COSC 3P91 – Assignment 4 – 7242530

NICHOLAS PARISE, Brock University, Canada

1 DESIGN OVERVIEW

The general overview is that with each loop of the engine, it calls the thread pool to run methods in change of all aspects of the vehicles in parallel. Then at predetermined intervals the engine unlocks the ability for the player objects to communicate with the user. These player objects are being controlled in their own thread that deals with communication and authenticating the user in the database.

1.1 Threading

There are several instances of threading in this program. a thread is used to handle the server Socket, and each client that connects, and a thread pool is used to move the vehicles.

Every loop of the update method in the game engine calls a method called "RunThreadPool" that creates a thread pool, adds all the runnable tasks to the pool and then blocks the game engine thread until all the tasks are done. To make this possible all methods that were previously used to update the vehicle position were changed into Runnables.

1.2 Server

The server package handles the communication with the game engine to the players. It consists of two classes and one enum. The execution starts with engine who spawns a thread of "ThreadedServer". This class is in an infinite loop waiting for new connections, and when a new connection is found it adds a new user to the players array, spanws a new thread of "ConnectionThread" and passes this player to the constructor. ConnectionThread is where all the actual communication takes place. The class starts in "Authenticate" mode and prompts the user for the username and password, if the supplied credentials are found in the database then the player is authenticated and the mode is switched. The server then (depending if it is unlocked) data is sent to the user.

1.3 Client

The client is a very simple package, it consists of an enum, a helper class and the comms class which houses the brains. When the comms class is loaded, it takes in the message from the server and using the helper class gets the state and the data in the message.

the class uses a switch case for the state and depending on which state it is does the corrisponding action:

```
Push: print the message to the user
Reply: print the message and take in input
Authenticate: it sends the credentials to the server
Kill: print out message and close connection
```

Author's address: Nicholas Parise, Brock University, 1812 Sir Isaac Brock Way, St. Catharines, ON, L2S 3A1, Canada.

Update date: 6 April 2024

1.4 Communication

The server communicates to the client through text in a formatted string as follows. On the client side, a simple helper class is used to decode the message

```
[Data]\t[State]
```

The client simply replies back with a string that follows this format, as the server already knows what state it is in.

```
[Data]
```

1.5 Database

The database is SQLite based. There is a class called "Database" that handles the communication with the database. The jar is from this GitHub: https://github.com/xerial/sqlite-jdbc and the code is based off the docs aswell.

The table is simple just has a few column

```
"CREATE TABLE IF NOT EXISTS users (\n"
                + " id integer PRIMARY KEY,\n"
                + " username text NOT NULL,\n"
                + " password text NOT NULL\n"
                + ");";
```

1.6 Using the software

There are two seperate programs

(1) Server
(2) Client

Server must be started first, then clients can be started. In the directory there are several shell scripts to start each but they are also listed below:

To run the Server

```
Windwows:
java -classpath ".;sqlite-jdbc-3.45.2.0.jar;slf4j-api-1.7.36.jar;build"
    Main.Main

Linux:
java -classpath ".:sqlite-jdbc-3.45.2.0.jar:slf4j-api-1.7.36.jar:build"
    Main.Main
```

To run the Client

```
java -cp build Client.Main
```

After opening the client after the server you will be met with a success message and then the first prompt from the server. In this example our vehicle has reached an intersection and we went forward

```
Connection accepted!
Authenticated
You've reached an intersection, which direction would you like to go?
You can go: FORWARD(F) RIGHT(R) Or WAIT(W)
F
Changed direction
```

playing the game is easy simply supply the characters the system requests.