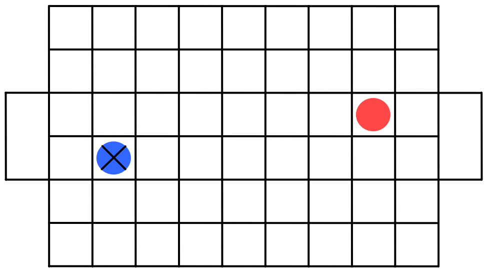
Machine Learning Assignment #2

Simple Grid Soccer
Description of Client/Server System



The Client/Server system for Simple Grid Soccer has four components:

- Soccer Server
- Soccer Monitor
- Log Player
- Simple (random) Client

All these programs have been written using Borland Delphi 7 and the source code will be available on the course web page.
The Soccer Server is the main component of this system. This server receives commands from both players, performs their command and then sends messages for both players. The command from players to the server should be just <u>one character.</u> This character should be an integer number between 0 and 4. The meaning of each number is as follows:

One Byte String Command	Soccer Command
‘0’	Hold (Stand)
‘1’	North
‘2’	South
‘3’	West
‘4’	East

In each cycle (game step) each player should send a one byte string as its command. If a player does not send a command, the server will choose a command at random for player in that cycle. So it is wise to send a command at each cycle!
The length of each cycle can be determined using the parameter Wait Interval on the form of Soccer Server. The default value has been set to 1000 ms. **<u>But please note that this value will be just 10 ms during the tournament! It means that your player must choose an action and send its command in less than 10 ms after receiving the server message.</u>**

After updating the environment a message containing the state of the environment will be send from server to both players. This message is a 26 bytes string and has the following format:

- Byte #1: Player's row; an integer number in the range [1,6]
- Byte #2: Player's column; an integer number in the range [1,9]
- Byte #3: Opponent's row; an integer number in the range [1,6]
- Byte #4: Opponent's column; an integer number in the range [1,9]
- Byte #5: Ball Owner;
    - 'P' means that the ball owner is the player
    - 'O' means that the ball owner is the opponent
- Byte #6: Current state of the game;
    - 'C' means that the game is running
    - 'L' means that the player scored (Goal for player)
    - 'R' means that the opponent scored (Goal for opponent)
    - 'F' means that the game has been finished
- Bytes 7 to 16 (10 bytes); cycle# (game step) in 10 digits, e.g. 0000000017
- Bytes 17 to 21 (5 bytes); Player's current score in 5 digits, e.g. 00003
- Bytes 22 to 26 (5 bytes); Opponent's current score in 5 digits, e.g. 00002

Using this information the agent (player) is able to model the state of the environment and make a decision based on this state.

Another important issue is that your player will always be supposed to be the left player and its opponent will be the right player, i.e. your player will always be initialized on the left side of the field and its opponent will be on the right side. So it is not necessary for you to handle the side of your player (This issue has been handled by the server using the symmetric property of the field).

For each game a log file will be created. This log file can later be used (by Log Player which will be described later) for debugging your code and evaluating the performance of your agent.
After setting the required fields (port number, wait interval and the log file name) on the server form you can run the server by clicking on the 'Initialize Server' button.
The length of each game is defined in "UnitSServer.pas" by GAME_LENGTH constant. In order to specify the length of the game, change its value and recompile the server code. However please remember that the value of 10000 will be used as the length of the game during the tournament.
That's enough for the server. Let's go to the next part, Soccer Monitor.

Soccer Monitor is a program to monitor a game while playing. You can use it for debugging purposes while the game is running. In order to use Soccer Monitor first click on the Initialize button and then click SendInitInfo button on the Soccer Monitor. Please note that you should run the server before using the monitor. If everything goes well, a message on the server form will be shown indicating that the monitor has been connected properly.

The next part is Log Player, a useful program for watching games that have been played before. Using the Log Player is trivial.

Competitions will be played using the provided game server. Your program will be required to communicate to this server using a TCP socket connection. Otherwise, there are no restrictions on the language. An example to show how a program should connect to the server, send and receive messages, etc. has been written using Borland Delphi 7. This program at each cycle receives a message from the server, updates its world model and sends back a random command to the server. If you prefer to use Delphi to develop your program you can easily put the decision-making structure in "GetCommand" function of its source. In this way you can get rid of Socket Programming issues. But if you choose to use some other programming language, you are responsible for implementing your own TCP interface to the server. **Remember that you should send your player's name (preferably your own name) as the first message (before sending any commands) to the server. The format of this message is an "I:" followed by the name. For example a player with name "Ali" should send "I:Ali" as its first message to the server.** In fact this is an initialization message by which the server can detect the player. For further information please refer to the source code of Client program. The server will start the game after both players connect properly.

If you have any questions please send an email to afkanpour@ce.sharif.edu.

By the way as I wrote all these stuff just in two days there may be (with high probability!) some bugs in the code. I would be grateful if you report any bugs of the programs to me.

Any news regarding this assignment including bug-fixing, answers to common questions, etc. will be announced on the course web page.