Assignment 5
Group 1
Part 1 Questions

April 6th, 2018

Bianca Angotti (bangotti 1417422)
Andrew McKernan (amckerna 1432175)
Arjun Kalburgi (kalburgi 1388774)

How do I handle starting and serving two different games?

Once the server is running, players can connect over the server to play in two-player game modes. When a client is connected to the server, there will be a list of available “rooms” that can be played in. Once two players join a room, the game will begin for those players. Two more players can join another room and play there at the same time. The server computer will communicate between the two and host all the game related information.

How do I start new servers?

Run server driver file on host computer, passing it the number of users it should expect, along with the hostname and port. This server would initiate a new database, so that once games are started or completed, they will be saved in this database for future review (game stats).

How can a client connect to a game?

A client can connect to an existing server by running the client driver file, along with the hostname and port of the server it is connecting to. At this point in time, the GUI will display all of the “rooms” that are on the server. Users can select to either join an available room (max of two users per room) or play a game against AI (which occurs locally). If the user tries to join a room that is full, they will be met with a message that their chosen room is occupied, and given the option to join another one.

What happens when only one client connects, what happens when three or more try to connect?

When only one client connects, the client should be made aware that there is currently no other clients present on the server, and the option given to them to either wait for another client to connect or to exit the server. If three or more try to connect, they will be able to see the list of available rooms. If say, three or more try to connect to the same room, any clients past the second connected client should be barred from connecting, and be notified that the selected server room is full and that they should attempt to connect to another one.

What synchronization challenges exist in your system?
There are several synchronization challenges in this system. For one, each client must ensure that it's local version of the game board is the same as the one on the server, and when a move is made, both clients are updated as to the new state of the game. As well, both clients must be notified when the game ends, to ensure that one client does not attempt to continue playing when there is no one left to play with.

How do I handle the exchange of turns?
When a turn ends, the client on one machine (Client1) should send their turn data to the database. The server should see and note this change, and notify the other client (Client2) that their turn has begun after updating their board to reflect the other client's action. At this point, Client1 should begin waiting for the server to update them as to Client2's action. This cycle will continue until the game ends, either due to a victory or draw, due to a connection failure, or because one Client has decided to end the game and save it for later.

What information does the system need to present to a client, and when can a client ask for it?
The system needs to present the current state of the game and whether or not it is the client's turn. The client can ask for the game state to be updated if it is the client's turn and the client makes a move on the game board.

What are appropriate storage mechanisms for the new functionality? (Think CMPUT 291!)
For the new functionality, it is best to use a database to store and manage the data through SQL, more specifically MySQL. It is important to be able to add, modify, and query data, such that we are able to display user stats, and continue saved games. Information to save in the database would include information such as usernames of players and who won.

What synchronization challenges exist in the storage component?
In the storage component, it is important to ensure that both clients have access to the most up to date version of the game that is present on the server. If one client was looking at a board that was actually not up to date and playing based on that, it could cause all sorts of unexpected behaviour, from losing a move to causing an error. As well, when a game ends, any saved version of the current game must be removed from the database. A client should not be able to 'resume' a completed game, or even see that such a game exists on the database. Thus, a challenge here will be ensuring that a client has an option to resume a game that might be saved, but does not have this option when the game ends and another is started.

What happens if a client crashes?
If a client crashes and they are playing a two player game, the other client should be alerted, so they do not think that their game is hanging. This could happen after a timeout period, to ensure that the client has really crashed, and has not just disconnected momentarily. The game state will be saved to the server each time a turn is taken, so the game will be able to be resumed in the case of an unexpected failure, even if the user did not exit properly.

What happens if a server crashes?
If a server crashes, clients should be alerted of the closure of the game, so they do not think that the other player is unresponsive. The clients should also be notified that their game progress may not have been saved, since the unexpected closure of the server may have prevented the game from saving properly.

What error checking, exception handling is required especially between machines?
Between machines, it is necessary that both machines are checking on a regular basis that they are still connected to the server, to ensure that the server has not been prematurely terminated. Error checking code is needed here in order to ensure that the game does not crash or perform some other unexpected behaviour on the unforeseen termination of a connection. As well, each client must be prepared to receive incorrect information about the state of the board from the server. If there is a synchronization error, and one player ends up playing a move on an incorrect board, the client must be prepared to handle it by having error handling code that signifies such an error has occurred. At this point, the game should be restarted.

Do I require a command-line interface (YES!) for debugging purposes????? How do I test across machines? And debug a distributed program?
Command line interfaces are integral for testing and debugging purposes, as it can provide much more detail on the inner workings of the program, printing key data and information at different points of the game. It is easier to see the stage by stage transition, thereby realizing where an error may have occurred. Testing across machines can become more difficult as there can be general issues with connecting and staying connected. Socket errors should be logged and shown to us for debugging purposes.

What components of the Ruby exception hierarchy are applicable to this problem?
RuntimeError is applicable as we implement them as exception handling routines that need to be called and handled by the controller for game functionality. ArgumentError would be applicable for incorrect hostname or lack of providing a port for the server and client. There are also standard errors (or encoding errors) that could be thrown during the serialization of the game class to be stored in the database.