The project is a turn-based educational game that pits java bots (uploaded by the users) against each other or single-bot challenges. Our part of the project deals with bot verification, testing mode, graphical display and playback mode.

The project has three major game modes: testing mode (in which the user can test their bots against other bots, user input, or single-player challenges), challenge mode (in which the server will run a game using only an uploaded bot or bots), and playback mode (in which the client will replay a finished testing mode or challenge mode game using data retrieved from the server).

GENERAL:

1. The Game Display Module (part of the client) must be able to accept game states and commands from the server, and use them to draw game states and animations with sprites, including the map and characters. Additionally, the module must be able to display errors.

2. The server must be able to accept uploaded java programs (referred to as *bots*) from users, verify that they are not potentially malicious, and save them to its bot database.

3. The server must be able to save all game states, commands, and errors to its game state database.

4. At the beginning of a replay or testing mode game, the server must be able to pass the initial game state to the client.

TESTING MODE:

5. At the beginning of any character’s turn, the system must be able to handle input from them as appropriate (based on their given input type).

a. If the character is controlled by user input, the client must be able to prompt the user for their input and then send it to the server, then end the character’s turn and wait for the server’s response.

b. If the character is controlled by client-side bot, the client must be able to prompt the user to modify their bot, if desired. The client must also allow for the user to run the bot once, then send this output to the server, then end the character’s turn and wait for the server’s response.

i. If the client-side bot takes too long to produce output, the client must be able to produce an error to notify the user that the bot takes too long to run.

c. If the character is controlled by server-side bot, the client must be able to prompt the user to proceed to the next turn. If the user accepts, the client must send a message to the server. The server must then run the bot and accept its input.

6. When prompting the user for input at the beginning of a character’s turn, the client must also allow for the user to undo or redo previous moves.

7. After accepting input from the client or from a server-side bot, the server must be able to pass it to the Game Evaluation Engine, which will return either a new game state or an error.

a. If the server received an error, it must pass that error to the client, which must display it. If the error was from player input or a client-side bot, the client must then prompt the user to retry their input. Otherwise, the server must proceed to the next turn.

b. If the server received a new game state, it must pass that game state to the client, which must display it. The system must then begin the next character’s turn, or terminate the game if needed.

8. After a testing mode game has run to completion, the client must prompt the user to view a replay of it in playback mode.

CHALLENGE MODE:

9. The server must be able to automatically run a challenge mode game to completion using a given map and uploaded bots.

10. The server must be able to pass a command produced by running an uploaded bot to the game evaluation module, which will return a new game state.

a. If a bot’s input produces an error, the server must treat its command as an “idle” command and pass this command to the game evaluation engine.

11. Once the game ends, the server must be able to send the client(s) the game data (states, commands, and errors). The client must then enter replay mode and allow the user to watch the game.

PLAYBACK MODE:

12. The server must be able to send the client a sequence of game states and commands for a requested completed game (either testing or challenge mode), as well as any errors produced.

13. The client must be able to accept this data and use the Game Display Module to replay the game.

14. Playback mode must also allow the user to play or rewind the game one turn at a time.

ADDITIONAL BOT VALIDATION:

14. If a testing mode or challenge mode game exceeds a large number of turns (ie. over 500), the server must be able to automatically terminate it (as a precaution against endless stalemates).

15. If a server-side bot takes too long to produce output (ie. over 500 milliseconds), the server must be able to terminate it and produce an error.

**Glossary**

**Bot:** A java program that takes a game state and returns an action. Users can upload bots to the server, which will ensure they are safe and then store them in its bot database. Challenge mode is run by the server using bots. Testing mode can also be used to test bots.

**Client-Side Bot:** A bot hosted on the client, used for testing mode.

**Public Bot:** A default bot provided by the server.

**Player Bot:** A bot created by a player.

**Server-Side Bot:** A bot hosted on the server, used for challenge or testing mode.

**Bot Database:** A server database of all uploaded server-side bots that the server has determined are safe.

**Challenge:** An online game mode run by the server that is played by one or more server-hosted bots. Game info (game states, commands, and errors) from challenge mode will be stored in the server’s game state database. Upon completion of a challenge mode game, users can view it in playback mode.

**Character:** The in-game representation of a player or bot. A character is controlled by commands.

**Client:** The web-application used by the user. The client is responsible for running the game display module and sending user input to the server in the form of commands and uploaded bots.

**Command:** The move a character makes on their turn, which changes the game state. Commands are provided by a player or bot.

**Fast-Forward:** In playback mode, speeds up the game animations.

**Game:** An instance of testing or challenge mode.

**Game State Database**: A server database that stores all game states, animations, and errors for all games.

**Game Display Module:** A part of the client written in Phaser which takes game states, animations, and errors, and displays them. The game display module also runs playback mode, allowing the player to view replays of finished games.

**Map:** An arena in which testing or challenge mode takes place. A map contains walls, empty space, character spawn points, and other objects.

**Mode:** Testing, challenge, or playback mode.

**Pause:** In playback mode, temporarily stops playback of the game being watched.

**Play:** In playback mode, starts or resumes playback of the game being watched.

**Playback Mode:** A game mode which allows the user to view a replay of a testing mode or challenge mode game that has already been played. The game info (game states, animations, and errors) will be passed to the client by the server when the user requests a particular game.

**Rewind:** In playback mode, replays the game being watched in reverse.

**Server:** Software that is hosted on a remote machine. The server is responsible for verifying the safety of bots, storing bots and game info in its game state database, receiving input from the clients and sending output back to them, and passing info to the game evaluation engine.

**System:** All components of the project, including the client and server-side modules.

**Testing Arena:** A map designed for testing mode.

**Testing Mode:** A game mode designed to allow the user to test their bots. Testing mode will allow the client to specify a type of input for each character (user input, client-side bot input, or server-side bot input). Testing mode will not run automatically like challenge mode but instead after each turn will prompt the user to provide input or move to the next. Testing mode will also allow the user to undo or redo moves.

**Turn:** A single game state update, in which the server receives input and returns a new game state as well as any relevant commands and errors.

**User:** Human who plays the game using the client.