Moonbase

- a Java Swing exploration platformer game.

Complete Documentation

Presented to you by:

Jun Yen Leung – junyenle@usc.edu

Ethan Barker – esbarker@usc.edu

Uri Rahimi – urahimi@usc.edu

Jake Leventhal – jglevent@usc.edu

Table of Contents:

High-Level Requirements page 3

Technical Specifications page 4

Detailed Design Document page 7

Testing Document page 19

Deployment Document page 23

High-Level Requirements

**Overview:**

Moonbase will be a side-scrolling platformer-style narrative adventure game. The core gameplay will involve controlling a character capable of basic movement and navigating through a space station to find your way back to Earth. The game can be played either alone, or with another player.

**Section 1: Title Screen, Log In, Registration**

Upon loading the game, players should be shown a title screen with game art and be given the option to log in, register, or continue as a guest. Choosing to register will allow them to choose a unique username and provide a password for logging in in the future. After registration, the player should be redirected to the log in screen. Choosing to log in will allow the player to enter their username and password. These particulars will be checked against the database and the player will be allowed to log in only if a match is found. After logging in, the player will have the option to play either a one or two player game.

**Section 2: Agents**

**2a: Player Character (PC)**

The PC should be able to move left and right, as well as be able to perform two layers of jump (short jump and long jump).

**2a: Non- Player Characters (NPCs)**

The NPCs should be located throughout the world and display a block of text when a PC interacts with them.

**Section 3: Environment**

The environment should consist of tiles that the player can walk on and background art that is purely aesthetic. Located throughout the environment should be several teleporters that should be used as both navigation tools and checkpoints. At the end of the environment, there should be a rocket ship that takes a PC back to earth.

**Section 4: Additional features**

**4a: GAME OVER**

GAME OVER occurs when the player beats the level. A “game over” message should be displayed along with a fun graphic of the player traveling back to earth.

Technical Specifications

**Section 0: Misc. Notes**

* Platform: Moonbase will run on any system with Java. It will be run as an executable JAR.
* Moonbase will use keyboard controls.

**Section 1: Title Screen, Login, Registration**

Title Screen / Login / Registration (8 Hours)

* + The main title screen needs to have four buttons: login, register, continue as guest, and quit game.
    - If the login button is clicked, the user needs to be brought to a screen with a form that takes two text fields: username and password. Here, there must be a back button which returns the user to the previous screen, and a login button that submits the form. Clicking this login button should check the username and password inputs against the database. If the username and password are valid, the user will be logged in. If not, an error message should be displayed.
    - If the register button is clicked, the user needs to be brought to a screen that has a username field, password field, register button, and back button. The back button should send the user back to the previous screen.
    - If the “continue as guest” button is clicked, we will let the user proceed and to the main game as a “guest user” that can only play the 1-Player Game.
    - Once a user is logged in, there should be two additional options on the menu screen, 1-Player Game and 2-Player Game.
    - Selecting 1-Player or 2-Player Game on any appropriate screen should begin the level.
    - If the quit game button is pressed, the application should exit.

**Section 2: Agents**

**2a: Player Character (PC)**

PC Actions (8 Hours)

* + The PC should have the following abilities within the game world:
    - If the up button is pressed, the PC should jump. Holding down the button should result in a higher jump.
    - If the ‘left’ button is pressed, the PC should move to the left.
    - If the ‘right’ button is pressed, the PC should move to the right.

PC Art/Animations (8 Hours)

* + PC art and animation will be drawn, vectored, and interpreted as sprite sheets.

**Section 3: Environment**

**3a: Player-acting environment**

Environment Design (8 Hours)

* + The level will be created by making a grid of tiles in a tile map. Tiles need to be drawn 120x120 pixels in size.

**3b: Non-player-acting environment**

Environment Design (8 Hours)

* + Non-player-acting means environment features that are purely aesthetic.
  + There needs to be a painted background to add an element of world immersion to the game.

**3c: Player / Environment Bridging**

By this we mean the elements that help connect the player on the other side of the screen to the environment we have created.

Player Camera (8 Hours)

* + The Player Camera (PLAYCAM) determines which elements of the LEVEL are visible on the player’s screen.
  + The PLAYCAM should be centered on the PC and move with the PC.
  + The width of the PLAYCAM should be the width of the player’s screen, up to a maximum of 1920 pixels.
  + This will be done by updating the x and y positions of the PLAYCAM’s center every few frames and setting it equal to the PC’s x and y positions.
    - We do this every few frames and not every frame so that we can achieve a delayed camera following effect (for style purposes).

**Section 4: Additional features**

**4a: GAME OVER**

GAME OVER Event (2 Hours)

* + GAME OVER occurs when the player wins.
  + The player should be taken to a screen that says “Game Over” with an animation showing the player traveling back to Earth.

**Section 5: Database**

Database (8 Hours)

* + We will use a MySQL database hosted on one of the project member’s desktop computer.
  + The database (DB) will have one collection:
    - It will consist of all registered players and their login details. For instance,

{

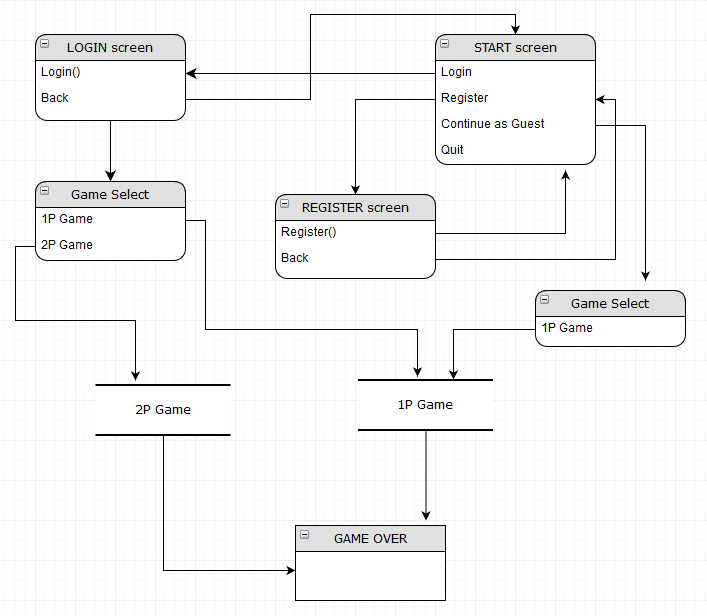
username : “ttrojan”,

password : “fighton”

}

This collection should be indexed by username for quick querying when players try to log in.

Detailed Design Document

**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**

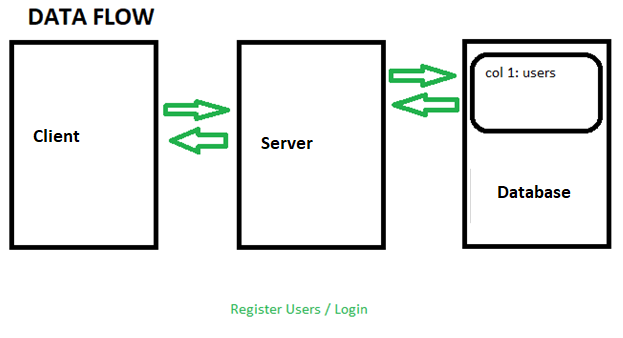
**3a: Database Schema**

* 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**

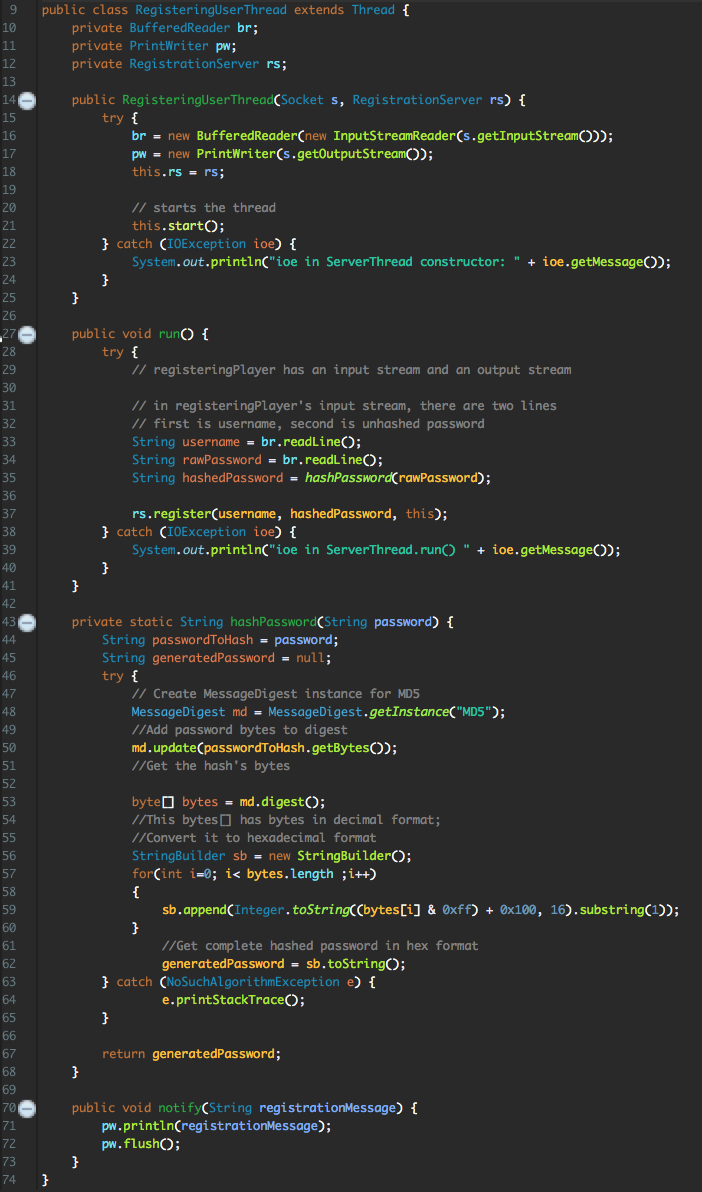
****

**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**

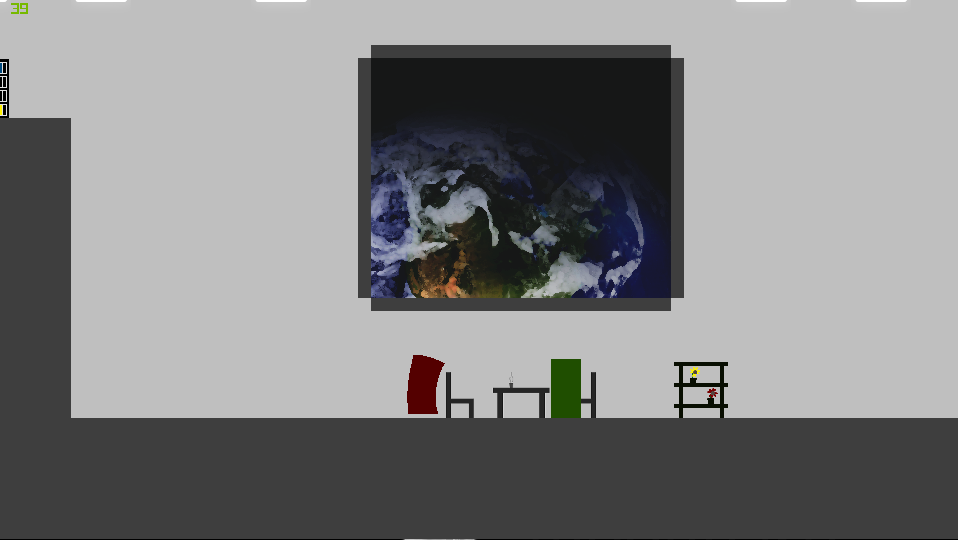
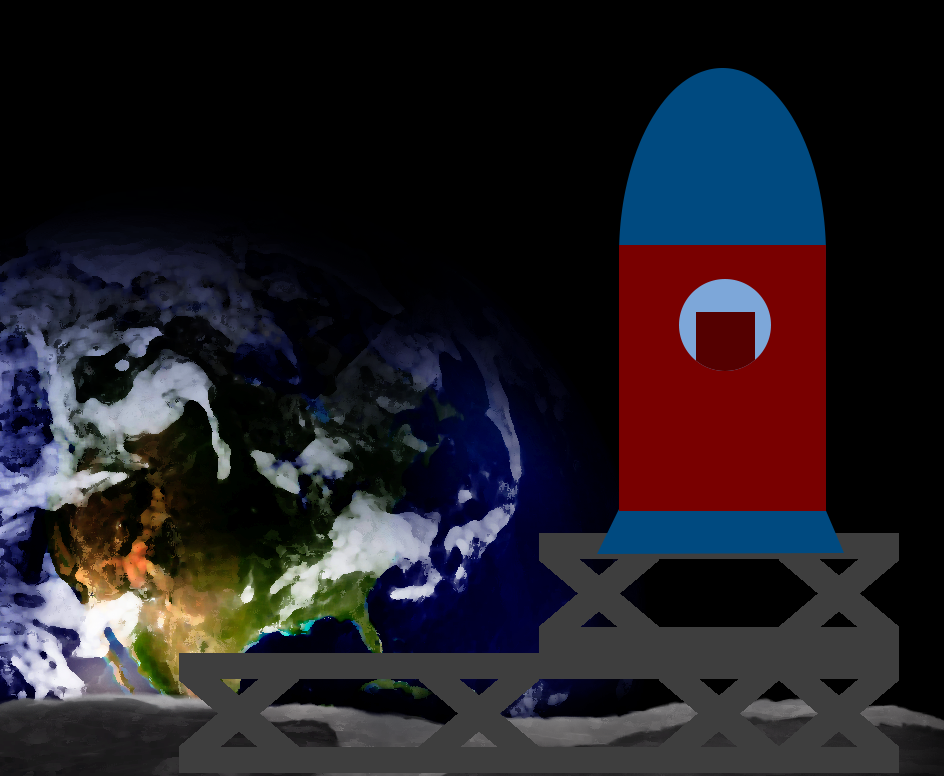
* The application will run in full-screen only. The GUI will be designed for 1080p (1920x1080 pixels) screens.

**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:

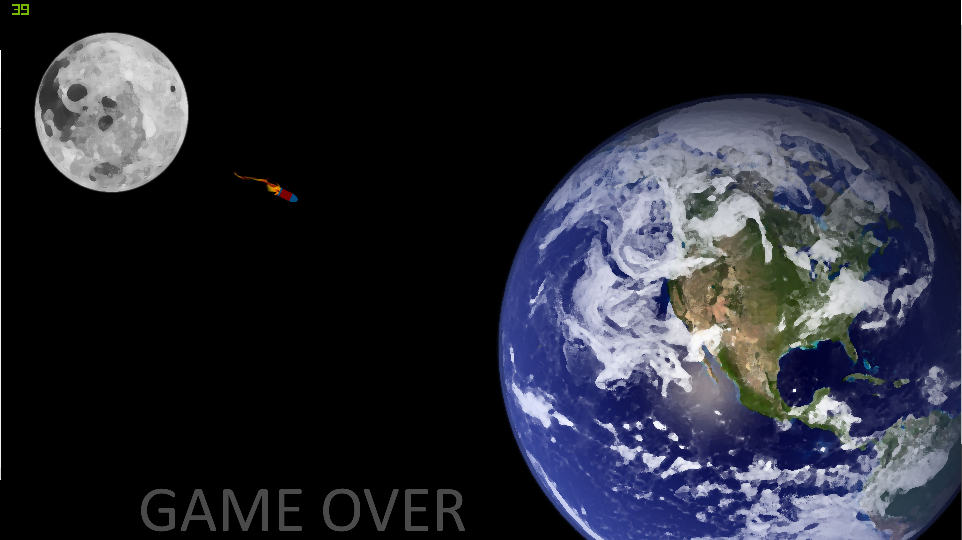
****

**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.

Testing Document

**White Box Tests**

Login and Registration

* Test: Attempt to login without any text in either input field.  
  Expected Behavior: Error message should display to notify user that the username/password combination is invalid. User should be able to attempt to login again.
* Test: Attempt to login without a password.  
  Expected Behavior: Error message should display to notify user that the username/password combination is invalid. User should be able to attempt to login again.
* Test: Attempt to login without a username.  
  Expected Behavior: Error message should display to notify user that the username/password combination is invalid. User should be able to attempt to login again.
* Test: Attempt to login with the wrong password.  
  Expected Behavior: Error message should display to notify user that the username/password combination is invalid. User should be able to attempt to login again.
* Test: Attempt to login with a nonexistent username.  
  Expected Behavior: Error message should display to notify user that the username/password combination is invalid. User should be able to attempt to login again.
* Test: Attempt to login with a nonexistent password *and* nonexistent username.  
  Expected Behavior: Error message should display to notify user that the username/password combination is invalid. User should be able to attempt to login again.
* Test: Attempt to login with a valid username and password.  
  Expected Behavior: User should be logged in successfully and game should be playable by logged-in user.
* Test: Attempt to register without any text in either input field.  
  Expected Behavior: Error message should display to notify user that a username and password are required. User should be able to attempt to register again.
* Test: Attempt to register with a username over 25 characters.  
  Expected Behavior: Error message should display to notify user that usernames can only be up to 25 characters long. User should be able to attempt to register again.
* Test: Attempt to register without any text in the username input field.  
  Expected Behavior: Error message should display to notify user that a username is required. User should be able to attempt to register again.
* Test: Attempt to register without any text in the password input field.  
  Expected Behavior: Error message should display to notify user that a password is required. User should be able to attempt to register again.
* Test: Attempt to register with an existing username.  
  Expected Behavior: Error message should display to notify user that the supplied username is already taken. User should be able to attempt to register again.
* Test: Attempt to register with a unique username and a password.  
  Expected Behavior: User should be successfully registered. User should be redirected to the Login screen.

Gameplay

* Test: Press the right arrow key button.  
  Expected Behavior: The PC should move to the right.
* Test: Press the left arrow key button .  
  Expected Behavior: The PC should move to the left.
* Test: Press both left and right arrow key buttons.  
  Expected Behavior: The PC should not move.
* Test: Press the up button while on the ground.  
  Expected Behavior: The PC should jump and should land on the ground afterwards.
* Test: Press the up button while in the air.  
  Expected Behavior: Nothing should happen. The PC should continue their original jump.
* Test: Attempt to walk off screen to the left and right.  
  Expected Behavior: The PC should collide with an invisible wall and be unable to walk off screen.
* Test: Attempt to jump while outside the space station.  
  Expected Behavior: The PC should jump higher than normal due to reduced gravity.
* Test: Walk into flag on space station.

Expected behavior: The PC should be teleported to the corresponding teleportation location and should not be able to re-enter the teleporter.

* Test: Fall off the bottom of the level after reaching a checkpoint.  
  Expected Behavior: The PC should be immediately teleported back to the last checkpoint that they interacted with.
* Test: Walk near NPC.  
  Expected Behavior: A dialogue box should be displayed above the NPC.
* Test: Walk into the space station.  
  Expected Behavior: The player should be brought to a game over screen.

Post-Gameplay

* Test: Complete the game and return to the main menu.  
  Expected Behavior: After reaching the game over screen, the user should be able to return to the main menu by selecting the return to menu button.

**Black Box Tests**

Login and Registration

* Test: Attempt to register with control characters in the username (e.g. \t, \n).  
  Expected Behavior: The game should not interpret these characters and instead register the user with those characters as plaintext (e.g. “Dave\nMiller”). The database should not have any problems storing this username.
* Test: Attempt to register with control characters in the password (e.g. \t, \n).  
  Expected Behavior: The game should not interpret these characters and instead register the user with those characters as plaintext (e.g. “Dave\tPW”). The database should not have any problems storing this password.
* Test: Attempt to register with foreign characters in the username.  
  Expected Behavior: An error message should be displayed if any non-UTF8 character is used. Otherwise, user should be successfully registered.
* Test: Attempt to register with foreign characters in the password.  
  Expected Behavior: An error message should be displayed if any non-UTF8 character is used. Otherwise, user should be successfully registered.

Gameplay

* Test: The player can interact with the appropriate area in the game and is stopped by the walls.   
  Expected Behavior: Walls should keep players from interacting with the surroundings that are not meant to be part of normal game play.
* Test: The player can reach all of the intended game space within the game  
  Expected Behavior: The player should be able to reach all of the corners, floating blocks and obstacles that they may encounter throughout game play.

Post-Gameplay

* Test: Reach the end of the level.  
  Expected Behavior: game over screen should be displayed.
* Test: Reach the end of the game
* Expected Behavior: The user should be able to finish the game and end. After that, the user is shown an appropriate end of game message.

**Stress Tests**

Login and Registration

* Test: Attempt to register multiple users at the same time.  
  Expected Behavior: All new users should be registered successfully.
* Test: Attempt to register 100 users.  
  Expected Behavior: All new users should be registered successfully.
* Test: Attempt to Login multiple users at the same time.  
  Expected Behavior: All users should be logged in successfully.
* Test: Attempt to Login 100 users at the same time.  
  Expected Behavior: All new users should be logged in successfully.
* Test: Attempt to Login a user while registering a user.  
  Expected Behavior: The new user should be registered successfully and the logged in user should be logged in successfully.
* Test: Attempt to Login or register rapidly (e.g. mashing the Login button).  
  Expected Behavior: Only one request should be served per second.

Gameplay

* Test: Leave the game running for an hour.  
  Expected Behavior: Game should still run without problems
* Test: Run the game alongside other applications (web browser, etc.)  
  Expected Behavior: Game should still run smoothly.

Post-Gameplay

* N/A

**Unit Tests** (broadly categorized for organization – specific tests mostly covered in White / Black Box sections)

Login and Registration

* Test: Login Validation Tests – All white box tests on invalid login inputs.  
  Expected Behavior: As specified in the white box tests.
* Test: Registration Validation Tests – All white box tests on invalid registration inputs.  
  Expected Behavior: As specified in the white box tests.
* Test: Data Flow Tests – All white box and black box tests on inserting and retrieving data.  
  Expected Behavior: As specified in the white and black box tests.

Gameplay

* Test: Movement Functionality Tests – All white box tests on PC movement.  
  Expected behavior: As specified in white box tests.

Deployment Document

Requirements:

* Tomcat 9.0 (<https://tomcat.apache.org/tomcat-9.0-doc/appdev/installation.html>)
* MySQL (<https://dev.mysql.com/doc/refman/5.7/en/installing.html>)
* Java SE Development Kit 8 (<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>)
* Eclipse Oxygen (<https://wiki.eclipse.org/Eclipse/Installation>)

Database Setup:

* Create a new MySQL Connection.
* Open the MySQL Shell and connect to this connection.
* Run the provided setup.sql script.

Server Setup:

* Start Eclipse and import the provided Server project (included as a ZIP file).
* If necessary, change the user and password for the MySQL Connection in the RegisterState and LoginState files (we have set the server to connect to root@localhost with password: “root” by default).
* Run Server.ServerDriver.java on a Tomcat server.

Game Setup:

* Change the IP address stored in String host in States.StateManager.java to the IP address of the server.

Game Run:

* Run the Driver.GameStarter.java file.

Distribution:

* Export the Eclipse project as an executable JAR.
* Run the JAR to launch the game.