CI601 – THE COMPUTING PROJECT

Project Supervisor: Robin Heath

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

Introduction

The Casino-Style Arcade Game Framework takes an original approach to creating web-based arcade games by offering a user-friendly and efficient platform. This innovative framework is built on a monorepo system, encompassing all necessary infrastructure for game development along with a small suite of pre-built games. The introduction provides a thorough overview of the project, exploring its inception, the motivations behind it, and its structured methodology.

Project Overview

This framework introduces a streamlined development experience for casino-style arcade web games, focusing on backend simplicity and handling all the standard features needed in every game, such as data fetching per round and UI responsiveness on various devices. It provides a rich yet simple development environment where games can be built and identified through a unique game ID.

Report Structure Overview

This report will explore the various stages of the project from its conceptualization to its technical execution and evaluation. Each section is designed to give a detailed look at different sections of the project, revealing the motivations, methods employed, challenges faced, and the results achieved. This will include an in-depth analysis of the framework's architecture, the integration of games, and the outcomes of the testing and evaluation phases.

Aims and Objectives

Primary Aim

The primary goal of the Casino-Style Arcade Game Framework is to simplify the development process for creating engaging casino-style arcade web games. This framework is designed to provide developers with a streamlined platform that effectively manages common game functionalities. These include fetching data, designing user interfaces, optimizing responsive layouts, and managing assets, all through referencing a unique game ID for each game.

Specific Objectives

The project targets the creation of four distinct games as a demonstration of the framework's capability to support diverse game mechanics and features efficiently. These showcase games will highlight the adaptability of the framework to different types of gameplay and its potential to serve as a robust foundation for further game development.

Target Audience

The intended users of the Casino-Style Arcade Game Framework span a broad spectrum, from novice developers seeking to learn game development to experienced professionals looking for an efficient way to produce games. The framework is also geared towards educational environments where students can experiment with game development without the steep learning curve associated with more complex systems.

Functionality

At its core, the framework simplifies the development process by abstracting the complexities typically involved in game development. It automates essential tasks such as data management, UI configuration, and responsive design. This abstraction allows developers to concentrate on creating compelling game content and engaging user experiences.

Scalability

Designed with scalability in mind, the framework is built to accommodate expansion. Future enhancements might include releasing it as a node package, which would facilitate easy updates and the integration of additional features or games, ensuring the framework remains adaptable and up-to-date.

Evaluation Criteria

The success of the project will be measured by the framework's effectiveness in enabling rapid and efficient game development. Evaluation will focus on the ease of use, the flexibility of the framework's tools, and user satisfaction with the development process. These factors are essential in determining the impact and practicality of the framework in real-world game development scenarios.

Deliverables

Overview of Components and Features

The final output of the Casino-Style Arcade Game Framework includes a comprehensive suite of components that streamline the process of creating casino-style arcade games. Each component is integral to simplifying different aspects of game development, ranging from data management to user interaction and the visual presentation of the games.

App Controller

Following the Model-View-Controller (MVC) architecture principles outlined by **Reenskaug (1979)**, the App Controller acts as the core of each game's operations. It manages the foundational activities such as loading assets, setting up the game views (Stage, UI, and Game), and the game loop. It also adjusts the game's views to accommodate different screen sizes, ensuring a seamless experience across all devices.

Connection Model

Serving as the link between the game and its metadata, the Connection Model provides developers with easy access to essential game information such as rules, payouts, and setup configuration. These would typically reside on a remote server, but in this framework, they are incorporated within the local repository.

User Interface (UI)

The User Interface component is standardized across games, offering a consistent look and feel while displaying essential information like game rules, the player's balance, and betting options. It also includes interactive controls for playing rounds and adjusting bets, tailored to ensure usability across both mobile and desktop platforms.

Game View

This component renders the visual elements of the game, such as graphics and animations, based on the data provided by the Connection Model. It is designed to be extendable, allowing developers to customize the visual experience for each game without altering the underlying gameplay mechanics.

Stage View

The Stage View manages the overall layout of the game elements within the framework's rendering environment, using PixiJS **(PixiJS, 2024)**. It ensures that all visual components are correctly scaled and positioned on various display sizes, maintaining graphic quality and interaction consistency across different devices.

Functionalities of the Framework

The Casino-Style Arcade Game Framework provides a robust array of features designed to streamline the development of web-based casino-style games. These functionalities not only simplify the technical process but also enhance the overall user experience by focusing on ease of use and efficient game performance.

Efficient Game Development

Following the findings of **Salen and Zimmerman (2004)** on game design and mechanics, the framework abstracts common development tasks, which allows developers to focus on creating unique gameplay and engaging visuals. Key aspects like data fetching, UI management, and game logic are handled systematically, reducing the repetitive coding effort and speeding up the development cycle.

Modular Architecture

As detailed by **Shaw and Garlan (1996)** regarding software architecture, the inherent modularity of the framework promotes code reusability and makes it easy to add new features or games without modifying existing code. This design supports a scalable development environment where components can be independently updated or replaced as needed.

Optimized Performance

As discussed by **Parisi (2012)** in his exploration of 3D rendering with WebGL, using technologies like Pixi.js **(PixiJS, 2024)** for rendering and GSAP **(GreenSock, 2024)** for animations ensures that the games not only look appealing but also run smoothly. The framework optimizes the rendering of graphics and animations, ensuring high performance across different devices and browsers.

Resource Management: Intelligent management of graphical resources ensures that games load quickly and perform well, even on devices with limited capabilities.

Developer Freedom

While the framework provides a structured approach to game development, it also offers flexibility, allowing developers to customize and extend the games according to their creative visions. Developers can experiment with different game mechanics and visual styles within the provided structure.

Project Planning

Project planning was crucial in successfully developing the Casino-Style Arcade Game Framework. This section outlines the methodical steps taken from the conception of the idea through to the iterative development, highlighting the planning tools and methodologies that were instrumental in guiding the project.

Development Process

The development journey began with a clear identification of the scope, objectives, and technical requirements needed to build the framework. Establishing these early on was vital for getting the project started.

Initial Conceptualization

The project began with research and brainstorming to explore potential game mechanics, visual designs, and the overall technical architecture. This stage was essential in forming a robust foundation for all subsequent design and development efforts.

Prototyping

Prototyping was a critical early step that allowed for the testing and validation of initial concepts and ideas. These prototypes provided practical insights into what worked well and what needed refinement, shaping the direction of the framework development.

Iterative Development

The project took a flexible, iterative approach to development, allowing for continual refinement and enhancement of the framework. Regular evaluation of progress and functionality helped to improve features and ensure the framework met the design goals.

Methodology

An agile methodology, as adapted by **Cohen and others (2004)**, was tailored to suit a single-developer project, ensuring flexibility and responsiveness to changes as the project evolved. This approach supported a dynamic development environment where modifications could be implemented based on ongoing assessments and self-reviews.

Agile Principles

Following **Beck and Andres' (2004)** principles of Agile Software Development, adopting agile practices aided a streamlined development process, where quick iterations and regular assessments ensured the project remained on track and aligned with planned objectives.

Project Management Tools

Effective task management was achieved using digital tools designed to organize and prioritize the workflow, crucial for maintaining project momentum and meeting deadlines.

Trello

A personal Kanban board was implemented using Trello, which helped visualize the workflow and manage tasks efficiently. This tool was pivotal in organizing tasks into categories such as 'Backlog', 'In Progress', and 'Testing', which provided a clear and manageable view of the project's progress at any moment.

Task Management

Prioritizing tasks based on urgency and importance allowed for efficient management of the development process. This approach ensured that key components were developed and tested in a logical order, optimizing the use of time and resources.

Self-Review and Testing

Regular self-reviews and rigorous testing were integral to the project, providing opportunities for self-evaluation and refinement of the framework.

Self-Reviews

Periodic reviews of the project's progress against objectives allowed for adjustments and refinements, ensuring the development stayed aligned with the initial goals.

Testing Phases

Comprehensive testing phases, including unit and integration tests, were conducted to ensure the functionality and stability of the framework. These tests were crucial for identifying and resolving issues before the final deployment.

Technical Details

This section provides an overview of the technologies and tools employed in the development of the Casino-Style Arcade Game Framework. It highlights how each technology contributes to the functionality and performance of the games, ensuring an efficient and robust gaming experience.

Technologies Used

The framework utilizes a combination of modern technologies that are well-suited for developing interactive and high-performance web-based games.

Node.js

As the runtime environment, Node.js **(Node.js, 2024)** facilitates server-side scripting and manages the game's backend operations efficiently. It plays a crucial role in the development of scalable network applications.

Pixi.js

This is a powerful 2D WebGL renderer that allows for high-quality graphic rendering. Pixi.js **(PixiJS, 2024)** is essential for creating visually appealing and dynamic graphical content in games.

GSAP (GreenSock Animation Platform)

GSAP **(GreenSock, 2024)** is used for creating smooth, high-performance animations within the games. Its robust features enhance the visual appeal and interactive elements of the game experience.

Krita

Employed for asset creation, Krita **(Krita, 2024)** supports the design of detailed and high-quality game graphics such as sprites and backgrounds.

Development Environment Setup

Setting up a conducive development environment is crucial for efficient workflow and productivity. The following steps outline the environment setup:

Installation of Node.js

Installing Node.js **(Node.js, 2024)** sets the foundation for using JavaScript on the server side and managing dependencies through npm (Node Package Manager).

Dependency Installation

Using npm, all necessary dependencies specified within the package.json file are installed. This ensures that all tools and libraries are available for the development process.

Webpack Configuration

Configuring Webpack **(Webpack, 2024)** helps in bundling JavaScript files and managing assets, which is vital for optimizing the load time and performance of the games.

Development Server Setup

A development server is set up using webpack-dev-server **(Webpack, 2024)**, which facilitates hot reloading. This feature speeds up the development process by automatically reloading the game when any changes are made to the code.

The integration of these technologies and tools ensures that the Casino-Style Arcade Game Framework not only supports the development of engaging and responsive games but also provides a flexible and powerful development environment for game creators.

Framework Architecture

The architecture of the Casino-Style Arcade Game Framework is designed to be flexible and modular, supporting seamless development and integration of various arcade-style games.

Monorepo Structure

The framework is organized within a single, unified codebase — a monorepo — that encapsulates both the infrastructure for game development and the games themselves. This setup streamlines version control and dependency management:

Source Code Organization

All source code, including game logic, user interface components, and asset files, is centrally located, making it easier to maintain and update.

Database Integration

A mock JSON database is used to simulate game data interactions, allowing developers to implement and test game functionalities without needing an external database connection.

Detailed Directory Structure

The framework's directory structure is organized as follows to promote ease of access and modularity:

- **src**: This directory contains the source code for the framework and games.
    - **database**: Holds 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**: Houses components within the game, such as buttons, text displays, and views.
            - **stage**: Manages the PixiJS **(PixiJS, 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.

Integration and Modularity

The framework's modular design supports easy integration of new games and features:

Extensibility

Developers can independently develop and seamlessly integrate new game modules or features into the framework.

Scalability

The modular structure allows the framework to scale up, accommodating more complex games or additional features as needed.

This architecture ensures that the Casino-Style Arcade Game Framework is not only capable of supporting a diverse range of game types and styles but also adaptable to future advancements in game development technologies and methodologies, providing a robust foundation for developers to build engaging, high-quality arcade games efficiently.

Games Overview

The Casino-Style Arcade Game Framework includes four distinct games, each designed to showcase the framework's capabilities and the diversity of gameplay it supports. This section provides a brief overview of each game and explains how outcomes are predetermined by the system upon a player placing a bet.

Tricky Cups

Description

Players attempt to track a ball shuffled between several cups. The objective is to identify the cup hiding the ball after shuffling.

Game Mechanics

Upon placing a bet, the outcome (win or lose) is immediately determined by the system. The cups then shuffle, and the ball is placed under the cup where the player guessed (if they are predetermined to win).

Rules

Players must select the cup they believe contains the ball; a correct guess, as determined at the bet placement, wins triple the bet amount.

Higher or Lower

Description

This game challenges players to predict whether the next card drawn will be higher or lower than the current card.

Game Mechanics

The outcome is decided the moment the bet is placed. Players see the first card and choose higher or lower. The next card shown will support the predetermined win or loss.

Rules

The game uses a standard deck of cards. If the player's prediction aligns with the predetermined outcome, they double their bet; otherwise, they lose their wager.

Bombs Away

Description

Players guess the time at which a bomb will explode to win a payout.

Game Mechanics

Once a bet is made, whether the player wins or loses is pre-established by the system, and this dictates whether the player’s chosen time is accurate or not. The player selects a time, and if they are predetermined to win, the bomb explodes after their selected amount of time has passed.

Rules

Correct guess yields a payout of five times the bet amount, highlighting the luck-based nature of the game.

Wheel of Payouts

Description

Players spin a wheel to land on various payout segments.

Game Mechanics

The segment on which the wheel will stop is predetermined upon bet placement. The spin is visually appealing, but the outcome is already fixed.

Rules

The wheel contains segments with different payout multipliers. Landing on any segment other than zero, as pre-determined, results in a win according to the multiplier.

Connection Model and Game Outcomes

The Connection Model plays a critical role in managing the outcomes of each game. It retrieves game data, rules, RTP (Return to Player) percentages, payout structures, and other essential parameters from the mock JSON database (see Figure 1). The model ensures that the outcomes, although displayed through engaging and interactive gameplay, are predetermined once a player places their bet, maintaining the integrity and fairness of the gaming experience (see Figure 2).

Pre-determined Outcomes

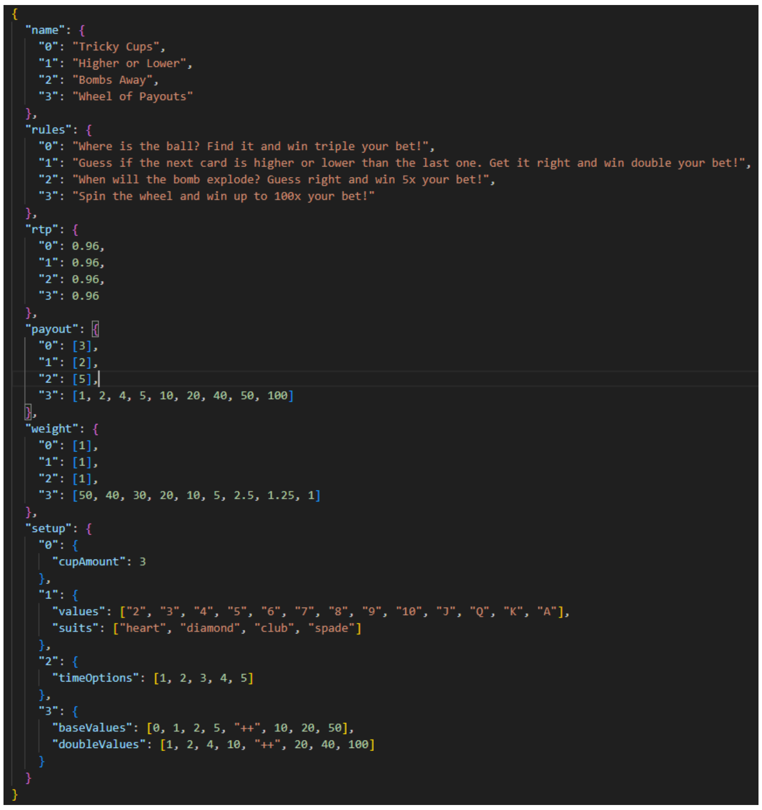
Upon bet placement, the Connection Model immediately determines the result based on the game's RTP, payout, and payout weights. This ensures that while the gameplay appears random, the outcomes are consistent with the expected game probabilities.

Integration with Game Mechanics

The Connection Model seamlessly integrates these outcomes with the game mechanics, ensuring that the visual and interactive elements of the game reflect the predetermined results accurately.

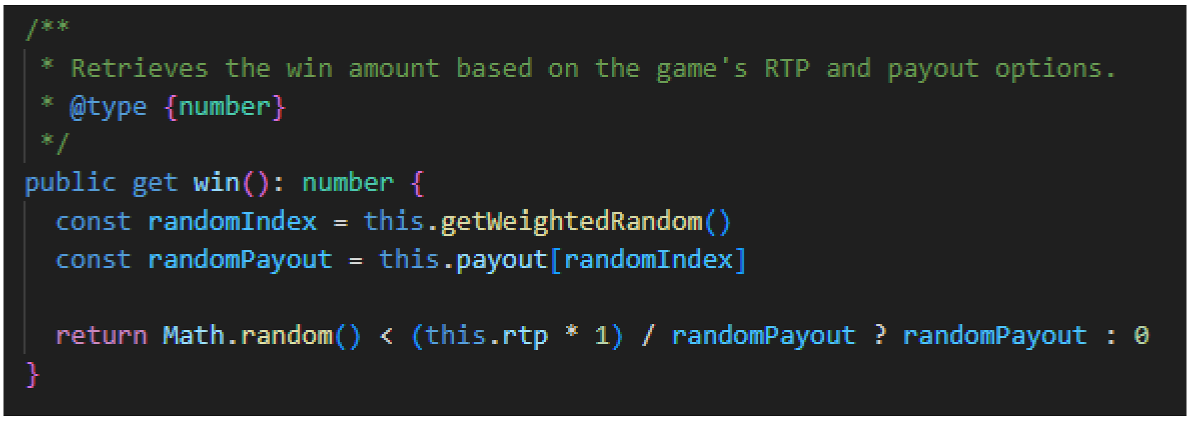
This structure and methodology ensure that players have an engaging yet fair gaming experience, with the excitement of the gameplay enhanced by the visual and interactive sophistication of each game, despite the outcomes being pre-established by the system.

Figure 1



```json
{
  "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
}
```

Technical Implementation

The technical implementation of the Casino-Style Arcade Game Framework involves a structured approach that uses the capabilities of modern web technologies and the architectural features of the framework. This section outlines how the framework was implemented, highlighting the development processes, setup, and integration techniques used.

Framework Utilization

The framework provides a solid foundation for game development by offering essential components such as the App Controller, Stage View, and Game View. These components simplify the complexity involved in game development, allowing for more focus on game-specific logic and user interaction.

App Controller

Manages the initialization of all other components, managing the game loop, and handling resizing events to ensure the game displays correctly across different devices.

Stage View

Manages the PixiJS **(PixiJS, 2024)** application and stage, crucial for rendering game elements efficiently and managing their layout on various screen sizes.

Game View

Implements the visual aspect of the games, where each game's specific visuals are rendered according to the game logic and user interactions.

Development Process

The development process within the framework involved a modular approach, allowing individual game components to be developed and tested independently before integration.

Setup and Configuration

Each game within the framework has its own entry point, facilitating modular development and testing. The Webpack **(Webpack, 2024)** configuration is crucial here, ensuring that assets are bundled efficiently, and that the development environment supports hot reloading.

Game Logic Implementation

Using TypeScript, game logic is implemented in a structured manner. This setup enhances maintainability and scalability by enforcing type safety and clear coding practices.

Connection Model Integration

The Connection Model is integral to the framework, handling the retrieval of game data and ensuring that game outcomes are predetermined yet appear to be generated by player input.

Data Fetching and Processing

The model fetches game rules, RTP percentages, and payout structures from the mock JSON database. It then processes this data to determine the outcomes of each game round, aligning with the pre-established probabilities.

Integration with Game Mechanics

This integration ensures that the gameplay mechanics are consistent with the predetermined outcomes, providing a seamless user experience where the visual elements of the game accurately reflect the results.

Challenges and Solutions

During development, several challenges were encountered and subsequently addressed to enhance the framework's functionality and user experience.

Asset Management

Managing multiple game assets efficiently was a challenge, addressed by implementing a structured asset management system and leveraging Webpack **(Webpack, 2024)** for optimizing asset delivery.

Game Balancing

Ensuring that the games were both enjoyable and fair required iterative testing and adjustments. This process involved fine-tuning the game mechanics and payout structures to achieve the desired balance.

Cross-browser Compatibility

To ensure that the games performed consistently across different browsers, extensive compatibility testing was conducted. Issues were resolved by optimizing the code.

This structured technical implementation not only ensures that the framework operates efficiently but also that it remains adaptable and easy to enhance with future technologies or additional games.

Testing

Testing was a critical phase in the development of the Casino-Style Arcade Game Framework, ensuring that both the framework and the individual games performed as expected under various conditions. This section details the comprehensive testing approach employed, including unit testing, integration testing, and user testing, to validate the functionality and user experience of the games.

Testing Approach

The testing strategy employed multiple techniques, designed to cover all aspects of the game framework from individual components to the full user experience.

Unit Testing

In line with best practices for software testing described by **Myers and others. (2012)**, focused on the smallest parts of the application, unit tests were written for individual functions and components. This ensured that each element performed its designated functions correctly and errors could be isolated quickly.

Integration Testing

After unit testing, integration tests were conducted to ensure that different parts of the application worked together seamlessly. This was crucial for verifying the interactions between the Connection Model, Game Views, and Controller logic.

User Testing

User testing provided direct feedback on the gameplay experience and interface design. A diverse group of users played the games and provided insights into the intuitiveness of the interface and the enjoyment of the game.

User Testing and Feedback

Engagement and Enjoyment

Feedback from users was vital in assessing the engagement level and enjoyment of the games. It helped refine game dynamics and UI elements to enhance player satisfaction.

Performance and Responsiveness

User feedback also included performance aspects such as loading times, game responsiveness, and graphical smoothness across various devices and browsers.

Self-Testing Procedures and Outcomes

As the sole developer, conducting self-testing was essential to maintaining a high standard of quality before wider user testing phases.

Functional Testing

Comprehensive tests were performed to validate all game functionalities, including the correct implementation of game rules, payout mechanisms, and the user interface.

Cross-Browser Compatibility

Games were tested across multiple browsers to ensure consistent performance and functionality, addressing any compatibility issues found.

Performance Optimization

Performance tests identified bottlenecks in the game's operation. Optimizations were then applied to enhance the efficiency of game rendering and logic processing.

Testing Outcomes

The structured approach to testing ensured that the framework and games were robust, performant, and provided a good user experience. Issues identified during testing were systematically addressed, leading to improvements in the framework's stability and the overall quality of the games.

Evaluation

The evaluation of the Casino-Style Arcade Game Framework focused on assessing its performance, effectiveness, and the overall user experience it provides. This section reflects on the framework's strengths, areas for improvement, and the impact of the project management process on the final outcomes.

Framework Performance and Effectiveness

The framework was designed to provide a robust and scalable solution for developing web-based arcade games, and its performance was measured against several key criteria:

Modular Architecture

One of the framework's major strengths is its modular design, which facilitates easy expansion and customization. This architecture allowed for rapid development of new games and features, significantly reducing the time and effort required.

Integration with External Libraries

Utilizing well-supported external libraries like Pixi.js **(PixiJS, 2024)** and GSAP **(GreenSock, 2024)** enhanced the framework's capabilities, allowing for the creation of visually appealing and dynamic games.

TypeScript Integration

The use of TypeScript brought improved maintainability and scalability to the project, helping prevent common bugs and enhancing developer productivity.

Identified Weaknesses

Despite the successes, the evaluation also highlighted areas needing improvement:

Limited Database Support

The current implementation uses a mock JSON database, which limits the ability to simulate more complex game dynamics and player interactions that a real database could offer.

Lack of Multiple Game States

The framework currently does not support multiple game states or levels, which could restrict its use in developing more complex games that require such features.

Project Management Evaluation

The project management approach was largely effective, but there were notable areas for enhancement:

Testing and Quality Assurance

While adequate, the testing processes could be expanded to include more automated testing to catch issues earlier in the development cycle.

Documentation and Knowledge Sharing

Improving documentation and establishing a method for sharing knowledge and best practices could further enhance the framework's usability and adoption.

Future Directions

Based on the evaluation, several potential enhancements have been identified to improve the framework's functionality and user experience:

Database Integration

Integrating a real database would enable the development of games with dynamic content and enhance the overall game experience.

Advanced Game State Management

Adding support for multiple game states would allow developers to create more complex and engaging games.

Performance Optimization

Continuous focus on optimizing performance would ensure smoother gameplay and enhance responsiveness across devices.

Conclusion

The evaluation concludes that the Casino-Style Arcade Game Framework meets many of the initial objectives, providing a solid foundation for building arcade-style games. While it succeeds in modularity and ease of use, the path forward includes strengthening database capabilities and expanding the framework's feature set to support a wider range of game types.

Future Work

The evaluation of the Casino-Style Arcade Game Framework has highlighted several areas for future development that could enhance its capabilities and extend its applicability. This section outlines potential enhancements and expansions to improve the framework further.

Database Integration

To enhance the complexity of the games developed using the framework, integrating a real database is a priority. This would allow for dynamic content generation and management, enabling more sophisticated game dynamics such as player progression and state persistence across sessions.

Advanced Game State Management

Developing a more comprehensive game state management system would allow the framework to support games with multiple levels, stages, or modes. This would make the framework suitable for more complex game designs that require advanced state management.

By supporting multiple game states, the framework could accommodate a broader variety of game types, appealing to a wider developer audience.

Extensible Component Library

Expanding the framework's component library to include a wider range of pre-built elements and modules would significantly speed up the development process for game developers.

Enhanced Documentation and Tutorials

Improving the documentation and providing comprehensive tutorials would make the framework more accessible, especially to new users or those less familiar with game development.

Community Collaboration

Establishing a robust community around the framework can lead to more feedback, improvements, and innovations.

Open-Source Contributions

Making the framework open source and encouraging contributions can help accelerate its development and adoption, incorporating a wide array of ideas and approaches.

Performance Optimization

Continual efforts to optimize the performance of the framework will ensure that the games developed with it run smoothly on a variety of devices, providing a better end-user experience.

Gamification Research

Investing in research on gamification techniques and new interactive elements can lead to innovations within the framework, enhancing its appeal and effectiveness.

Conclusion

The planned future work aims to build on the current successes of the Casino-Style Arcade Game Framework by broadening its capabilities and making it more versatile and user-friendly. Each of these areas presents opportunities for significant enhancements that could further solidify the framework's position as a leading tool in game development.

Conclusion

The development of the Casino-Style Arcade Game Framework has resulted in a robust and flexible tool that aids the creation of engaging web-based arcade games. This conclusion reflects on the project's achievements and the insights gained throughout the development and evaluation phases.

Summary of Key Findings and Insights

The framework has successfully demonstrated its capability to simplify the game development process significantly. By employing a monorepo architecture and utilizing modern technologies such as Node.js **(Node.js, 2024)**, Pixi.js **(PixiJS, 2024)**, and GSAP **(GreenSock, 2024)**, it has provided a strong foundation for developers to create high-quality games efficiently.

Efficiency and Flexibility

The framework's modular design and the inclusion of a mock JSON database have proven effective in streamlining the development process, allowing developers to focus more on game design and user experience.

User Engagement

Games developed using the framework have been well-received, indicating that the user interface and game mechanics are both intuitive and entertaining.

Reflection on the Overall Experience

The project has been a valuable learning experience, highlighting the importance of thorough planning, continuous testing, and iterative development. Despite some challenges, such as managing game assets and ensuring cross-browser compatibility, the solutions implemented have enhanced the framework's robustness and usability.

Challenges and Overcoming Them

Addressing the initial challenges with asset management and compatibility issues provided critical lessons in adaptability and problem-solving.

Final Remarks

While the framework meets many of the initial objectives, there is potential for further enhancement and expansion. The planned future work, including integrating a real database and expanding the component library, promises to further increase the framework’s utility and appeal.

The framework is well-positioned to evolve and adapt to new technologies and user needs, continuing to serve as a valuable tool for game developers around the globe. The experiences and lessons learned from this project will undoubtedly influence future developments in the field of game development.

References

- Node.js (2024) Node.js Official Documentation. Available at: https://nodejs.org/en/docs/ (Accessed: 22 April 2024).
- PixiJS (2024) PixiJS 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).
- Cohen, D, Lindvall, M. (2004). An Introduction to Agile Methods. Advances in Computers 62(66):1-66.
- Reenskaug, T. (1979). Models-Views-Controllers.
- Salen, K, Zimmerman, E. (2003). Rules of Play: Game Design Fundamentals. MIT Press.
- Parisi, T. (2012). WebGL: Up and Running. O'Reilly Media.
- Myers, G J, Sandler, C, Badgett, T. (2012). The Art of Software Testing, 3rd Edition. Wiley.
- Shaw, M, & Garlan, D. (1996). Software Architecture. Perspective on an Emerging Discipline. Prentice Hall.
- Beck, K, Andres, C. (2004). Extreme Programming Explained: Embrace Change, 2nd Edition. Addison-Wesley Professional.

Appendices

Appendix A

01/12/2023

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

07/03/2024

Received advice on the structure of the project.

26/04/2024

Demonstrated latest developments of the project and received advice on the final report.

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
- Pre-built games: https://kn215.brighton.domains/arcade/games