Project 5

CS 342 - Team 4

Allen Chung - achung26@uic.edu
Jordan Lee - jlee799@uic.edu
Sean Ennis sennis3@uic.edu
Thomas Kubik - tkubik3@uic.edu

Purpose:

The purpose of this project is to play a game with four people each on their own client connecting to a server using sockets and multithreading. The game is specifically a fighting game where each players stats is dependent upon the items they are able to pick. The goal is to create this fighting game where four people connect to the server and enter a username and then they roll a die to see who will pick items first and then rotate picking items such as sword, helmet, chest armor and a usable item. After every item has been picked rounds of fighting will commence where each player can either attack a player, defend themselves or use their usable item. They will take damage based on the strength of the attacker and their own defense stat based on their armor they chose. The round continue until either everyone is dead or one person survived.

High Level Entities:

There are two main high level entities. These are the list of client threads in the ConnThread class and the object StateOfGame which has four Player objects which holds all the info on all the players.

The Array list of client threads in ConnThread is used to keep track of the different clients connected to the server. It only allows four people to be connected.

The StateOfGame is also very important because as the game progresses there is a lot of information that needs to be kept track of and updated. When clients connect the usernames of the clients are stored in this StateOfGame as well as the rolls each player rolls and the items they choose. This StateOfGame object is also used to calculate the stats of the players based on the items they chose and then is updated to have the player's attack and defense stats. As the game progresses and players make moves those moves are temporarily saved in StateOfGame and the health of the players is updated as they take damage. StateOfGame is also important to the client because the GUI elements are updated based on changes to the StateOfGame object.

Low Level Design of Entities:

There are two main entities. These are the ArrayList of client threads in the ConnThread class and the object StateOfGame which has four Player objects which holds all the info on all the players.

The Array list of client threads in ConnThread is used to keep track of the players ID's, socket, ObjectOutputStream and ObjectInputStream. This ArrayList allows the clients to be sent Strings from the server as well as allowing the server to receive Strings from the clients. The object is an ArrayList so we can know how many clients are connected by using the size method for an ArrayList and so we can iterate through it to send a String to either the one client that has the matching id of who we want to send to or iterate through while sending the String to everyone in the list. It acts like an observer design pattern. A risk would be having to iterate through the list to send it one person while the list might be very long and this would slow the server down.

The StateOfGame is used to keep track of all the four players information. This is important to allow processing of moves to be made such as when a player is attacked we need to know the strength of the attacker and the defense stat of the player being attacked to calculate how much damage is dealt and then update the health. All of these things must be kept track of and this is done easily through the StateOfGame object which holds four player objects which contains the information for each player. Some risks is it doesnt allow for more than one game to be played at a time.

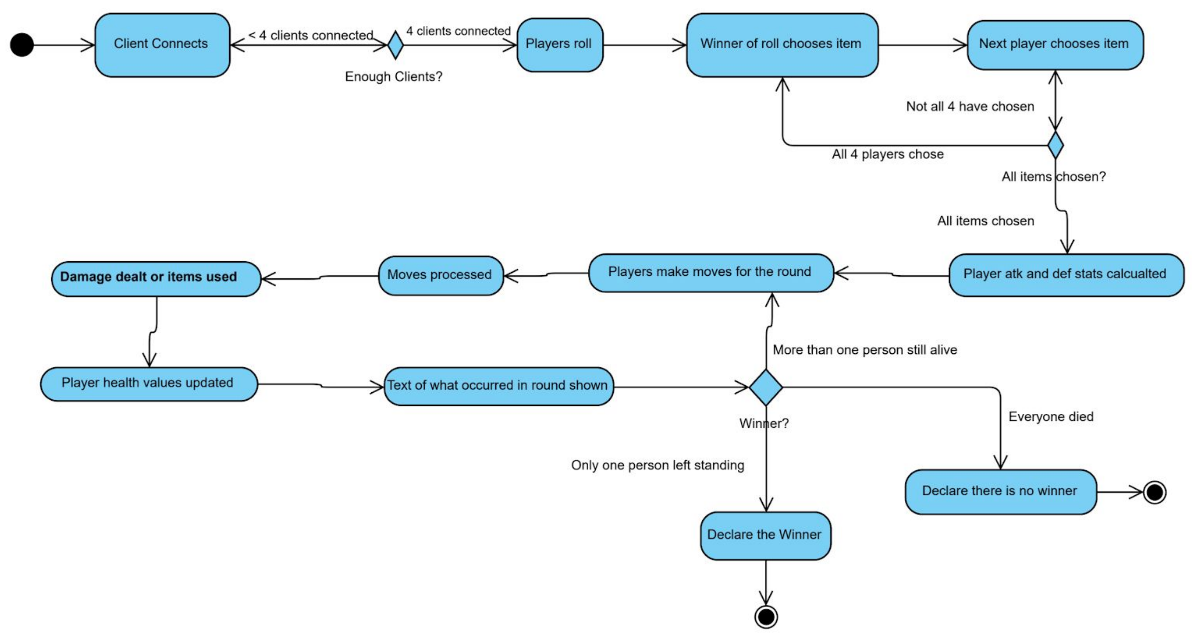
Benefits, Assumptions, Risks/Issues:

One of the benefits for our design is that it allow for multiple players to easily connect to the same game through sockets and threads and can play the game together. The javaFX allows the players to somewhat see the actions of the other players during the game. The javaFX GUI we implemented was very simple and very easy for any user to understand.

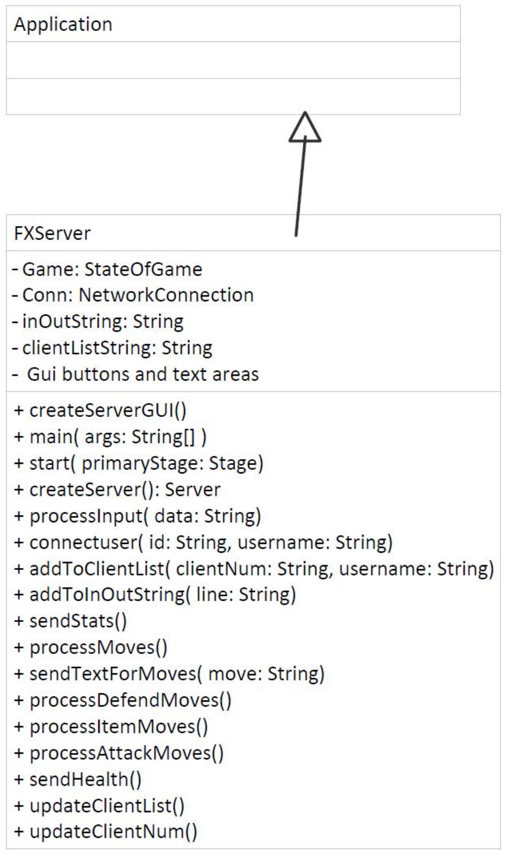
We assume that there will be four players because the game can only be played with four players. We also assume that nobody quits in the middle of a round because the game would still wait for that player to make their pick or move anyway. Quitting only works at a specific time because the server needs to take that player out of the game. There is the assumption that every player types in a username because the game and client GUI doesn't force you to type one in before proceeding.

There's a risk that the game flow might be disrupted if someone suddenly quits the game and the server doesn't get correctly notified. Another possible issue was if more than 4 players tried to join the game, but this was resolved. There is also the risk that a player will put in the wrong port and ip address and is unable to connect to the server. If a player doesn't type in a username, the username for that player remains null which will cause a problem because other players won't be able to attack that player, and this also has the potential of causing a segmentation fault in the server-client communication. A major unresolved issue is that once a player's health reaches 0, the game is unable to distinguish who's still alive, and causes an error that ends the game.

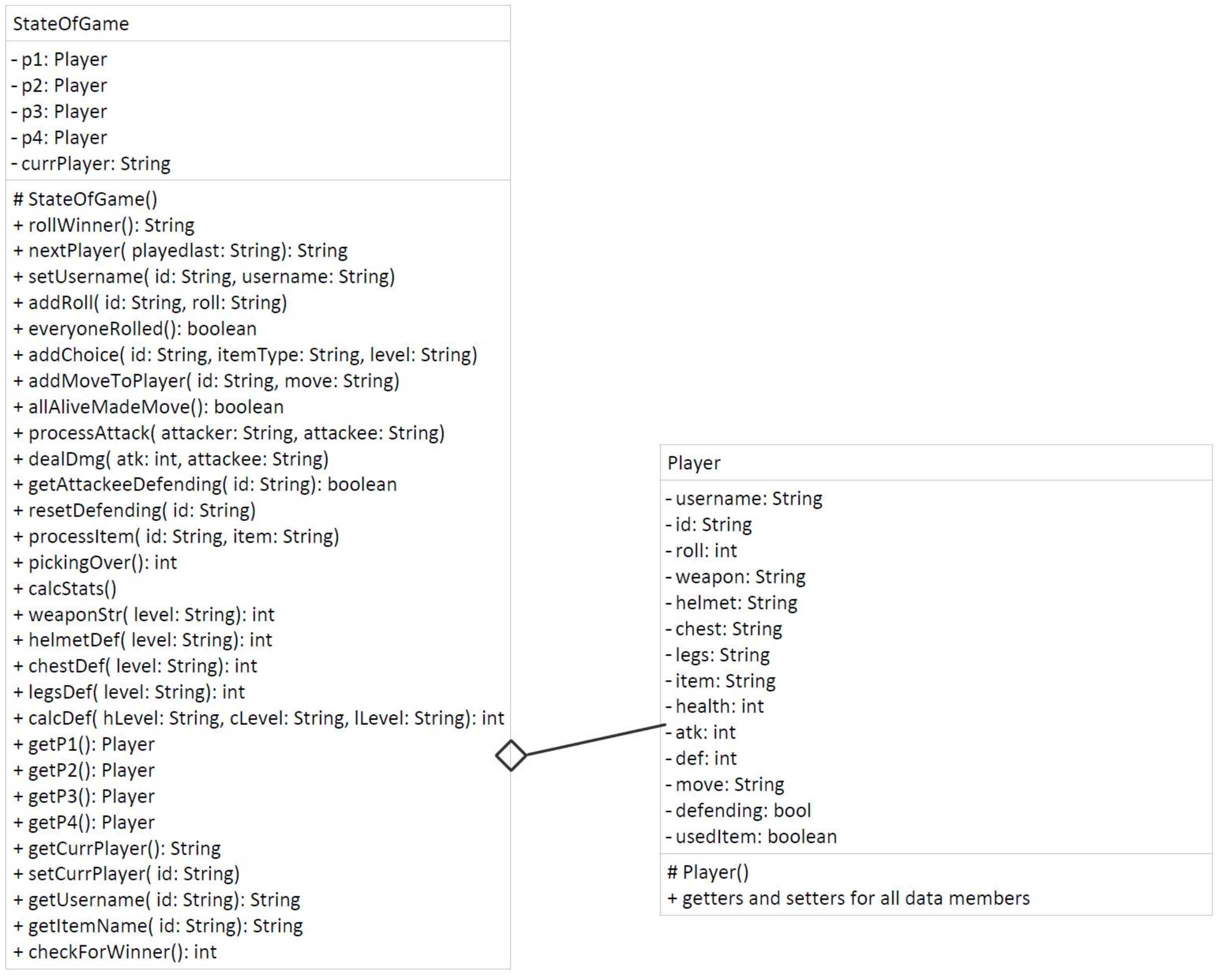
Activity Diagram


Client Connects
< 4 clients connected
4 clients connected
Enough Clients?
Players roll
Winner of roll chooses item
Next player chooses item
Not all 4 have chosen
All 4 players chose
All items chosen?
All items chosen
Player atk and def stats calcualted
Players make moves for the round
Moves processed
Damage dealt or items used
Player health values updated
Text of what occurred in round shown
More than one person still alive
Winner?
Everyone died
Only one person left standing
Declare there is no winner
Declare the Winner

Server UML



Application
FXServer
- Game: StateOfGame
- Conn: NetworkConnection
- inOutString: String
- clientListString: String
- Gui buttons and text areas
+ createServerGUI()
+ main(args: String[])
+ start(primaryStage: Stage)
+ createServer(): Server
+ processInput(data: String)
+ connectuser(id: String, username: String)
+ addToClientList(clientNum: String, username: String)
+ addToInOutString(line: String)
+ sendStats()
+ processMoves()
+ sendTextForMoves(move: String)
+ processDefendMoves()
+ processItemMoves()
+ processAttackMoves()
+ sendHealth()
+ updateClientList()
+ updateClientNum()


StateOfGame

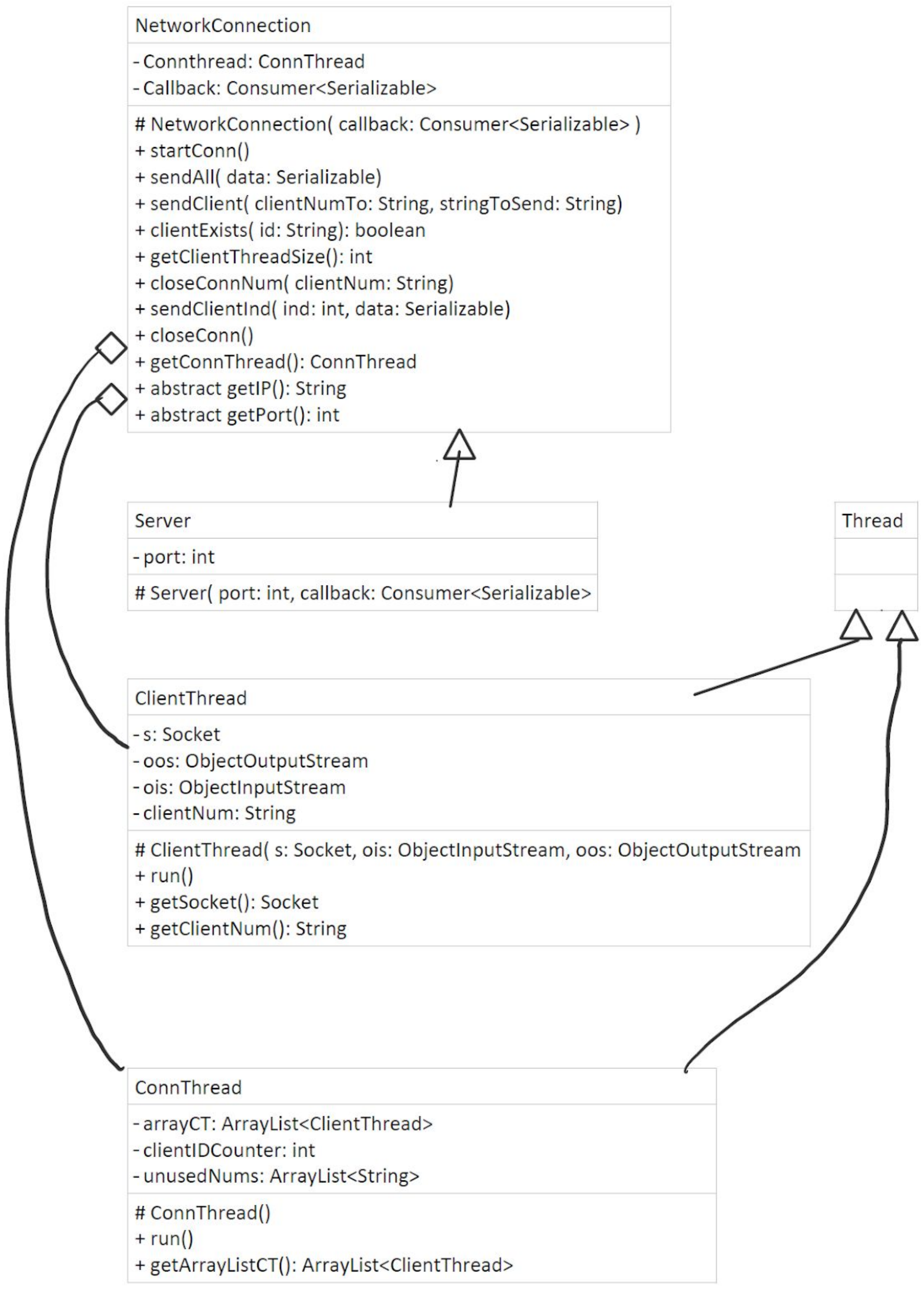
- p1: Player
- p2: Player
- p3: Player
- p4: Player
- currPlayer: String

StateOfGame()
+ rollWinner(): String
+ nextPlayer(playedlast: String): String
+ setUsername(id: String, username: String)
+ addRoll(id: String, roll: String)
+ everyoneRolled(): boolean
+ addChoice(id: String, itemType: String, level: String)
+ addMoveToPlayer(id: String, move: String)
+ allAliveMadeMove(): boolean
+ processAttack(attacker: String, attackee: String)
+ dealDmg(atk: int, attackee: String)
+ getAttackeeDefending(id: String): boolean
+ resetDefending(id: String)
+ processItem(id: String, item: String)
+ pickingOver(): int
+ calcStats()
+ weaponStr(level: String): int
+ helmetDef(level: String): int
+ chestDef(level: String): int
+ legsDef(level: String): int
+ calcDef(hLevel: String, cLevel: String, lLevel: String): int
+ getP1(): Player
+ getP2(): Player
+ getP3(): Player
+ getP4(): Player
+ getCurrPlayer(): String
+ setCurrPlayer(id: String)
+ getUsername(id: String): String
+ getItemName(id: String): String
+ checkForWinner(): int

Player

- username: String
- id: String
- roll: int
- weapon: String
- helmet: String
- chest: String
- legs: String
- item: String
- health: int
- atk: int
- def: int
- move: String
- defending: bool
- usedItem: boolean

Player()
+ getters and setters for all data members


NetworkConnection
- Connthread: ConnThread
- Callback: Consumer<Serializable>
# NetworkConnection(callback: Consumer<Serializable>)
+ startConn()
+ sendAll(data: Serializable)
+ sendClient(clientNumTo: String, stringToSend: String)
+ clientExists(id: String): boolean
+ getClientThreadSize(): int
+ closeConnNum(clientNum: String)
+ sendClientInd(ind: int, data: Serializable)
+ closeConn()
+ getConnThread(): ConnThread
+ abstract getIP(): String
+ abstract getPort(): int
Server
- port: int
# Server(port: int, callback: Consumer<Serializable>
Thread
ClientThread
- s: Socket
- oos: ObjectOutputStream
- ois: ObjectInputStream
- clientNum: String
# ClientThread(s: Socket, ois: ObjectInputStream, oos: ObjectOutputStream
+ run()
+ getSocket(): Socket
+ getClientNum(): String
ConnThread
- arrayCT: ArrayList<ClientThread>
- clientIDCounter: int
- unusedNums: ArrayList<String>
# ConnThread()
+ run()
+ getArrayListCT(): ArrayList<ClientThread>

Player
- username: String - id: String - roll: int - weapon: String - helmet: String - chest: String - item: String - health: int - atk: int - def: int - move: String - defending: bool - usedItem: boolean
Player() + getters and setters for all data members

NetworkConnection
- Connthread: ConnThread - Callback: Consumer<Serializable>

# NetworkConnection(callback: Consumer<Serializable>) 	
+ startConn()
+ sendAll(data: Serializable)
+ sendClient(clientNumTo: String, stringToSend: String)
+ clientExists(id: String): boolean
+ getClientThreadSize(): int
+ closeConnNum(clientNum: String)
+ sendClientInd(ind: int, data: Serializable)
+ closeConn()
+ getConnThread(): ConnThread
+ abstract getIP(): String
+ abstract getPort(): int

Client UML

FXClient
- ipAdd: String
- portNum: int
- idNumStr: String
- idNumInt: int
- userName: String
- players: ArrayList<Player>
- availableItems: ArrayList<Button>
- didUseItem: boolean
- GUI elements
- setItemSelectButtonActions()
- setPlayButtonActions()
- disableMoveButtons()
- createConnect(primaryStage: Stage)
- createItemSelectScene(primaryStage: Stage)
- createPlayScene(primaryStage: Stage)
+ disableItemButtons()
+ disableMoveButtons()
+ removeFromAvailableItems(type: String, rarity: String)
+ main(args: String[])
+ start(primaryStage: Stage)
+ createClient(): ClientRPS
+ processInput(data: String)
+ setID(id: String)
+ setUsername(id: String, username: String)
+ setRoll(id: String, numRolledStr: String)
+ choose()
+ setPick(id: String, pickType: String, pick: String)
+ setAtkStat(id: String, atkStat: String)
+ setDefStat(id: String, defStat: String)
+ updateHealth(id: String, health: String)
+ startNextRound()
+ winner(winnerID: String)
+ receiveMessage(message: String)
+ sendUsername(username: String)
+ rollDie()
+ pickItem(pickType: String, pick: String)
+ attack(playerToAttack: String)
+ usernameToID(name: String): String
+ defend()
+ useItem(itemUsed: String)
+ quit()

ClientNetworkConnectionRPS
- Connthread: ConnThread - Callback: Consumer<Serializable>

# NetworkConnection(callback: Consumer<Serializable>) 	
+ startConn()
+ send(data: Serializable)
+ closeConn()

ConnThread
- socket: Socket - out: ObjectOutputStream
+ run()

ClientRPS
- ip: String - port: int
ClientRPS(ip:String, port: int, callback: Consumer<Serializable>) + getIP(): String + getPort(): int