ICS 167 - Project 1: Multiplayer Board Game – *Sorry!*

For this project, you are required to code the client-server communications that take place for a multiplayer board game: *Sorry!*.

Sorry! is a turn based multiplayer board game. For this project, we have programmed it as a client/server model. The code for the game logic will be given to you. You are only required to handle the exchange of communications between the client and the server, specifically the **client side**. The purpose being to test your knowledge of socket communication using the transport layer protocol **TCP**. As part of the challenge, you will be required to understand the code and figure out what specific packets need to be sent from the client to the server at each instant in time for the game play. You will not be responsible for the server responses. Those are already coded and included in the executable. **IMPORTANT**: do not change any of the game code, **ONLY** insert your client communication code.

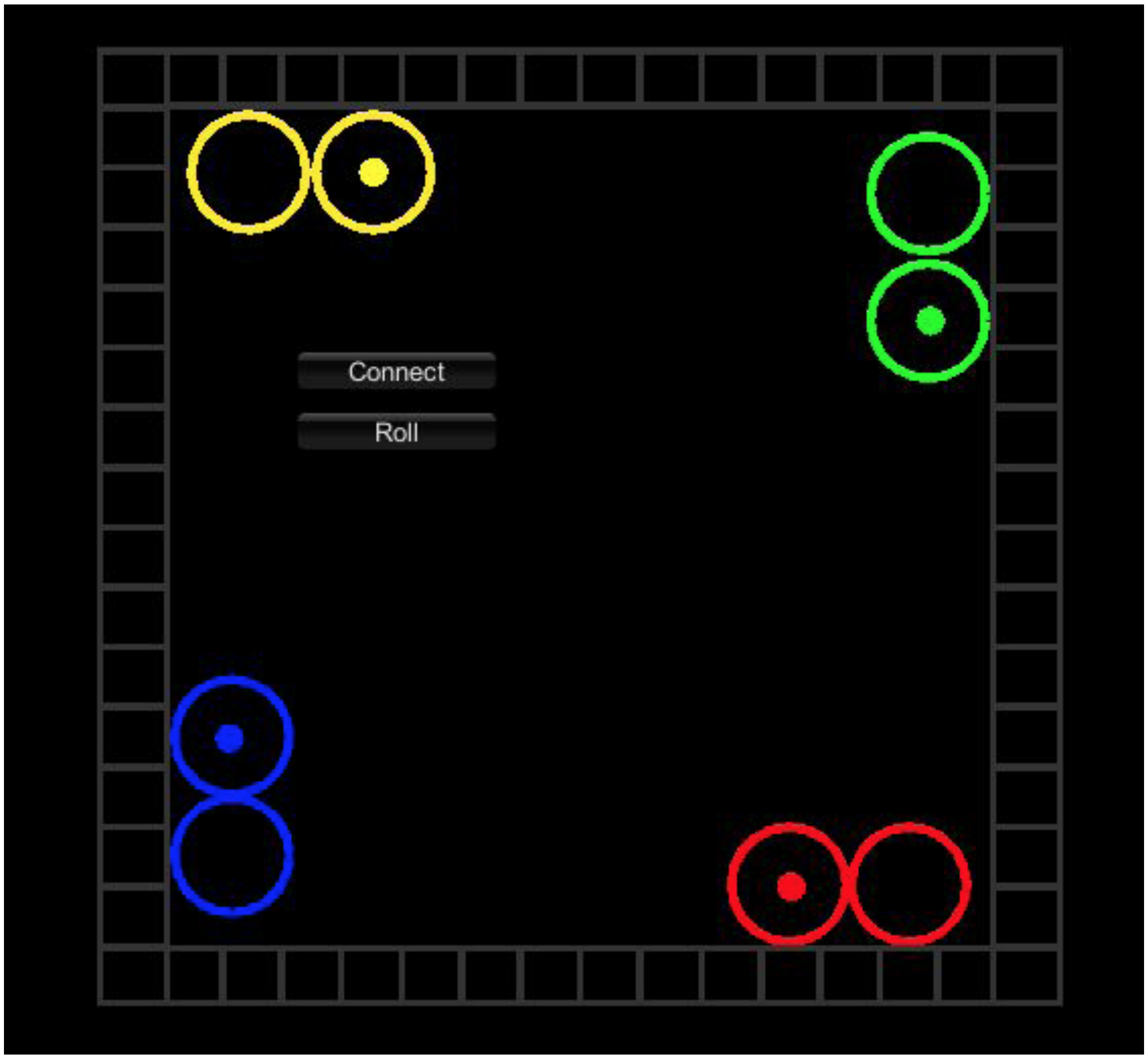
Please pay close attention to the code. When programming clients for networked games, the specific client that you launch is an arbitrary client with respect to the server; therefore, the client that you develop must be able to handle any of the player strategies that are assigned to you by the server.

Sorry is a 2 – 4 player game. Both a 2-player server and a 4-player server will be given to you. The 2-player server is being provided to help you get started and troubleshoot the communications. You must, however, complete the game with 4 players for credit.

Game Objective (see Sorry! for more detailed instructions)
The objective is to be the first player to get all four of the color pawns from the **Start** location to the **Home** location. A stack of cards is used to instruct how the players can move around the board. Each client is assigned a color and 4 pawns. The pawns have to move around the board in a clockwise (forward) direction unless instructed to move backwards. At each turn, a player (client) draws a card from the deck and follows the indicated instructions. Most moves are required; some can be or must be ignored (forfeit turn). The first player to get his/her pawns into the Home location, wins. There are rules on how a pawn may enter the Home location.

In our game, we have only one pawn per player, we use a circle to indicate the Start location and an adjacent circle to indicate the Home location. A roll refers to taking a card from the stack and the board has no safety zones. We do enforce the ending rule, which requires an exact move count into the Home location.

The Board as we have programmed it:



The game has been programmed so that:
Client 1 = Green
Client 2 = Red
Client 3 = Blue

Client 4 = Yellow

The order of the client number is the order in which the clients connect to the server (i.e., the first client to connect to the server is client #1, etc.).

After all the clients connect, the server indicates that the game is ready to be started. A turn signal is then sent to client #1. Then the turn is delegated to client #2, etc.. After a client receives the turn signal, it must press (click) the roll button, which triggers a roll command to the server. The server sends the outcome of the roll command to all the clients, including the client that pressed the button. The client then clicks on the corresponding pawn and moves it based on the roll outcome. All clients see the outcome of the move after the client has sent the new pawn locations to the server and the server has passed that on to the other clients.

Rules of the Game
Pawns may only move out of Start with a roll of 1 or 2. Any forward move is done in the clockwise direction. If you land on a square occupied by another pawn (from another client), you bump that pawn back to *their* Start.

The 11 possible rolls (cards) are:

1. "You can move a pawn from the start or move forward 1 space."
2. "You can move a pawn from the start or move forward 2 spaces."
3. "Move forward 3 spaces."
4. "Move backward 2 spaces."
5. "Move forward 5 spaces."
6. "Move forward 7 spaces."
7. "Move forward 8 spaces."
8. "Move forward 10 spaces or move backward 1 space."
9. "Move forward 12 spaces."
10. "Move forward 11 spaces or switch places with an opponent."
11. "Sorry!" This roll allows you to bump an opponent's pawn back to the *its* Start but you stay where you are.

Forfeiting a turn: to indicate that you are forfeiting your turn**,** move the pawn somewhere off the board (anywhere other than a square or a circle). A client can only *willingly* forfeit a turn under two circumstances: not want to bump another pawn or not able to bump another pawn, due to the absence of pawns on the board The voluntary "forfeit" only applies to rolls "10" and "11".

Note that if a move forward is not possible (i.e., a move beyond/past your home), the game will not automatically forfeit because some rolls allow you to go forward or backward. In that case you must also willingly forfeit your turn. The game must send packets to the server after every move until winning.

Incorrect Moves: The game will not allow you to move to an incorrect square. If you have miscalculated your move, the game will keep your pawn in its current location until you have figured out the correct move. E.g., if you want to move forward three squares, but place your pawn accidently in one square further, the game will just refuse to execute your move action until you move it the correct number of squares forward. Please look to the code to see this functionality.

Winning: To **win** you must be the first to end the game. Ending the game means reaching Home, i.e., landing on the circle adjacent to your Start circle. Note however that you must obtain the exact number of moves to land in the circle. If you draw a card that requires a backwards move you have to do as instructed. If you draw a card that forces you to land in a square beyond the Home location, you must forfeit your turn. If you have the correct roll to land yourself into the "Home" circle, then you have to move your pawn to that position.

Server - Client interactions
Each packet that is sent to the server from the client and each packet sent to the client from the server is only **one byte**. The rightmost bit is the least significant bit.
The first 2 bits of a byte indicate a **specific game command**,
The next 6 bits constitute the **data** related to the specific command.

```
Byte: 0 0 | 0 0 0 0 0 0
Init and End command:
    If command = 0 0
        If game has not started
            data = 1 – 4, indicates client number (server to client)
            data = 5, start game (server to client)
        Else
            data = 1- 4, indicates client that has won
Turn command:
    If command = 0 1
```

data = 1 – 4, indicates turn for that client (server to client)
data = 0, indicates done with turn (client to server)

Roll command:

If command = 1 0

data = 1 – 11, roll number (server to client)
data = 0, roll trigger (client to server)

Move command:

If command = 1 1

data = 1 – 60, position (square number) on the board (client to server after client has moved, or server to clients, indicating new positions of all clients after a client has made a move)

Note that a voluntary forfeit is indicated by sending your current location to the server, indicating no move. All pawn positions stay as is. That too needs to be communicated to the server. By clicking on any part of the board other than a square or a circle, you are choosing a voluntary forfeit. That action will be recognized by the code (in conjunction with the roll state) and you will have to generate the move packets that indicate no move on your part and that of the other pawns too.

Packet flow:
After the server detects that all the clients have connected, the server sends an **Init** packet to each client, designating their client number.
The server then sends an **Init** start game command to all the clients.
This is followed by a **Turn** command to all the clients indicating it is client #1's turn.
Client #1 clicks on roll, which triggers a **Roll** packet from that client to the server. The server then sends a **Roll** packet indicating the result of the roll to all the clients.

The player then moves the pawn on the board to the instructed location (a specific square or off the board if forfeiting turn). **Move** packets are sent from the client to the server giving the updated positions of each pawn (whether they changed or not).
Note: if there are only two players, only 2 move commands must be sent, if 4 players are playing, 4 packets need to be sent. The order of the sent packet indicates the client # associated with that move. (i.e. In order from client #1 to client # 2 or #4)
After the server receives the move commands from the client and receives the end of **Turn** command, it sends the received **Move** commands to all the

other clients telling the clients to update their pawn positions in order from client #1 to the max number of clients (either 2 or 4).
It then sends a **Turn** command to the next client.
This process is repeated until a client has won.
When a client wins, the only difference will be that instead of sending an end of **Turn** command, it will send an **Init** command with its client number. The server, instead of sending the next client the **Turn** command will send to all the clients the **Init** command with the winning client number. All clients interpret that **Init** command as the WIN command as it is received whilst they are in *play* game state, so the game state (whether the game has started or not) must be recorded by every client.

Installation:
Each group will be assigned a Windows 7 VirtualBox Environment on the tristram.ics.uci.edu server. One member from each group must email the TA Afshin Mahini (amahini@uci.edu) the names of all the team members. You will then receive an IP address to enable a remote desktop into the server. You will also be given a username and a password to login to your server. When you first login to the windows 7 environment, please change the assigned username and password to one of your own choice for the group.

The project package that is on the class website (zip format), includes the 2-player server, ServerSorry2p.exe, the 2-player client project (in Unity) and the Pthread folder. You will need to install/unzip this package on your server.

Then you must install the VC++ Redistributable 2010 package from the Microsoft website on your server.

Next, you must copy the pthread dlls found in the pthreads-w32-2-9-1-release/Pre-built.2/dll/x86 and paste them into your Windows/System32 folder on the server.

Then open up the command prompt and navigate to the location of your ServerSorry executable. The Sorry Server expects to receive a port number as an argument: ServerSorry <port number>.
For example, in the command prompt type :

- ServerSorry2p 4645

You may choose any port number on the server starting from port # 4000.