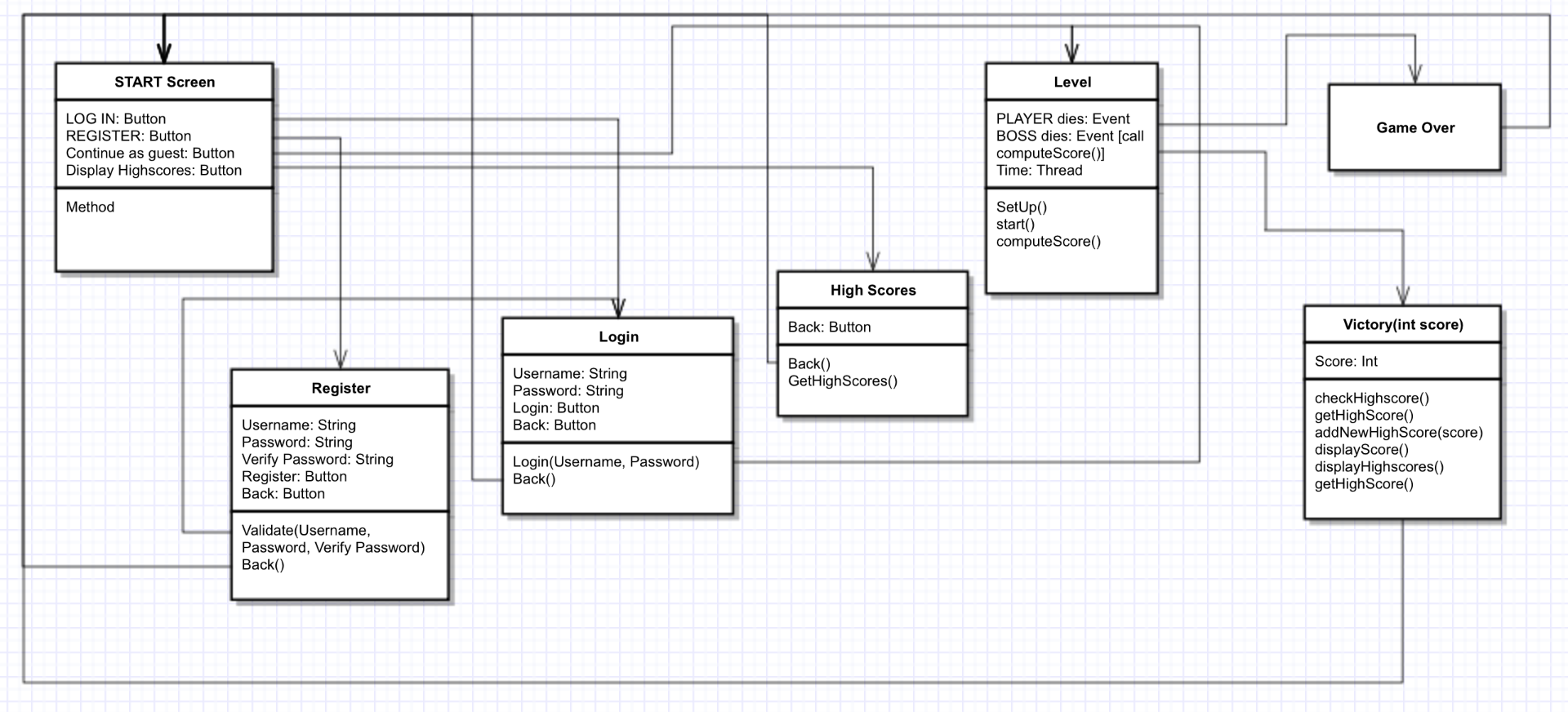
***Coliseum - Detailed Design Document***

Jun Yen Leung - junyenle@usc.edu

Ethan Barker - esbarker@usc.edu

Jake Leventhal - jglevent@usc.edu

Uri Rahimi - urahimi@usc.edu

**Section 1: Broad Program Structure (Class Structure)**

**1a: Flow and Overview**

* **Program Flow From Application Run:**
  + **START screen →** 
    - Has Login, Register, Continue as guest, and Quit game buttons (functionality below).
    - **Login screen (on Login button click)**
      * Has Username field, Password field, Login button, and Back button.
      * Login button calls login() function, which will check the username and password against the server.
      * Further explained in data section below.
      * If successful, redirects to LEVEL SELECT.
      * Back button redirects to START screen.
    - **Register screen (on Register button click)**
      * Has Username field, Password field, Register button, and Back button
      * Register button calls validate() to check Username and Password fields’ validity (check password not null and that username not taken).
        + If validated successfully, calls on the RegisteringUserThread on the server to register user.
      * Further explained in the data section below.
      * If successful, redirects to START screen.
      * Back button redirects to START screen.
    - **Continue as Guest (button only, no screen - on Continue as guest button click)**
      * Redirects player to LEVEL SELECT with 2P game disabled.
    - **Quit game button**
      * Exits the application.
  + **LEVEL SELECT →**
    - Player will be able to start a 1P or 2P game (if logged in only).
    - Starting the game will transition the application to the OnePGame or TwoPGame, depending if 1P or 2P was selected.
      * In the case of 2P game, new ServerThreads will be started to listen for player input and broadcast server responses.
  + **LEVEL →**
    - The LEVEL itself refers to the game level selected from the LEVEL SELECT screen.
    - The player will be able to move and interact with objects as they make their way through the level. Upon beating the level, they will be shown the GAME OVER screen.
  + **GAME OVER →** 
    - Upon beating the game, the GAME OVER screen will display with a small animation and the text “Game Over”.

**1b: Game State Management**

* + - The game will transition between menu, display, level, etc. states (that is, any state of the application which has different functionality from any other) by accessing GameStates from a vector in the StateManager. The StateManager will act as a manager for all states – that is, all functions in any state of the game are called through the manager. For instance, if we wish to draw the current frames of a certain GameState, we will actually instruct the StateManager to call the desired function on the current state (which it will keep track of as an integer index of the GameState vector).
    - There will be a GameWindow which will instruct the StateManager to update its current state every ~25 milliseconds.
    - GameState will be an abstract class to facilitate simple creation of new states that extend from it during development.
    - These states will use KeyListeners to detect key input and Graphics2D to draw their output. They will also individually keep track of their Maps, assets, and Agents.
      * Maps are level design instructions encoded as text files. States are able to interpret these files and transform them into arrays of positions.
      * Assets like player animations and tile art will be read by States as BufferedImages to lessen the processing that needs to be done each frame.
      * Agents are any element that could possibly move or act uniquely upon the player. Classes that will extend from Agent include the player character, checkpoints, NPCs, and portals.

**Section 2: Hardware / Software Requirements**

**2a: Programs / Tools**

* Game logic will be written in Java and drawn with Java Swing.
* Sketches, concept art, and game art will be drawn in Adobe Photoshop using Wacom Intuos 4.
* Model vectoring will be done in Inkscape / Photoshop and converted into animation sprite sheets in Inkscape.
* Voice lines will be recorded in Audacity.
* Soundtrack will be developed in MuseScore.

**2b: Libraries**

* Java Database Connectivity (JDBC)
* Java Swing

**2c: Programming Languages**

* Server logic: Java
* Client logic: Java

**Section 3: Data Management**

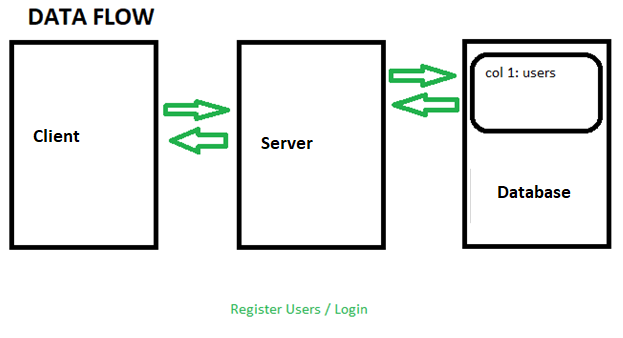
**3a: Database Schema**

* Single-collection MySQL database.
  + Users collection to hold registered users’ names and hashed passwords (indexed by username).
  + See 3b for more details on data structure.

**3b: Data Structure**

* The database collection will store usernames and hashed passwords that we will be accessing and modifying as users are registering.
* Sample login data collection:{  
   username: “ttrojan”,  
   password: hashed password here  
  },  
  {  
   username: “bbruin”,  
   password: hashed password here  
  }

**3c: Pushing and Pulling Data**

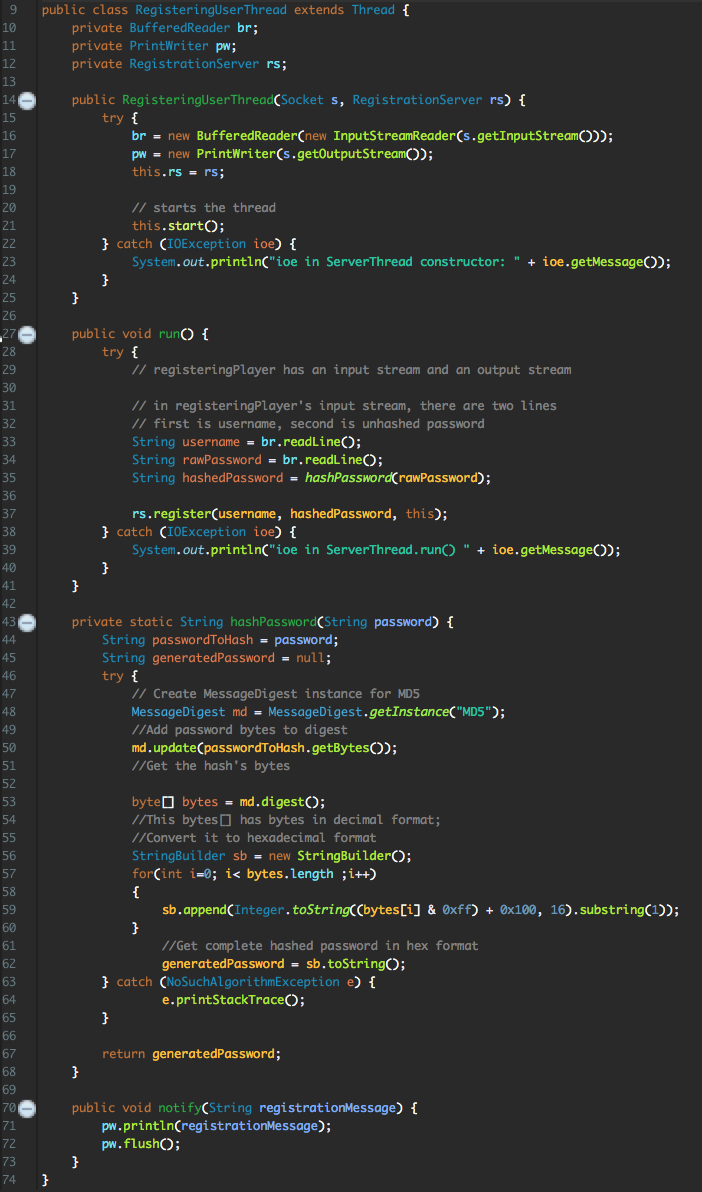
****

**3ci: User Registration**

* At the home screen (start screen), users will be prompted to either login or register which both will be making two separate calls.
  + Registering: There will be three fields for the user to enter data into. First, the user will need to input a username, password, and password confirmation. Upon registering, a validate() function will be called to make sure that the user has entered valid input.
    - If the username that the user is registering with is already taken, an error next to the username field will be displayed.
    - If the password field contains text shorter than 6 characters, an error will be displayed next to the password field.
    - If the text in the password field is not equal to the text in the confirm password field, an error message will be displayed next to the confirm password field.
    - If all the fields are valid, a RegisteringUserThread object is created to represent the user that is registering. This thread contains member variables for a PrintWriter, a BufferedReader, and a reference to the registration server.
      * When the RegisteringUserThread is created, the username and password to register with is read from its socket’s input stream.
      * The password is then hashed so that I can be stored securely using the RegisteringUserThread’s hashPassword(String rawPassword) function.
        + This function performs bitwise operations on the password to transform it to a 32 character hashed version of the Password.
      * The registration server’s register(String username, String password, RegisteringUserThread registeringUser) method is then called.
        + This adds the new username and password to the MySQL database and then notifies the user of successful or failed registration via the registeringUser’s notify(String registrationMessage) method.

This method simply flushes the registrationMessage to the RegisteringUserThread’s output stream so that it can be displayed on the client’s front end.

* + - After the registration is successful, we will trigger a text-to-speech event notifying the player that registration was a success.
    - The code for what RegisteringUserThread, including hashPassword(String rawPassword) and notify(String registrationMessage) functions.

****

* + - After registration, redirect to the login screen.

**3cii: User Login**

* The user login will simply check and verify that the username and hashed password matches a corresponding existing entry in the username/password database table via the login() method.
  + This will make a call to the LoginServer which will make the actual database check.
    - If there is a match for the username and the hashed password, the user will be logged in.
    - Otherwise, an error message will be displayed notifying the user that the username and password combination is invalid.

**Section 4: Graphics**

**4a: GUI Design**

**4a0: Global Specifications**

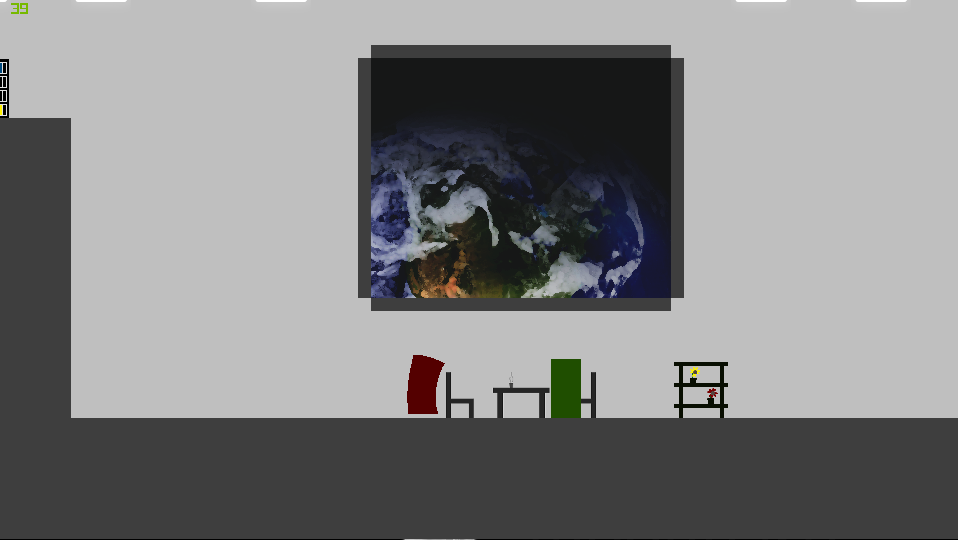
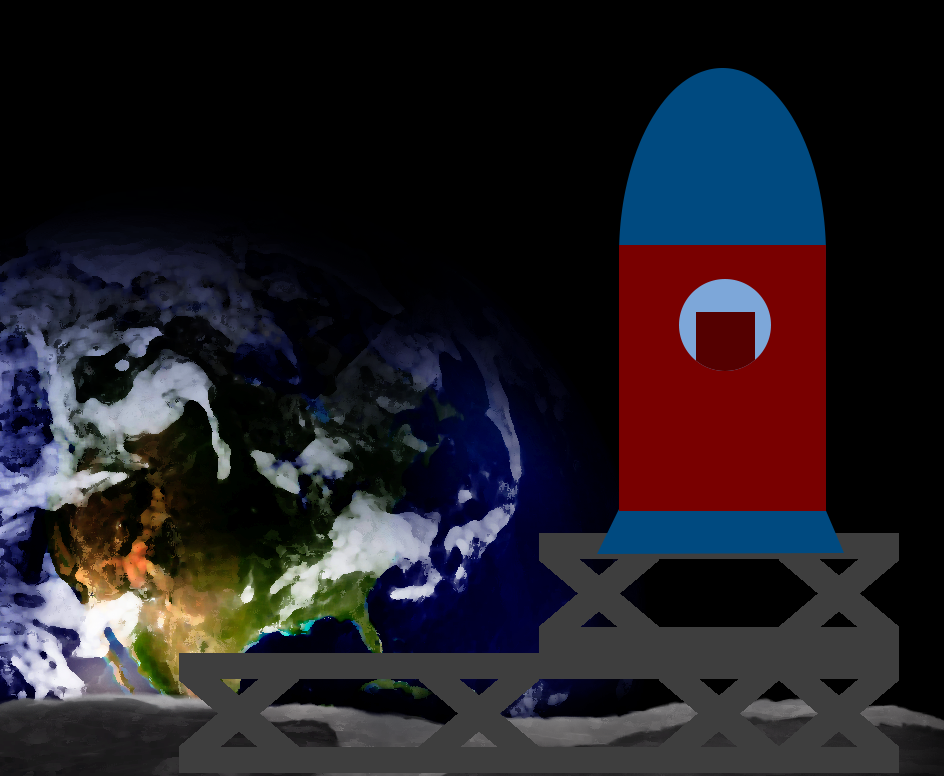
* The application will run in fullscreen only. The GUI will be designed for 1080p (1920x1080 pixels) screens.

**4ai: Pre-LEVEL**

* The first screen shown to the user (START screen) should have four buttons: Log in, Register, Continue as Guest, and Quit Game.
* Behind these buttons, there should be START screen art.
* The START screen should look like this:

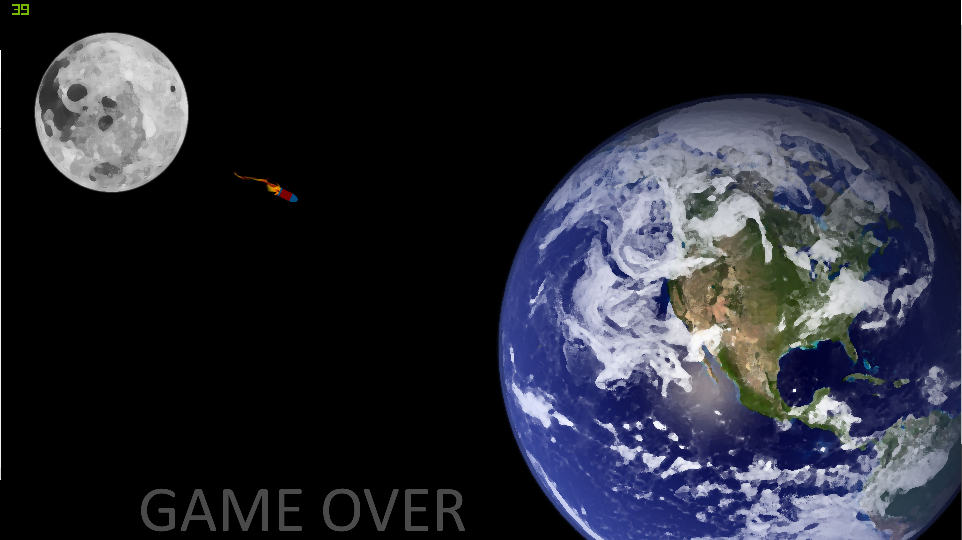
****

**4aii: LEVEL**

* The playing arena on which the actual gameplay will take place (LEVEL) will be a long platform with background art and tiles drawn in the midground. The player will be placed at the beginning of the LEVEL. The camera will follow the player as they move.
* ****The LEVEL should look like this
* The LEVEL’s win condition is to reach the spaceship at the end of the LEVEL. The spaceship will have a short takeoff animation.

**4aiii: Post-LEVEL**

* The post-LEVEL screen will display “Game Over” text and background art.
* This screen will be displayed when the player beats the level by getting to the spaceship at the end.
* The Game Over screen should look like this:

****

**4b: Modelling and Animation**

**4bi: Environment**

* The LEVEL art will be painted in Photoshop, as will the tiles. The tiles will be placed using Graphics2D.

**4bii: Player Character (PC)**

* The PC will be drawn by hand, vectored in Photoshop / Inkscape and converted into sprite sheets for animation in Inkscape.
* PC actions will be animated in the same way.

**Section 5: Sound**

**5a: Soundtrack**

* All soundtracks will be written in MuseScore and edited in Audacity.
* All soundtracks will be converted to WAV and played using javax.sound

**Section 6: Game Mechanics**

**6a: Player Character (PC)**

**6ai: Movement**

* The player will be able to move left, move right, and jump. Moving left or right is accomplished by hitting the left or right arrow keys, and jumping is accomplished by hitting the up arrow key. Holding the up arrow key for longer results in a longer jump. Players will only be able to jump when in contact with a surface below them.

**6b: Collision**

* Tiles will be labelled as “blocking” or “non-blocking”. Non-blocking tiles are tiles that the player can move through. For example, the grey wall tiles are non-blocking. Blocking tiles block player movement. For example, walls and the floor are constructed from blocking tiles.
* Collision will be computed separately in the x and y directions. Colliding with an object in either of these directions will reset that direction’s velocity vector to zero as well as prohibit movement in that direction. Collision will be computed every frame.