Project 4: Word Guessing Game

Group #12: Bart Blazik, Sergiu Buruiana, Jakob Karol, Brian Yu

Server Description

For this project, the server implementation was a simple threaded server, similar to project 3. When the server program begins, the user inputs a port for the server to listen to on the localhost. When a client connects to the server port, a new client thread is initiated in the server program.

Each client thread has five main parts: a socket to which the client connects, an integer that represents the client number, an input, and output connection, and a GameInfo object. A GameInfo object contains information about the state of the game, as well as some functions for the game logic. Upon initialization, the five fields are initialized, and the GameInfo object is immediately sent out to the client program.

Each client thread runs in an infinite loop with three main steps. First, it waits for its respective client to send a GameInfo object. We then run **checkGuess()** in the GameInfo object. This method takes one parameter, which is the letter that the client user guessed, checks whether it's in the word currently being guessed, and updates the game state accordingly. After this, it sends the GameInfo object back out to the same client.

Client Description

The client program begins with a scene where the user can input the IP address and port they wish to connect to. There is also a button to confirm these and begin the game; once this is pressed, the client connects to the chosen address and port. Assuming these connect to the server properly, the game then shifts to a scene where the player can choose which of the three categories they wish to choose from. Once a category is selected, a GameInfo object within the client program randomly generates a word within the category and displays it on the screen as underscores. The game proceeds as the overview describes, with the player guessing one letter of the word each until they either guess six incorrect letters or correctly guess the word.

The client code itself is rather simple; after its initialization, it runs in an infinite loop of waiting to read a GameInfo object, then does **callback.accept()** on the GameInfo so that the UI can work with it in the Platform.runLater.

UI/UX

The UI was made using a template builder, but the components were kept simple enough that someone with the skills learned in class could have built it exactly as we did. Both UIs were created with FXML using the Controller design principle.

Collaboration

In our team of four, each of us chose what to do based on our strengths and weaknesses; the person who was good at JavaFX did the UI, the person who was good at writing wrote the

final report, and so on. We talked and collaborated through a Discord group chat, and kept our code in a Github repository with a separate branch for each member of the team.

Bart Blazik:

- Wrote server code
- Assembled final report

Sergiu Buruiana:

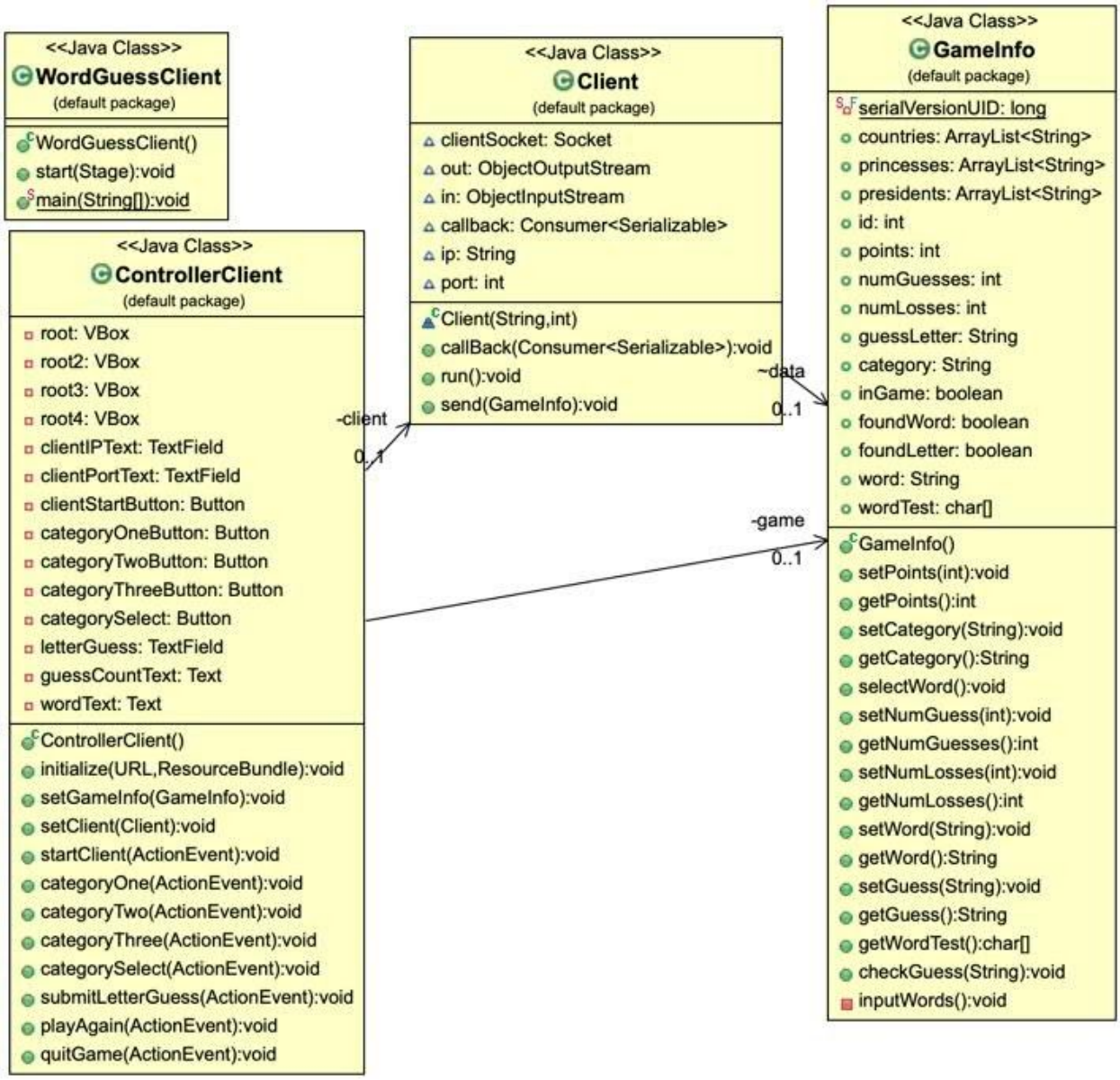
- Wrote client code
- Cleaned up miscellaneous code
- Created wireframes

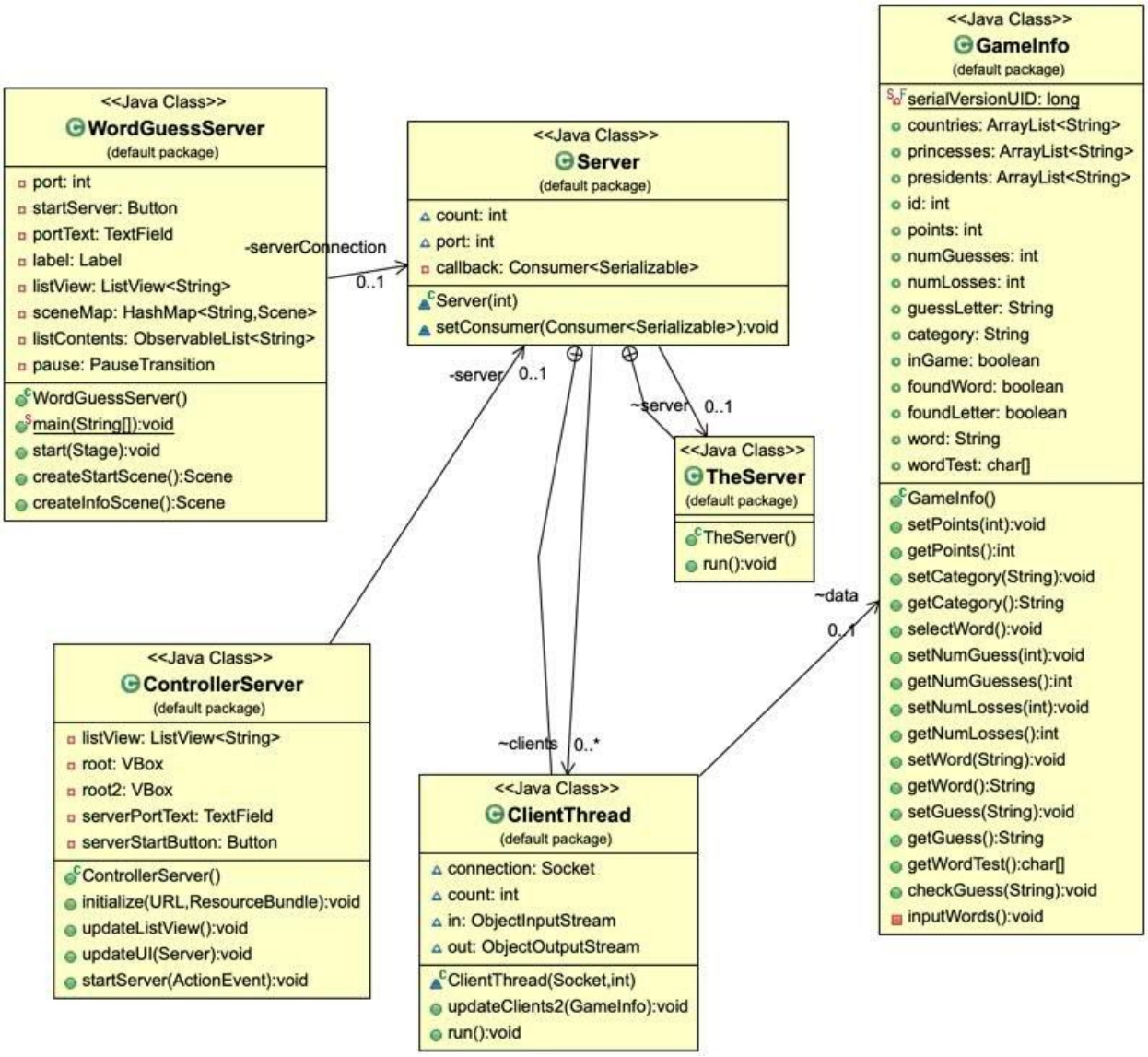
Jakob Karol:

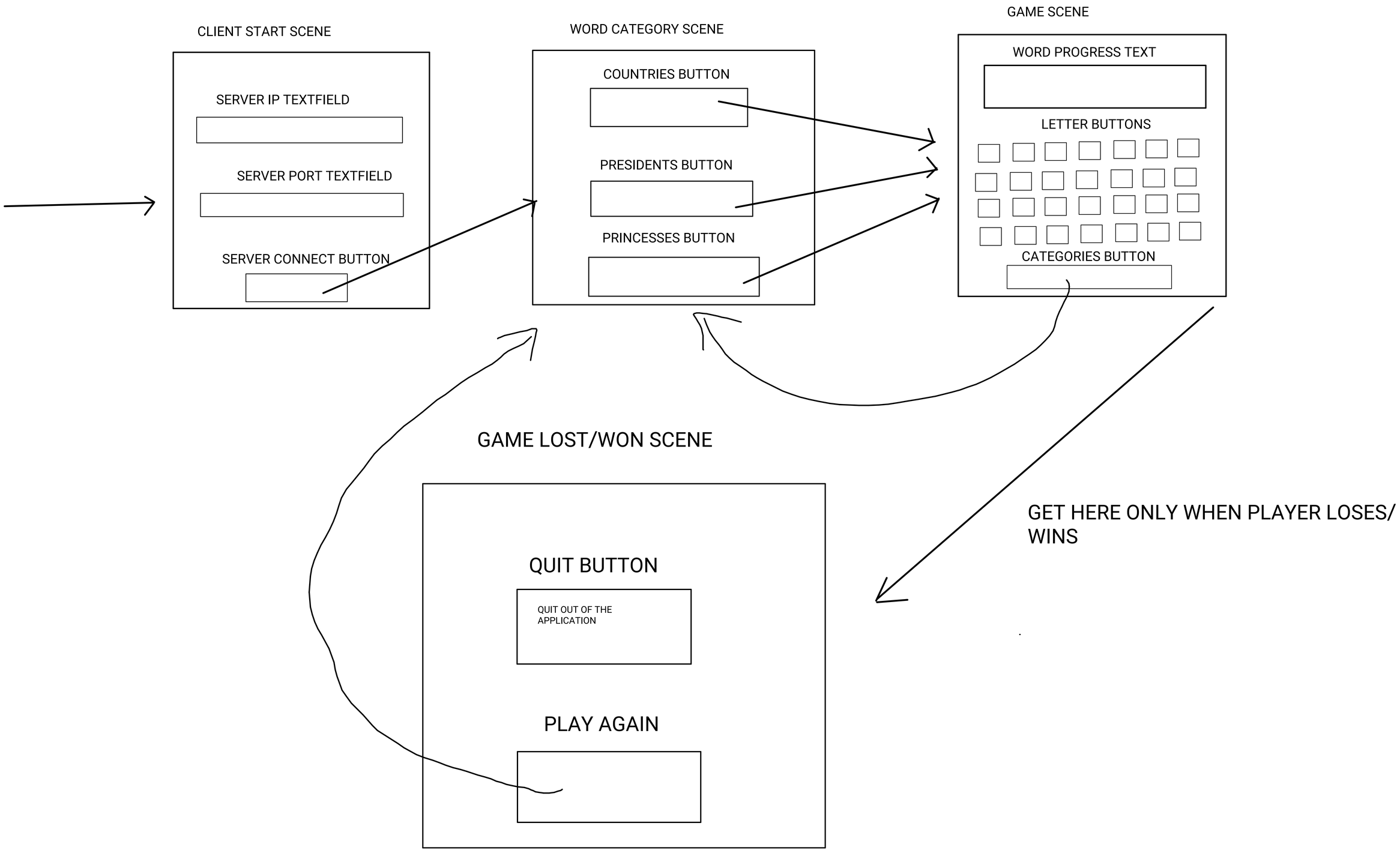
- Created GameInfo object and test cases
- Handled communication between server and client
- Created UML and activity diagrams

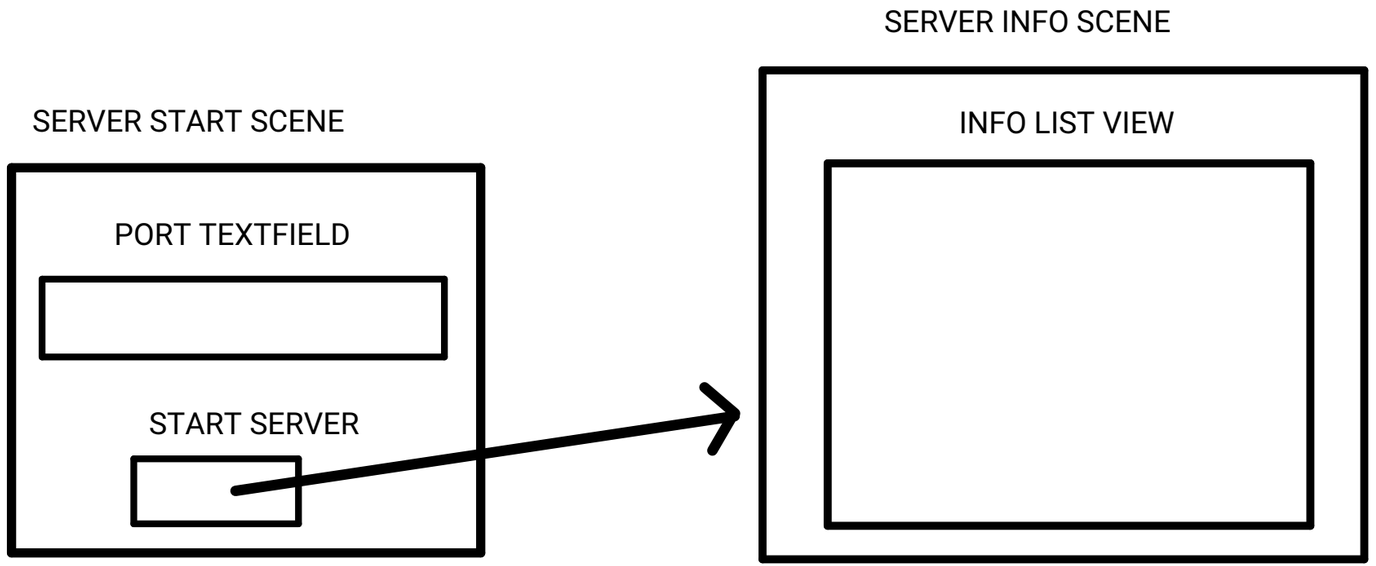
Brian Yu:

- Assembled UI for server and client programs.
    - Created and styled using CSS files
    - Did the core functionality of buttons and scene changes
    - Used FXML and controller classes to do controller design pattern

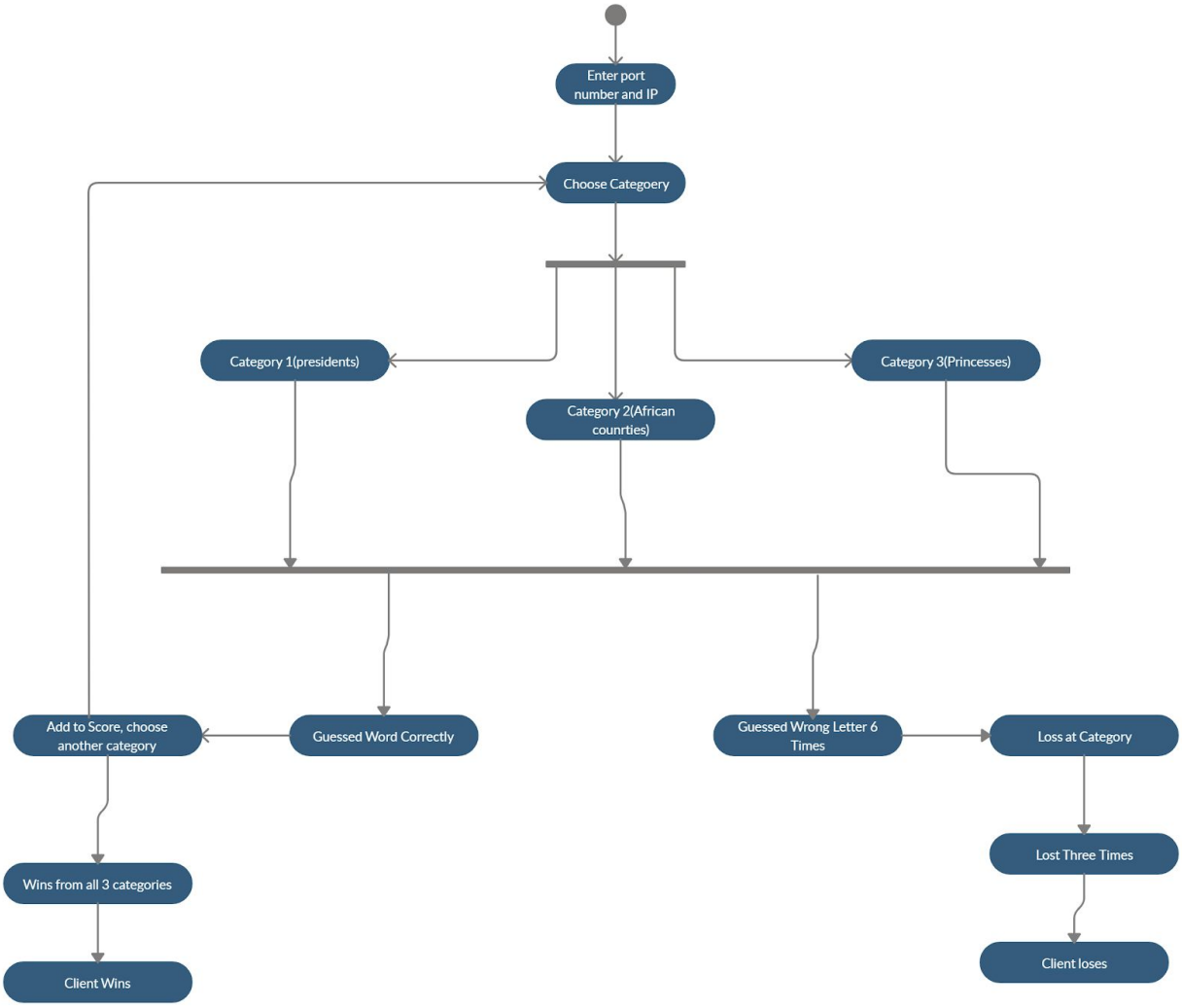

<<Java Class>>
WordGuessClient
(default package)
WordGuessClient()
start(Stage):void
main(String[]):void
<<Java Class>>
Client
(default package)
clientSocket: Socket
out: ObjectOutputStream
in: ObjectInputStream
callback: Consumer<Serializable>
ip: String
port: int
Client(String,int)
callBack(Consumer<Serializable>):void
run():void
send(GameInfo):void
<<Java Class>>
GameInfo
(default package)
serialVersionUID: long
countries: ArrayList<String>
princesses: ArrayList<String>
presidents: ArrayList<String>
id: int
points: int
numGuesses: int
numLosses: int
guessLetter: String
category: String
inGame: boolean
foundWord: boolean
foundLetter: boolean
word: String
wordTest: char[]
GameInfo()
setPoints(int):void
getPoints():int
setCategory(String):void
getCategory():String
selectWord():void
setNumGuess(int):void
getNumGuesses():int
setNumLosses(int):void
getNumLosses():int
setWord(String):void
getWord():String
setGuess(String):void
getGuess():String
getWordTest():char[]
checkGuess(String):void
inputWords():void
<<Java Class>>
ControllerClient
(default package)
root: VBox
root2: VBox
root3: VBox
root4: VBox
clientIPText: TextField
clientPortText: TextField
clientStartButton: Button
categoryOneButton: Button
categoryTwoButton: Button
categoryThreeButton: Button
categorySelect: Button
letterGuess: TextField
guessCountText: Text
wordText: Text
ControllerClient()
initialize(URL,ResourceBundle):void
setGameInfo(GameInfo):void
setClient(Client):void
startClient(ActionEvent):void
categoryOne(ActionEvent):void
categoryTwo(ActionEvent):void
categoryThree(ActionEvent):void
categorySelect(ActionEvent):void
submitLetterGuess(ActionEvent):void
playAgain(ActionEvent):void
quitGame(ActionEvent):void
-client
0..1
~data
0..1
-game
0..1


<<Java Class>>
WordGuessServer
(default package)
port: int
startServer: Button
portText: TextField
label: Label
listView: ListView<String>
sceneMap: HashMap<String,Scene>
listContents: ObservableList<String>
pause: PauseTransition
WordGuessServer()
main(String[]):void
start(Stage):void
createStartScene():Scene
createInfoScene():Scene
-serverConnection
0..1
<<Java Class>>
Server
(default package)
count: int
port: int
callback: Consumer<Serializable>
Server(int)
setConsumer(Consumer<Serializable>):void
-server
0..1
~server
0..1
<<Java Class>>
TheServer
(default package)
TheServer()
run():void
<<Java Class>>
GameInfo
(default package)
serialVersionUID: long
countries: ArrayList<String>
princesses: ArrayList<String>
presidents: ArrayList<String>
id: int
points: int
numGuesses: int
numLosses: int
guessLetter: String
category: String
inGame: boolean
foundWord: boolean
foundLetter: boolean
word: String
wordTest: char[]
GameInfo()
setPoints(int):void
getPoints():int
setCategory(String):void
getCategory():String
selectWord():void
setNumGuess(int):void
getNumGuesses():int
setNumLosses(int):void
getNumLosses():int
setWord(String):void
getWord():String
setGuess(String):void
getGuess():String
getWordTest():char[]
checkGuess(String):void
inputWords():void
~data
0..1
<<Java Class>>
ControllerServer
(default package)
listView: ListView<String>
root: VBox
root2: VBox
serverPortText: TextField
serverStartButton: Button
ControllerServer()
initialize(URL,ResourceBundle):void
updateListView():void
updateUI(Server):void
startServer(ActionEvent):void
~clients
0..*
<<Java Class>>
ClientThread
(default package)
connection: Socket
count: int
in: ObjectInputStream
out: ObjectOutputStream
ClientThread(Socket,int)
updateClients2(GameInfo):void
run():void


CLIENT START SCENE
SERVER IP TEXTFIELD
SERVER PORT TEXTFIELD
SERVER CONNECT BUTTON
WORD CATEGORY SCENE
COUNTRIES BUTTON
PRESIDENTS BUTTON
PRINCESSES BUTTON
GAME SCENE
WORD PROGRESS TEXT
LETTER BUTTONS
CATEGORIES BUTTON
GAME LOST/WON SCENE
QUIT BUTTON
QUIT OUT OF THE APPLICATION
PLAY AGAIN
GET HERE ONLY WHEN PLAYER LOSES/ WINS


SERVER INFO SCENE
SERVER START SCENE
INFO LIST VIEW
PORT TEXTFIELD
START SERVER

`


Enter port number and IP
Choose Categoery
Category 1(presidents)
Category 2(African counrties)
Category 3(Princesses)
Add to Score, choose another category
Guessed Word Correctly
Guessed Wrong Letter 6 Times
Loss at Category
Wins from all 3 categories
Lost Three Times
Client Wins
Client loses