

## 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 and integrated the client code with the client UI
- 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