CI601 – THE COMPUTING PROJECT

Project Supervisor: Robin Heath
Second Reader: Saeed Malekshahi Gheytassi

Abstract

This report introduces a Casino-Style Arcade Game Framework for rapid development of web-based arcade games. Built with Node.js and a monorepo architecture, the framework allows developers to create engaging games quickly using limited resources. It features a mock JSON database to support game logic and incorporates tools like GSAP and Pixi.js for enhanced functionality. The effectiveness of this framework is demonstrated through four games: Tricky Cups, Higher or Lower, Bombs Away, and Wheel of Payouts. This report details the framework's design, implementation, and potential for future enhancements.

Kai Nicholas (student)
20816811

Table of Contents

1. Introduction

1.1. Project Goals

The main goal of the Casino-Style Arcade Game Framework was to create a developer-friendly library of components that simplify and increase the efficiency of the development of web-based arcade games. The framework was designed to address the common challenges faced by developers, such as the need for efficient development and asset management. By using a monorepo architecture and implementing modern web technologies like Node.js **(Node.js, 2024)**, the framework aims to provide a strong foundation that makes development as easy and efficient as possible.

1.2. Achievements

The project successfully delivered a fully functional game development framework that significantly reduces the time and challenges associated with creating new arcade games. Key achievements include:

1.2.1. Responsiveness

The framework ensures that games look good and work well on any screen size. It automatically adjusts the game's display during window resizing, and after app initialisation, providing a consistent user experience across different devices.

1.2.2. Prototype Games

As proof of concept, four games were developed using the framework: Tricky Cups, Higher or Lower, Bombs Away, and Wheel of Payouts. Each game demonstrates uses of the framework's capabilities, such as handling screen sizes and resizing, handling user interactions, and processing game outcomes.

1.2.3. Asset Management

Webpack **(Webpack, 2024)** automates the process of asset management. When building a game, Webpack **(Webpack, 2024)** scans the assets folder, identifies all images, and adds them to a generated asset manifest. This manifest is then used by Pixi.js **(Pixi.js, 2024)** to pre-load assets before the game starts, ensuring that all visual elements are available and cached upon game launch.

1.2.4. Technology Integration

The use of technologies like GSAP **(GreenSock, 2024)** for animations and Pixi.js **(Pixi.js, 2024)** for graphics rendering ensures that the games have visual appeal.

2. Methodology

2.1. Overview

The approach to developing the Casino-Style Arcade Game Framework was focused on making the process efficient and manageable. The project used a clear structure and modern web technologies like Node.js **(Node.js, 2024)** to build a reliable framework for game development.

2.2. Project Management

The project was managed using agile methods which helped keep track of progress and made it easier to adjust as needed. Using a monorepo approach helped in managing the project's multiple components and games from one place, aiding in version control and managing games' dependencies on framework components.

2.3. Technologies Used

The Casino-Style Arcade Game Framework utilises several key technologies that are core to its functionality, from server-side operations to graphic rendering and asset management.

2.3.1. TypeScript

Functionality

TypeScript is exclusively used across the framework to add types to JavaScript. This helps improve code quality by catching errors early in development, making the code easier to manage and debug.

Web Development

Web-based games can be accessed on any device with a web browser, including desktops, tablets, and smartphones, without the need for multiple versions of the game. Players do not need to download and install games with the framework.

2.3.2. Node.js

Functionality

Node.js **(Node.js, 2024)** is used for all backend processes in the framework. It runs the server-side scripts efficiently. This was used as the backend over Deno **(Deno, 2024)** due to familiarity and longer lifetime.

Package Management

Node.js **(Node.js, 2024)** comes with Node Package Manager, which is used to install and manage the other core technologies the framework relies on.

2.3.3. Webpack

Functionality

Webpack **(Webpack, 2024)** is responsible for compiling the game's assets and scripts through reference to the game's ID. It prepares and links everything needed for each game to run, including generating an asset manifest for image pre-loading by Pixi.js **(Pixi.js, 2024)**.

Development Support

The framework uses Webpack's dev-server for development, allowing hot reloading. This lets developers see updates in real-time without needing to refresh the page, speeding up the development and testing process.

2.3.4. Pixi.js

Functionality

Pixi.js **(Pixi.js, 2024)** is used for rendering game graphics. It supports fast, efficient rendering across all modern web browsers using WebGL. The framework uses Pixi.js **(Pixi.js, 2024)** for displaying all visuals.

Components Used

The framework imports components from Pixi.js **(Pixi.js, 2024)** which are response for preparing the HTML canvas (the application module), displaying text, and organizing graphics in containers.

2.3.5. GSAP (GreenSock Animation Platform)

Functionality

GSAP **(GreenSock, 2024)** is used for animations within the games. It allows for complex animation sequences that are smooth and visually appealing, enhancing the interactive aspects of game elements.

2.3.6 Krita

Functionality

Krita **(Krita, 2024)** is used to create the visual assets for the game prototypes. It offers advanced digital painting and illustration tools, essential for designing detailed game graphics.

2.4. Development Approach

The design of the framework focused on reusable components. This setup made it easier to adapt and expand the framework as new games were developed.

2.4.1. Prototyping

The project started with creating a prototype game called "Tricky Cups." This first game tested the framework's basic design and setup. It made sure that the foundational features worked well before adding more complex features. The game's and the frameworks' development then continued simultaneously.

2.4.2. Iterative Development

The project was carried out in stages, each involving design, development, and testing of new features or improvements. This step-by-step approach ensured that the framework could grow and adapt through each cycle.

2.5. Testing

Testing was a continuous part of the project, beginning from the start of the project to ensure everything worked as expected.

2.5.1. Continuous Testing

The first game, "Tricky Cups," was used to test the framework from the beginning of its development. This helped catch and fix problems early, which made the development smoother.

2.5.2. Framework Updates

Updates to the framework were tested using all the prototype games made so far. This approach made sure that new changes did not break any existing functionalities and that the framework remained compatible with older versions.

2.5.3. User Testing

The user testing served as a form of blackbox testing, in which the testers had no knowledge of the internals of the prototype games, only the rules of each game. Development builds of the prototype games were deployed publicly for distribution amongst the testers. Web developers and friends played the prototype games to provide insights on how the games felt and performed. Their feedback was necessary for improving the design, especially the user interface and how the games responded during play.

For example, one tester had inspired the implementation of support for multiple payouts, claiming that one payout per game was not engaging enough. This also led to the development of the prototype game, Wheel of Payouts, which showcases this capability of the framework.

2.6. Learning and Adaptation

Adapting based on experience was a key part of the project. Each phase brought new lessons, and adjustments were made to tackle any challenges faced. This ongoing learning helped enhance the project continuously.

3. Product Description

3.1. Overview of the Framework

The Casino-Style Arcade Game Framework is a set of tools designed for quickly creating web-based arcade games. It uses modern web technology to build a strong foundation for developing interactive casino-style arcade games.

3.2. Components of the Framework

The framework includes several key components, each with a specific role. These components work together to provide a complete gaming experience.

3.2.1. App Class (Controller)

Purpose and Functionality

The App class is the main controller for the game. It manages how the game starts, runs, and responds to user actions. It coordinates all parts of the game to make sure they work together smoothly.

Implementation Details

This class sets up necessary parts like the ConnectionModel for retrieving game data and calculating round outcomes, StageView for the display, and specific game controllers and views. It manages game events and connects actions to the right components.

3.2.2. StageView Class (View)

Purpose and Functionality

The StageView class handles how the game looks on the screen. It makes sure that all visual elements of the game adjust to fit different screen sizes and resolutions properly.

Implementation Details

Using Pixi.js **(Pixi.js, 2024)**, this class manages the Pixi.js **(Pixi.js, 2024)** app and stage for the game's graphics. It changes the size and layout of visuals based on the screen to keep the game looking good on any device.

3.2.3. Game Class (Controller)

Purpose and Functionality

The Game class provides a basic structure for all games in the framework. It sets standard methods and properties that all specific games built with the framework must use.

Implementation Details

Different games extend this class to add their own rules and gameplay mechanics. It lays out common functions like setting up the game, entering the game's play round state.

3.2.4 GameView Class (View)

Purpose and Functionality

The GameView class updates what players see during a game, based on the game's changing conditions. It shows animations and player feedback visually.

Implementation Details

It works closely with its Game class to reflect game changes visually. Using Pixi.js **(Pixi.js, 2024)**, it manages animations and displays for game elements.

3.2.5. ConnectionModel Class (Model)

Purpose and Functionality

The ConnectionModel class fetches settings for each game from a database. It makes sure every game starts with the right settings according to the rules stored in the database. It also calculates whether a player has won, and the payout to return.

Implementation Details

It uses a JSON-based mock database **(see Figure 1)** to get data specific to each game, such as rules and payout information. This class uses the RTP, payouts, and weights data to calculate the outcome of a round **(see Figure 2).**

Figure 1


```
{
  "name": {
    "0": "Tricky Cups",
    "1": "Higher or Lower",
    "2": "Bombs Away",
    "3": "Wheel of Payouts"
  },
  "rules": {
    "0": "Where is the ball? Find it and win triple your bet!",
    "1": "Guess if the next card is higher or lower than the last one. Get it right and win double your bet!",
    "2": "When will the bomb explode? Guess right and win 5x your bet!",
    "3": "Spin the wheel and win up to 100x your bet!"
  },
  "rtp": {
    "0": 0.96,
    "1": 0.96,
    "2": 0.96,
    "3": 0.96
  },
  "payout": {
    "0": [3],
    "1": [2],
    "2": [5],
    "3": [1, 2, 4, 5, 10, 20, 40, 50, 100]
  },
  "weight": {
    "0": [1],
    "1": [1],
    "2": [1],
    "3": [50, 40, 30, 20, 10, 5, 2.5, 1.25, 1]
  },
  "setup": {
    "0": {
      "cupAmount": 3
    },
    "1": {
      "values": ["2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"],
      "suits": ["heart", "diamond", "club", "spade"]
    },
    "2": {
      "timeOptions": [1, 2, 3, 4, 5]
    },
    "3": {
      "baseValues": [0, 1, 2, 5, "++", 10, 20, 50],
      "doubleValues": [1, 2, 4, 10, "++", 20, 40, 100]
    }
  }
}
```


Figure 2

```
/**
 * Retrieves the win amount based on the game's RTP and payout options.
 * @type {number}
 */
public get win(): number {
  const randomIndex = this.getWeightedRandom()
  const randomPayout = this.payout[randomIndex]

  return Math.random() < (this.rtp * 1) / randomPayout ? randomPayout : 0
}
```

3.3. Game Flow Overview

This overview explains how the game operates from start to finish within the Casino-Style Arcade Game Framework, outlining each main step in the game lifecycle.

3.3.1. Initialization Process

Starting the App Controller

The game begins with the creation of the App controller. This controller manages the entire game.

Setting Up ConnectionModel

Next, the App controller creates the ConnectionModel, which loads the mock database including default bank amounts and bet levels.

3.3.2. Boot Sequence

Loading Assets

The framework uses Pixi.js **(Pixi.js, 2024)** to load all game assets listed in the asset manifest.

Setting Up the Stage

The Stage controller is created, setting up the StageView. This view initializes the Pixi application for the game's graphics.

Initializing UI

Once the Pixi application is initialised, the UI controller is created and sets up the UIView, which shows the current bank amount and bet level.

Preparing Game and WinText

The Game controller is then created, setting up the GameView. The WinText view is also prepared at this point.

Resizing for the Screen

An initial resize function invocation adjusts all game elements to fit the device screen properly.

3.3.3. Gameplay Interaction

User Input

The App controller waits for the player to start the game.

3.3.4. Handling Play Events

Checking the Bet

When the player starts the game from the UIView, the App controller checks if the bank covers the bet. If not, the game does not proceed.

Calculating Payout

If the bet is covered, the ConnectionModel figures out the payout for the round.

Updating the Bank

The bet amount is deducted from the bank, and the UI is updated with the new total.

Disabling UI

The UI elements are disabled to prevent further actions during the game play.

3.3.5. Game Execution

Activating Game Controller

After disabling the UI, the Game controller's play method is called with the payout amount, allowing the game to show the outcome creatively.

Showing Wins

The WinText view then displays the total win through an animation.

3.3.6. Post-Game Updates

Re-enabling the UI

Once the win animation finishes, the UI shows the updated totals and is enabled again for more gameplay.

This process ensures a seamless game experience from the moment the game starts to when it ends, handling each aspect of the game effectively from the mock backend setup to frontend display.

3.4. System Architecture

The Casino-Style Arcade Game Framework uses a monorepo structure. This means all parts of the project, such as framework components and game components, are kept in one place. This setup makes it easier to manage dependencies by having everything related to the project accessible in one repository. Within this structure, key classes such as App, Game, GameView, and ConnectionModel interact effectively to manage game operations. The App class sets up and controls the game, Game and GameView handle the logic and display, and ConnectionModel fetches and applies game data.

3.4.1. Framework Directory Structure

The framework's directory structure is organized as follows, thought out for ease of access and modularity:

src: This directory contains the source code for the framework and games.

- **database**: Stores the mock JSON database file (database.json) used to store game metadata such as names, rules, RTP (Return to Player) percentages, payouts, and setup configurations.
- **framework**: Contains the core components of the framework.
    - **app**: Includes controllers and views responsible for managing the game logic and user interface.
        - **controller.ts**: Implements the main controller for handling game logic and interactions.
        - **game**: Contains components within the game, such as buttons, text displays, and views.
        - **stage**: Manages the Pixi.js **(Pixi.js, 2024)** application and stage for rendering game elements.
        - **ui**: Handles user interface components like buttons and text displays.
    - **connection**: Contains database-related modules and interfaces.
        - **database**: Defines the interface for interacting with the mock JSON database.
        - **model.ts**: Implements the database model for fetching game metadata.
- **games**: This directory stores individual game modules, each contained within its own subdirectory named after the game's index. Each game directory (0, 1, 2, 3, etc.) contains an app.ts file defining the game's entry point, an assets directory for storing game assets (e.g. images), and game-specific components organized into subdirectories (e.g. ball, cups, table) along with their respective view files.

dist: Contains the bundled JavaScript files, assets, and the index.html file used for running the game.

package.json: Defines project dependencies and scripts for development, building, and documentation generation.

tsconfig.json: Configures TypeScript compiler options.

webpack.config.js: Configures webpack **(Webpack, 2024)** for bundling game modules and assets.

3.5. Design Principles

The development of the framework was guided by several important principles:

3.5.1. Modularity

All components are built to be used in different games without needing changes. This allows developers to add new features or games easily.

3.5.2. Scalability

The framework can handle more or more complex games over time without losing performance.

3.5.3. Ease of Use

The framework is designed to be easy for developers to use and expand, with simple interfaces and clear code documentation.

3.6. Implementation Process

3.6.1. Initial Setup

The project started by setting up the necessary tools and libraries like Node.js **(Node.js, 2024)** and Pixi.js **(Pixi.js, 2024)**. A monorepo was created to organize all project resources effectively.

3.6.2. Development of Components

Components were built step-by-step:

- The App class manages the game from start to finish.
- The Game and GameView classes take care of the game mechanics and how they are shown on screen.
- The ConnectionModel gets and manages settings and rules for the games from a database.
- During development, issues like responsive design and smooth state transitions were addressed through updates to the Game and GameView classes.

3.6.3. Integration and Testing

The components were put together step by step to check for any issues and to make sure they work well together. Tests were done continuously. This included testing each part alone and together with others to ensure the whole system was effective. Feedback from user testing helped improve the player experience.

3.7. Requirements and Specifications

The framework meets these needs:

3.7.1. Performance Criteria

It works smoothly on all supported devices and browsers, loading and running games quickly.

3.7.2. Target Platforms

Designed for modern web browsers to reach a broad audience.

3.7.3. User Interaction Specification

The framework allows for games to be user-friendly, with simple and intuitive controls.

3.8. Prototype Games

Four prototype games were developed which utilise the Casino-Style Arcade Game Framework, each designed to test different aspects of the framework's capabilities. Below are descriptions of four prototype games developed to demonstrate the framework's versatility and functionality.

3.8.1. Tricky Cups

At the start, players see three red cups placed on a table. A ball is briefly shown under the middle cup before all cups are shuffled at a high speed. Players must then guess which cup contains the ball. The probability of choosing correctly is 0.96 in 3. If the player guesses correctly, they win three times their bet.

3.8.2. Higher or Lower

This game begins with two piles of cards, both are face down. The dealer reveals a card from the left pile to the player. If this card is the highest or lowest in the deck, it's returned and replaced. The player then predicts whether the next card from the right pile will be higher or lower than the first card. With a 0.96 in 2 chance of being correct, successful guesses double the player's bet.

3.8.3. Bombs Away

Upon starting, players are presented with a bomb and five buttons representing times from one to five seconds. Players choose when they think the bomb will explode. The player has a 0.96 in 5 chance of making a successful guess, which results in the player winning five times their bet. After making their choice, the bomb's fuse is lit, and the bomb explodes at one of the possible times.

3.8.4. Wheels of Payouts

Players are greeted by a spinning wheel divided into eight sections labelled with different multipliers: 0x, 1x, 2x, 5x, 10x, 20x, 50x, and "++". When the wheel stops, landing on a multiplier applies that multiplier to the player's bet as winnings. Landing on "++" upgrades all values on the wheel for a free, subsequent spin. The probability of landing on a section that awards a payout greater than 0x is about 48%.

4. Research

4.1. Literature Review

This section reviews important books, scholarly articles, and academic materials that have helped to shape the development of game frameworks, focusing especially on those designed for games. The review is divided into three main topics: game design principles, framework development, and user experience in gaming.

4.1.1. Game Design Principles

Effective game design is crucial for creating engaging and interactive games. **"Rules of Play: Game Design Fundamentals" by Katie Salen and Eric Zimmerman (2003)** discusses how to create games that offer meaningful play through well-designed systems, aesthetics, and mechanics. This book has guided the project's approach to understanding how games operate and interact with players.

Jesse Schell's "The Art of Game Design: A Book of Lenses" (2008) provides a method for looking at game design from different perspectives. This approach has helped shape the project's modular design, making it easily adaptable to different types of games.

4.1.2. Framework Development

"Game Engine Architecture" by Jason Gregory (2009) explores the essential architectural components needed to build robust game engines. The insights from this book have influenced the project's use of scalable and flexible systems that can support various games.

Articles in the ACM Digital Library show the advantages of using HTML5 and WebGL in game development. For example**, "Creating a web-based, 2-D action game in JavaScript with HTML5" by Weeks (2014)** explains how HTML5 lets developers create games that can run directly in web browsers without extra plugins. **"An approach to WebGL based distributed virtual environments" by Zhang and Gracanin (2013)** discusses how WebGL improves graphics in games, allowing for more "dynamic and attractive online virtual environment (VE) applications" **Zhang and Gracanin (2013)**. Based on these findings, our project uses Node.js **(Node.js, 2024)** for backend processes and Pixi.js **(Pixi.js, 2024)** for graphics rendering.

4.1.3. User Experience in Games

Don Norman's "The Design of Everyday Things" (2013), while not specific to games, discusses design principles for creating user-friendly interfaces. These principles have been applied to the framework's interface design to improve usability and accessibility.

"Game Usability: Advice from the Experts for Advancing the Player Experience" by Katherine Isbister and Noah Schaffer (2008) focuses on enhancing player interaction in video games. This book's insights into user experience have informed the framework's strategies for usability testing and interface design.

4.2. Competitive Analysis

This section looks at Phaser **(Phaser, 2024)**, an existing product in the market, most like the Casino-Style Arcade Game Framework. Phaser **(Phaser, 2024)** is another framework that uses Pixi.js **(Pixi.js, 2024)**.

4.2.1. Feature Set Comparison

Phaser **(Phaser, 2024)** is known for its extensive features that support a wide variety of game development needs, with a focus on physics and playable characters. While Phaser's **(Phaser, 2024)** features are beneficial for creating complex games, they can be more than what's necessary for simple arcade-style games.

Strengths of Phaser

- Offers detailed tools for physics.
- Has a large community and many learning resources.
- Works across different platforms and devices.

Weaknesses of Phaser

- The complexity of Phaser **(Phaser, 2024)** might be challenging for beginners or those interested in making simpler games.
- Takes more time to learn and integrate into projects.

In contrast, the casino-style arcade framework is designed to be simpler, which is ideal for developers who want to quickly make and launch casino-style arcade games. It simplifies many common development tasks like loading assets, managing game loops, and adjusting for different screen sizes.

4.2.2. Market Position

Phaser **(Phaser, 2024)** holds a strong position in the open-source game development market because of its versatility and comprehensive features. However, its broad functionality may not suit developers who need simpler solutions.

The Casino-Style Arcade Game Framework can attract developers looking for easier tools to create simple arcade games, especially in niche markets like casino-style games.

It is also well-suited for educational purposes, helping newcomers learn game development basics quickly.

4.2.3. Advantages Over Competition

Like the arcade game framework, Phaser **(Phaser, 2024)** uses Pixi.js **(Pixi.js, 2024)** for graphics rendering, which is excellent for fast and flexible visuals. However, Phaser adds many more features, which might not be necessary for all developers.

Phaser **(Phaser, 2024)** includes many advanced features, but this project focuses on performance and ease of use, especially for web games. This makes it more suitable for running games on less powerful devices and simpler platforms.

5. Critical Review

5.1. Achievements

The Casino-Style Arcade Game Framework has successfully simplified the development of web-based arcade games. It has made a positive impact on both developers and players by improving how games are built and played.

5.1.1. Development Efficiency

Description

The framework makes starting new games easier by handling many complex tasks automatically. These include setting up the PIXI application, adjusting to different screen sizes, loading assets, managing game loops, calculating round outcomes, and displaying win messages.

Impact

This automation saves developers time and effort, allowing them to focus more on designing games and enhancing gameplay instead of dealing with repetitive technical setup tasks.

5.1.2. Player Experience

Description

The framework ensures that games adjust well to different device screens, making them accessible and enjoyable on any device.

Impact

Feedback from players has been very positive, particularly about how smoothly the games run and how good they look across different devices.

5.2. Areas for Improvement

Some aspects of the framework could be improved.

5.2.1. Lack of Real Database Communication

Description

The framework uses a mock database, which limits its ability to handle live data updates that a real database would be able to. It's also loaded directly into the ConnectionModel, which would be extremely bad practise if the database contained real sensitive data.

Proposed Enhancements

Adding support for live database interactions would mitigate any security concerns and would also reduce the size of the final game build.

5.2.2. Limited Game States

Description

Currently, the framework only supports basic game states, which restricts the complexity of game scenarios.

Proposed Enhancements

Introducing additional game states could allow developers to create more detailed and varied game experiences.

5.2.3. Audio Management

Description

There is no support for managing audio in the framework, which means games cannot have sound effects or music without manual solutions.

Proposed Enhancements

Developing an audio management component would allow games to include sound, enhancing the player experience.

5.2.4. Integration of MVC Architecture

Description

The framework uses an MVC architecture, but the role and integration of the Game Controller are unclear.

Proposed Enhancements

Better defining and integrating MVC components could make the framework's structure clearer and more effective.

5.2.5. Reusable Components

Description

The framework would benefit from more reusable components that games could easily incorporate.

Proposed Enhancements

Creating a set of common game components for use across different games would help developers build games faster and keep a consistent quality.

5.2.6. Specific Game Types

Description

The framework could offer specialized versions of the base game class for different types of games, which would standardize features common to each genre.

Proposed Enhancements

Developing specific subclasses for genres like puzzles or strategy games could simplify the development process for these types of games.

5.3. Lessons Learned

Developing the Casino-Style Arcade Game Framework provided important insights:

5.3.1. Importance of User Feedback

Insights

User feedback was crucial in guiding the development of the framework. It highlighted areas where the user interface and game interactions needed improvement.

Future Application

In future projects, incorporating user feedback from the start and throughout will be a key focus. This ensures that the games are developed with the player's experience in mind, adapting to their needs and preferences.

5.3.2. Need for Thorough Testing

Insights

The project showed the need for thorough testing, especially when new features are added or when multiple components are integrated. Some initial testing was not enough, leading to issues with scalability and performance.

Future Application

Future development will include a stronger emphasis on early and detailed testing. This will involve both automated and manual tests to catch issues early and fix them before they affect players.

5.4. Future Implications

The experiences from this project set the direction for future enhancements within the framework:

5.4.1. Advancements in Technology Integration

Plans

The framework plans to include newer technologies, particularly for managing real-time data and enhancing interactive designs.

Expected Outcomes

By integrating these advanced technologies, the framework will be better at supporting the development of complex and interactive games. This will make the games more dynamic and responsive to player actions.

5.4.2. Expansion of Game Types

Plans

The framework aims to support a wider variety of games. This includes plans for handling more complex game states and adding audio management features to offer more immersive experiences.

Expected Outcomes

Supporting a broader range of games will help attract more developers and players, expanding the framework's market presence. It will also foster a larger and more active community around the framework.

6. Conclusion

6.2. Summary of Key Points

The main goal of the Casino-Style Arcade Game Framework was to make it easier to create web-based arcade games, especially casino-style games. The framework uses modern web technologies to help developers build engaging games more efficiently. It simplifies many common development tasks, such as managing game loops, adjusting to screen sizes, and loading game assets.

6.3. Achievements and Impact

The project has seen significant achievements. It has made the game development process faster and less resource intensive. This is because the framework handles many basic game development tasks automatically. Games made with this framework run smoothly on different devices, providing a good experience for players. Feedback from those who tested these games has been very positive, particularly about how easy and enjoyable the games are to play. Technological tools like Node.js **(Node.js, 2024)**, Pixi.js **(Pixi.js, 2024)**, and GSAP **(GreenSock, 2024)** have been crucial in achieving these results, ensuring that games are not only functional but also visually appealing.

6.4. Lessons Learned

During the development of the framework, a lot was learned from analysing the competitors and listening to user feedback. For example, while some platforms offer a wide range of features, there is a specific demand for a simpler tool focused on casino-style games. User feedback was essential for improving the framework, providing insight into what works well and what could be better. This has highlighted the importance of designing with the user in mind and continuously testing and updating the product. The technologies chosen worked well, supporting the goals for efficient and high-quality game development.

6.5. Future Directions

In the future, there are several different ways to improve the Casino-Style Arcade Game Framework. Expanding the framework to include more types of games could attract a broader range of developers. Adding real-time database capabilities would allow the framework to support dynamic content and more complex interactions, which would be crucial for games that require real-time updates and multiplayer features. Implementing audio management within the framework could create more immersive experiences for players, with sound effects and music.

6.6. Final Thoughts

The Casino-Style Arcade Game Framework has achieved its goals by contributing to simplifying the process of creating web-based arcade games. By making game development easier and more accessible, the framework serves as a model for future tools aimed at increasing efficiency and engagement in game design. It showcases the power of modern web technologies and user-focused design in producing engaging gaming experiences. The framework could inspire future innovations in the gaming industry, encouraging the development of creative and technically advanced gaming solutions.

7. References

- Deno (2024) Deno Documentation. Available at: https://docs.deno.com/ (Accessed: 25 April 2024).
- Node.js (2024) Node.js Documentation. Available at: https://nodejs.org/en/docs/ (Accessed: 22 April 2024).
- Pixi.js (2024) Pixi.js Documentation. Available at: https://pixijs.com/docs (Accessed: 22 April 2024).
- GreenSock (2024) GSAP Documentation. Available at: https://greensock.com/docs/ (Accessed: 22 April 2024.
- Webpack (2024) Webpack Documentation. Available at: https://webpack.js.org/concepts/ (Accessed: 22 April 2024).
- Krita (2024) Krita Official Documentation. Available at: https://docs.krita.org/en/ (Accessed: 22 April 2024).
- Salen, K, & Zimmerman, E. (2003). Rules of Play: Game Design Fundamentals. MIT Press.
- Schell, J. (2008). The Art of Game Design: A Book of Lenses.
- Gregory, J. (2009). Game Engine Architecture.
- Weeks, M (2014). Creating a web-based, 2-D action game in JavaScript with HTML5. ACM Digital Library.
- Zhang, X, Gracanin, D. (2013). An approach to WebGL based distributed virtual environments. ACM Digital Library.
- Norman, D. (2013). The Design of Everyday Things: Revised and Expanded Edition.
- Isbister, K, & Schaffer, N. (2008). Game Usability: Advice from the Experts for Advancing the Player Experience.
- Phaser (2024) Phaser Documentation. Available at: https://phaser.io/docs/ (Accessed: 23 April 2024).

8. Appendices

8.1. Appendix A

8.1.1. 01/12/2023

Discussed general idea for project and demonstrated a very early prototype. Received advice on the upcoming interim report.

8.1.2. 07/03/2024

Discussed concerns on current project structure. It was decided that the server-side framework used was unnecessary and a more lightweight solution would be sufficient.

8.1.3. 26/03/2024

Demonstrated latest developments of the project, such as all the prototype games made: Tricky Cups, Higher or Lower, Bombs Away, and Wheel of Payouts. Discussed the meaningfulness of the data in the mock database, how the wins are calculated, how the weights worked, and the new structure of the framework.

8.1.4. 25/04/2024

Received additional advice on report based on current progress.

8.2. Appendix B

- YouTube demonstration: https://www.youtube.com/watch?v=4GjNcQBPV2E
- GitHub repository: https://github.com/KaiKaiKaiKaiKaiKaiKai/arcade-game-framework
- Code documentation: https://kn215.brighton.domains/arcade/docs
- Documentation word count: roughly five thousand (including referenced code)
- Pre-built games: https://kn215.brighton.domains/arcade/games