Arcade Mania

[1]Mayur Prabhakar Sangale, [1]Divya Kharche, [1]Prasad Satish Deshpande
[1]Information Systems, Northeastern University
{sangale.m, Kharche.d, Deshpande.prasa}@northeastern.edu

***Abstract*—The final project is a game that involves two players taking turns to engage in a fight competition. There are six playable characters to choose from, namely Gladiator, Kasumi, Akuma, Viking, Thor, Ninja. Each player is randomly allocated three player from the pool of six, and they battle using one another player at a time. Every player has two abilities, and the effects of the abilities vary depending on the player they are used against. Players can strategically use abilities that favor their match-up or retreat from unfavorable positions by switching the other players. The game ends when one player loses all three players. An event-driven approach is used to design the game, where the game state is updated when the player clicks on the ability buttons. A global Boolean variable is used to determine the player's turn status, and the game's user interface is conditionally rendered based on the value of this variable. Over 10 classes and their associated properties and methods are used to create a 2-player fighting environment where the end-user interacts with method contracts solely through the player class.**

Outcomes:

- **Developing a turn-based fighting game for two players with added cosmetic features, including animations, sounds, dialog boxes, and transitions..**
- **Creating concrete factory classes that will provide the necessary components such as players, Abilities, MediaPlayers, and Animations.**
- **Implementing the player interface to define the expected behaviors for the player class.**
- **Utilizing composition of classes extensively to achieve the game design.**
- **Utilizing data structures such as HashSet and ArrayLists to allocate and store players.**
- **Appreciating the event-driven approach used in the game algorithm to manage and progress the game state.**

Keywords - Java, JavaFX, SceneBuilder, game, animations, MySQL database

I. Problem Description

The gaming industry is witnessing a surge in popularity for complex 3D simulation games that boast realistic graphics. However, these games suffer from long loading times and steep learning curves, making it difficult for players to get into them. Furthermore, due to their heavy resource consumption, players are compelled to upgrade their hardware.

In contrast, games like Minesweeper and Solitaire are straightforward and easy to play, with fast startup times and no learning curve. They can be enjoyed in any setting due to their short duration.

The objective of the Arcade Mania is to provide players with a simple and entertaining game that can be played between two individuals without any special hardware requirements. Players can choose their favorite character and compete against another character, engaging them both mentally and physically.

The goals of the Arcade Mania are to:

- Creating an engaging and entertaining game that is simple, fast-paced, and interactive.
- Ensuring that the game has a minimal learning curve so that players can start playing immediately without requiring additional knowledge or skills.
- Developing a game that does not require any additional knowledge or specialized equipment.

II. Analysis (Related Work)

In addition to being a turn-based 2-player game, our project also belongs to the popular RPG (role-playing-games) genre. As a result, it incorporates elements inspired by existing games in this category. Specifically, we drew inspiration from the "Pokemon" game series, which has a similar turn-based fighting mechanism. During the game's development, we studied the Pokemon series and sought to address its shortcomings by incorporating our own innovative features.
According to Mombach et al.'s research [5], the Pokemon game snippets exhibit a user interface with two Pokemon characters engaged in combat. However, there were no attempts made to enhance user interactions, as the game's interface merely featured a static dialog box.

Our project aimed to incorporate common UI elements found in games like Pokemon, such as a 1-on-1 face-off, health bars, ability panes, and dialog boxes. However, we went a step further by adding transitions, animations, and media that correspond to the dynamic state of the game. This was done to make the game more lively and engaging for the players.

III. System Design

Our project utilizes four main screens – Login, Home, Game, and End - each of which is rendered by a corresponding controller class: LoginController, HomeController, GameController, and EndController. To facilitate the transfer of scene and stage information between these screens, we defined a parent class called SceneController. This class contains methods for setting the appropriate scene and is extended by all the controller classes. A UML diagram is provided below to illustrate this structure, with some methods omitted for conciseness.

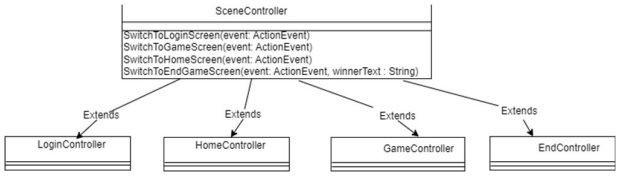

Figure 1. *Inheritance used to communicate between controllers*

To achieve the design objective, our project relies heavily on class compositional relationships. This hierarchical system architecture is illustrated in the diagram below, which omits some methods and fields for the sake of brevity:

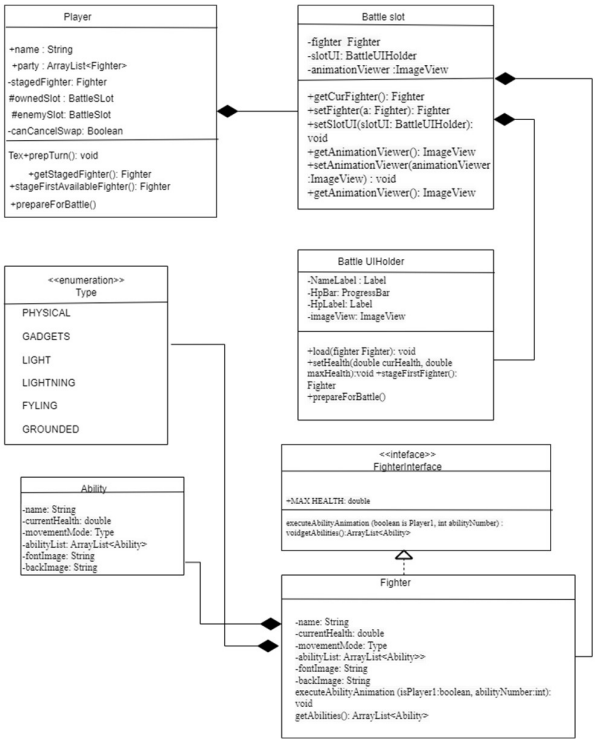

Figure 2. *Compositional relationships and abstraction displayed in class diagram*

To reduce the level of tight coupling between the producer code and the consuming client code, we extensively employed factory design patterns in our project. This was done to provide Players, Abilities, Animations, and Media through common factory classes.

IV. Implementation

A. Scenes

- The game has four screens: Login, Home, Game, and End, created using FXML files.
- The SceneController class acts as the parent controller and manages screen transitions on the stage.
- The LoginController, HomeController, GameController, and EndController extend the SceneController class and implement the Initializable interface, allowing them to execute tasks upon screen loading, like setting up event handlers, media, and animations.

B. Animations

- Functions were developed on the home page to showcase different superheroes and their superpowers.
- The implementation involved utilizing classes like TranslateTransition, ScaleTransition, and FadeTransition.
- A SequentialTransition class was used to ensure that the transitions occurred in a specific order.
- The FXML annotation was utilized within the controller to directly access image views created through Scene Builder.
- The MediaPlayer class was employed to play media and handle MalformedURLException.
- The start game button was enabled through setOnFinished event handler.
- Animations were triggered for game actions during the gameplay, facilitated through the AnimationFactory class.
- The AnimationFactory class contains methods that execute different animations, taking image views and a boolean value indicating which player's turn it is as input parameters. As the image views of both the player and the superpower need to be changed, the direction of these animations is determined by whose turn it is.

C. Game

- The Interface defines attributes and methods for a player such as getAbilities, executeAbilityAnimation, health, etc.
- These methods also add the list of abilities generatedvia the AbilityFactory.
- The class consists of static and action images for a player.
- The initializePlayersParty selects random player objects and sets them up on the stage.
- Ability class consists of name, type, and base damage for an ability.
- BattleUIHolder manages player health and displays required data.
- The BattleSlot determines damage taken or inflicted based on the opponent.
- The Player class has a list of assigned player objects, a battle slot, and ability to swap.
- MediaPlayerFactory supplies media based on a given ability.
- The Event handling heavily handles the game.
- The buttons that activate superpowers have been given a setOnAction event handler that executes different actions upon being clicked, such as disabling the buttons during animation, playing media, computing damage, displaying a dialog box, and more.
- The setOnFinished event handler is responsible for actions that can only be performed after the superpower animation has finished.
- It resets the player image to a static position and the superpower image view as well..
- The setOnAction event handler also handles swapping players and ending the game if all players have lost their health, using methods like toggleSwapMenu, isOver, and canFight.

Characters	**Weak Against** **(Abilities)**	**Strong Agains** **(Ability Used: Opponents)**
Gladiator	Akuma sword, ninja katana	*viking_axe*
Kasumi	viking_spear, akuma_sword	ninja_shuriken, thor
Ninja	gladiator_sword, akuma wings	kasumi_sword, viking_axe
Akuma	viking_axe, ninja_katana	flying-shield, akuma_sword
Viking	kasumi_gun, ninja_shuriken	*akuma wings*
Thor	flying-shield	*viking_spear*

Table 1: This table describes the fighters and their weaknesses and superpowers so that the players can play strategically and it is also to create damage relative to the opponent

V. EVALUATION

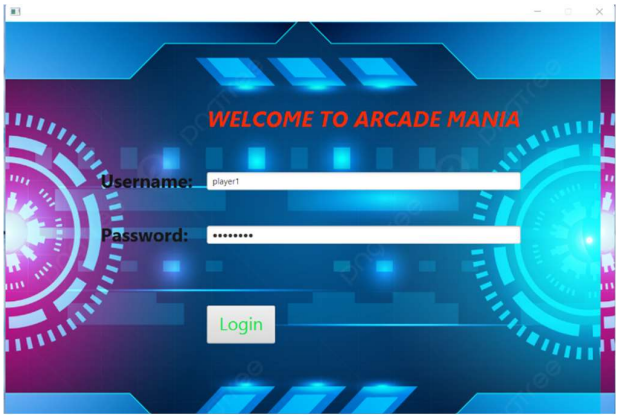

Figure 3: Login page

Figure 3 shows the login page having username and password and the login button where players can login and authenticate themselves

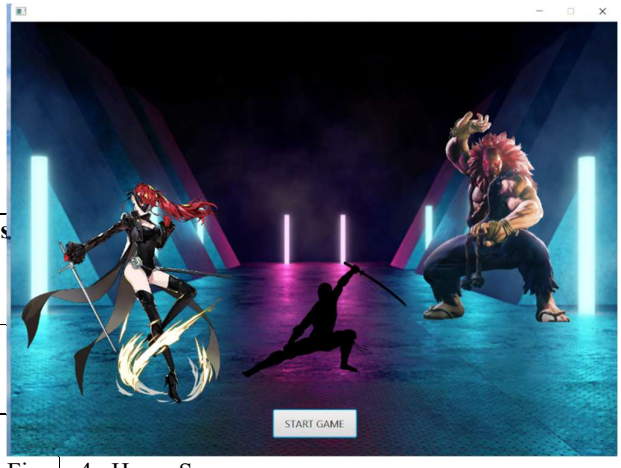

Figure 4 : Home Screen

Figure 4 shows the homepage displays animated entries highlighting their unique superpowers in an engaging manner, which is a fun way to captivate the user's attention from the start.

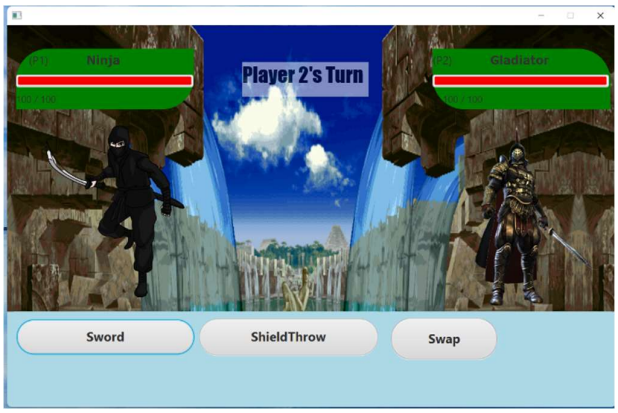

Figure 5: Game screen at beginning

Figure 5 displays the start of the game screen, as soon as one of the superpower buttons is clicked, the onAction event will trigger.

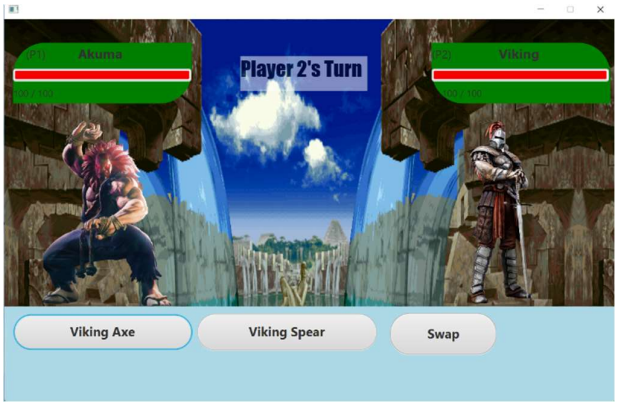

Figure 6: Akuma v/s Viking

As shown in Figure 6, Akuma and viking has started the fight

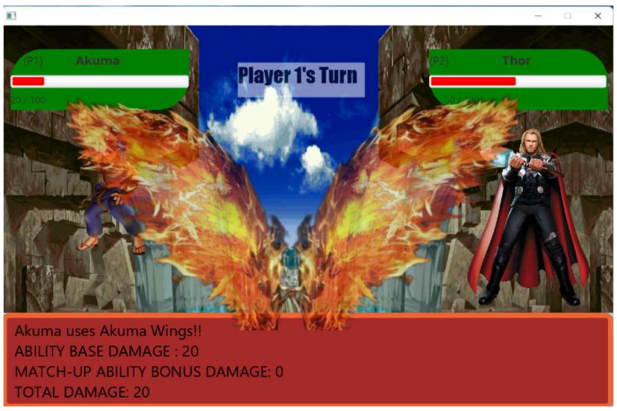

Figure 7: Akuma v/s Thor

Figure 7 shows the Akuma uses Akuma wings and the ability base damage is 20 and total damage is 20

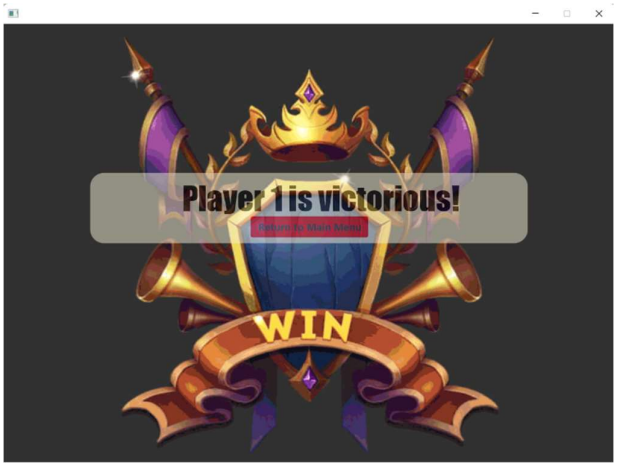

Figure 8: The end screen

Figure 8 shows the end screen indicating the victory of player 1 with amazing images in the background

This is how the game goes on and the players will last until the health bar lasts and the player having more score wins

Feedback:

User 1 provided positive feedback on the game, stating that it was enjoyable and required strategic thinking to determine the most effective abilities to use against opponents and when to switch between them. They also found the user interface easy to use and navigate.

User 2 commented on the appealing animations, and appreciated how the sound effects enhanced the overall experience.

User 3 praised the game's attention to detail, noting that the dialog boxes, animations, and backgrounds were well-designed and contributed to the overall ambiance of the game.

VII. Discussion (Reflection)

Through our project, we were able to develop a game featuring players with a user-friendly interface and enjoyable music. Although we created a traditional-style game, we were able to incorporate characters from a popular fighters that many people appreciate. Developing the game gave us an understanding of how challenging it is to create games, as we had no prior experience with game development or programming logic.

This game is designed for those who enjoy playing strategy games, as each character has a weakness against certain opponents and the player must decide which character to use.

Moving forward, we discovered some areas of improvement for the game, such as implementing a system for unlocking characters to keep the user engaged. Additionally, due to time constraints, we were unable to implement a session storage feature for user score and win data.

If we had more time, we could have improved the game by adding session storage through a database, enabling us to match players of similar skill levels using an Elo rating system. We could have also added more unique characters with varied powers and animations, as well as introducing new game modes such as a 2v2 or a Player vs Enemy mode for multiple players to challenge a common overpowered foe. Overall, we believe this game has the potential to serve as a fun and engaging stress-reliever for students and a means to improve social interaction between friends.

VIII. Conclusions and Future Work

We have developed a simple and lightweight multiplayer game that requires minimal resources and can be enjoyed by anyone. The game was created using a factory model, where music and transitions were stored, making it easy to play and not overly addictive. We believe this game could serve as a fun stress-reliever for students and enhance interaction between friends.

However, one drawback of the game is that all characters are pre-unlocked, which may fail to engage the user. Also, due to time constraints, we were unable to implement session storage to store user scores and win data.

Furthermore, we could have added more characters with unique abilities and animations. To enhance the gaming experience, we could have included a 2v2 mode where two players could team up against other two players and a Player vs Enemy mode where multiple players could play against a common overpowered enemy without any weakness.

IX. Job Assignment

The job assignment for the project is as follows:

- **Mayur Prabhakar Sangale:** Authetication, JDBC Connnectivity, Swap players, Player Damage, HomeScreen
- **Divya Khache:** Background Effects, Event Handling, Media & Sound addition, VFX, Unique Characters, Dialog Box
- **Prasad Deshpande:** Health Bar, Selective Weapons, Animation, Randomized Player

References

[1] Chong-Wei Xu. Java Programming and Game Development HelloWorld, *Learning Java with Games,*2018. 46(1): p. 3-15

[2] JavaFX playing audio - javatpoint. www.javatpoint.com. (n.d.). Retrieved December 10, 2022, from https://www.javatpoint.com/javafx-playing-audio

[3] Eden-Rump, E. (2021, March 12). How to set the JavaFX scene background. Eden Coding. Retrieved December 10, 2022, from https://edencoding.com/scene-background

[4] Transition in JavaFX. Transition (JavaFX 2.2). (2013, December 9). Retrieved December 10, 2022, from https://docs.oracle.com/javafx/2/api/javafx/animation/Transition.html