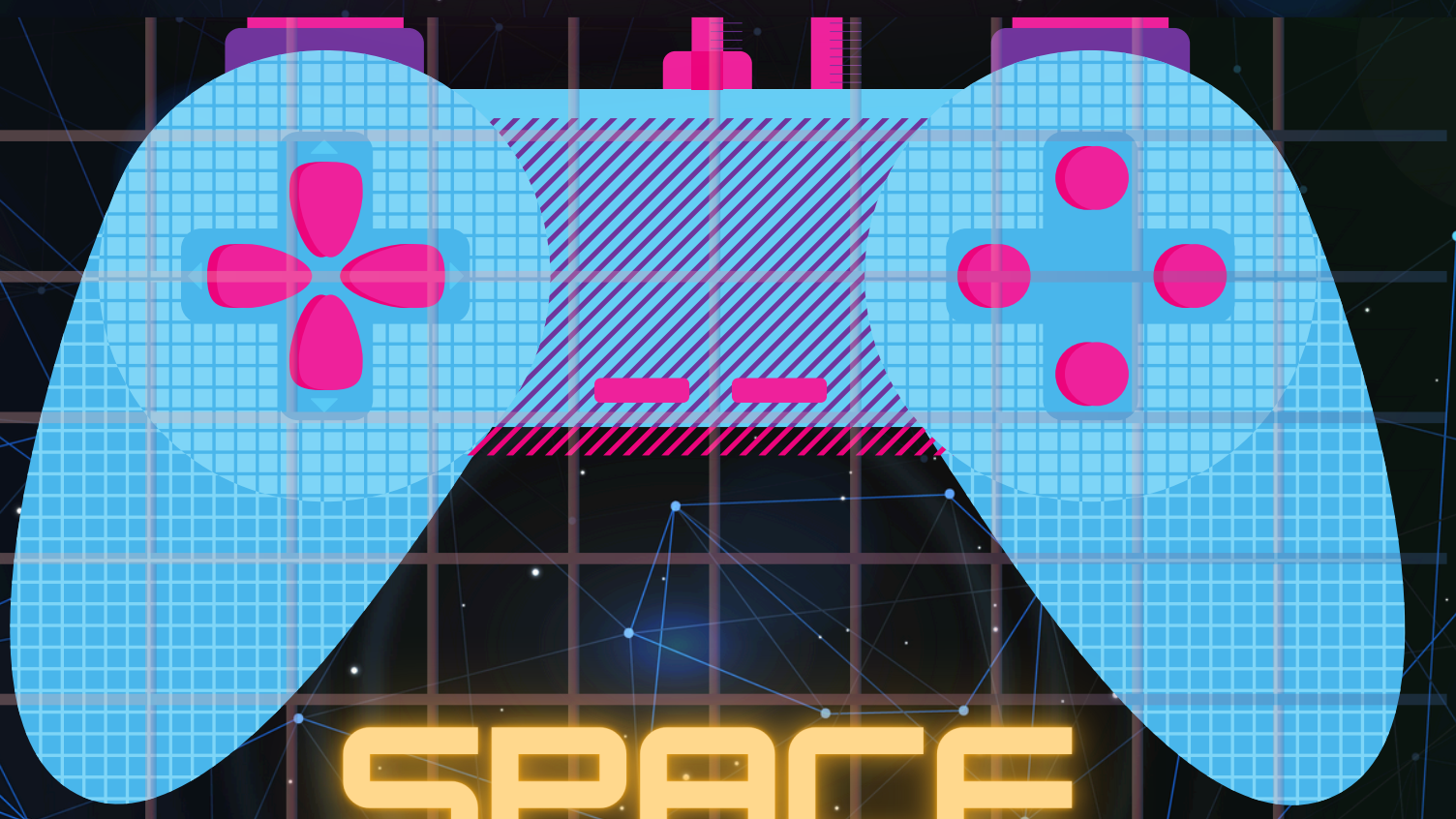
BATCH 2022-23

CEP REPORT
(SE-326)

GAME



SPACE INVADER

SUBMITTED TO
SIR ALI AKHTER

SUBMITTED BY
ZAINAB RAZA (CS-22054
DOA AHMED (CS-22067)
HIBA MAHBOOB (CS-22068)
AFFAF ARIF (CS-22069)

DEPARTMENT : CIS
SECTION: B
YEAR: THIRD

1.ABSTRACT:

Space Invaders " is a 2D arcade-style game developed using JavaScript and HTML5 Canvas. In the game, players control a spaceship that moves horizontally and shoots projectiles at waves of incoming invaders. The goal is to defeat all invaders before they reach the player. If they reach the player, the player loses and game ends. The game features an interesting background and a scoring system. It includes basic player controls, collision detection, and a user interface with a start screen, score display, and game-over screen. The project aims to demonstrate game development techniques while providing an engaging and challenging gameplay experience.

2. INTRODUCTION

2.1 BACKGROUND

Space Invaders is a legendary arcade game first introduced in 1978, renowned for its simple yet addictive gameplay. This project reimagines the classic game as a web-based version, developed using JavaScript and HTML5 Canvas. It incorporates responsive spaceship controls, wave-based enemy mechanics, smooth animations, and real-time collision detection to provide a dynamic and engaging experience. By drawing inspiration from the original, this project recreates the nostalgic charm of the classic arcade era while offering a modern gameplay experience accessible in web browsers.

2.2 OBJECTIVES

The objective of this project is to develop a modern Space Invaders game with enhanced features while maintaining the essence of the original. Key goals include implementing wave-based invader spawning, smooth animations, and real-time collision detection for a seamless gameplay experience. The game will incorporate various mechanics such as bombs with dynamic movement, background music for user enjoyment, and interactive scoring labels. Additionally, it aims to provide an intuitive user interface with a start screen, score tracker, and restart functionality. By focusing on these features, the project seeks to deliver a challenging yet enjoyable game that appeals to players of all skill levels.
objective.

2.3 SCOPE

The Space Invaders game is a JavaScript-based game that features interactive gameplay, with responsive controls for navigating a spaceship, an engaging wave-based enemy system, smooth animations, and real-time collision detection. It is built with JavaScript and HTML5 Canvas. This game allows players to control a spaceship, defending against waves of invaders descending from the top of the screen and large bombs that are being generated in various places. The goal is to eliminate all invaders while avoiding enemy projectiles and protecting the player’s life. There is functionality to display the player’s score, showcasing their abilities. This game offers several benefits to players. It creates an engaging and enjoyable gaming experience. It has interactive controls and fast- paced gameplay, the game enhances players’ reflexes and hand-eye coordination as they move the spaceship to dodge projectiles and eliminate waves of invaders. As people are nostalgic for classic arcade games, this version of Space Invaders provides a fun way to relive the experience right in a web browser. The game’s design recreates a nostalgic image for players to remember old arcade game

3. REQUIREMENT ANALYSIS

3.1 FUNCTIONAL REQUIREMENTS

3.1.1 GAMEPLAY FUNCTIONS

1. **Move Spaceship:** The player must be able to move the spaceship horizontally across the screen to dodge enemy projectiles and position for attacks. The spaceship should respond to 'a', 'd' and 'spacebar' to move left or right without lag or delay.

2. Shoot: Allows the player to fire projectiles at invaders. When the player presses the "shoot" button 'spacebar', a projectile is launched from the spaceship.

3. Collision Detection: Detects collisions between projectiles, invaders, and the player's spaceship. The game should accurately detect collisions and trigger appropriate responses, such as ending the player's life upon collision with an invader or awarding points upon hitting an invader.

4. Score Points: Awards points to the player for destroying enemies or achieving objectives. Each enemy hit increases the player's score. This score is displayed on the screen in real time and updates dynamically.

5. Lose Life: If the spaceship collides with an enemy or a projectile, the player loses their life and the game ends.

3.1.2 USER INTERFACE FUNCTIONS

1. Display Score : This shows the player's score on the screen during gameplay. The score should be updated in real time, located prominently on the game screen for easy visibility.

3.1.3 ADMINISTRATIVE FUNCTIONS

1. Configure Game Settings: Allows the administrator to adjust game settings, such as difficulty, player lives, and sound options. The administrator can change parameters to fine-tune game difficulty or enable/disable certain features (e.g., background music).

2. Manage Score Data: Allows the administrator to access and modify score data. The administrator can reset scores, view past game data, or adjust scoring rules as needed.

3.1.4 AUDIO AND VISUAL EFFECTS

1. Background Music: Plays background music during gameplay. The background music file should start on game launch and loop throughout the gameplay.
2. Sound Effects: Plays sound effects in response to specific actions, like shooting, collisions, and game over. Each action should trigger the appropriate sound effect (e.g., shooting or explosion sounds) with minimal delay.

3.1.5 GAME OVER AND RESTART FUNCTIONS

1. Display Game Over Screen: Shows a "Game Over" message when the player's life has ended. A screen appears with the final score and options to restart the game.

2. Restart Game: Allows the player to reset the game from the beginning. Upon selecting the restart option, the game resets to its initial state, including resetting the score, lives, and level to the default starting values.

3.2 NON- FUNCTIONAL REQUIREMENTS

1. Sound Effects: All sound effect files (such as enemyShoot.wav, gameOver.mp3, shoot.wav, bonus.mp3, and explode.wav) should play within 5 milliseconds of triggering the related game events to ensure immediate feedback for actions like shooting, game over, and explosions.

2. Game Over Sound Delay: The gameOver.mp3 file, which is 79.3 kB, should play within 3 milliseconds of the player losing all lives, to enhance user experience by providing immediate audio feedback.

3. Memory Usage: The game should operate using a maximum of 100 MB of RAM during typical gameplay on standard devices. This limitation ensures that the game remains accessible on a broad range of hardware without memory overflow issues.

4. Game Start-Up Time: The game should load and initialize within 5 seconds on standard hardware to minimize user waiting time and provide a fast experience.

5. Frame Rate: The game must maintain a frame rate of at least 60 frames per second (fps) during normal gameplay to ensure smooth animations and reduce latency in player actions.

6. Collision Detection Response: Collision events, such as between a projectile and an invader or spaceship, must be processed within 50 milliseconds to maintain the real-time, interactive nature of the game and give accurate visual feedback.

7. Input Response Time: User input actions such as moving the spaceship or firing a projectile, should be registered within 30 milliseconds to provide an accurate control experience. This is because any noticeable delay can negatively impact gameplay.

3.3 USE CASE

GAMEPLAY FUNCTIONS

Player Movement:

As a player, I want to move the spaceship horizontally across the screen using the 'a' and 'd' keys so that I can dodge enemy projectiles and position myself to attack invaders.

Acceptance Criteria: The spaceship must move smoothly without lag or delay and stop at screen boundaries.

Shooting Projectiles:

As a player, I want to press the 'spacebar' to shoot projectiles at the invaders so that I can destroy them and score points.

Acceptance Criteria: The projectile should launch instantly, travel upwards, and detect collisions with invaders or objects.

Collision and Scoring:

As a player, I want accurate collision detection so that I can eliminate invaders and avoid losing the game due to undetected impacts.

Acceptance Criteria: Points are awarded when projectiles hit invaders, and the game ends if the spaceship collides with an enemy or projectile.

Game Over and Restart:

As a player, I want to see a Game Over screen with my final score when I lose so that I know my progress.

Acceptance Criteria: The Game Over screen displays a restart option to reset the game to its initial state.

User Interface Functions

Score Display:

As a player, I want my score to be visible during gameplay so that I can track my performance in real time.

Acceptance Criteria: The score updates dynamically and is prominently displayed.

Game Start and Restart:

As a player, I want to easily start or restart the game from a menu screen so that I can quickly engage in gameplay.

Acceptance Criteria: Start and restart buttons should work without delay and reset all game variables.

AUDIO AND VISUAL EFFECTS

- Audio Feedback:

As a player, I want sound effects for actions like shooting or collisions so that I receive immediate feedback during gameplay.
Acceptance Criteria: Sound effects play within 5 milliseconds of events like shooting or game over.

- Background Music:

As a player, I want looping background music during gameplay to enhance the atmosphere.
Acceptance Criteria: The background music starts automatically at game launch and loops seamlessly.

3.4 Data Requirements

- Gameplay Data:

Player position (x, y) for movement.
Invader position and state (alive or destroyed).
Projectile positions and velocities.
Collision state (projectile-invader, invader-player).

- Score Data:

Player's current score, dynamically updated.
Final score displayed upon game over.

- Game State Data:

Game status (active or over).
Wave progression or number of invaders left.

- Performance Metrics:

Frame rate tracking for smooth animations
Collision detection response times

- Input response times Audio Assets:

File paths for background music and sound effects (e.g., gameOver.mp3, shoot.wav).
Volume levels for each sound type.

- Visual Assets:

Background image and game screen components (start menu, score display, game over screen).
Animations and particle effects for explosions and collisions.

4. System Design

4.1 Architecture Overview

The architecture of the Space Invaders game follows a model where all gameplay, rendering, and user interactions are handled directly within the browser. The system is structured to provide an interactive and smooth gaming experience by utilizing several key components that work together seamlessly.

System Components:

User Interface (UI): Built with HTML5 and CSS, the UI includes game components such as the start screen, score display, and game-over screen. Canvas API: The game uses the HTML5 Canvas API to render dynamic 2D graphics, including the spaceship, invaders, projectiles, explosions, and background animations.

Game Logic: The core gameplay logic is written in JavaScript. This includes handling player input, updating the game state, managing enemy movements, collision detection, and tracking scores. Audio Management: Sound effects and background music are handled by the Howler.js library. It has sounds triggered for actions like shooting, enemy movements, explosions, and the game-over event.

Data Flow: Data flows through the system as player actions (e.g., moving the spaceship or shooting) trigger updates to the game state. The game state includes variables for player position, invader positions, projectiles and score. These updates are immediately reflected on the canvas in real time. For example, when a projectile hits an invader, the collision is detected, the score is updated, and an explosion animation is triggered.

User Interaction: Players interact with the game via the keyboard, using the 'a' and 'd' keys to move the spaceship left and right, and the spacebar to shoot projectiles.

SYSTEM FLOW:

The game starts at a start screen with a button to begin the game.
Once the game starts, the player can move the spaceship, shoot projectiles, and engage with waves of invaders.
The player loses when their spaceship collides with an invader or an enemy projectile, at which point a "Game Over" screen appears.

4.2 Technology Stack

- **FRONTEND:**

HTML5: Structure and game canvas.
CSS: Styling and layout for UI components.
JavaScript: Core logic for gameplay, animations, and interactions.

- **GRAPHICS RENDERING:**

HTML Canvas API: 2D rendering for all visual elements.

- **AUDIO MANAGEMENT:**

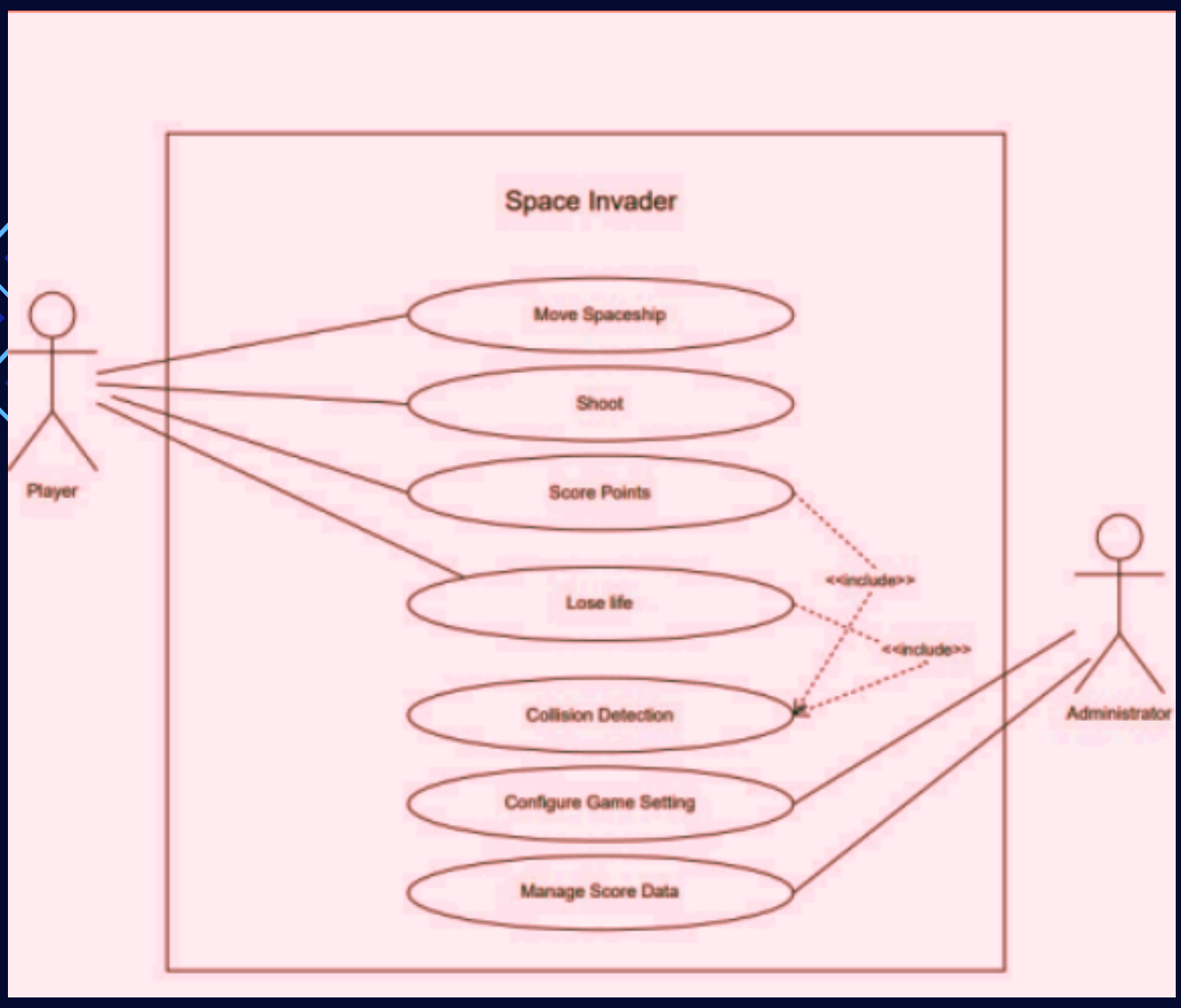
Howler.js: Handling sound effects and background music.

- **DEVELOPMENT TOOLS:**

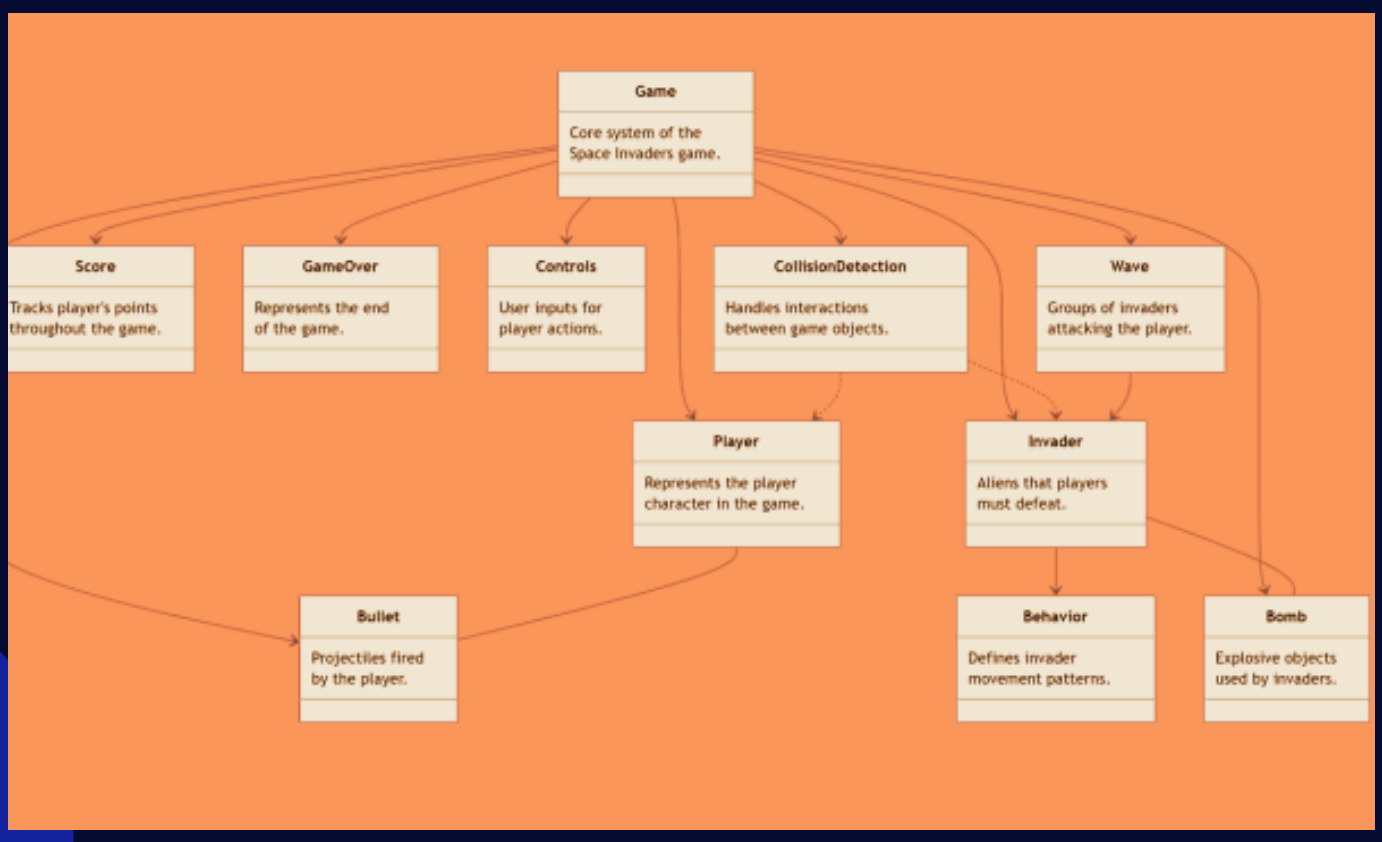
Code editor: VS Code.

4.3 UML Diagrams

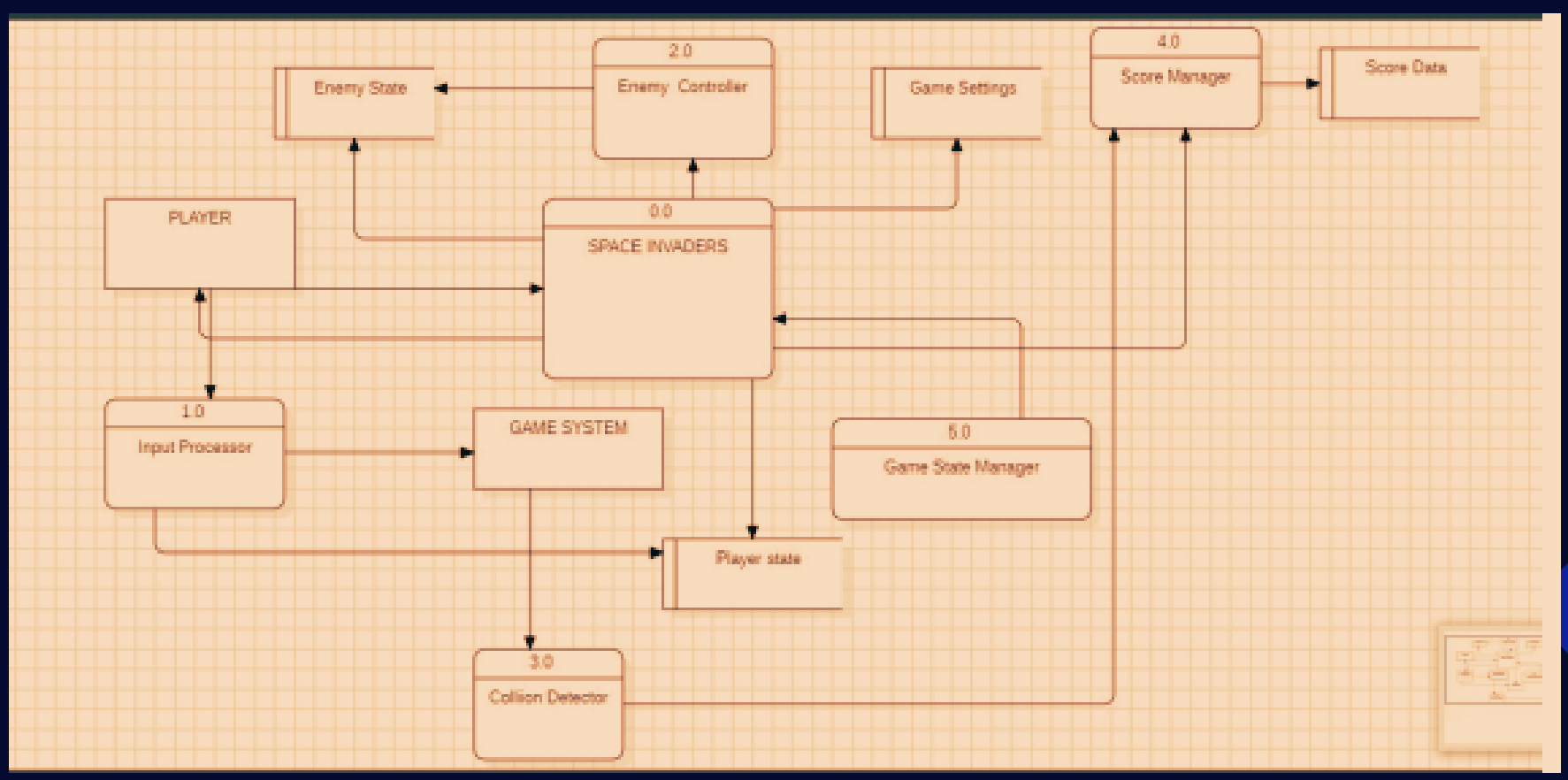
1. Use Case Diagram:



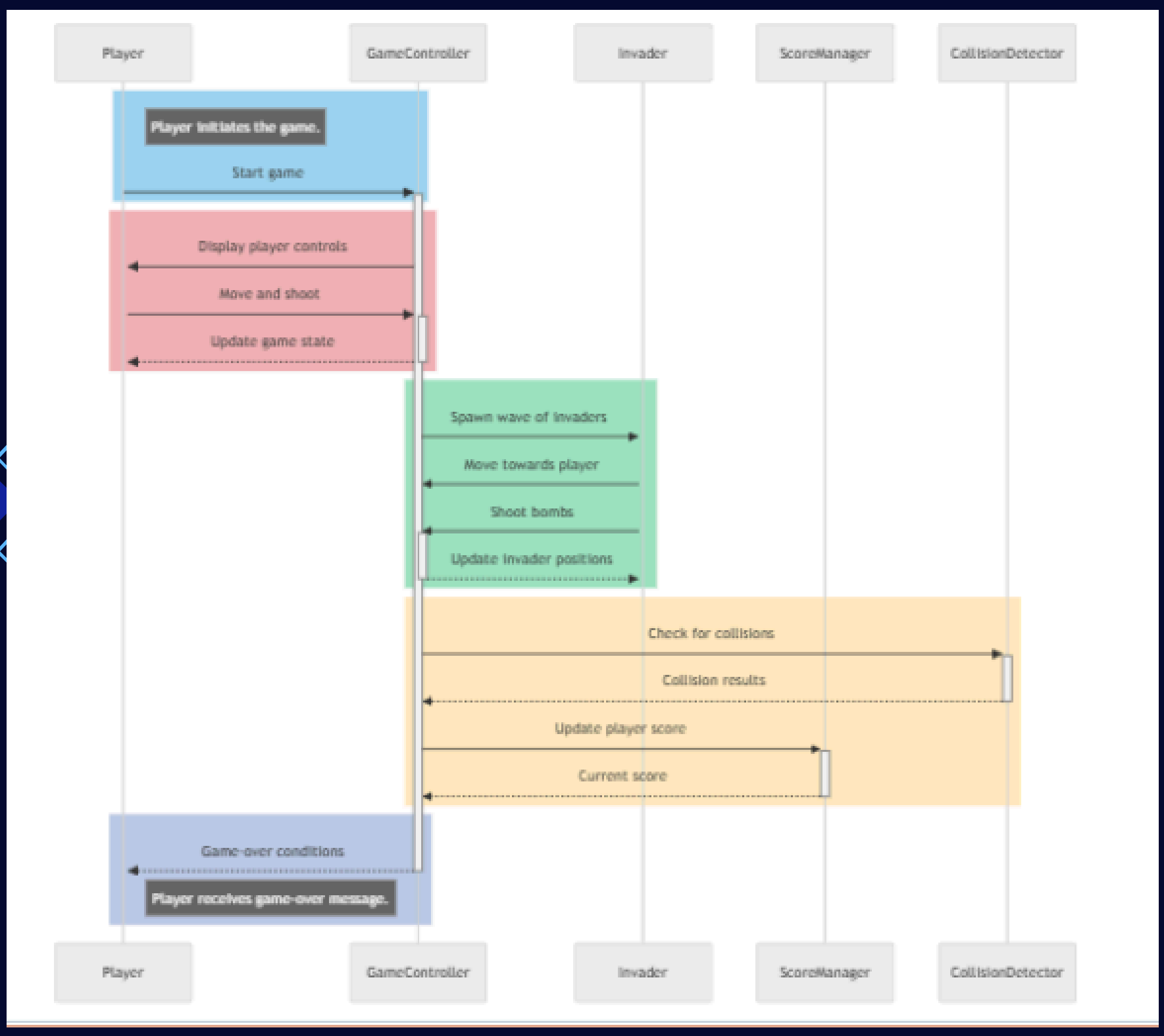
2. Class Diagram:



3. Data Flow Diagram:



4. Sequence State Diagram:



5. Implementation

5.1 Development Environment

The Space Invaders game was developed in a browser-based environment, using modern web technologies. The primary development tools and platforms used in this project are as follows:

Code Editor: The project was developed using Visual Studio Code (VS Code), which provides features like syntax highlighting, IntelliSense, and Git integration.
Web Browser: The game was tested and run in modern browsers like Google Chrome and Mozilla Firefox, both of which fully support HTML5 Canvas and JavaScript, ensuring cross-browser compatibility.

Libraries Used:

- HTML5 Canvas: The game uses the Canvas API for rendering 2D graphics, including objects like the spaceship, invaders, and projectiles.
- Howler.js: Used for managing sound effects and background music in the game. It allows for smooth audio playback with minimal delay.

Operating System: The game was developed and tested on both Windows and Mac OS platforms.
File Organization: The project files are organized into directories:

- css: Contains styling files for the game's interface.
- js: Holds JavaScript files for the game logic, including classes for different entities (Player, Invader, Projectile, etc.).
- audio: Includes sound files used in the game for background music and sound effects.
- img: Stores image assets for visual elements like backgrounds and buttons.

5.2 CODE STRUCTURE

The code structure is modularized to maintain clarity and facilitate future development. The files and directories are organized as follows:

- HTML (index.html): Contains the structure of the game, including the <canvas> element for rendering and links to external resources like styles and scripts. It also defines the UI components, such as the start screen and game-over screen.
- CSS (index.css): Provides the styles for the UI, such as positioning the score display, the start screen, and the game-over screen.
- JavaScript:
  - audio.js: Contains all the sound management logic. It initializes audio files for background music and sound effects using Howler.js.
  - index.js: The main game loop and logic are implemented here. It includes features like player input handling, collision detection, scoring, and updating the game state.
  - classes: Contains various classes that define the game entities:
    - Player.js: Handles the player's spaceship, its movement, and shooting behavior.
    - Projectile.js: Defines the projectiles fired by the player, including their movement and collision detection.
    - Invader.js: Represents invader enemies, including their movement and shooting behavior.
    - Grid.js: Manages the organization and movement of invader grids.

- Bomb.js: Handles randomly falling bombs that add difficulty to the gameplay.
- Particle.js: Manages particle effects, such as explosions and visual feedback.
- InvaderProjectile.js: Defines projectiles fired by invaders.

img: Contains image files such as the background image, game buttons, and other graphical assets for the user interface.
audio: Holds all the audio files used in the game, including sound effects and background music (e.g., shoot.wave, explode.wav, gameOver.mp3).

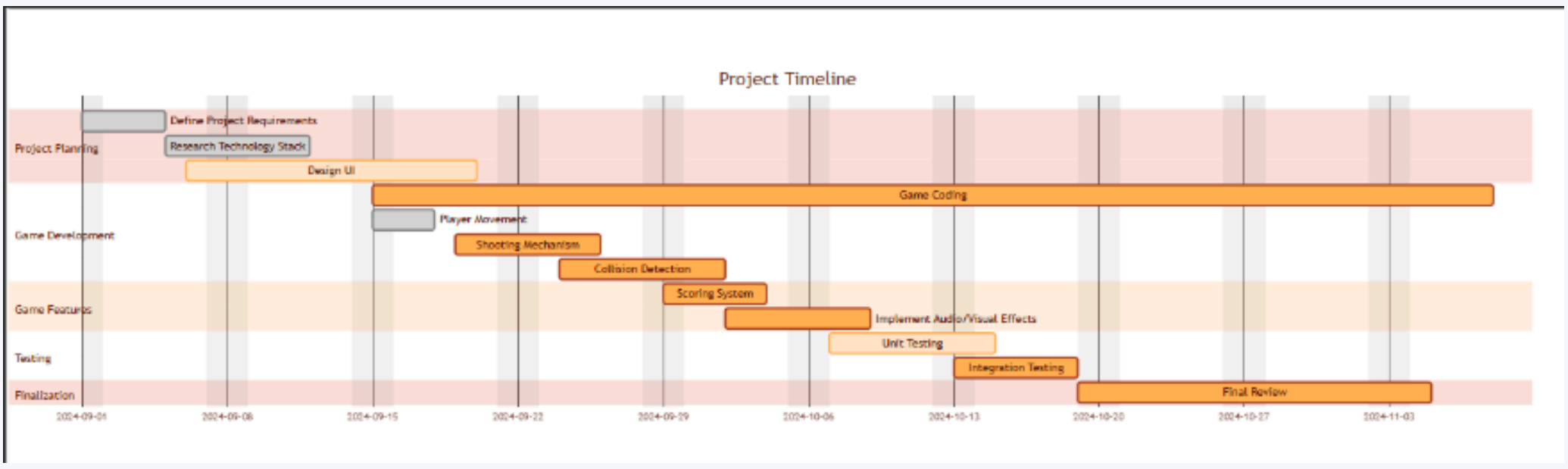
5.3 Key Features

- Player Movement: The spaceship moves smoothly left and right using the 'a' and 'd' keys, staying within the screen boundaries to avoid going off-screen.
- Shooting Mechanism: The player can shoot projectiles by pressing the spacebar, with projectiles moving upwards to hit invaders.
- Collision Detection: Real-time detection of collisions between projectiles and invaders, awarding points for hits and ending the game if the spaceship collides with an invader or projectile.
- Score System: The score is dynamically updated in real time, increasing with each enemy destroyed, and displayed at the top-left corner of the screen.
- Enemy Wave System: Invaders appear in waves, with increasing speed and spawn rate as the game progresses, raising the difficulty.
- Bombs and Power-Ups: Randomly falling bombs must be avoided.
- Game Over and Restart: When the player loses their life, a Game Over screen appear with the option to restart the game, resetting the score and life.

Audio and Visual Feedback: Sound effects are played for actions like shooting and explosions, and background music enhances the gaming atmosphere, with visual effects like explosions and score labels providing instant feedback.

6. Project Plan

6.1 Project Timeline



6.2 Resource Allocation

In this section, list all the team members, their roles, and the hardware/software resources required for the successful completion of the project.

Team Members and Roles:

- Hiba – Project Lead, Game Logic Development. Hiba will oversee the development of core game logic, including handling player controls, collision detection, and score tracking.
- Affaf – UI Designer and Audio Manager. Affaf will design the user interface, including the start and game-over screens. She will also manage the integration of audio features using Howler.js.
- Zainab – Graphics and Testing . Zainab will handle game graphics, such as designing sprites for the spaceship, invaders, and explosions. She will also manage unit testing and debugging.
- Doa – Backend Development .Doa will handle backend functionalities like ensuring smooth integration between different components, and working on performance optimization.

Hardware and Software Resources:

Hardware: Computers with internet access for coding, testing, and deployment. Devices (PCs or laptops) with modern web browsers (Chrome, Firefox) for testing the game across different platforms.

Software:

Visual Studio Code: For writing and debugging the code.

Google Chrome/Firefox: For testing the game in different web browsers.

Howler.js: For managing audio in the game.

HTML5 Canvas: For rendering the game environment.

Git/Github For sharing files and collaborating as a team.

6.3 Risk Management

Risk management involves identifying potential risks, analyzing their impact, and defining strategies to mitigate them.

- Risk: Project Delays

Impact: Delays in development could affect the overall project timeline, leading to incomplete features or missed deadlines.

Mitigation: Set realistic deadlines and prioritize tasks based on their complexity. Regularly track progress using the Gantt chart and adjust as needed.

- Risk: Technical Challenges

Impact: Issues with cross-browser compatibility or audio integration could disrupt gameplay and cause delays.

Mitigation: Test the game early and often on different browsers (Chrome, Firefox, Safari) and ensure that all core features, including audio, are working as expected. Use libraries like Howler.js to handle audio compatibility across browsers.

- Risk: Lack of Resources or Team Availability

Impact: Team members may not be available due to personal or other commitments, which could slow down development.

Mitigation: Ensure team members have clear responsibilities and maintain open communication. If needed, adjust tasks based on availability and allocate extra time for key tasks.

- Risk: Bugs and Performance Issues

Impact: Bugs and performance issues (e.g., frame rate drops) could impact the game's usability and user experience.

Mitigation: Conduct unit and integration testing regularly, especially during the development phase. Use tools for performance testing, and optimize the code to handle a smooth frame rate.

- Risk: Scope Creep

Impact: The scope of the project may expand beyond the original plan, leading to missed deadlines or overcomplication.

Mitigation: Clearly define the scope at the beginning of the project and stick to it. Prioritize essential features and avoid adding unnecessary functionality unless time permits.

9. RESULTS AND ANALYSIS

9.1 PERFORMANCE METRICS

The performance of the Space Invaders game was evaluated based on several predefined metrics. The game successfully maintained a stable frame rate of 60 frames per second (fps) during normal gameplay, ensuring smooth animations and responsiveness. Collision detection between projectiles and invaders, as well as between the player's spaceship and projectiles, was processed within 50 milliseconds, providing accurate real-time feedback. Input response times for actions like moving the spaceship and firing projectiles were under 30 milliseconds, creating a seamless control experience. Additionally, audio playback performed reliably, with sound effects triggered within 5 milliseconds of corresponding actions, and background music looping without noticeable interruptions.

9.2 USER FEEDBACK

User feedback was collected from individuals who tested the game, focusing on aspects like gameplay, controls, and audio-visual elements. Testers found the gameplay engaging and challenging, particularly appreciating the wave-based enemy system and the introduction of power-ups like the "Bomb." The controls were described as responsive and intuitive, with smooth movement and shooting mechanics. Audio and visual effects were highly rated for enhancing immersion, particularly the explosions and background music. Some testers suggested including more levels and difficulty settings for extended play.

9.3 COMPARISONS

The developed game was compared to the original Space Invaders and similar browser-based games. Compared to the original, the new version featured modernized graphics, smoother animations, and additional features like power-ups and a dynamic scoring system. Against other web-based games, the Space Invaders project was praised for its responsive controls, engaging gameplay, and polished visual effects, though it was noted that adding more levels and a leaderboard could enhance its competitive appeal.

10. CHALLENGES AND SOLUTIONS

The development of the Space Invaders game presented several challenges, which required thoughtful solutions to ensure the project's success. One of the primary challenges was managing time constraints. With a strict project deadline, overlapping phases like coding, testing, and debugging made it difficult to allocate sufficient time to each task. This issue was mitigated by creating a detailed Gantt chart, breaking tasks into smaller milestones, and regularly tracking progress to stay on schedule. Team members also collaborated effectively to divide responsibilities and address delays when they occurred.

Another challenge was conducting integration testing after the coding phase. Ensuring that all individual components (e.g., player controls, scoring system, collision detection, and audio) worked seamlessly together was a complex process. Initial integration tests revealed bugs, such as incorrect score updates and delayed collision responses. To address these issues, the team used a systematic approach to isolate and resolve conflicts between modules. Testing was performed incrementally, focusing on one feature at a time to confirm its compatibility with the others.

Technical challenges, such as maintaining consistent browser compatibility and achieving real-time collision detection, were also significant. Browser inconsistencies in handling audio playback were resolved by utilizing Howler.js, ensuring reliable audio across all modern web browsers. For collision detection, optimized algorithms were implemented to process events within 50 milliseconds, maintaining the game's real-time feedback and interactive nature.

11. FUTURE WORK

The Space Invaders game provides a solid foundation for further development and feature enhancements. One significant improvement would be the introduction of additional levels, allowing players to progress through increasingly challenging stages. Each level could feature unique enemy formations, faster invader movement, and varying difficulty settings, keeping players engaged and motivated to improve their skills.

Another enhancement would be the integration of a high score system. A leaderboard, powered by a backend database, could store and display players' scores, enabling them to compete globally or locally. This feature would encourage competitive gameplay and increase replayability as players strive to outperform others.
The game could also be expanded to include a multiplayer mode, allowing two or more players to either cooperate or compete. In cooperative mode, players could team up to defeat invaders together, while in competitive mode, they could race to score the highest points within a given time. Multiplayer functionality could be implemented using WebSockets or similar technologies for real-time communication, enabling smooth interactions between players.

12. CONCLUSION

The Space Invaders project successfully achieved its primary objective of developing a functional, engaging, and visually appealing game. Core features such as responsive controls, real-time collision detection, wave-based enemies, and dynamic scoring were implemented effectively, providing players with a challenging and enjoyable experience. The project demonstrated the team’s ability to apply programming and design skills to a practical task, highlighting the importance of structured planning and testing in game development. While there is scope for future improvements, the current version serves as a testament to the potential of browser-based games and the learning outcomes achieved through this project.