player. At the moment, only *RandomPlayer* and *HumanPlayer* are implemented. You can play against the random or another human player by running the server and then launching two instances of *Clientplayer*.

Each player must implement the *Player* interface. It contains two methods, *GetConnection* (which you can cut and paste from *RandomPlayer*) and *MakeMove*. The main loop which waits for messages from the server takes place in the constructor which you can also copy from *RandomPlayer*. *MakeMove* takes as input the board state from the server must return a string representing a valid move. Note that you are responsible for ensuring that this method returns a move within the allowed time period. Your code does not have to check whether the end of the game has been reached, as this will be handled by the server (i.e. *MakeMove* will only be called if the game has not ended and there are still valid moves available to make).

You have to implement three AI players for the game of the Hex.

- A MinMax player (note that this player will probably be too slow to run on any boards larger than 3x3)
- A Monte Carlo Player
- A Monte Carlo Player that uses UCT's Bandit Heuristic to explore moves more effectively (i.e. you do not have to implement the full UCT algorithm).

You can test your players against the random player and against each other. Investigate performance of different board sizes with different response times. Eventually your players will be evaluated against a player I wrote.

The due date for this assignment is 19 May 2021, however, manage your time by implementing at least one player every week.

For next week, make your own random player with the following characteristics:

- If it is possible to win in on the current move, the player will make the winning move.
- If possible, it will always place a stone on a randomly selected hexagon next to a hexagon occupied by its opponent, otherwise it places a stone on an a randomly selected empty hexagon.

The purpose of this task is to familiarize yourself with the code and is thus not for marks.