Age of Pokemon: A Pokemon-themed strategy game from Age of War

Viet Tin Le (1585762) *
Department of Computer Science
Frankfurt University of Applied Sciences
Frankfurt, Germany
viet.le@stud.fra-uas.de

That Nhat Minh Ton (1588341) *
Department of Computer Science
Frankfurt University of Applied Sciences
Frankfurt, Germany
that.ton@stud.fra-uas.de

Tri An Yamashita (1590012) *
Department of Computer Science
Frankfurt University of Applied Sciences
Frankfurt, Germany
tri.yamashita@stud.fra-uas.de

***Abstract*—Age of War is an engaging, fast-paced, and family-friendly board game. However, like many traditional board games, it requires players to be physically present, which limits accessibility. This project aims to expand the reach of the game by developing a digital version using Java, JavaFX, SceneBuilder, and CSS. While preserving the core gameplay mechanics, we enhance the user interface and experience by incorporating a Pokemon theme, a humorous and refreshing twist. To improve the the user experience and overall gaming experience, we integrate detailed manuals, interactive tutorials, adjustable difficulty levels, and visually engaging scenes, including menus, settings, and result screens. Finally, we conduct experiments to evaluate key gameplay aspects, including average playtime, score distribution, and strategic patterns in Age of War. Empirically, we have illustrated two key points: first, that player order does not significantly influence game outcomes, and second, that the new difficulty levels maintain game balance while reducing playtime and diversifying the gameplay experience. The source code is available at https://github.com/tinvietle/aop.git**

I. Introduction

The physical edition of Age of War [1] was originally released in 2014 by Fantasy Flight Games. It is a dice-based strategy game designed for 2 to 6 players, with an average playtime of 15 to 30 minutes. Players compete to conquer castles and control territories by rolling dice that represent different types of troops. Success in the game depends on a combination of strategic planning, resource management, and risk assessment, along with a bit of luck, as players must carefully allocate their forces while hoping for favorable dice rolls. Age of War incorporates all the essential features of a traditional board game—it is accessible, adaptable for multiple players, brief yet engaging. However, like most conventional board games, it has a significant limitation: players must be physically present to play together. In today's digital era, online accessibility is vital, allowing people from different locations to connect virtually and enjoy the experience of board games together. In this project, we aim to create a fully functional digital adaptation of Age of War that stays true to the original gameplay mechanics while introducing a modern, lighthearted twist through Pokemon-themed elements, called Age-of-Pokemon. The game is being developed using Java and JavaFX to create a smooth gameplay, a responsive user interface, and an immersive player experience by implementing a range of game scenes, including a main menu, registration menu, in-game battle screen, results display, help screens, and settings menu. Additionally, we are integrating animations, sound effects, user-friendly controls, and guided walkthrough instructions to improve engagement and accessibility. Alongside an intuitive and visually appealing design, our development approach also puts strong focus on well-structured logic through object-oriented programming principles. Beyond delivering a fun and interactive digital version of Age of War, this project also serves as an opportunity for our team to deepen our knowledge in game development, UX/UI design, and collaborative software engineering, applying best practices to create a polished and enjoyable gaming experience.

*These three authors contributed equally.

II. Problem Description

A. Core game definition

Age-of-Pokemon is a fan-made version of the original game called Age-of-War. This is a quick multiple-player game with 14 cards and 7 dice.

Fig. 1: Original Age of War board game with cards and dice

Each card has a given point, and catching every Pokemon in a group will grant a bonus point. A player can obtain a card by rolling the dice and gets the card from the field or steal it from another player by meeting the card dice requirements.

The game ends when all 14 cards have been captured and the player with the highest scores wins the game.

B. Original Game's Rules

The player starts their turn by rolling all seven dice. They can either start conquering a castle or lose one dice for each reroll. To conquer a castle, the player chooses a face-up castle card and completes only one requirement line by matching its symbols with their dice. They then roll the remaining dice. If they can fill a line, they do not lose a dice for reroll. The turn ends when the player either conquers the castle by completing all combat lines or rolls their last die. The game then proceeds to the next player.

The game ends after all castles have been conquered. Each player then scores points based on the castles they own, plus bonus points if they have a group of castles of the same color. The individual with the most points wins the game.

If there is a tie, the player who has captured the most castles wins. If there is still a tie, the player who has captured the most clans wins. If there is still a tie, the tied players split the winnings.

C. Our adapted game

Age-of-Pokemon is a reimagined of Age-of-War, incorporating elements inspired by Pokemon [3]. In our adaptation, a Pokemon represents a castle, and a group represents a clan

Instead of normal playing cards, the game employs animated GIFs of Pokemon characters to provide a more dynamic and visually attractive experience. Similarly, to keep with the Pokemon motif, dice have been substituted with Pokeballs. The game also uses Pokemon-inspired typefaces and music to immerse players in the themed setting. We also add in a catch Pokemon video when the player successfully meet all the requirements.

Despite these changes, all of Age-of-War's original rules remain in place, ensuring that the essential mechanics and gaming experience are intact.

D. Development Aim

In order to finish this project, we propose to complete these requirements:

- **Obtain Pokemon resources:** find and download the Pokemon GIFs and their corresponding capture videos from online repositories.
- **Design a theme:** develop a cohesive color theme and font to use throughout the game.
- **Implement game rules in Java:** replicate the game rule set by coding the mechanism and logic inside Java.
- **UI Adaptation:** binding the elements of the scene to match the size and ratio of different screen size.
- **Support Scene for How to Play the Game:** create sufficient scenes to guide users on how to play the game, control the music, and adjust the window size. Ensure that these instructions are comprehensive to help users play the game independently.
- **Update every week:** we will meet every week to understand more about the game and assign each member some tasks to finish the game in two months.

III. Related Work

A. Similar Game

The Age of Pokemon takes inspiration from the Age of War, where the roll-off mechanism is unique and creates a balance between strategy and luck. Players must decide which dice to keep after each roll and which to reroll to satisfy the requirement of any battle line on the card, creating a funny gameplay experience. These are similar games with dice-based game designs.

1) King of Tokyo: King of Tokyo [4] is a game where players will play as monsters to occupy the city of Tokyo. These monsters will fight each other, and the monster that can survive last will win. These monsters will gain points and actions through the values of the dice, including attack, recovery, energy, and victory points. We can see that rolling the dice is the primary mechanism in this game. Players can choose which dice to keep and which dice to roll again up to two times, giving players the tactical options they feel are reasonable. The similarity is the ability to balance luck and strategy; players have to think of different strategies based on the results of the dice rolls. However, the game is more complicated than other strategies to preserve their health and the ability to coordinate between different card skills, increasing the depth of the game.

2) Zombie Dice: Zombie Dice [5] has a similar dice-rolling mechanism to that of Age of Pokemon. Players must roll the dice to collect "brains" to avoid shotgun blasts. Players can play each roll safely by keeping their score or risk rolling again to get more valuable rewards. This game has unpredictable moves thanks to the distribution of many dice, challenging players to take risks to get high rewards to win. Compared to Age of Pokemon, this game seems less strategic but is easy to understand and almost entirely based on luck.

3) Yahtzee: Similar to the main lifeblood of the game, Yahtzee [6], lets players roll five dice three times to create combinations to get high scores. Players must decide the most reasonable strategies based on the dice results after each roll to bring victory to themselves. Although the game mechanisms may sound similar, Age of Pokemon has more interesting elements and more vivid images than just using combinations of dice.

B. Inspriration

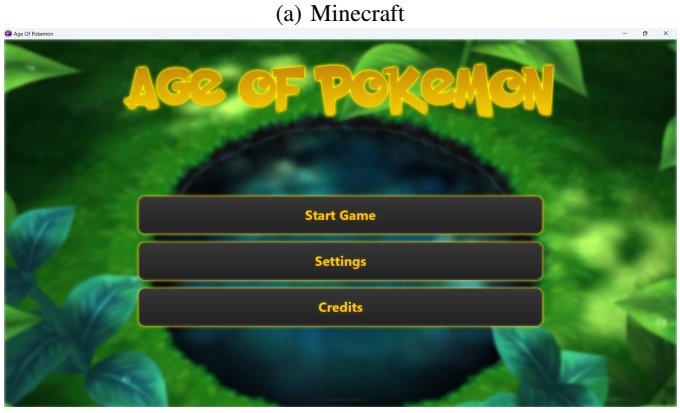
Our team has referred to the designs used by various famous games:

- The images, video, and sounds related to the game are all taken from the internet and related to Pokemon [1]. The information has been carefully selected, making the game look good and attractive to Pokemon fans. The pleasant music and sound effects are all available in famous Pokemon games, especially those of Nintendo and Pokemon Company. Our team has searched for

reputable and legal sources and selected the music many fans love in Pokemon games. In addition, the sounds of the Pokemon are all taken from the YouTube platform; our team has cut the voices of the Pokemon that have been dubbed and brought into the game to increase the livelines. In addition, we choose a font that many Pokemon game publishers also pick, The Pocket Monk.

- Inspired by the launch of the game Balatro [2] in 2024, which has become a hot topic globally with its straightforward but stunning designs and a plethora of awards for Best Independent Game, Best Debut Indie Game, and Best Mobile Game, We have applied their designs to the game guide page in Age of Pokemon. We have applied dark backgrounds with eye-catching patterns in bright colors to create an attractive and easy-to-see, clear and neat layout to help players feel not confused and not complicated.
- Minecraft [7], a game everyone has heard of, has an effortless menu scene design. Put a title on top, then three rectangular buttons, and behind are animations related to the game. This simple yet attractive and friendly look has greatly inspired us in our game's menu scene.

(a) Minecraft



(b) Age of Pokemon

Fig. 2: Comparison of Game Menu Screens between Minecraft and Age of Pokemon

- The classification of sound settings into master volume, background music volume, SFX, and voice is something anime-style games with voiced characters often use, such as Genshin Impact [8] or Fate/Grand Order (FGO) [9].Our team has evaluated this idea and determined that it aligns well with our game.
- Not only do we take inspiration from famous games, but we are also inspired by Korean comic Omniscient Reader Viewpoint [10] with content related to system usage. The system designs are visually striking, and our team used them as a reference to explore suitable color tones and combinations.

C. Game Balance

Like many other board games, the Age of Pokemon employs strategic decision-making to determine the best option. Due to the dice-rolling mechanism, players are forced to choose between taking chances and staying safe. The gameplay is enjoyable, and because it is dependent on luck, it is simple to alter a rule and keep the game balanced.

IV. Teamwork

A. Team Roles and Responsibilities

1) Viet Tin Le–Team Leader, UX/UI Developer:

- As the Team Leader, Tin is responsible for overseeing the overall progress of the project and ensuring that all team members stay on track toward the final goal. He will facilitate team meetings and guide discussions to encourage collaboration and problem-solving. In cases where differing opinions arise, he will act as the moderator, guiding the team toward a consensus and making the final decision when necessary. Additionally, he will manage task assignments and ensure that the workflow remains efficient and well-organized.
- As a UX/UI Developer, Tin will design and develop a visually appealing and user-friendly interface for the game. His role includes crafting intuitive menus, interactive elements, and an engaging layout that enhances the player experience. He will also be responsible for designing clear and accessible onboarding instructions for new users, ensuring a smooth gaming experience.

2) That Nhat Minh Ton-Code Manager, Logic Developer:

- As the Code Manager, Minh is responsible for maintaining the overall structure and quality of the codebase. He ensures that the code remains readable, reusable, and well-organized by structuring the working directory effectively, refactoring repetitive code into maintainable functions and classes following object-oriented programming (OOP) principles. Additionally, he oversees the GitHub repository, managing version control, handling pull requests, resolving merge conflicts, and ensuring smooth collaboration among team members.
- As the Logic Developer, Minh is in charge of translating the game's mechanics and rules into functional Java code. He will implement the game logic, ensuring smooth integration between gameplay mechanics and UI components. His role involves handling game state management, user actions, and scoring systems, ensuring that the game progresses correctly based on player interactions.

3) Yamashita Tri An–UX/UI Developer, Researcher:

- As the UX/UI Developer, An is responsible for designing a seamless and engaging user experience by developing various game scenes that guide players through different stages of the game. His work extends beyond the core gameplay to include essential menus and interfaces, such as the start menu, registration menu, results screen, and settings menu. He carefully crafts both major and minor design elements, ensuring consistency in fonts, sounds, transitions, and visual effects to create an immersive and polished experience.
- As the Researcher, An takes an active role in playtesting the game, identifying potential issues, and providing feedback to enhance gameplay and user interaction. With his extensive experience, he researches and analyzes successful design patterns from other games, drawing inspiration to refine mechanics, improve usability, and introduce creative solutions. His work helps ensure that the game remains engaging, functional, and enjoyable for a diverse audience.

B. Collaboration Approach

In this project, we adopt a Feature-Driven Development (FDD) approach, an agile methodology that emphasizes the development of small, functional features within specific areas of responsibility. The stages of our project include:

1) Data Collection: While it is not an official step in the FDD methodology, data collection is an indispensable phase in any agile project. At this stage, we were introduced to the project objectives but had yet to select a board game to implement or fully understand how to use Java for game development. Therefore, during this phase, each team member will research various board games, both recommended and popular, to create a list of potential games along with proposals for their implementation. Simultaneously, the team will familiarize itself with JavaFX, a Java library used for game development, to gain a clear understanding of the possible outcomes for this project. At the end of this step, we had acquired sufficient knowledge to move forward with the project and decided to implement the Age of War board game.

2) Modeling and Feature List Development: Following FDD principles, we started by developing an overall model which outlines the mechanics of the game, the user interface, and the user experience. We used mind mapping to visualize relationships between different components, ensuring an organized approach to feature development. Based on this model, we created a feature list, breaking the project down into three sets of features, each assigned to a team member according to their expertise and interests. Each member also became the class owner for the classes associated with their assigned feature set, ensuring accountability and consistency in development.

3) Feature Implementation and Iterative Development: Each team member evaluated the complexity and dependencies of their assigned features to determine the optimal implementation order. Since each feature typically requires 1 to 2 weeks for completion, we scheduled weekly or biweekly meetings depending on the team's progress, to synchronize updates, discuss challenges, and adjust feature lists as needed. Between meetings, we maintained seamless communication through detailed documentation, ensuring transparency in ongoing improvements. During meetings, team members presented their progress, outlined remaining issues, and planned subsequent feature implementations. If a feature depends on another team member's work, we discuss the collaboration approach during meetings to ensure smooth integration and efficient progress. Completed features were merged into the main branch through pull requests, with the code manager handling conflicts. This structured, feature-driven workflow ensured steady progress while maintaining flexibility for refinements and additional features when necessary.

V. Proposed Approach

In developing our game, we have chosen to use JavaFX alongside SceneBuilder due to their efficiency in creating interactive and visually appealing applications. JavaFX provides a powerful framework with built-in support for graphics, animations, and event-driven programming, making it well-suited for game development. SceneBuilder further enhances the development process by allowing for intuitive, drag-and-drop UI design, reducing the need for manual coding and providing a strong foundation for frontend development. To further enhance the visual appeal of the game, we apply CSS styling to the FXML-based game scenes. By integrating CSS, we can efficiently manage the appearance of UI elements such as colors, fonts, backgrounds, and animations, ensuring consistency across different screens while simplifying design adjustments. To provide a clearer understanding of our project's code, the following section presents some of our core algorithms using pseudo-code.

A. Visual Elements

1) Binding: To ensure a scalable and visually consistent user experience, proper binding is crucial. This guarantees that all elements in the game scene maintain proportional sizes regardless of the screen dimensions.

```
public void bind_element(element, container){
    /*
    Parameters:
    - element: The UI component to be bound (e.g.,
        ImageView, Pane).
    - container: The parent pane that contains the
        element.

    Output:
    - The element resizes dynamically when the
        container's size changes.
    */
    element.fitWidthProperty().bind(container.
        widthProperty());
    element.fitHeightProperty().bind(container.
        heightProperty());
}
```

The function ensures that when the screen size changes, the container and its child elements adjust proportionally.

2) Scene Switching: As the game expands with multiple scenes to enhance user interaction, efficient scene management becomes vital. When transitioning between scenes, two key aspects must be maintained: smooth visual transitions and the retention of essential data from the previous scene for a seamless return.

```
public void switchScene(double durationIn, double
    durationOut) {
    /*
    Parameters:
    - durationIn: The fade-in duration in seconds.
    - durationOut: The fade-out duration in seconds.

    Output:
    - Saves the current screen state before
        switching.
    - Applies smooth transition effects (fade-out
        and fade-in).
    */

    // Store the current scene data for later
        retrieval.
    savePreviousScene(currentScreen);

    // Apply fade-out effect.
    fadeOut(durationOut);

    // Apply fade-in effect for the new scene.
    fadeIn(durationIn);
}
```

This ensures a smooth transition and preserves game state when switching scenes.

3) Overlay Effect: During gameplay, there are moments when we need to convey additional information to players. Instead of switching to a different screen and then returning, which could disrupt the experience and cause inconvenience, we implement an overlay effect. This approach dims the current game scene while displaying important messages in the same game scene.

```
/*
Parameters:
- welcomeOverlay: The overlay pane displaying dialog
    messages.
- messageContent: The text field that holds the
    dialog message.
*/
public class DialogController {
    private StackPane welcomeOverlay;
    private Text messageContent;

    /*
    Parameters:
    - text: The message to be displayed in the
        dialog.

    Output:
    - Updates the messageContent with the provided
        text.
    */
    public void customizeDialog(String text) {
        messageContent.setText(text);
    }

    /*
    Parameters:
    - visible: A boolean indicating whether the
        overlay should be shown or hidden.

    Output:
    - Displays or hides the overlay based on the
        input.
    */
    public void setVisibility(boolean visible) {
        welcomeOverlay.setVisible(visible);
    }
}
```

This implementation of DialogController allows for seamless integration of in-game messages, such as walkthrough guides for new users. The customizable text ensures that relevant information is provided at different stages of the game.

B. Logic Elements

1) JSON Reader: Many aspects of the game, such as rules and Pokemon attributes (name, capture requirements, group, etc.), are predefined. Instead of hardcoding these data, which are not scalable or maintainable, we use JSON files to store game-related information.

```
public List<Pokemon> readPokemonData(String
    difficulty) {
    /*
    Reads Pokemon data from a JSON file based on the
        game difficulty.

    Parameters:
    - difficulty: A string representing the
        difficulty level (e.g., "easy", "medium", "
        hard").

    Output:
    - A list of Pokemon objects parsed from the JSON
        file.
    */

    // Retrieve the correct file path based on
        difficulty.
    String filePath = getFilePath(difficulty);

    // Parse JSON data into a list of Pokemon
        objects.
    List<Pokemon> pokemonList = JsonMapper.parse(
        filePath, new TypeReference<List<Pokemon>>()
        {});

    return pokemonList;
}
```

This function allows dynamic game configuration by loading Pokemon data from external JSON files.

2) Dice Controller: The dice controller is a fundamental component of the game, responsible for handling rolling mechanics and ensuring valid interactions.

```
public class DiceController {
    /*
    Attributes:
    - dice: A list of dice objects used in the game.
    - totaldice: The total number of dice available
        to the player.
    */

    private List<Dice> dice;
    private int totaldice;

    public void rollButton() {
        // Ensure rolling is a valid action.
```

```
        if (!isRollValid()) return;

        // Retrieve selected Pokeballs.
        List<Pokeball> savedPokeballs =
            getChosenPokeballs();

        // Check if any requirement line is
            satisfied.
        boolean meetsRequirement = savedPokeballs.
            anyMatch(pokeball -> meetsAnyRequirement
            (pokeball));

        // Penalize the player if no requirement is
            met.
        if (!meetsRequirement) {
            totaldice--;
        }

        // Adjust the dice arrangement.
        reorganizedice();

        // Reroll dice that were not locked.
        rerollRemainingdice();
    }

    public void endButton() {
        // Ensure ending is a valid action.
        if (!isEndValid()) return;

        // Retrieve selected Pokeballs.
        List<Pokeball> savedPokeballs =
            getChosenPokeballs();

        // Check if any requirement line is
            satisfied.
        boolean meetsRequirement = savedPokeballs.
            anyMatch(pokeball -> meetsAnyRequirement
            (pokeball));

        // If the player caught a Pokemon, update
            the player and Pokemon data.
        if (meetsRequirement) {
            pokemon.updatePlayer(pokemon);  //
                Update the player's Pokemon.
            player.updatePokemon();  // Update the
                player's data with the caught
                Pokemon.
        }

        // Reset dice for the next round.
        resetdice();

        // Proceed to the next player's turn.
        nextTurn();
    }
}
```

VI. Implementation Details

A. Application Structure

The game is built with Maven to assist with file construction, and compiled by JavaFX version 23 [2]. To prevent code merging issue, we utilize Github to keep track of our work and collaboration. Fig 3 shows the overall working directories in Github where Maven plays a crucial role in organizing folders and defining library versions. It is worth noticing that a `.gitignore` file is used to make sure team members do not accidentally push to Github the game assets that are too heavy, instead, we will store those files on Google Drive and manually manage them.

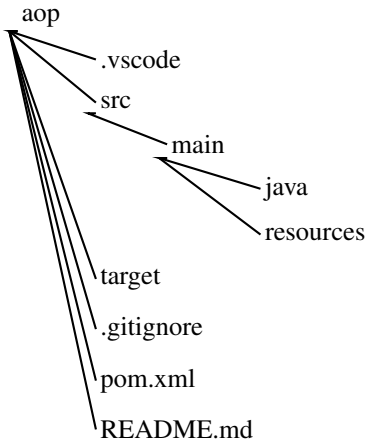


Fig. 3: Overall Project Folder Structure

Fig. 4 illustrates the Java source code structure of the game. The `java\com\example` folder contains subfolders for each scene, where each subfolder includes the Java files that compile the respective scenes. There are a total of seven scenes: a menu scene to open the game, a register scene to know how many players, a game scene to play the game, a capture scene to show the video clip when successfully catching a pokemon, result scene to show the final scores.

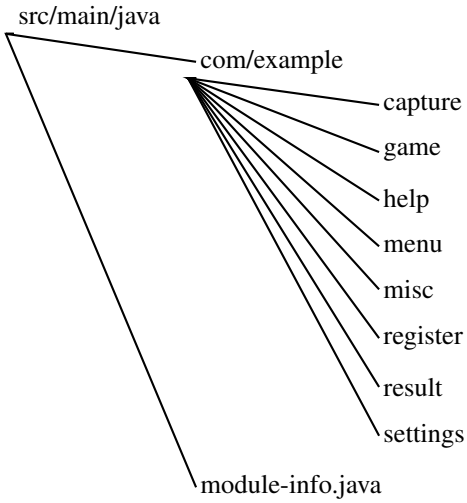


Fig. 4: Java Source Code Folder Structure

Fig. 5 shows the `resources\com\example` directory, which contains subfolders storing `FXML` and `CSS` files. These files are used for the design layout and styling of each scene, so that every scene is consistent throughout the game. The `FXML` file is used to make the layout, put images, scaling with the help of Scene Builder app. The `CSS` is used for decorating objects and each scene have a stylesheet to maintain the appearance.

To enable interaction between Java and `FXML` files, a `module-info.java` file is created. This file defines the required dependencies, permits access to other packages, and exports specific modules as needed.

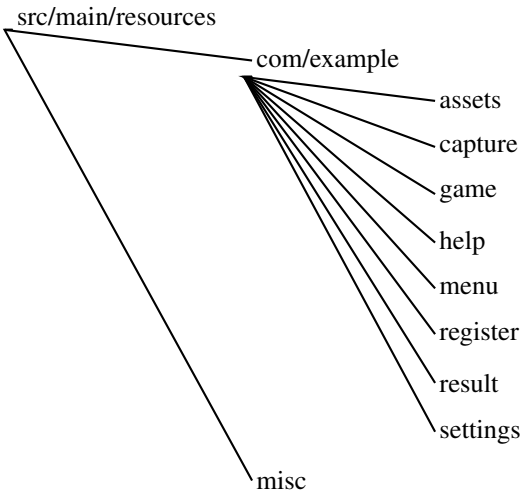


Fig. 5: Resources Folder Structure

The application begins execution from `Main.java` with menu screen as the first screen. Based on the user's selection (e.g., game, settings, credits), the corresponding controller Java file and `FXML` file are loaded dynamically.

B. GUI Details

There are seven scenes in total.

The game opens with a menu screen that has three buttons: one to launch the game, one to change the music or Pokemon sound effects loudness, and one to load the credits, which thank the creators and mentors for their effort. To establish a distinct contrast between the foreground and background, the video backdrop behind the buttons is blurred.

In the register scene, users can enter the number of players, which can range from two to six, and submit a short, unique name for each player. An `HBox` is used in this scenario to vertically center every element.

The game switches to the game scene once the player names have been submitted. A `BorderPane` is utilized in this scene to arrange the layout. Using a `Pane` that permits unrestricted object movement, Pokemon are positioned at random locations. The scene is divided into three areas by the `BorderPane`: the map and Pokemon are shown in the center, controls for rolling the dice are located at the bottom, and a menu bar at the top allows users to access support or reload the help menu. A video clip appears in a pop-up scene when a Pokemon is captured.

The scores and rankings for each player are shown in the result scene when the game is over. This scenario uses a `VBox` with two nested `HBox` components to automatically align the content. Users can launch a new game by clicking on the button in the bottom left.

Every element in a scene's `Pane` is meticulously connected to its parent element. This guarantees that the elements stay in their proper layout locations and dynamically change in size as the user resizes the scene.

Additionally, a specific `CSS` file is used to style each scene. The `Pane` has the CSS connected to it, which allows components with the right tags to apply the proper styling and support dynamic binding effects.

C. Binding Scene

Binding elements to scenes' sizes is essential in this product. Since each person has a different computer configuration, binding to the computer configuration ratio is necessary. Our team did two configurations: the first was according to the maximum ratio of the user's computer; the other was according to the ratio of 800 x 500. We applied the technique to almost every elemens in all the scenes in the game. We divided the scene elements into three categories for binding, including things related to FXML, fonts, and padding. To get the ratio right, we had to adjust each ratio one by one. The example of a binding is in Fig. 6

(a) Game screen of size 1600x1000

(b) Game screen of size 800x500

Fig. 6: Dynamic UI for different screen size

D. Class diagram

Figure 7 illustrates the overall game structure. The GameController serves as the central component, managing the core gameplay, player interactions, and connections with other controller classes. The Player class stores information about player scores, captured Pokemon, and player names. The

Pokemon class defines Pokemon details with the requirements processed in Requirement and Line classes,. The Requirement class specifies all conditions for Pokemon, while the Line class details each specific line of the requirement.

Various controllers, including MenuController, RegisterController, GameController, SettingController, and HelpController, manage different game scenes. These controllers integrate with UI components defined in FXML files and use JavaFX elements such as Pane, VBox, and HBox to organize the interface. SoundManager manages background music, sound effects, and Pokemon sounds, while VideoPlayer handles video loading. The ResultDisplay component displays player rankings. Finally, the App class initiates the game, and the Utils class manages error handling and user warnings in specific scenarios.

Firstly, the whole code utilizes `JavaFX` library to build UI, manage animation, apply decoration and tie the scene to the screen size. The version we use in our code is 23 because this version has been built to fix the scaling issue for different laptop screen size on Window 11. The library includes a number of functions such as `javafx.scene` to define variables of the components of a scene, such as `BorderPane, Button, Alert, Text, HBox, VBox`. These variables are then used in various functions, such as `onClicked`, `hover`, `MouseEvent` and `setOnEndOfMedia`, to trigger interactive behavior or are used to link internal components to external components so that whenever the scene size changes, the components retain their respective positions and sizes. Additionally, we also use the JavaFX library for animating components using the `Transition` package. By giving it the parameters, we can make our game more engaging to users.

The other library that is also important to build the logic of the game is the `java.util`. This library has advanced data structures, such as `ArrayList`, `HashMap`, and `List`, which enables coders to make effective object management and algorithm implementation. Our game has a lot of Pokemon and players; therefore, with the help of this library, we can switch from the traditional style of using string and integer arrays to using dynamic arrays and easier object manipulation.

E. Important Code Snippets

To load the scene, we need to load the FXML which is the UI of the scene. This FXML file is linked to the corresponding Java Controller file which has the defined functions and variables of the components. The loaded FXML is then passed to the function to set it as the new scene or to the controller to use the functions of that controller.

```
public static Parent loadFXML(String fxml) throws
    IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(App.class
        .getResource("/com/example/" + fxml + ".fxml
        "));
    return fxmlLoader.load();
}
```

To make the game more lively, we add background sound to our game by calling Media and MediaPlayer function and setting infinite loop. This function also applies to SFX and Pokemon voice sound

```
public void playBGM(String bgmPath) {
    try {
        Media media = new Media(getClass().
            getResource(bgmPath).toString());
        mediaPlayer = new MediaPlayer(media);
        mediaPlayer.setVolume(volume * masterVolume)
            ;
        mediaPlayer.setCycleCount(MediaPlayer.
            INDEFINITE);

        // Add error handling for media player
        mediaPlayer.setOnError(() -> {
            System.err.println("Media error: " +
                mediaPlayer.getError().getMessage())
                ;
        });

        mediaPlayer.play();
    } catch (Exception e) {
        System.err.println("Error playing BGM: " + e
            .getMessage());
    }
}
```

To store the requirements of the Pokemon from a JSON file, the following code reads the JSON file and stores all Pokemon as an `ArrayList`.

```
public ArrayList<Pokemon> readPokemons() {
    ObjectMapper objectMapper = new ObjectMapper();

    try {
        // Parse JSON into a List of Pokemon objects
        ArrayList<Pokemon> pokemons = objectMapper.
            readValue(
            new File("src\\main\\resources\\com\\
                example\\assets\\stocks\\pokemon.
                json"),
            new TypeReference<ArrayList<Pokemon>>()
                {}
        );

        // Print all Pokemon
        return pokemons;
    } catch (IOException e) {
        e.printStackTrace();
        return new ArrayList<>(); // Return an empty
             list in case of an exception
    }
}
```

To bind the inner components to the outer Pane, we use the `bind, widthProperty, heightProperty` function from `javafx`. This ensures that the game adapts properly to any screen ratio.

```
dicePane.prefWidthProperty().bind(borderPane.
    widthProperty());
dicePane.prefHeightProperty().bind(borderPane.
    heightProperty().multiply(76.0 / 500.0));
```

In order to make the dice rolling animation run simultaneously, we apply `Thread` for code to execute without waiting and `CountDownLatch` to enable the next function only after all the threads are finished.

```
new Thread(() -> {
    rollEachDice(diceImage);
```

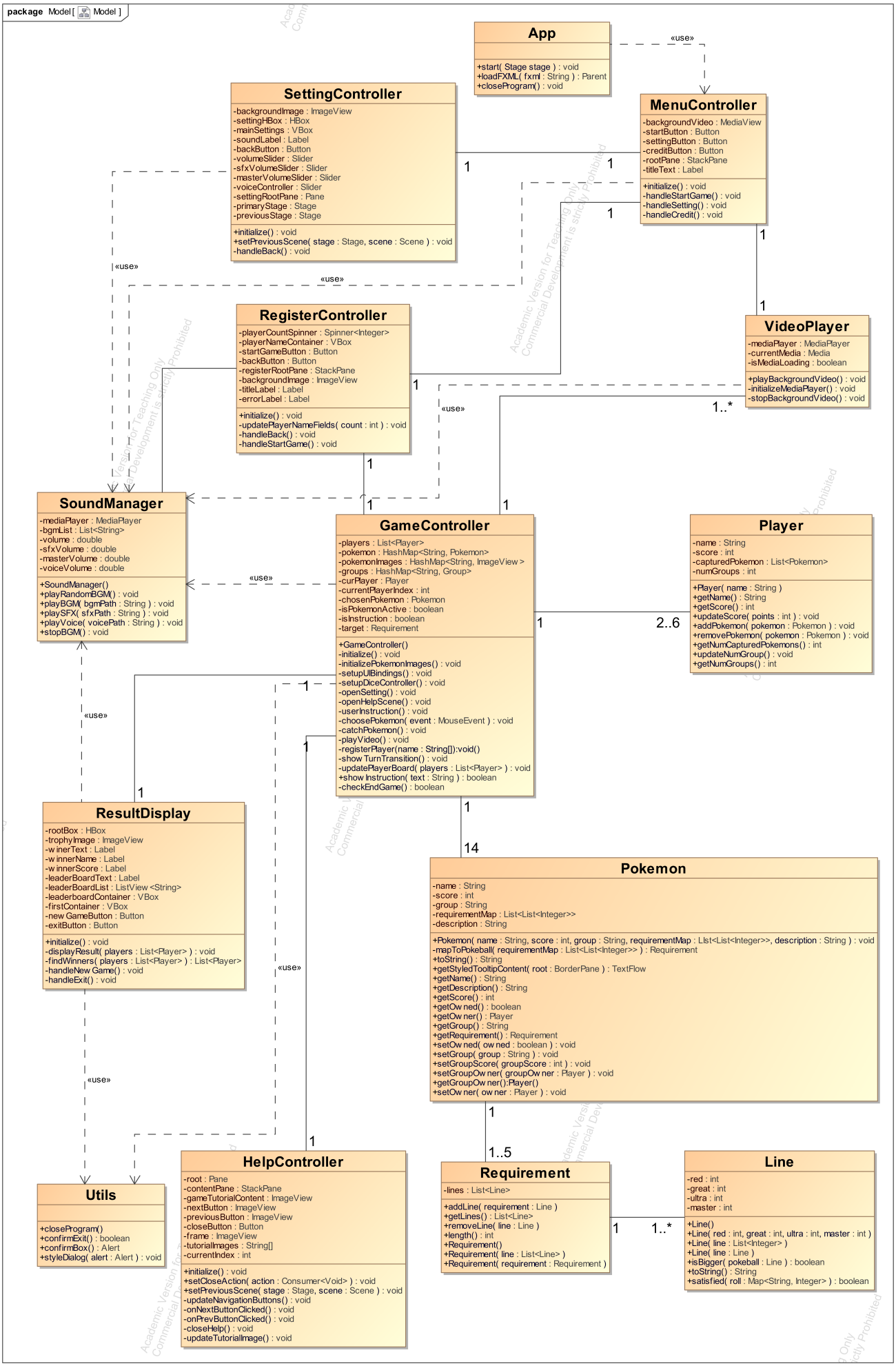


Fig. 7: Class diagram of Age of War game

```
    latch.countDown();
}).start();
```

VII. Experimental Results, Statistical Tests, Running Scenarios

In this section, we present the results of our experiments, focusing on parameter variations within the game and their corresponding statistical data. We conducted a total of nine test sessions, each involving our three team members as participants. For each session, we recorded the game duration, the total number of turns, and the final scores of the players in positions one, two, and three. To minimize bias, we rotated the order of participants across games while maintaining consistent playstyles, aiming to maximize scores by targeting high-value Pokemon. The comprehensive data is detailed in Table I.

A. Player Performance Analysis

In Age of Pokemon, players roll dice and select Pokemon in a predetermined sequence. This turn order may provide the first player with an advantage, as they have the initial opportunity to capture smaller groups of Pokemon more efficiently. Conversely, players in subsequent positions benefit from observing their opponents' strategies, allowing them to adapt their tactics accordingly. To assess fairness, we analyzed the impact of player order on game outcomes. The table II summarizes the results of nine test sessions, with Player 1, Player 2, and Player 3 representing the roll sequence.

Our analysis indicates that player performance is relatively consistent with the mean, median, and standard deviation of the scores that one player can get across different positions, suggesting that turn order has a minimal effect on the final outcomes. This conclusion is further supported by correlation analyses of experimental parameters in Figure 8.

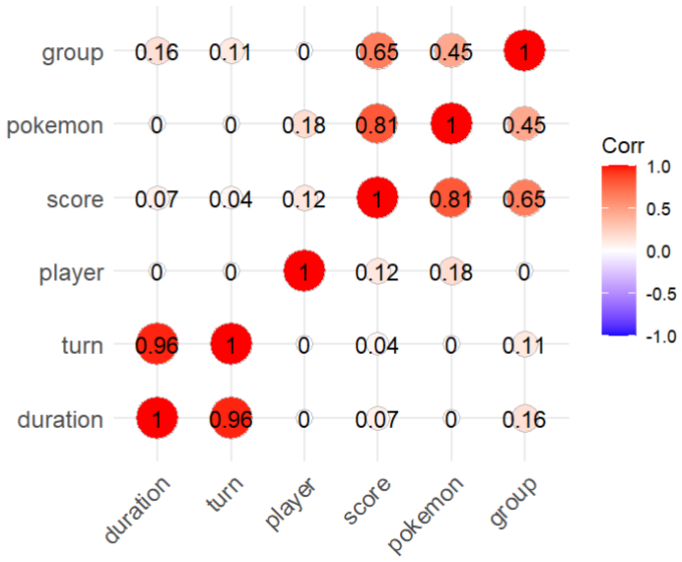


Fig. 8: Heatmap displaying the Correlation between Parameters in the Experiment

The heat map visualization reveals a weak correlation between player position and final scores, Pokemon captured, and group formations. This suggests that the game mechanics are unbiased regarding player order, thus supporting the game's fairness.

B. Impact of Difficulty Levels

To enhance the gaming experience, we introduced two additional difficulty settings—Easy and Normal—alongside the original Hard level. While Age of War is designed for 15–30-minute gameplay, initial test sessions revealed that the Hard level could extend up to 45 minutes due to unfavorable dice rolls and frequent Pokemon thefts among players. Our objective with the new difficulty levels was to determine if simplifying Pokemon capture mechanics could reduce overall game duration. We conducted three test sessions for each difficulty level, evaluating the effects on game duration, number of turns, and player performance. The summarized results are presented in the Table III.

Our findings show that reducing difficulty levels leads to shorter game durations, which correspondingly decrease the number of turns—a relationship confirmed by positive correlation in Figure 8. However, difficulty levels appear to have minimal impact on player performance metrics, such as final scores, Pokemon captured, and group formations. This suggests that the new difficulty settings maintain the balance and fairness of the original game. Additionally, we observed that easier difficulty levels influenced player behavior, promoting more aggressive strategies due to simpler stealing mechanics. Given our limited sample size, further testing is recommended to validate this observation.

Overall, our analysis concludes that the Easy and Normal difficulty levels preserve game balance while offering reduced complexity and shorter playtimes. These modes are ideal for new players to familiarize themselves with the game mechanics or for quick, casual sessions.

VIII. Conclusion and Future Work

A. Teamwork

Teamwork played a pivotal role in the successful development of our project. With clearly defined roles and responsibilities, each member contributed based on their strengths, encouraging a collaborative and efficient environment. Viet Tin Le led the project while developing a user-friendly and visually appealing interface. That Nhat Minh Ton managed the codebase, implemented game logic, and maintained code quality. Yamashita Tri An focused on crafting seamless user experiences and conducting research to refine gameplay mechanics. Adopting a Feature-Driven Development (FDD) approach, the team efficiently managed the project by breaking it down into stages, including data collection, modeling, feature list development, and iterative feature implementation.

Weekly task planning, meetings, and GitHub-based collaboration ensured smooth communication and effective problem-solving. Although the team faced challenges, such as limited experience with Maven on Visual Studio Code, JavaFX, or

Game	Mode	Duration (s)	Turn	P1 Score	P1 Pokemon	P1 Group	P2 Score	P2 Pokemon	P2 Group	P3 Score	P3 Pokemon	P3 Group
1	Easy	780	7	11	5	1	17	6	2	5	3	1
2	Easy	807	8	12	5	2	12	5	1	11	4	2
3	Easy	555	5	8	4	0	11	6	1	12	4	1
4	Normal	1275	13	12	5	2	14	5	1	7	4	1
5	Normal	1004	7	12	4	1	9	4	1	13	6	1
6	Normal	1192	9	8	3	2	12	4	2	18	7	2
7	Hard	1516	15	12	4	2	14	7	2	7	3	0
8	Hard	1506	15	8	4	1	7	3	1	21	7	3
9	Hard	1592	16	13	5	1	10	3	1	11	6	1

TABLE I: Summary of Game Statistics.
P1, P2, P3 denote correspondingly the first, second, and third player in the game.
Score, Pokemon, and Group denote the scores, the number of captured Pokemons,
and the number of captured Groups of each player in one game.

Player	Mean Score	Median Score	SD of Score
P1	10.7	12	2.01
P2	12.0	12	3.22
P3	12.8	12	5.00

TABLE II: Summary of Player Scores: Mean, Median, and Standard Deviation. Each value represents the performance of an individual player.

organizing files when the project grows, we overcame these difficulties by leveraging combined efforts and successfully delivered a fully functional and visually attractive final product.

B. Insights

During the project, we gained valuable experience in game development. Working on turn-based mechanics like dice rolling, adding sound effects and background music, and designing user-friendly scenes with CSS helped us explore new programming techniques and better understand game logic. Debugging challenges also sharpened our technical skills.

Beyond programming, we learned how to collaborate effectively, balance tasks based on each other's strengths, and solve problems quickly and efficiently. Every challenge we faced was met with thoughtful solutions and clear presentations, helping us grow both as developers and as a team.

C. Future Development

1) Customizable Gameplay and Enhanced UI: Since the rule is easy to understand, our future development will introduce a customizable gameplay setting. Players can adjust the rules, add some humorous features in this luck-based game, and develop the game in many unpredictable ways. A similar idea can be seen in a game called Tranquility on a board game website called boardgame-arena [12] displayed in Fig. 9.

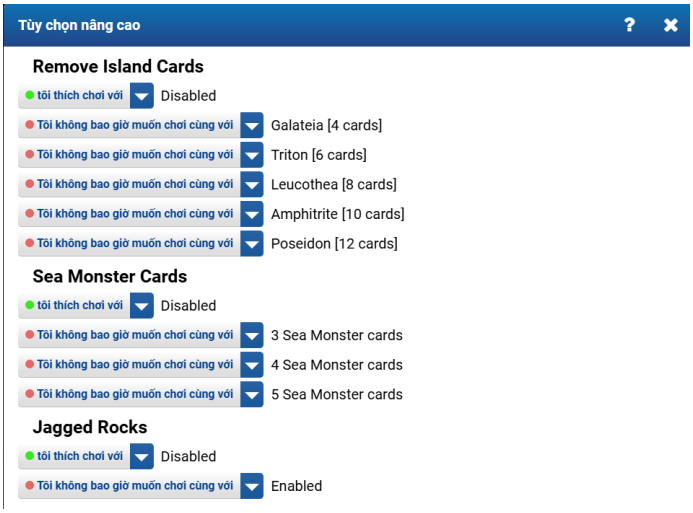


Fig. 9: Adavanced Setting in Tranquility

After adding advanced settings, players can feel new and exciting every time they play. This feature will increase the interaction between players and the game and attract them to play again.

In addition, our team wants to add other themes related to Pokemon and inspired by other games to increase the color and variety of the game, helping players feel most comfortable before entering this game. In addition, we want to upgrade the interactivity and add more thorough instructions to help players understand more clearly about the game.

2) AI Integration and Player Engagement: One of our team's key goals for future development is the implementation of AI-powered bots. Using AI will increase the experience in single-player and multiplayer modes. Our team will also research more achievements and rewards to motivate and

Mode	Mean Duration (s)	Mean Turn	Mean Score	Mean Pokemon	Mean Group
Easy	714.0	6.7	11.0	4.7	1.1
Normal	1157.0	9.7	12.2	4.8	1.6
Hard	1538.0	15.3	11.8	4.8	1.4

TABLE III: Summary of Player Performance by Game Mode

attract players. Our team can take parameters based on player choices to provide analysis guides that players may find useful and can put into use in the future games to increase the strategic nature of Age of Pokemon.

3) Future Development for cooperative mode: Adding team play mode will lead to many changes because players can share resources. Developing abilities for cards to enhance tactical diversity is possible because the game is straightforward and well-balanced. They will have to research combinations of skills to make reasonable decisions and be able to help their team. In addition, players must carefully calculate their actions and bring benefits to teammates instead of personal interests, and victory will depend on the ability to coordinate and analyze opponents to come up with reasonable strategies. This idea stems from the unique gameplay of Unmatched [11], a board game in which players transform into superheroes and form a faction to fight the other faction. Superheroes in Unmatched all have unique skills, and players must coordinate to devise the best strategy to defeat other players.

REFERENCES

[1] "Age of War." Accessed: Feb. 02, 2025. [Online]. Available: https://www.fantasyflightgames.com/en/products/age-of-war/

[2] Wikipedia Contributors, "Balatro," Wikipedia, Jan. 30, 2025.

[3] "Pokemon Legends: Arceus — Official Website." Accessed: Feb. 02, 2025. [Online]. Available: https://legends.arceus.pokemon.com/en-us/

[4] "King of Tokyo," IELLO Games. Accessed: Feb. 02, 2025. [Online]. Available: https://iellogames.com/jeux/king-of-tokyo/

[5] "Zombie Dice." Accessed: Feb. 02, 2025. [Online]. Available: https://www.sjgames.com/dice/zombiedice/

[6] "Yahtzee," Wikipedia. Jan. 19, 2025. Accessed: Feb. 02, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Yahtzee&oldid=1270496404

[7] "Welcome to the official site of Minecraft," Minecraft.net. Accessed: Feb. 02, 2025. [Online]. Available: https://www.minecraft.net/en-us

[8] "Genshin Impact – Step Into a Vast Magical World of Adventure." Accessed: Feb. 02, 2025. [Online]. Available: https://genshin.hoyoverse.com/en/

[9] "Fate/Grand Order," Wikipedia. Jan. 25, 2025. Accessed: Feb. 02, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Fate/Grand_Order&oldid=1271644091

[10] "Omniscient Reader - Episode 3," www.webtoons.com. Accessed: Feb. 02, 2025. [Online]. Available: https://www.webtoons.com/en/action/omniscient-reader/episode-3/viewer?title_no=2154&episode_no=4

[11] "Unmatched Collection - Restoration Games," Restoration Games, Aug. 26, 2024. https://restorationgames.com/unmatched/ (accessed Feb. 03, 2025). [

[12] "Play Tranquility online from your browser," Board Game Arena, 2020. https://boardgamearena.com/gamepanel?game=tranquility (accessed Feb. 03, 2025).