Pacman Java Game

A PROJECT REPORT
for
Mini-Project 2 (ID201B) Session (2024-25)

Submitted by

Aditya Chaudhary
(202410116100010)
Aaryan Gautam
(202410116100003)
Akash Choudhary
(202410116100015)

Submitted in partial fulfilment of the Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Mr. Vipin Kumar (Assistant Professor)

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206 (MAY-2025)

CERTIFICATE

Certified that **Aditya Chaudhary (202410116100010), Aaryan Gautam (202410116100003) and Akash Choudhary (202410116100015)** has/have carried out the project work having **“Pacman Java Game”** MINI PROJECT - 2 (FULL STACK DEVELOPMENT)
(ID201B)) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Mr. Vipin Kumar	**Dr. Akash Rajak**
Assistant Professor	**Dean**
Department of Computer Applications	**Department of Computer Applications**
KIET Group of Institutions, Ghaziabad	**KIET Group of Institutions, Ghaziabad**
An Autonomous Institution	**An Autonomous Institution**

Pacman Java Game

ABSTRACT

Pacman Java 2D Game is an engaging desktop-based arcade application developed using Java and 2D graphics libraries, aimed at recreating the classic Pacman experience with modern coding practices. The game allows users to navigate the iconic Pacman character through a maze while avoiding ghosts and collecting pellets, thereby offering both nostalgia and fun gameplay.

The primary objective of this project is to enhance understanding of game development fundamentals using Java, including object-oriented programming, event-driven design, collision detection, and sprite animation. The game logic incorporates intelligent ghost movement, score tracking, and level progression to ensure a challenging and interactive experience for players.

By implementing clean Java architecture and leveraging Java 2D for rendering graphics, the Pacman Java 2D Game ensures smooth gameplay and efficient performance across platforms. It serves as a practical learning project for aspiring developers looking to build foundational skills in game development and graphical user interface (GUI) programming.

This project contributes to the digital entertainment space by offering a lightweight and accessible game that mimics the classic Pacman charm while being a strong example of Java-based 2D game development.

ACKNOWLEDGEMENT

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Mr. Vipin Kumar** for his guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Akash Rajak**, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Aditya Chaudhary

Aaryan Gautam

Akash choudhary

TABLE OF CONTENTS

List of Figures

Figure No.

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Pacman Java 2D Game is a desktop-based arcade game designed to provide an engaging and nostalgic experience by recreating the classic Pacman gameplay using Java and 2D graphics. The game utilizes object-oriented programming principles, event-driven logic, and Java’s built-in libraries to render smooth animations, detect collisions, and manage user inputs for real-time interaction. Players navigate Pacman through a maze, avoiding ghosts and collecting pellets to progress through increasingly challenging levels.

The game features an intuitive interface that allows users to start and control the game seamlessly. The core logic includes features such as ghost AI for dynamic enemy movement, point tracking, lives system, and level progression. The implementation relies on sprite animation, game loop mechanisms, and collision management to simulate realistic and responsive game behavior, ensuring an interactive and entertaining experience.

To achieve performance and visual quality, the game is optimized with Java 2D rendering techniques and structured in a modular codebase to maintain clarity and scalability. The project benefits not only casual gamers but also programming learners and developers aiming to understand the essentials of game development using Java. By integrating animation, logic, and user interaction, Pacman Java 2D Game simplifies learning through hands-on application, making it an ideal project for both educational and recreational purposes.

1.2 PROBLEM STATEMENT

In the realm of learning programming and game development, beginners often struggle to find practical projects that combine logic, graphics, and interactivity in a simple yet engaging format. Many learning resources lack integration between core programming concepts and real-world applications, resulting in limited understanding and motivation.

Classic games like Pacman offer an ideal platform for this purpose, but they are often complex or outdated when attempted from scratch. Students and aspiring developers frequently face difficulties implementing smooth animations, real-time interactions, and intelligent game mechanics in Java, especially without structured examples. **Pacman Java 2D Game** addresses this challenge by offering a simplified yet fully functional version of the original game, built using Java's 2D graphics libraries. This project provides an interactive and rewarding way to explore object-oriented programming, graphical rendering, and basic game AI.

1.3 OBJECTIVE

- **Introduce Game Development Fundamentals** – Provide a hands-on project that teaches core programming concepts such as loops, conditions, classes, and objects in an interactive setting.

- **Build Logical and Visual Integration** – Help learners understand how logic and graphics work together by implementing real-time game loops, animations, and collision detection.

- **Implement Classic Game Mechanics** – Recreate original Pacman features such as ghost AI, score tracking, pellet collection, and level progression using Java.

- **Enhance Programming Skills** – Strengthen object-oriented design skills and improve coding structure, modularity, and debugging practices through a complete Java-based application.

- **Promote Interactive Learning** – Engage users through visual feedback and responsive controls, making programming more exciting and relatable for beginners and enthusiasts alike.

1.4 SCOPE

Pacman Java 2D Game aims to modernize the learning experience for beginner programmers and game development enthusiasts by providing a fully functional replica of the classic arcade game using Java. The project is designed to cater to students, educators, and aspiring developers, offering a fun and structured way to explore core programming concepts through hands-on application.

The project's scope includes the development of a standalone desktop application that enables players to control Pacman through a maze, collect pellets, avoid ghosts, and progress through levels. By implementing Java 2D graphics and object-oriented principles, the game emphasizes modular coding, collision detection, sprite animation, and event-driven programming, ensuring an immersive and educational experience.

Pacman Java 2D Game will simulate game logic with real-time user interaction, AI-controlled enemy movement, and score/life tracking mechanics to mimic the original gameplay. The system may also include multiple levels of increasing difficulty, sound integration, and keyboard input support for interactive control. Additionally, the project can be extended to support multiplayer functionality or enhanced graphical elements for richer visual appeal.

The project has the potential to serve as a foundation for more advanced games by incorporating features such as level editors, save/load functionality, and game state management. Furthermore, scalability options include adapting the game for web or mobile platforms, integrating learning modules for educational use, and releasing source code for open-source collaboration in coding communities.

By combining the excitement of retro gaming with practical Java programming, Pacman Java 2D Game enhances both engagement and skill-building, making it an essential learning tool for those interested in software development and 2D game design.

1.4 FEATURES

Classic Gameplay Mechanics: Implements the traditional Pacman experience where the player navigates a maze, collects pellets, and avoids ghosts to survive and score points.

• **Game Loop Logic**: Uses a consistent and efficient game loop to handle updates, rendering, and input processing in real-time.

User-Friendly Interface: Provides a simple and interactive interface with keyboard controls for seamless gameplay and a clean display of scores and lives.

• **Collision Detection**: Detects interactions between Pacman, walls, pellets, and ghosts to manage game logic such as point accumulation and life loss.

• **Animated Sprites**: Displays animated movements for Pacman and ghosts, enhancing visual appeal and improving player immersion.

• **Sound Effects Integration**: Incorporates classic arcade sound effects for actions like eating pellets, ghost encounters, and game over scenarios.

• **Multiple Difficulty Levels**: Offers varying levels of difficulty by adjusting ghost speed and AI complexity to keep the gameplay challenging.

• **Score and Life Tracking**: Continuously updates and displays the player’s score and remaining lives during gameplay.

• **Modular Code Design**: Structures the project using Java classes for entities like Player, Ghost, Maze, and GameEngine, supporting easy maintenance and scalability.

• **Cross-Platform Compatibility**: Designed to run smoothly on desktop systems across different operating systems using standard Java libraries.

1.5 HARDWARE/ SOFTWARE USED IN PROJECT

☐ ***Hardware Requirement***

S. N.	Description
1	PC with 5 GB or more Hard disk.
2	PC with 2 GB RAM.
3	PC with core i3 or above processor.

Table 1.1 Hardware Requirement

☐ **Software Requirements**

S. N.	Description	Type
1	Operating System	Windows 7 or above
2	Programming Language	Java SE Development Kit
3	IDE	NetBeans, IntelliJ IDEA
4	Graphics Library	Java AWT, Swing

Table 1.2 Software Requirement

1.6 BACKGROUND

With the growing interest in interactive learning and the popularity of retro arcade games, classic titles like Pacman remain an effective and engaging way to teach programming concepts. Many beginners and students find traditional textbook examples difficult to relate to, often losing interest in the process of learning how to code.

The Pacman Java 2D Game was conceptualized to bridge this gap by offering a fun, interactive, and visual approach to software development. By using core Java libraries such as AWT and Swing, the project introduces players to game design fundamentals including object movement, event handling, collision logic, and animation, all within a simple 2D framework.

As a result, the game not only recreates the nostalgic experience of the original Pacman but also serves as a practical project for students to understand Java programming in an applied manner. It demonstrates how theory can be converted into a working application, encouraging logical thinking and creativity.

CHAPTER 2

FEASIBILITY STUDY

The development and deployment of the Pacman Java 2D Game is highly feasible from both a technical and economic standpoint. From a technical perspective, the game is implemented using Java, a widely taught and well-supported programming language in academia and industry. It uses built-in libraries such as AWT and Swing, which provide sufficient functionality to develop interactive graphical user interfaces and basic 2D game elements like sprites, animation, and collision detection.

Economically, the project is suitable for student developers or small-scale projects, as it requires no expensive third-party tools or subscriptions. Java is open-source and widely available, and the IDEs used for development (such as NetBeans or IntelliJ) offer free community editions. The project can be built and run on any system meeting basic hardware requirements, making it accessible to a broad range of users, including students, educators, and beginners aiming to learn Java through hands-on game development.

2.1 ECONOMICAL FEASIBILITY

The development of the Pacman Java 2D Game is economically feasible, especially as a student project. The technologies required—namely Java SE, Java AWT, and Swing libraries—are open-source and free. IDEs like NetBeans or IntelliJ IDEA Community Edition allow for rapid development without incurring costs. No additional licenses or high-performance systems are required, keeping infrastructure expenses minimal.

Additionally, the game does not rely on any paid APIs or data subscriptions. The logic, assets, and mechanics can be custom-built or sourced from royalty-free repositories. The development and testing can be done on local machines, and distribution through platforms like GitHub further eliminates the need for costly hosting services. This makes it a practical and cost-efficient project suitable for learning and demonstration purposes.

- **Development Costs**: This includes time and resources for designing game mechanics, implementing movement logic, collision detection, and building the UI, all of which are achievable with free tools and minimal financial input.

- **Operational Expenses:** These are minimal recurring costs required to support the Pacman Java 2D Game. Since the game runs locally or as a standalone desktop application, there is no need for web hosting or domain registration. If uploaded for public access (e.g., on a website), hosting would be the only operational cost.

- **Revenue Streams (for future scope):** While the Pacman game is primarily an educational project, future revenue models could include offering it on app stores, integrating in-game ads, or selling upgraded versions with new levels, themes, or multiplayer support.

- **Projected Utility – & ROI:** The game holds strong educational value, helping beginners learn game development, Java programming, and UI design. Its simplicity makes it an ideal teaching tool, and if expanded commercially, it could yield modest returns through ads or premium versions.

- **Maintenance Costs:** Once developed, maintenance is limited. Occasional updates might be needed to fix bugs, optimize code, or improve gameplay experience. Java's backward compatibility ensures long-term usability with minimal code changes.

- **Hosting Fees:** If the game is distributed as a downloadable file or shared via GitHub, there are no hosting costs. Publishing to platforms like itch.io or GitHub Pages is free, while paid hosting (if required) would be minimal and optional.

- **Scalability** : The Pacman game is designed with modular Java classes, allowing easy scalability for future enhancements. As new features or levels are added, the structure supports expansion without disrupting existing functionality.
- **Cost-Effectiveness of Technology Stack** : The project utilizes the Java programming language and free, open-source tools, ensuring a highly cost-effective development process while offering robust capabilities for game logic, graphics, and user interaction.

2.2 TECHNICAL FEASIBILITY

The Pacman Java game project is technically feasible due to the use of a well-established and widely supported programming language. Java provides object-oriented features, extensive libraries, and strong community support, making it ideal for game development. The game is built using core Java concepts such as classes, objects, inheritance, and interfaces, providing a solid foundation for both learning and execution.

Graphics and game interface are implemented using Java's built-in libraries such as Swing and AWT, enabling the creation of a responsive and interactive game environment. These libraries support smooth animations, keyboard event handling, and collision detection. The game can be compiled and executed on any standard Java Runtime Environment (JRE), allowing broad compatibility across systems.

Additionally, the game is lightweight and can run on low-resource machines without the need for high-end hardware. The source code is easy to modify, making the application suitable for educational purposes or further development into more complex game projects. Overall, the technical components of the Pacman game are not only feasible but also efficient and scalable for academic or personal use.

Technical feasibility refers to the examination of the technological aspects involved in the successful execution of a Pacman game project. Here's a detailed explanation of the components:

- **Programming Language**: Java is used, which is highly stable and suitable for building 2D games with clear logic and structure.
- **Graphics and Event Handling Libraries**: Utilizes Java libraries such as AWT and Swing.

- **Game Framework**: Developed using core Java, including the AWT and Swing libraries for implementing game loops, event listeners, and rendering the GUI for player interaction.
- **Frontend Technologies**: Utilizes Java's built-in GUI libraries (AWT/Swing) to create a responsive game interface with interactive controls and smooth animation.
- **Game Type**: Implements classic 2D arcade-style game mechanics involving maze navigation, point collection, and collision detection with enemies.
- **Asset Management**: Makes use of lightweight image and sound files (e.g., `.png`, `.wav`) that are freely available or self-created, reducing asset dependency issues.
- **System Requirements**: Can be run on any standard machine with the Java Runtime Environment (JRE); no high-end graphics hardware or special configurations are needed.
- **Hosting & Deployment**: Can be packaged into a standalone `.jar` file and distributed easily across devices; optionally deployable on educational or game-sharing platforms.
- **Security Considerations**: Contains no sensitive data handling; runs entirely on the client-side, eliminating server-side risks and simplifying security.
- **Scalability & Maintenance**: Built using modular Java classes, enabling future additions such as new levels, difficulty settings, or multiplayer support without refactoring the core game logic.

2.3 OPERATIONAL FEASIBILITY

Operational feasibility refers to the evaluation of how effectively the Pacman Java game can be implemented and used in a practical, real-world setting. This involves examining whether the game can function smoothly from the player's perspective and how well it aligns with the intended learning and entertainment objectives.

The Pacman game is designed to offer an engaging and accessible user experience. Players interact with the game through keyboard controls to navigate the maze, collect points, and avoid enemies, offering a familiar and intuitive gameplay structure. Since it is developed using core Java, the game does not require internet connectivity or additional software installations beyond the Java Runtime Environment, making it suitable for use in offline environments such as classrooms, training labs, or personal systems.

Its lightweight design ensures smooth performance even on low-end devices, while the modular codebase allows easy customization. Educators and learners can modify the code to experiment with object-oriented programming concepts or game mechanics, enhancing its educational value.

Overall, the Pacman Java game is not only operable across various platforms but also effective as both a learning tool and a source of entertainment, supporting continued use and future expansion without significant operational barriers.

Key Components of Operational Feasibility:

- **User-Friendliness**: Designed with a simple and interactive interface using arrow key controls, making it easy for users of all ages to play without technical knowledge.
- **Target Audience**: Suitable for students, beginner programmers, educators, and retro game enthusiasts looking to learn or enjoy classic arcade games.
- **Ease of Use**: Requires only basic keyboard input (arrow keys) for navigation; users can start playing immediately without needing any configuration.
- **Minimal Training Required**: No prior experience or technical training is required to run or play the game; instructions are intuitive and minimal.
- **Automated Gameplay Logic**: The game loop, enemy movement, scoring, and collision detection are handled automatically in the background, providing seamless gameplay.
- **Maintenance Simplicity**: Occasional updates may include new levels, improved graphics, or bug fixes, which can be easily managed due to the modular structure.
- **Platform Independence**: Can be executed on any device with a Java Runtime Environment (JRE), ensuring wide accessibility across platforms.
- **Low Operational Supervision**: Once launched, the game runs autonomously without requiring manual monitoring or intervention during gameplay.
- **Alignment with Goals**: Serves as both an educational tool to understand Java programming and a fun gaming experience, fulfilling its core objectives effectively.

2.2 BEHAVIORAL FEASIBILITY

Behavioral feasibility evaluates how well the users and stakeholders are likely to accept, adopt, and interact with the Pac-Man Java game. It considers the willingness of players to adapt to the new game mechanics and their ability to use it effectively. The Pac-Man Java game is designed to offer a familiar, engaging, and time-tested arcade experience, which often involves simple, intuitive controls. By providing a fast, accurate, and visually appealing gameplay based on traditional arcade mechanics, the system promotes user enjoyment and ease of play. The simple and responsive interface enhances user engagement and encourages repeated play sessions. Furthermore, the game does not disrupt any existing workflows and can be accessed anytime, increasing the likelihood of user acceptance. Since the application addresses a common entertainment desire with a clear solution, behavioral resistance is expected to be minimal, making the project behaviorally feasible.

Key Components of Behavioral Feasibility:

- **User Acceptance:** High likelihood of adoption due to the game's intuitive nature and recognition of the classic Pac-Man mechanics.

- **Trust in Results:** Gameplay provides clear, immediate feedback on actions (e.g., eating pellets, being caught by ghosts), enhancing player confidence in the game's fairness.

- **Minimal Resistance:** Users face no significant behavioral or psychological barriers in understanding and playing the game.

- **Accessibility:** Can be accessed easily through a standard Java Runtime Environment (JRE) with no need for complex installation or setup.

- **No Workflow Disruption:** Does not interfere with existing computer usage habits or other activities—adds value as entertainment instead.

- **Familiarity with Inputs:** Uses simple, commonly known game controls (e.g., arrow keys, WASD), requiring no technical background.

- **Encourages Reuse:** Engaging and responsive gameplay encourages repeated play sessions by individual users and groups of friends.
- **Positive User Experience:** Intuitive controls and nostalgic visuals foster positive interactions and enjoyment.
- **Fulfills User Needs:** Effectively addresses a common desire for entertaining and classic arcade gaming.

CHAPTER 3

SOFTWARE REQUIREMENT SPECIFICATION

The Pac-Man Java game software requires a robust and user-friendly desktop application capable of providing an engaging and authentic arcade experience. The system will be developed using Java for backend processing and game logic, with standard Java AWT/Swing for graphics and user interface rendering. The frontend will utilize Java Swing components, ensuring a responsive and accessible user interaction. The application must be able to accept player input (e.g., keyboard controls), process game state updates efficiently, and display real-time animations. It should support deployment on standard desktop environments with minimal setup. Additionally, the system will require access to static game assets (e.g., maze layouts, sprites, sound files). Security requirements include basic input validation to prevent erroneous game states, with no sensitive user information stored, ensuring ease of maintenance and scalability.

3.1 FUNCTIONALITIES

The Pac-Man Java game offers a set of core functionalities designed to deliver an accurate and user-friendly arcade experience. The system allows players to input key directional commands (e.g., Up, Down, Left, Right) to control the Pac-Man character. It processes this input against the maze layout, ghost AI, and pellet consumption mechanics to determine the current game state and score. The platform provides real-time visual feedback and displays the score, lives, and game progression clearly. Additionally, the system supports game state initialization (e.g., new game, level reset), manages the placement of pellets and power-ups, and provides administrative access to reset high scores or debug game logic as needed. The interface is designed to be responsive and accessible across various desktop systems, facilitating ease of use for individual players, and casual gamers.

Key Components of Functionalities:

- **Player Input Module:** Collects player commands for movement (Up, Down, Left, Right) and game actions (e.g., pause, start).
- **Game State Validation:** Ensures that player inputs and game events (e.g., collisions, pellet consumption) are accurate and within reasonable game logic.
- **Game Logic Engine:** Utilizes the core Pac-Man rules to manage character movement, ghost AI, collision detection, scoring, and level progression.
- **Result Display:** Presents the current score, lives, level, and game-over/win states in a clear, understandable format on the user interface.
- **Game Asset Management Module:** Stores and manages game resources such as maze layouts, sprite sheets, sound effects, and high scores.
- **Administrative Controls:** Enables authorized users (e.g., developers) to reset high scores, adjust game settings, and maintain the system.
- **Responsive User Interface:** Designed for accessibility and ease of use across various desktop display sizes and resolutions.

3.2 USER AND CHARACTERISTICS

The Pac-Man Java game system is designed to serve a diverse range of users involved in interactive entertainment. Primary users include individual players seeking engaging and quick gaming sessions to enjoy their leisure time. Casual and experienced gamers can also benefit from the platform to evaluate their reflexes and strategic thinking efficiently without relying solely on external tools. Users are expected to have basic computer literacy and familiarity with common input methods such as keyboard controls. The system is built to accommodate users with minimal technical expertise by providing a simple and intuitive interface. Here are some key components:-

- **Individual Players:** Users looking to get quick, accurate entertainment from a classic arcade game for personal enjoyment.

- **Casual Gamers:** Players who enjoy light entertainment and quick, engaging challenges without extensive time commitment.
- **Arcade Enthusiasts:** Individuals who appreciate classic arcade games and seek an authentic digital recreation.
- **Basic Computer Literacy:** Users generally have basic knowledge of operating desktop applications and utilizing keyboard inputs.
- **Familiarity with Game Mechanics:** Users understand common arcade game elements like character movement, scoring, and level progression.
- **Non-Technical Users:** The system is designed to be intuitive and easy to use without specialized technical skills.
- **Administrative Users:** System maintainers (e.g., developers or designated administrators) who handle high score resets, game configuration, and asset management.
- **Higher Technical Skill for Admins:** Administrative users possess technical expertise to ensure system accuracy and smooth operation.

3.3 FEATURES OF PROJECT

- **Authentic Gameplay Experience:** Uses classic Pac-Man mechanics, including maze traversal, pellet consumption, power-ups, and ghost AI.
- **User-Friendly Interface:** Simple and intuitive graphical interface that requires minimal inputs for immediate gameplay.
- **Multiple Input Parameters:** Supports essential inputs such as keyboard arrow keys or WASD for detailed character control.

- **Real-Time Feedback:** Provides instant visual and auditory feedback immediately after the player performs actions (e.g., eating pellets, turning corners).
- **Game State Validation:** Ensures that player actions and game events are checked for accuracy and validity to avoid glitches or exploits.
- **Responsive Design:** Compatible with various desktop screen sizes and resolutions for easy access anywhere.
- **Periodic Content Updates:** Allows for the introduction of new maze layouts, ghost behaviors, or power-ups to maintain and improve player engagement.
- **Lightweight and Fast:** Optimized for quick loading and minimal resource consumption, ensuring smooth gameplay.
- **Secure and Private:** No sensitive personal data is collected or stored, ensuring user privacy.
- **Administrative Controls:** Enables authorized personnel to manage high scores, update game assets, and oversee system maintenance.

3.4 FEATURE OF ADMIN

- **User Management:** Ability to monitor and manage user accounts (if implemented for features like high scores or multiplayer) and access permissions if applicable.
- **Game Content Management:** Upload, update, and manage game assets (e.g., maze files, sprite sheets, sound files) for enhancing and improving the game.
- **Game Logic Tuning & Retraining:** Initiate and oversee the adjustment of game parameters (e.g., ghost speed, pellet count) to ensure updated and accurate game balance.
- **System Monitoring:** Track game performance, usage statistics (if opted-in by user), and detect any anomalies or errors.
- **High Score Management:** Update or reset global/local high scores on the platform as needed.

- **Error Handling and Debugging:** Identify, log, and resolve game bugs or issues to maintain smooth operation.
- **Security Management:** Implement and maintain security measures to protect the game system from unauthorized access or tampering (e.g., high score manipulation).
- **Backup and Recovery:** Manage game data backup processes (e.g., high scores, custom levels) and ensure recovery options are in place for system failures.

3.5 FEATURES OF USER

- **Simple Input Interface:** Easy-to-use controls (e.g., keyboard arrow keys, WASD) for seamless Pac-Man movement.
- **Quick Game Start:** Receive fast and accurate game loading and immediate play based on selected difficulty or level.
- **Real-Time Feedback:** Instant display of game state changes (score updates, ghost movements, power-up effects) without delay.
- **Input Validation:** Guidance and error messages (if applicable) to help players understand correct controls or valid game actions.
- **Accessible Anywhere:** Use the game on various desktop systems (Windows, macOS, Linux) through a standard Java Runtime Environment.
- **No Technical Skills Required:** Designed for users with minimal or no technical background, focusing on intuitive play.
- **Clear Game Presentation:** Game progress, scores, and visual elements are displayed in a straightforward and understandable manner.
- **Support for Multiple Game Modes/Levels:** Ability to choose different maze layouts or difficulty settings for diverse gameplay categories.

CHAPTER 4

SYSTEM REQUIREMENT

The Pac-Man Java game project requires a system with basic modern specifications to function efficiently. It should run on Windows, Linux, or macOS with at least a 1.0 GHz processor (or equivalent), 512 MB RAM, and 100 MB of free storage. The software stack includes Java Development Kit (JDK) 8 or higher for game logic and rendering, and Java AWT/Swing for the graphical user interface. External libraries are not strictly required for core functionality, but might be used for sound, image loading, or advanced UI components if needed. The application will be accessible via a standard Java Runtime Environment installed on the user's desktop machine.

4.1 FUNCTIONAL REQUIREMENT

The functional requirements of the Pac-Man Java game define the core operations that the system must perform to fulfill its purpose of providing an entertaining arcade experience. These include collecting user input for movement, validating game actions, processing the game state through a game logic engine, and displaying the real-time visual and auditory feedback. Additionally, the system should allow administrative control for high score management and game configuration, ensuring the platform remains accurate and up to date. The interface must be interactive, responsive, and user-friendly to support a wide range of users with minimal technical knowledge.

Key Components:

- Player input handling for movement (Up, Down, Left, Right)
- Game state validation module to ensure accurate game progression
- Core game logic integration for character movement, ghost AI, and collision detection
- Real-time display to show game board, Pac-Man, ghosts, pellets, and scores
- Admin dashboard for managing high scores and game settings

- Function to trigger game restart/level reset
- Error handling and user feedback messages for game events
- Responsive design for accessibility across desktop screens
- Secure access control for administrative functions (e.g., high score resets)

4.2 NON-FUNCTIONAL REQUIREMENT

The non-functional requirements of the Pac-Man Java game project define the quality attributes that ensure the system operates efficiently, reliably, and securely. These include aspects like performance, scalability, usability, and security. While they do not directly affect the functionality of the system, they are essential to ensure a smooth user experience, maintainability, and long-term stability of the application.

Key Components:

- **Performance:** The system should respond to user inputs (e.g., key presses) and update the game state (e.g., Pac-Man movement, ghost AI) in under 50 milliseconds for fluid gameplay.
- **Scalability:** The game should be able to run smoothly on various desktop configurations without significant degradation in performance, even with complex maze designs or additional game elements.
- **Usability:** The user interface should be intuitive and accessible to players with minimal technical expertise, allowing for immediate engagement.
- **Reliability:** The system should function consistently with minimal crashes or unexpected errors during gameplay.
- **Security:** High scores and any stored game preferences should be protected from unauthorized modification, and administrative functionalities should be accessible only to authorized personnel.
- **Compatibility:** The system should run smoothly on different desktop operating systems (Windows, macOS, Linux) provided a compatible Java Runtime Environment is installed.

- **Maintainability:** The codebase should be modular and well-documented to support future game updates, bug fixes, and new feature additions.

- **Availability:** The game should be launchable and playable at any time without external server dependencies or significant loading delays.

4.3 DESIGN GOAL

The design goals of the Pac-Man Java game project outline the intended structure, usability, and visual appeal of the application to meet player expectations and provide an engaging gaming experience. The system aims to be clean, responsive, and efficient, providing users with a seamless experience when navigating the maze and interacting with game elements. The design should support quick interactions, ensure game logic accuracy, and offer adaptability for future enhancements.

Key Components:

- **Simplicity:** Keep the user interface clean and easy to navigate, focusing on core gameplay elements.
- **Responsiveness:** Ensure the game's visual updates and controls react immediately to player input, providing fluid gameplay.
- **Accuracy:** Display game elements (e.g., Pac-Man, ghosts, pellets) and score clearly and consistently.
- **Modularity:** Use a structured codebase for easy feature updates (e.g., new mazes, ghost behaviors) and debugging.
- **Consistency:** Maintain a consistent visual theme, color scheme, and layout throughout the game.
- **Accessibility:** Support keyboard navigation for all user types and ensure game elements are visually distinct.
- **Feedback:** Provide instant visual and auditory feedback for player actions (e.g., pellet consumption sound, power-up animation).
- **Extensibility:** Allow space for adding new features like additional levels, different ghost AI patterns, or high score saving mechanisms in the future.

Use Case Diagram

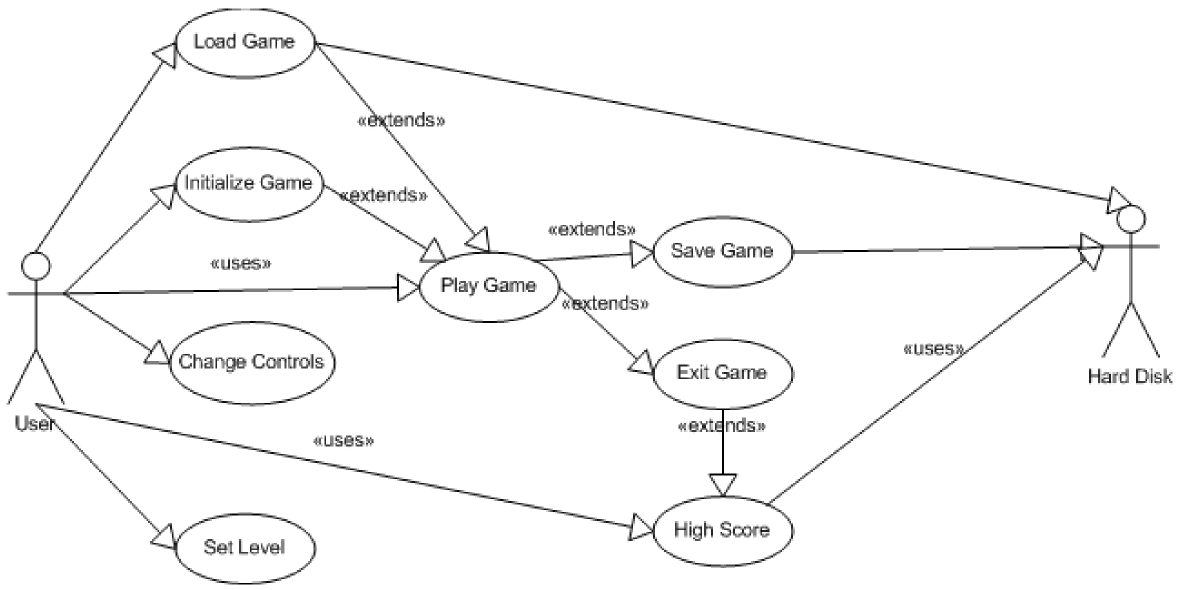
A Use Case Diagram is a graphical representation of the interactions between users (actors) and a system. It is part of **Unified Modeling Language (UML)** and is used to describe the functional requirements of a system. The diagram helps developers, designers, and stakeholders understand how different users interact with the system and what functionalities are available.

In the case of the Pac-Man Java Game, the primary actor is the **Player**, who interacts with the system to enjoy an arcade gaming experience. Other possible actors may include **Administrator**. The system itself consists of various **use cases**, such as **Start Game, Control Pac-Man Movement, Eat Pellets, Avoid Ghosts, Use Power Pellet, Track Score, Game Over, and Manage High Scores**. Each use case represents a specific function that the system performs in response to user interactions.

The relationships between actors and use cases are depicted using **associations (lines)**, showing which users perform which actions. Additional relationships like "**Include**" (indicating a mandatory sub-function) and "**Extend**" (representing optional behaviors) can be used to refine the diagram further.

By analyzing the Use Case Diagram, developers can identify user requirements, refine system functionalities, and ensure smooth interactions between different components. It serves as a crucial blueprint in system design, helping to build an intuitive and efficient platform like the Pac-Man Java Game.

USE CASE DIAGRAM



- **Fig 4.1: Use Case Diagram**

ER Diagram

An Entity-Relationship (ER) Diagram is a graphical representation that shows the structure of a database or information system. It identifies the *entities* (objects or concepts about which information is stored), the *attributes* (properties of those entities), and the *relationships* between the entities. While a Pac-Man game doesn't typically have a complex persistent database, an ER diagram can help us think about the game's data structure conceptually.

Entities (Conceptual):

- **Player:** Represents a player of the game.
    - Attributes: `PlayerID` (primary key, unique identifier), `Name`, `HighScore`, `LevelReached`.
- **Game:** Represents a single instance of a game played.
    - Attributes: `GameID` (primary key), `PlayerID` (foreign key, linking to the Player), `Score`, `Level`, `DateTime`.
- **Maze:** Represents a level layout.
    - Attributes: `MazeID` (primary key), `LayoutData` (how the maze is structured).

Relationships:

- **Plays:** A Player *plays* many Games. (One-to-many relationship from Player to Game).
- **Uses:** A Game *uses* one Maze. (One-to-one relationship from Game to Maze, or one-to-many if multiple levels are stored).

Explanation:

In this simplified conceptual ER diagram:

- The `Player` entity stores information about individual players, including their high scores and progress.
- The `Game` entity records each game session, linking it to the player and storing details like score and level.

Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation of the flow of data through an information system. It shows how data is input to and output from the system, where data is stored, and what processes transform the data. DFDs are a useful tool for understanding the system's functions and processes without detailing the underlying technology. They use specific symbols to represent external entities, processes, data stores, and data flows.

In the context of the Pac-Man Java Game, a DFD would illustrate how game data moves and is processed from player input to on-screen display and score management. We can typically identify different levels of DFDs (Level 0 for context, Level 1 for main processes, etc.).

Level 0 DFD (Context Diagram):

- **External Entities:**
    - **Player:** Provides input and receives visual/auditory feedback.
    - **Administrator:** Manages high scores and game settings.
- **System Boundary:** The entire "Pac-Man Java Game System."
- **Data Flows:**
    - `Player Input` (from Player to System)
    - `Game Display` (from System to Player)
    - `Sound Feedback` (from System to Player)
    - `Admin Commands` (from Administrator to System)
    - `High Score Data` (from System to Administrator, and possibly Administrator to System for updates).

Level 1 DFD (Main Processes):

This level breaks down the main system into its primary processes, data stores, and data flows.

- **External Entities:** (Same as Level 0: Player, Administrator)

- **Processes:**

1. **Handle Player Input:** Receives player input (e.g., key presses).
2. **Manage Game Logic:** The core game engine; processes game state changes (movement, collisions, ghost AI, pellet consumption).
3. **Update Game State:** Modifies internal game data based on `Manage Game Logic` results.
4. **Render Game Display:** Generates visual and auditory output for the player.
5. **Manage High Scores:** Handles reading, updating, and storing high scores.
6. **Configure Game Settings:** Processes changes to game parameters by the administrator.

- **Data Stores:**

  - **Game State Data:** Stores current positions of Pac-Man, ghosts, remaining pellets, lives, current score, level.
  - **Maze Data:** Stores the layout of the current maze.
  - **High Scores Database:** Stores persistent high score records.
  - **Game Configuration:** Stores settings like ghost speed, initial lives, sound volume.

DESIGN (OUTPUT)

The Design of **Pacman Game** refers to the final visual and functional representation of the system based on its requirements and design specifications. It includes various UI components, game logic, database structures (if applicable), and system interactions that ensure an engaging user experience.

1. User Interface (UI) Design:

- A classic arcade-style game screen with a maze layout, Pacman, and ghosts.
- Start and pause menus for controlling gameplay.
- A scoreboard displaying the current score, remaining lives, and level.
- Animations for Pacman movement, ghost reactions, and pellet consumption.
- Sound effects and background music to enhance gameplay immersion.

2. Game Mechanics Design:

- A tile-based maze system where Pacman navigates through corridors to eat pellets.
- Pathfinding algorithm for ghost AI behavior (e.g., chasing, fleeing, random walk).
- Collision detection system to manage interactions between Pacman, ghosts, and walls.
- Power pellets to allow Pacman to temporarily eat ghosts.
- Level progression by clearing all pellets on the current level.

3. System Functionality & Processing:

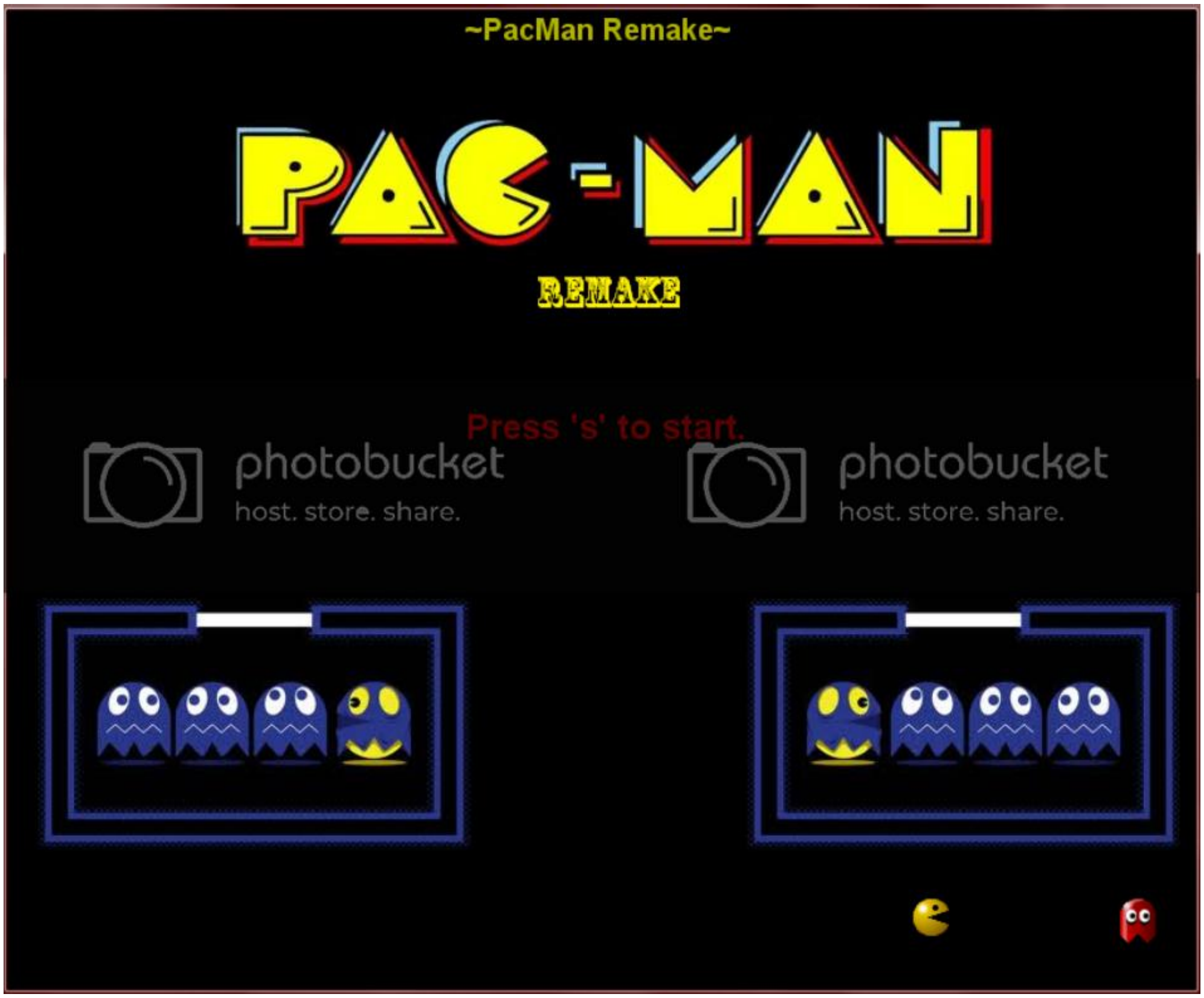
- Real-time game loop handling player input, rendering, and game state updates.
- Keyboard control for Pacman movement in four directions.
- Dynamic difficulty scaling by increasing ghost speed with higher levels.
- Score tracking system with high-score memory (optional local storage).
- Game over and win logic for determining end of gameplay**.**

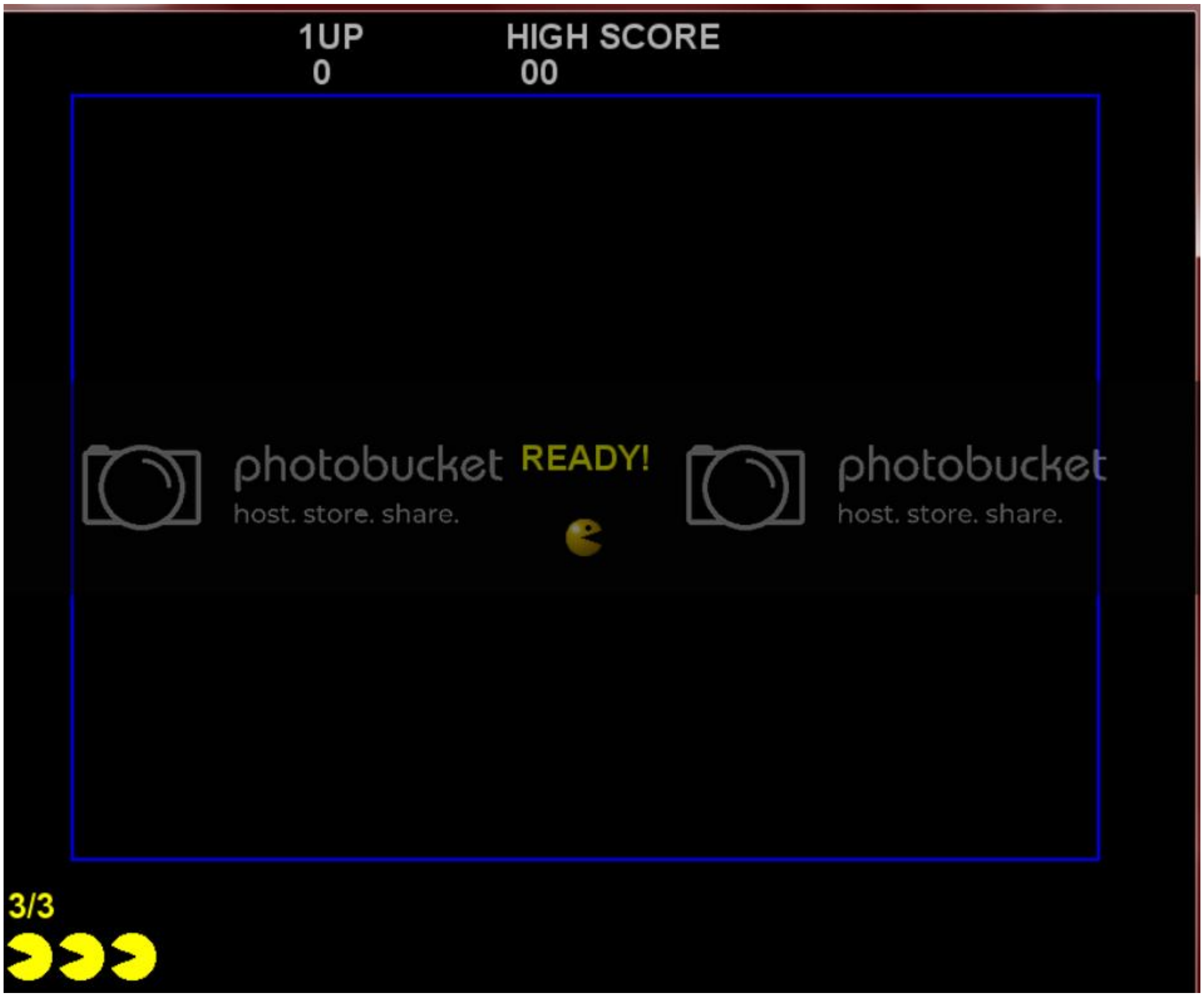
4. Technical Aspects:

- Frontend development using HTML5 Canvas or Pygame for visual rendering.
- JavaScript/Python-based backend logic for AI, collisions, and game rules.
- Optional database (like Firebase or SQLite) to store high scores or player profiles.
- Cloud deployment via platforms like GitHub Pages or Replit for easy access.

The design output ensures that Pacman Game delivers an interactive, nostalgic, and fun gaming experience while showcasing strong logic, responsiveness, and user engagement. Let me know if you need a visual wireframe or flowchart for this design!

PROJECT SCREENSHOTS


~PacMan Remake~
PAC-MAN
REMAKE
Press 's' to start.
photobucket host. store. share.
photobucket host. store. share.


1UP
0
HIGH SCORE
00
READY!
3/3
photobucket host. store. share.
photobucket host. store. share.

CONCLUSION

The Pac-Man Java Game is designed to be an innovative and user-friendly application that enables players to enjoy a classic arcade experience. By leveraging well-established game mechanics, responsive controls, and intuitive visual feedback, the system provides reliable and engaging gameplay, helping users enjoy their leisure time effectively. The platform ensures a smooth user experience through a modern, responsive interface, allowing seamless interaction across desktop devices. With a well-structured game engine and secure asset management, users can confidently immerse themselves in the game and receive satisfying play insights.

The system is built with a robust backend logic, integrating real-time game state updates and artificial intelligence to enhance the gameplay experience. By using an **object-oriented and scalable architecture**, the Pac-Man Java Game ensures high availability and optimal performance. The inclusion of **clear game loop integrations** further enriches the play model by incorporating real-time movement, collision detection, and scoring updates. This makes the Pac-Man Java Game a valuable tool for individuals looking for engaging entertainment at their fingertips.

Despite its advantages, the system faces challenges such as **balancing ghost AI difficulty, potential performance variations across different hardware, and competition from other gaming platforms.** However, continuous improvements in **game logic algorithms, animation rendering, and user feedback mechanisms** can enhance its precision and reliability. Expanding the game's features, such as integrating high score leaderboards or new level designs, can further increase its usability and market reach.

In conclusion, the Pac-Man Java Game serves as a **comprehensive and engaging arcade entertainment platform** that empowers players with a timeless gaming experience. With its **technological foundation, user-centric design, and commitment to accuracy**, it stands as a promising solution for individuals looking to enjoy classic gameplay. As the platform evolves, incorporating additional features and refining its core mechanics will ensure that it remains a competitive and reliable tool in the digital entertainment industry.

BIBLIOGRAPHY

- W3C. (n.d.). *Web Standards and Best Practices*. Retrieved from www.w3.org

- Mozilla Developer Network (MDN). (n.d.). *HTML, CSS, and JavaScript for Responsive UI*. Retrieved from developer.mozilla.org

- Documentation. (n.d.). *Building Scalable Web Applications*. Retrieved from dev

- Kelley Blue Book (KBB). (n.d.). *How to Determine Your score Value*. Retrieved from www.kbb.com

- Edmunds. (n.d.). *Pricing and Trends*. Retrieved from www.edmunds.com

- CarGurus. (n.d.). *Used Car Valuation and Pricing Trends*. Retrieved from www.cargurus.com