CSCI201 Final Project

Dragons & Fireballs

By Anay Patel, Dakota Palmer, David Pham, Jonathan Ohmsted

Team Members and Project Proposal:

Team Members

CP in charge: Nikita Pashintsev : pashints@usc.edu

- David Pham
    - davidvph@usc.edu
- Jonathan Ohmsted
    - johmsted@usc.edu
- Anay Patel
    - anaypate@usc.edu
- Dakota Palmer
    - dakotapa@usc.edu

Meeting Time: Fridays 5 - 7pm

Project Proposal

Name: Dragons & Fireballs

Concept: We are planning to create a player vs player projectile game where players will attempt to hit each other with some sort of object — fireballs in this case, but may be subject to change. On the server side, we'll be utilizing java on the back end to keep track of each players' positions and locations of the projectile positions when they launch one. Our GUI would consist of a 2-Dimensional layout with several players on the map and multiple obstacles spanning the map as well. We'll also be incorporating multi-threading within our code by supporting some sort of chat system within the game and perhaps the lobby — a staging area for players to generally be before entering an actual game.

High Level Requirements

We need to create a 2D game where multiple players will be able to move and shoot projectiles at one another. There will be a lobby where players enter prior to entering an actual game, and they will be given the option to login or sign up for an account with saved information (stats, name on the leaderboard, kills). They will then be given the option to connect to a game. After leaving the game, a board will be displayed containing information regarding kills and deaths of each player within the game. The gameplay goal of each player is to achieve as high a score as possible, which can be achieved through killing other players.

Technical Specifications

Sprites

- 1 hour
- Create sprites in Photoshop for player, projectile textures

Interface

- 6 hours
- Login page with username, password, and login button
- Signup page with username, password, and signup button
- Join game button that will place them in the game, connect to an IP address and port
- Stats/leaderboard page showing players with top scores

Database

- 2 hours
- The database will consist of one table for Users, which will store username, password, and score (kills and deaths)
- The Users table will be used for validating players on login and creating new players on signup

Server Functionality

- 6 hours
- Implementing a server that will allow multiple players to be playing at the same time
- Implementing multi-threading

Game Implementation

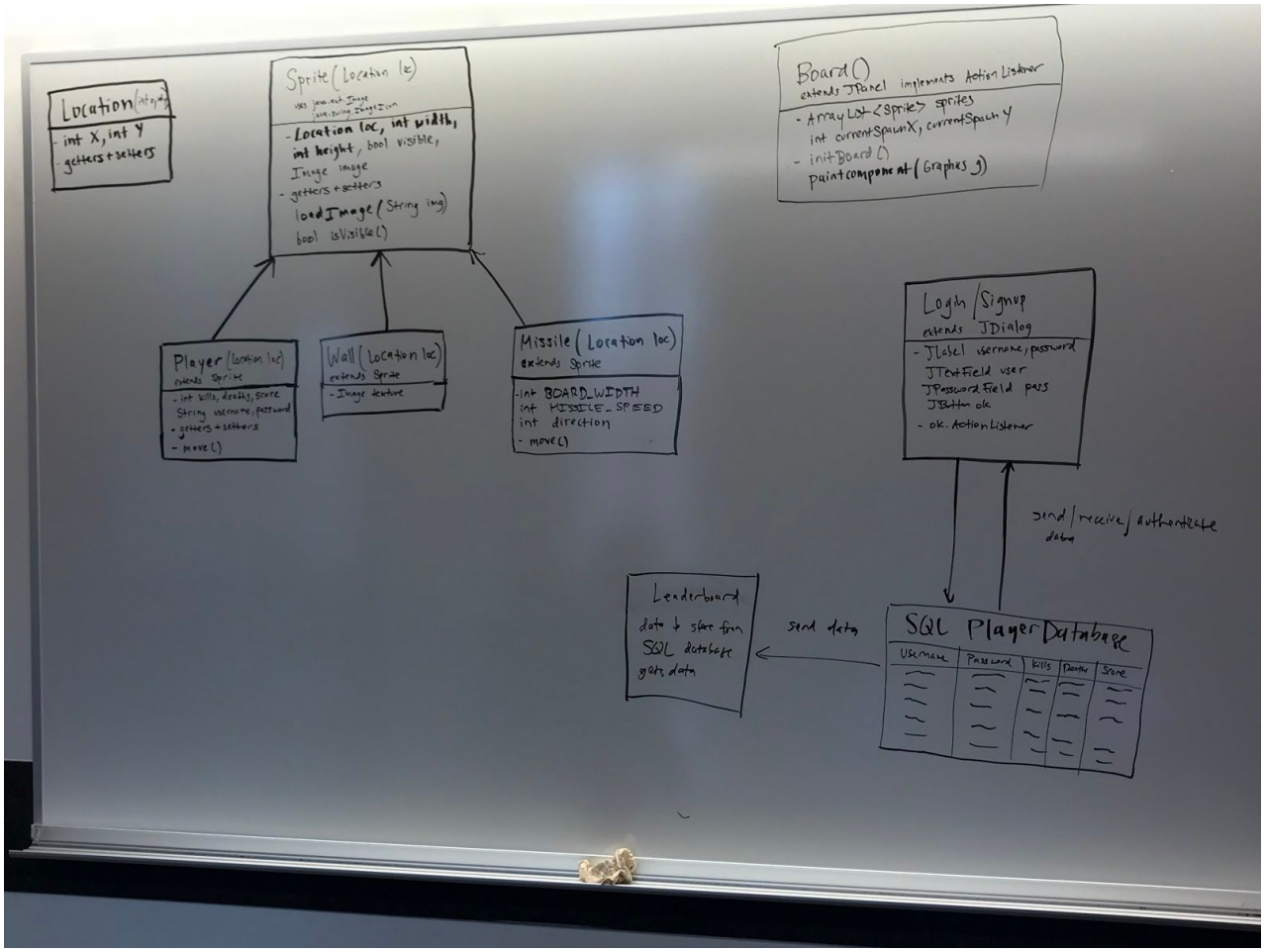
- 15 hours
- Projectile detection

- Player, projectile movement
- Score tracking system
- Respawning system

Gameplay Goal: to achieve as high a score as possible, which can be obtained by getting kills.

Detailed Design Document

Preliminary Class Hierarchy



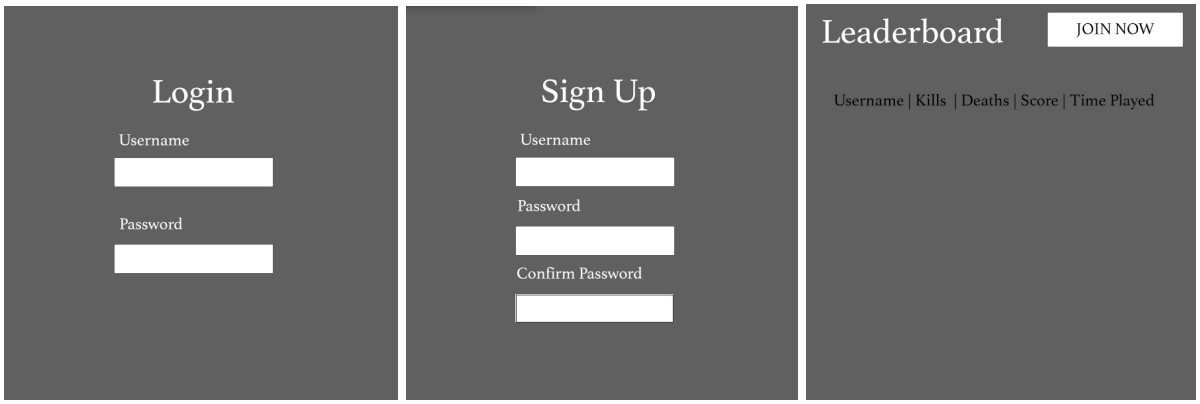
Detailed Class Hierarchy

- Client Package
    - Client extends Thread
        - Client()
            - Initializes GUI
        - JFrame applicationWindow
            - JPanels for each panel in the game (signup/login, join game, leaderboard, game board)

        - Connects to Server Thread
        - Use Object output/input streams to send/receive data
- Frames Package
    - Board extends JPanel implements ActionListener
        - Player player (contains player object)
        - initBoard()
        - Graphics
            - paintComponent()
            - doDrawing(Graphics g)
        - move()/actionPerformed()
            - Calls player's move function
    - Leaderboard extends JPanel
        - initLeaderboard()
            - Configures JPanel properties, front end design
            - Retreives data from SQL server
            - Displays it in Leaderboard Panel
    - Login extends JPanel
        - initLogin()
            - Configures JPanel properties, front end design
        - updateDatabase(Player p)
            - Updates a players information in the database when they have left the game
        - authenticate()
            - Retrieves data from SQL server
            - Validates/adds to the database depending upon whether the user is signing in or logging out
            - Uses AbstractAction to send/receive data from SQL server
    - MainFrame extends JPanel
        - initilizeConnectionPanel()
            - Configures JPanel properties, front end design
            - Uses AbstractAction to send/receive data from Server/Client

- Resources Package
    - Sprites .png files stored here
- Serialized Messages Package
    - GameMessage implements Serializable
        - GameMessage(int clientID, String protocol, String message)
        - Used to send message between server and client
- Server Package
    - GameServer extends Thread
        - GameServer(int port)
            - Trys to bind to port
        - initGameLoop()
            - Infinite game loop that runs the game when the server starts
            - Handles collisions between spirtes, handles locations of sprites, etc
        - run()
            - Sends/receives data from the client
- Spirtes Package
    - Player
        - Player(Client c)
        - Handles client key events
        - Stores location, kills, deaths, etc
    - Projectile implements Serializeable
        - Projectile(int x, int y, int id, String direction)
        - move()

Preliminary Design Ideas


Login
Username
Password
Sign Up
Username
Password
Confirm Password
Leaderboard
JOIN NOW
Username | Kills | Deaths | Score | Time Played

Testing Document

White box testing

- **Test the entire application while looking at the code**
- **Test Case 1 –** test the login functionality by specifying a username that exists and a password that does not match. The user should be taken back to the login page with a message “Invalid login.”
- **Test Case 2 –** test the login functionality by specifying a username that does not exist. The user should be taken back to the login page with a message “Invalid login.”
- **Test Case 3 –** test the login functionality by specifying a username that exists and a password that matches. The user should be taken to a page where they can join the game or look at their stats
- **Test Case 4** - Disconnect a player and make sure the database gets updated with their score
- **Test Case 5** - Have a player exit the game and make sure the database gets updated with their score
- **Test Case 6** - Test the signup functionality by trying to use a duplicate username that has already signed up, should display a message
- **Test Case 7 -** During sign up method, the username should be a max length of 12 characters and password should be a max of 8.

Black box testing

- **Test the entire application without looking at the code**
- **Test Case 1** - Gather multiple people to actually test the game by simply playing it normally, and getting feedback after
    - Start with a game where nobody joins mid-game
    - Start with a game where multiple people join mid-game
    - Gathering data on what they think about the functionality, look of the game, any lag, etc.

- **Test Case 2** - Gather multiple people to try and login and signup using various usernames, check to see if the functionalities of the login/signup page work properly.

Stress testing

- **Test the extensibility of the program by trying to find the limits**
- **Test Case 1 -** When the project is completed, try to add as many users as possible playing at the same time.
- **Test Case 2** - Try to move off the board with an entity in all four directions.
- **Test Case 3** - Make sure a projectile doesn't move off the board or through walls
- **Test Case 4 -** Ensure that all sprites are properly created and displayed moving correctly across the application board.

Unit Testing

- **Test individual functionality of the code by writing customized programming**
- **Test Cases : Client Application**
    - **Test Case 1**- spawn projectile at known velocity and position, verify it's position is accurately updated proportional to time
    - **Test Case 2** - Compare position/velocity values of players between client and server, make sure they match
    - **Test Case 3** - Compare position/velocity values of projectiles between client and server, make sure they match
- **Test Cases : Server Application**
    - **Test Case 1 –** insert SQL code in the username variable of the UserAuthentication.authenticate() method. Verify that the SQL code specified is not executed against the database.
    - **Test Case 2** - Input from user does not modify methods for accessing database (SQL Injection).

Regression Testing

- **When changes are made, make sure the changes don't affect other parts of the program**
- **Test Case 1** - when a user registers with an existing username, make sure the password they enter doesn't overwrite the existing user's
- **Test Case 2** - when a projectile hits a wall, the wall shouldn't magically disappear
- **Test Case 3** - when a projectile is shot, the player that shot it shouldn't also move

Deployment Document

- Installation of Java Platform (JDK) Required
    - http://www.oracle.com/technetwork/java/javase/downloads/index.html

How to run through jar file

- Download the .rar file that contains the game and unpack it
- Set up the SQL server
    - Make sure the SQL server on your machine has a username of “root” and a password of “root”
    - Make sure the SQL server is started
    - Execute the create_database.sql query in the rar archive within the sql folder, which creates a database called GameInfo with a table called Users (id, username, password, kills, deaths)
- If in Windows Environment
    - Open up command prompt and navigate to the directory of the unpacked rar file.
    - Run “java -jar server.jar”
        - Choose a port greater than 1024
    - Open another command prompt and navigate to the same directory and type “java -jar client.jar”
        - Specify IP address and port the server.jar is bound to.
- If in Mac Environment
    - Open up terminal and navigate to the directory of the unpacked rar file.
    - Run “java -jar server.jar”
        - Choose a port greater than 1024
    - Open another terminal window and navigate to the same directory and type “java -jar client.jar” OR double click the client.jar to open it
        - Specify IP address and port the server.jar is bound to.
- Playing the game

- Use the WASD keys to move
- Use space bar to shoot a projectile
- The goal is to hit the other players as much as you can with projectiles, and avoid the projectiles of other players
- Keep in mind that you can only see in a small radius around your player
- Press escape to view the leaderboard/sign out

How to run through eclipse

1. Make sure your build path is updated. If there are errors, there is a folder called "jar files" in the project folder. Update the build path such that those jar files are linked with the project.
2. SQL server
    - Make sure the SQL server on your machine has a username of “root” and a password of “root”.
    - Make sure the SQL server is started (this is very important, as you will not be able to create the database or login/signup to the game unless the server is running)
    - Execute the create_database.sql query in the project within the sql folder, which creates a database called GameInfo with a table called Users (id, username, password, kills, deaths)
    - This can also be done manually, but it is important that the schema name, the table name, and the column headers match those above.
3. Open the GameServer class and run it. In the console, choose a port greater than 1024.
4. Open the Client class and run it. When prompted, sign up for an account or log in. Make sure your username/password are greater than 6 characters when you sign up.
5. Once you are signed in, connect to "localhost" with the port that you specified when you ran the server. Click the join button.
6. Use the WASD keys to move around, and the spacebar button to fire.
7. Press escape to log out and view the leaderboard.

8. Try testing it with multiple clients on your computer.

Game Rules

1. Each player has 5 lives per round
2. Each player can fire up to 5 shots at once (once the shots have left the screen, they can fire again)
3. If a player runs out of lives, they will wait until there is only one player standing, after which the game will reset.
4. You can only see a small radius around you.
5. Once there is only one player left standing, a new round will commence and players will be able to re-connect.