Software Design Description for Asterisk

Version 1.0

Prepared by:
Rohith Mahesh (57)
Diyana Sadath (33)
Sathyaki Varma (61)
Asiya Muhammed (26)

In collaboration with TCS

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Purpose

This Software Design Document (SWDD) describes the architecture, design decisions, and components of *Asterisk*, a therapeutic 2D web-based mobile game. The purpose of this document is to provide a detailed blueprint for the implementation of the software, ensuring that all components align with the functional and non-functional requirements outlined in the Software Requirements Specification (SRS).

This document serves as a reference for:

- **Developers**, to implement the system efficiently.
- **Project Managers (TCS)**, to track progress and ensure adherence to design guidelines.
- **Testers**, to validate whether the implementation meets the specified design.
- **Therapists and Researchers**, to understand how the design supports rehabilitation objectives.

1.2 Scope

Asterisk is a web-based therapeutic game designed to enhance gross motor control, core stability, and hand-eye coordination in individuals with motor impairments, particularly cerebral palsy patients. The game provides an engaging, interactive environment where users tilt their mobile device to navigate a spaceship and collect stars in predefined physiotherapy-inspired movement patterns.

The system is designed to:

- **Utilize motion-based controls** via a smartphone's accelerometer and gyroscope.
- **Support multiple user classes**, including therapists, stroke survivors, and general users.
- **Provide real-time feedback and progress tracking** for therapists to monitor user improvement (Leaderboard System).
- **Stream sensor data via WiFi** to enhance motion tracking accuracy.
- **Ensure cross-platform compatibility**, allowing the game to run on both **PC browsers** while using Mobile Devices as a controller.

1.3 Overview

This document details the design and structure of *Asterisk* and is organized as follows:

- **Section 2: System Overview** – Provides a high-level description of the system and its functionality.
- **Section 3: System Architecture** – Describes the software's modular structure and interactions between components.
- **Section 4: Data Design** – Explains data structures, storage, and processing methods.
- **Section 5: Component Design** – Defines each software component and its responsibilities.
- **Section 6: Human Interface Design** – Details the user interface, including screen layouts and interactions.
- **Section 7: Requirements Matrix** – Maps functional requirements from the SRS to corresponding system components.

1.4 Reference Material

- **Software Requirements Specification (SRS) Version 1.0 for Asterisk**
- **IEEE Std 1016-2009** – IEEE Recommended Practice for Software Design Descriptions

1.5 Definitions and Acronyms

- **CP** – Cerebral Palsy
- **UI** – User Interface
- **SRS** – Software Requirements Specification
- **SDD** – Software Design Document
- **DFD** – Data Flow Diagram
- **HyperIMU** – A mobile application used for real-time motion tracking
- **Tilt Control** – Game mechanism where players control movement by tilting their device

2.0 SYSTEM OVERVIEW

Asterisk is a **therapeutic 2D web-based game** designed to improve **hand-eye coordination, upper body motor skills, and core stability** in individuals with **cerebral palsy and other motor impairments**. The game provides an engaging environment where users **tilt their mobile device** to navigate a **spaceship** and collect stars arranged in **physiotherapy-inspired movement patterns**.

Key Features:

- **Tilt-based motion control** using a mobile device's **accelerometer and gyroscope**.
- **Pre-designed therapeutic exercise patterns** that mimic physiotherapy movements.
- **Progress tracking and real-time feedback** for therapists and users.
- **Data streaming via USB/WiFi** using **HyperIMU** for precise motion tracking.
- **Customizable difficulty settings** to adapt to different motor skill levels.
- **A 3D-printed mobile holder** for improved device stability during gameplay.

System Context and Interaction

Asterisk operates as a **web-based mobile game**, requiring two devices for optimal use:

1. **A PC (or other computing device)** to run the game via a web browser.
2. **A smartphone (motion controller)** placed inside a **3D-printed ring holder** to detect tilt-based movements.

The system processes real-time **motion sensor data** from the mobile device and translates it into **game controls**. Users navigate the spaceship by tilting the phone, and the game provides feedback on **movement accuracy and stability**.

Primary Users and Their Roles:

- **Cerebral Palsy Patients** – Use the game as part of rehabilitation exercises.
- **Therapists** – Monitor patient progress and adjust therapy goals.
- **Stroke Survivors & Parkinson's Patients** – Use the game for motor skill rehabilitation.
- **Older Adults & General Users** – Engage with the game for light motor exercises and relaxation.

Operating Environment:

- **Platforms:** Web-based (accessible via PC browsers).
- **Devices:** Mobile phones (for tilt controls), PC (for game display).
- **Sensors:** Accelerometer & gyroscope for motion tracking.
- **Connectivity:** WiFi for real-time data streaming.
- **Software Dependencies:** HyperIMU (motion tracking), Unity (game rendering).

3.0 SYSTEM ARCHITECTURE

The system architecture of Asterisk is designed to leverage a smartphone's IMU (Inertial Measurement Unit) sensors to enable real-time spacecraft navigation. The system follows a modular, event-driven design, utilizing a UDP-based communication protocol to ensure low-latency data transmission between the IMU sensor module and the game engine.

This section describes the high-level structure of the system, how different subsystems interact, and the rationale behind our architectural choices.

3.1 Architectural Design

The system follows a client-server model, where the smartphone acts as the IMU-based controller, and the game engine interprets and executes spacecraft movements accordingly. The major components of the system are:

1. IMU Sensor Module (Client - Smartphone)

- **Reads real-time motion data** (accelerometer, gyroscope, magnetometer).
- Formats and transmits sensor data via **UDP packets** to the game engine.
- Provides a **lightweight interface** to configure sensor sampling rates and sensitivity.

2. Communication Layer

- Implements a **UDP-based networking protocol** for **fast and continuous** data streaming.
- Ensures efficient **packet handling** to reduce data loss and maintain synchronization.
- Uses a **non-blocking socket implementation** to prevent communication delays.

3. Game Engine Input Handler (Server - PC)

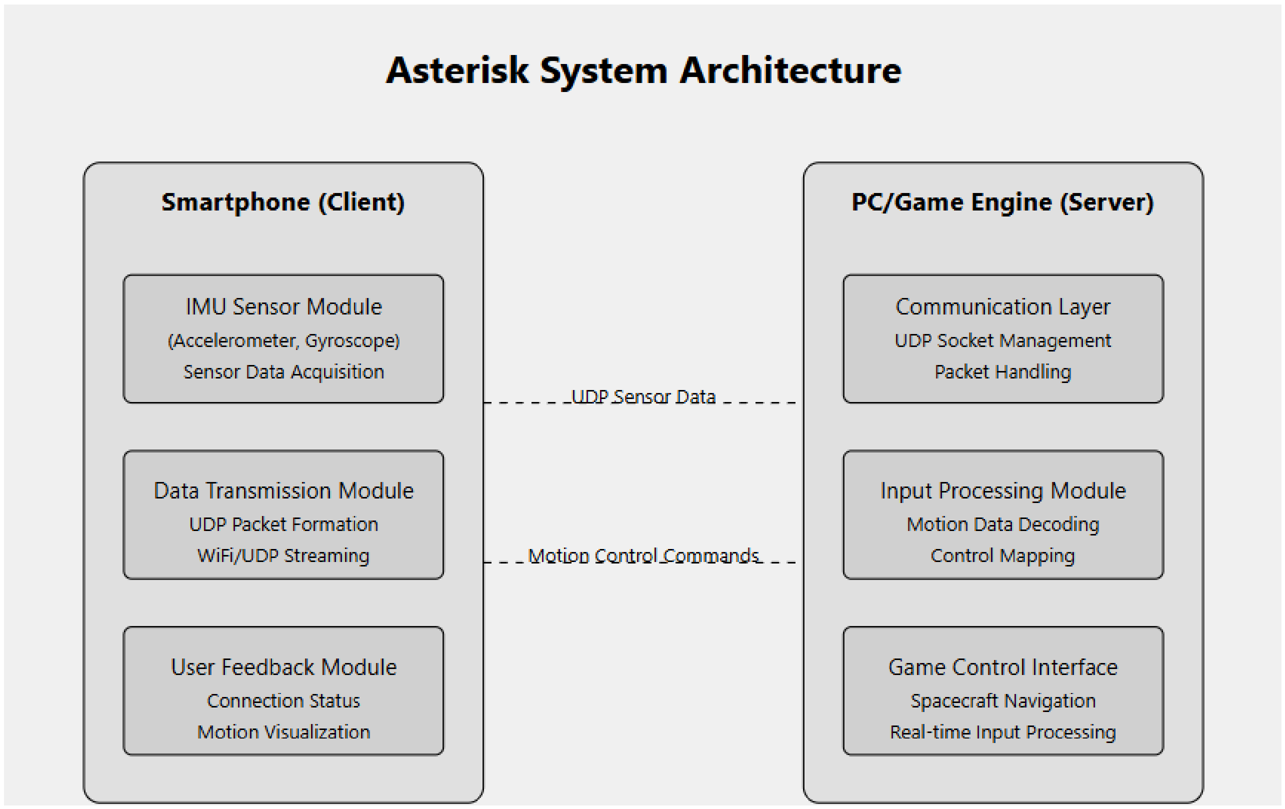
- Receives UDP packets, **decodes** motion data, and converts it into control inputs.
- Uses a **custom mapping algorithm** to translate IMU sensor values into spacecraft movement.

- Applies **thresholds and scaling factors** to fine-tune responsiveness.

4. User Feedback System

- Displays **real-time sensor data** and connection status on the smartphone UI.
- Implements an **error-handling mechanism** to detect packet loss and connection failures.
- Provides **visual cues** (e.g., movement indicators) to help users adjust their control motions.

System Architecture Diagram:



3.2 Decomposition Description

The system can be broken down into the following key subsystems:

A. Sensor Data Acquisition Subsystem (Smartphone)

- Collects IMU sensor readings (Accelerometer, Gyroscope, Magnetometer).
- Formats the data into structured UDP packets.
- Sends data over WiFi to the game engine in near real-time.

B. Networking Subsystem (UDP Communication)

- Establishes a **UDP socket connection** between the smartphone and game engine.
- Manages **data packet integrity** by handling out-of-order packets and network jitter.
- Uses a **fixed packet format** to ensure efficient parsing.

C. Input Processing Subsystem (Game Engine)

- Receives and decodes IMU data packets.
- Applies **custom transformation functions** to map sensor data to spacecraft movements.
- Filters out **unexpected motion spikes** to prevent erratic behaviour.

D. Control Mapping Subsystem

- Implements a **mapping logic** that translates IMU motion into game control inputs.
- Provides **adjustable sensitivity settings** to allow fine-tuning.
- Supports **multiple motion modes** (e.g., tilt-based, rotational, mixed).

E. User Feedback Subsystem

- Displays **real-time orientation** of the spacecraft based on motion inputs.
- Provides **status indicators** for network latency and connection health.
- Helps users calibrate their movements for smoother control.

3.3 Design Rationale

The architecture was chosen to balance real-time responsiveness, low latency, and ease of integration. The following design decisions were made:

1. **Why UDP instead of TCP?**
   - **UDP is faster** since it does not require handshaking or retransmissions.
   - **Low latency** is crucial for real-time motion control.
   - Packet loss is tolerable since minor data drops won't significantly affect gameplay.
2. **Why a Smartphone as the Controller?**
   - Modern smartphones have **high-precision IMU sensors**.

   - Avoids the need for **additional hardware**, making the system more accessible.
   - Wireless control allows **free movement** without being tethered to a device.
3. **Why Modular Subsystem Design?**
   - Allows **independent development** and debugging of different parts.
   - Makes it easier to integrate **future improvements** (e.g., haptic feedback, additional controls).
   - Simplifies testing by isolating network, input, and feedback components.

4.0 DATA DESIGN

4.1 Data Description

The Space Navigator system involves multiple data domains that support its therapeutic gaming functionality:

Primary Data Structures

1. **User Profile Data**
   - User identification
   - Therapeutic goals
   - Performance history
2. **Sensor Motion Data**
   - Accelerometer readings
   - Gyroscope measurements
3. **Game Session Metrics**
   - Gameplay duration
   - Movement accuracy
   - Star collection progress
   - Stability score
   - Difficulty level achieved
4. **Therapeutic Exercise Data**
   - Exercise pattern definitions
   - Movement complexity
   - Recommended difficulty levels
   - Progression tracking

Data Storage Considerations

- Local device storage for user profiles
- Cloud-based backup for performance metrics
- Encrypted storage of sensitive user information

4.2 Data Dictionary

User Profile Entities:

Entity	Type	Description
UserID	String	Unique identifier for user
UserType	Enum	Patient, Therapist, General User
MotorAbilityScore	Float	Quantitative measure of motor control
DiagnosedCondition	String	Medical condition (CP, Stroke, etc.)

Motion Tracking Entities:

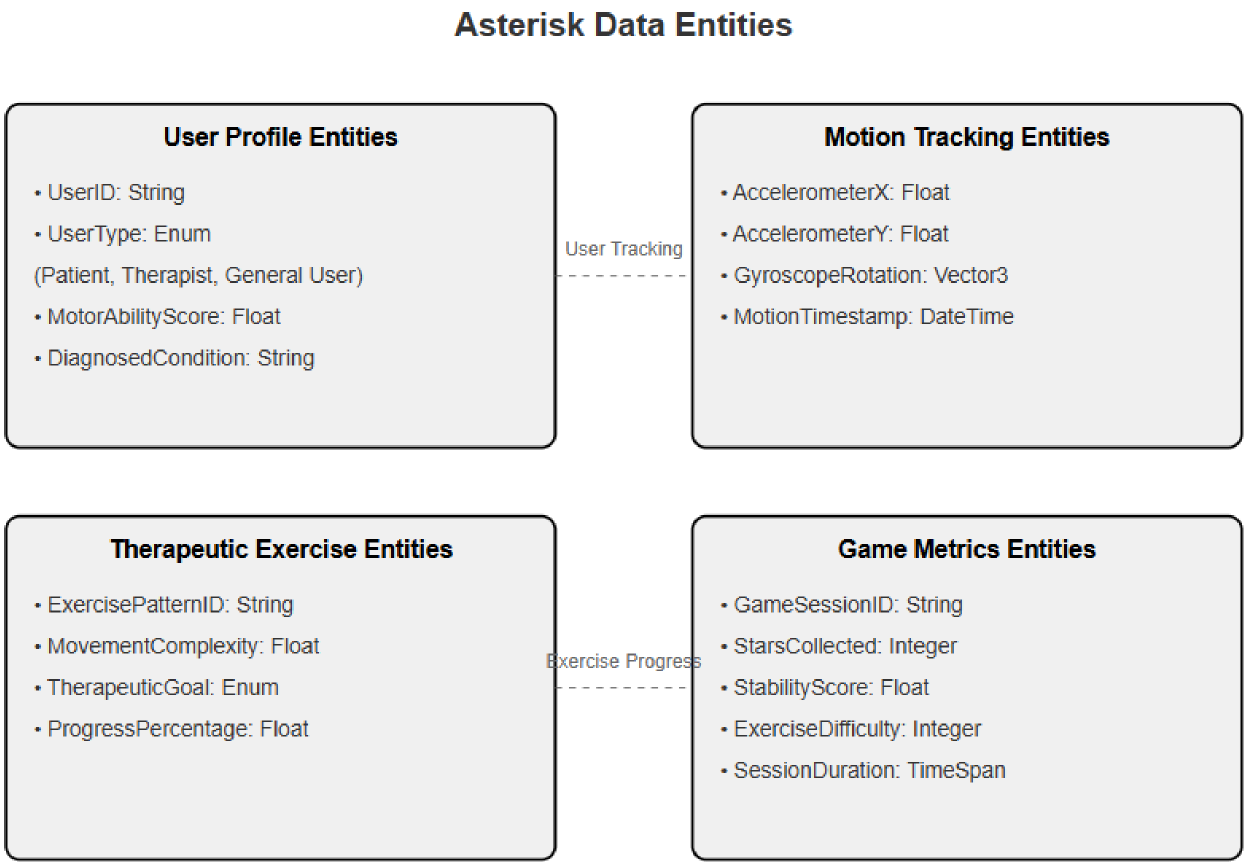
Entity	Type	Description
AccelerometerX	Float	Lateral motion acceleration
AcceleromaterY	Float	Vertical motion acceleration
GyroscopeRotation	Vector3	3D rotational movement
Motion timestamp	DataTime	Precise motion capture time

Game Metrics Entities:

Entity	Type	Description
GameSessionID	String	Unique session identifier
StarsCollected	Integer	Number of stars collected
StabilityScore	Float	Consistency of movement
ExerciseDifficulty	Integer	Current exercise challenge level
SessionDuration	TimeSpan	Length of gameplay session

Therapeutic Exercise Entities:

Entity	Type	Description
ExercisePatternID	String	Unique exercise identifier
MovementComplexity	Float	Difficulty of movement pattern
TherapeuticGoal	Enum	Specific skill targeted (coordination stability)
ProgressPercentage	Float	User's mastery of excercise


Asterisk Data Entities
User Profile Entities
• UserID: String
• UserType: Enum
(Patient, Therapist, General User)
• MotorAbilityScore: Float
• DiagnosedCondition: String
User Tracking
Motion Tracking Entities
• AccelerometerX: Float
• AccelerometerY: Float
• GyroscopeRotation: Vector3
• MotionTimestamp: DateTime
Therapeutic Exercise Entities
• ExercisePatternID: String
• MovementComplexity: Float
• TherapeuticGoal: Enum
• ProgressPercentage: Float
Exercise Progress
Game Metrics Entities
• GameSessionID: String
• StarsCollected: Integer
• StabilityScore: Float
• ExerciseDifficulty: Integer
• SessionDuration: TimeSpan

5.0 COMPONENT DESIGN

The **component design** section describes the **modular structure** of the Asterisk (Space Navigator) system, detailing its key software components, their responsibilities, and interactions. This design ensures **scalability, maintainability, and efficient real-time processing** of motion data for spacecraft navigation using a smartphone's IMU sensors.

5.1 Overview

The system follows a **modular architecture**, where each component has a well-defined role. The major components are:

1. **Sensor Data Acquisition Module** – Captures raw IMU data from the smartphone.
2. **Data Transmission Module** – Transmits IMU data packets to the game engine over a **UDP connection**.
3. **Data Processing & Interpretation Module** – Converts IMU data into actionable game control inputs.
4. **Game Control Interface** – Maps processed inputs to spacecraft navigation controls.
5. **User Interface Module** – Provides status updates, calibration options, and feedback to the user.

Each of these components interacts seamlessly to ensure **low-latency, real-time motion control**.

5.2 Component Breakdown

5.2.1 Sensor Data Acquisition Module

Description:

- This module collects real-time data from the smartphone's **accelerometer, gyroscope, and magnetometer**.

Responsibilities:

- Read sensor data at a configurable **sampling rate**.
- Normalize and format the raw data for transmission.
- Provide an option to **toggle sensors** (e.g., use only the gyroscope for rotation control).

Inputs:

- **Raw IMU sensor values** (Acceleration, Angular Velocity, Magnetic Field).

Outputs:

- **Processed sensor readings** ready for transmission.

5.2.2 Data Transmission Module

Description:

- This module establishes a **UDP-based communication channel** between the smartphone and the game engine.

Responsibilities:

- Open and maintain a **low-latency UDP socket**.
- Package IMU data into **structured packets** with timestamps.
- Ensure **efficient, loss-resistant data transmission**.

Inputs:

- **Processed sensor data** from the Sensor Data Acquisition Module.

Outputs:

- **UDP packets** containing IMU data.

5.2.3 Data Processing & Interpretation Module

Description:

- Converts raw sensor data into meaningful **motion control commands**.

Responsibilities:

- Apply **motion-to-control mapping algorithms**.
- Process sensor data to **filter noise and smooth transitions**.
- Adjust sensitivity based on **user calibration settings**.

Inputs:

- **UDP-received sensor data** from the Data Transmission Module.

Outputs:

- **Interpreted motion commands** for spacecraft navigation.

5.2.4 Game Control Interface

Description:

- This module integrates with the **game engine** to control the spacecraft's movement based on processed motion inputs.

Responsibilities:

- Translate interpreted **motion commands** into **game engine inputs**.
- Ensure **real-time responsiveness** with minimal input delay.
- Implement **failsafe mechanisms** in case of sensor errors or disconnection.

Inputs:

- **Processed motion control commands** from the Data Processing Module.

Outputs:

- **Direct game engine inputs** affecting spacecraft navigation.

5.2.5 User Interface (UI) Module

Description:

- The UI module provides **real-time feedback** to the user and enables **customization** of control settings.

Responsibilities:

- Display **sensor status** and **connection state**.
- Provide **calibration options** for fine-tuning sensitivity.
- Offer **visual indicators** of motion inputs for user guidance.

Inputs:

- **Processed motion data** and **connection status**.

Outputs:

- **Graphical feedback and control options** for the user.

5.3 Component Interaction & Data Flow

The **interaction between components** follows a structured pipeline:

1. **Sensor Data Acquisition Module** reads real-time IMU data.
2. **Data Transmission Module** packages and sends the IMU data over UDP.
3. **Data Processing & Interpretation Module** converts the data into meaningful game controls.
4. **Game Control Interface** receives commands and applies them to the spacecraft in the game engine.
5. **User Interface Module** provides real-time feedback and customization options.

Each component works **independently yet cohesively**, ensuring a **modular and scalable** system architecture.

6.0 HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The Space Navigator game interface is designed to be intuitive, accessible, and engaging for users with varying motor abilities, particularly those with cerebral palsy. The key interface design principles include:

Core Interface Features:

- Large, high-contrast buttons for easy interaction
- Simplified, clean design with minimal visual clutter
- Audio and visual feedback for all interactions
- Customizable UI elements to accommodate different user needs

Main Screen Components:

1. **Spaceship Control Area**
   - Central screen area where the spaceship navigates
   - Visible star patterns representing therapeutic movement paths
   - Real-time motion tracking visualization
   - Clear boundary indicators for motion challenges

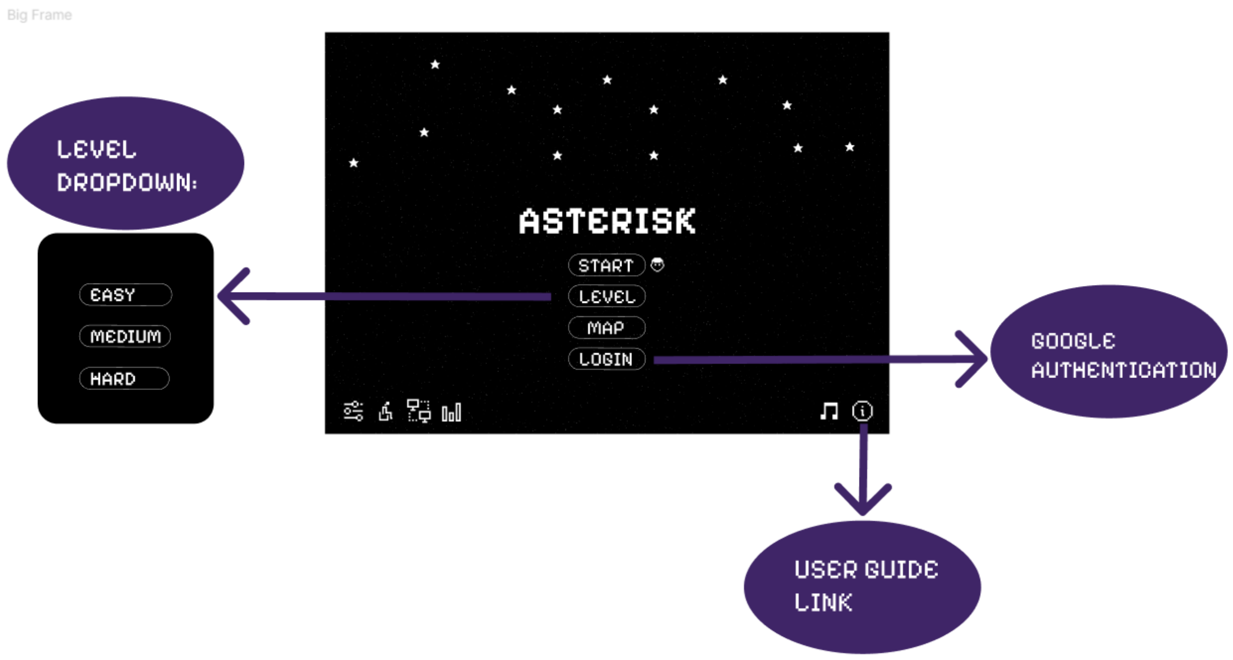
2. **Game Status Panel**
    - Current session progress
    - Star collection counter
    - Time elapsed
    - Stability score
    - Difficulty level indicator
3. **Navigation Elements**
    - Large, easily tappable menu buttons
    - Difficulty adjustment controls
    - Pause/Resume functionality
    - Help and tutorial access

Interaction Modes:

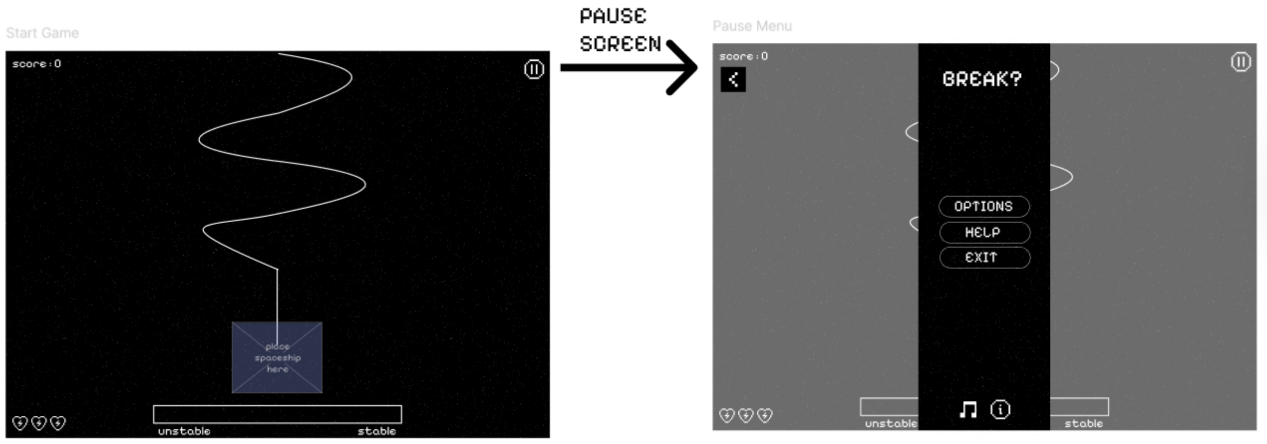
- Casual Mode: Structured exercises with precise tracking
- Tutorial Mode: Guided introduction to game mechanics

6.2 Screen Images

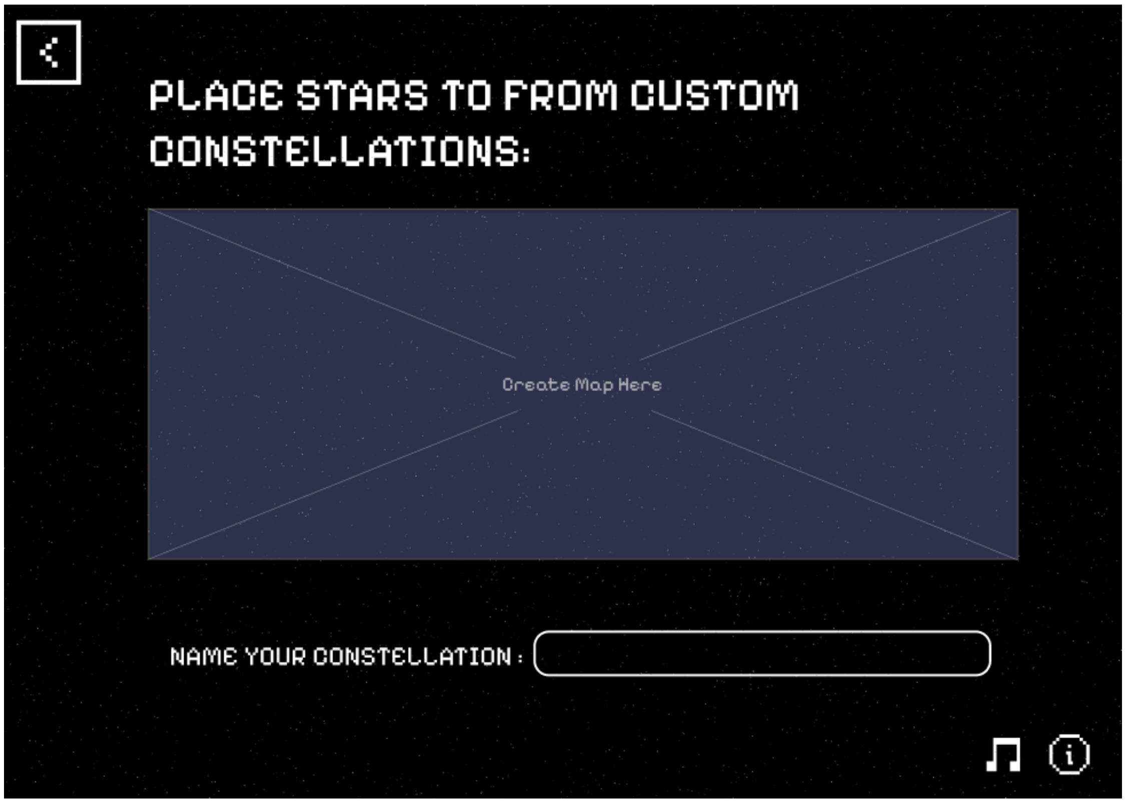
6.2.1 Home Screen



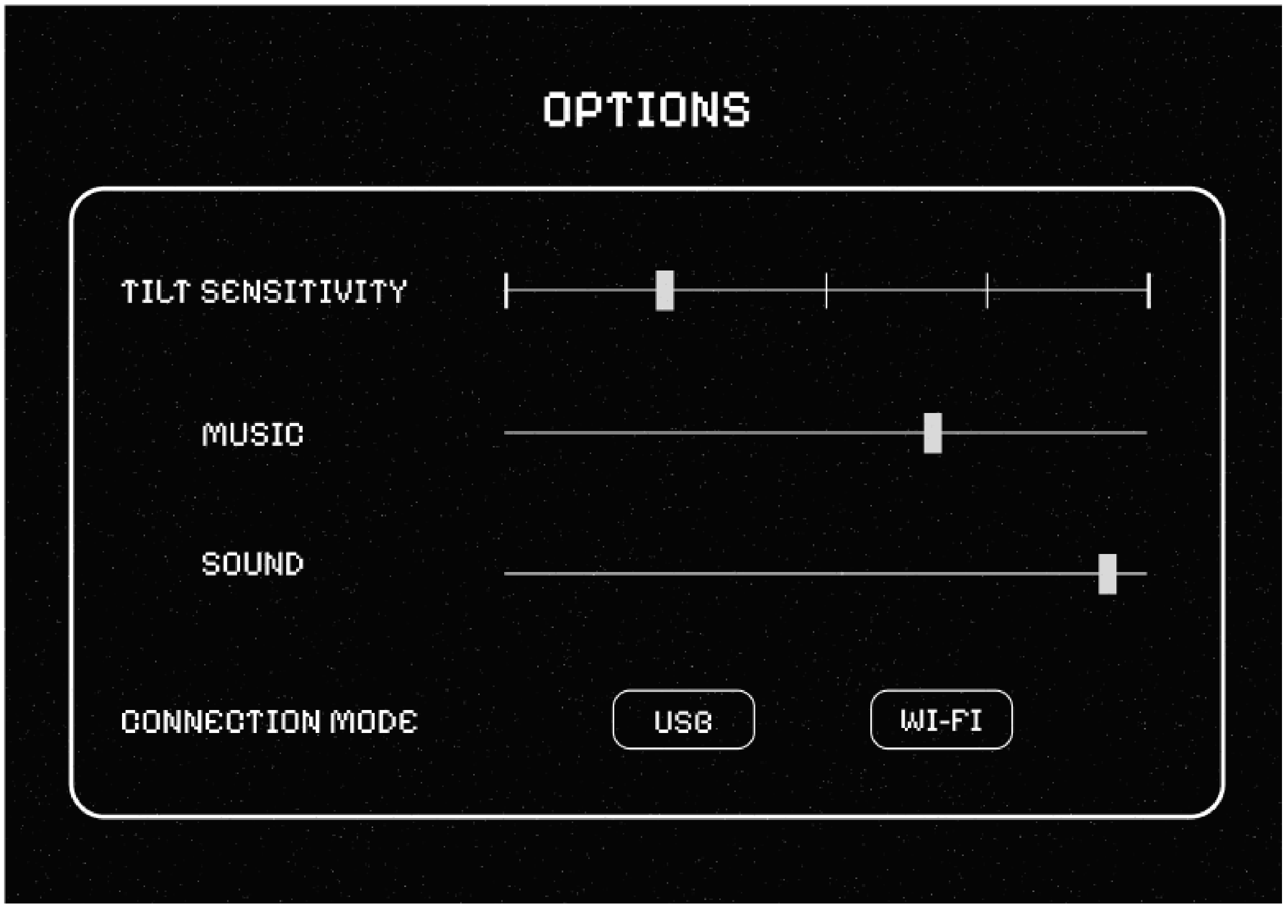
6.2.2 Gameplay Screen



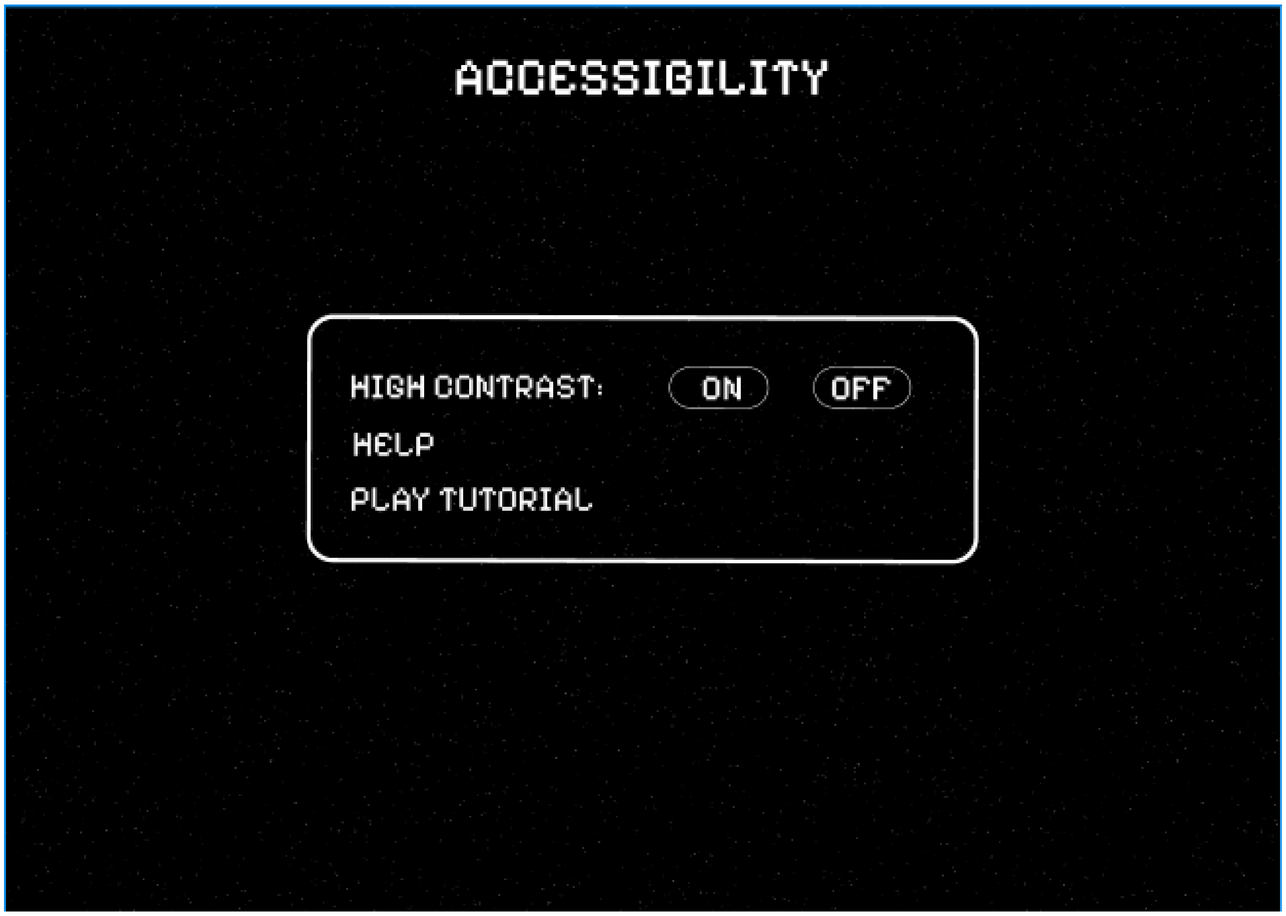
6.2.3 Custom Map


PLACE STARS TO FROM CUSTOM CONSTELLATIONS:
Create Map Here
NAME YOUR CONSTELLATION :

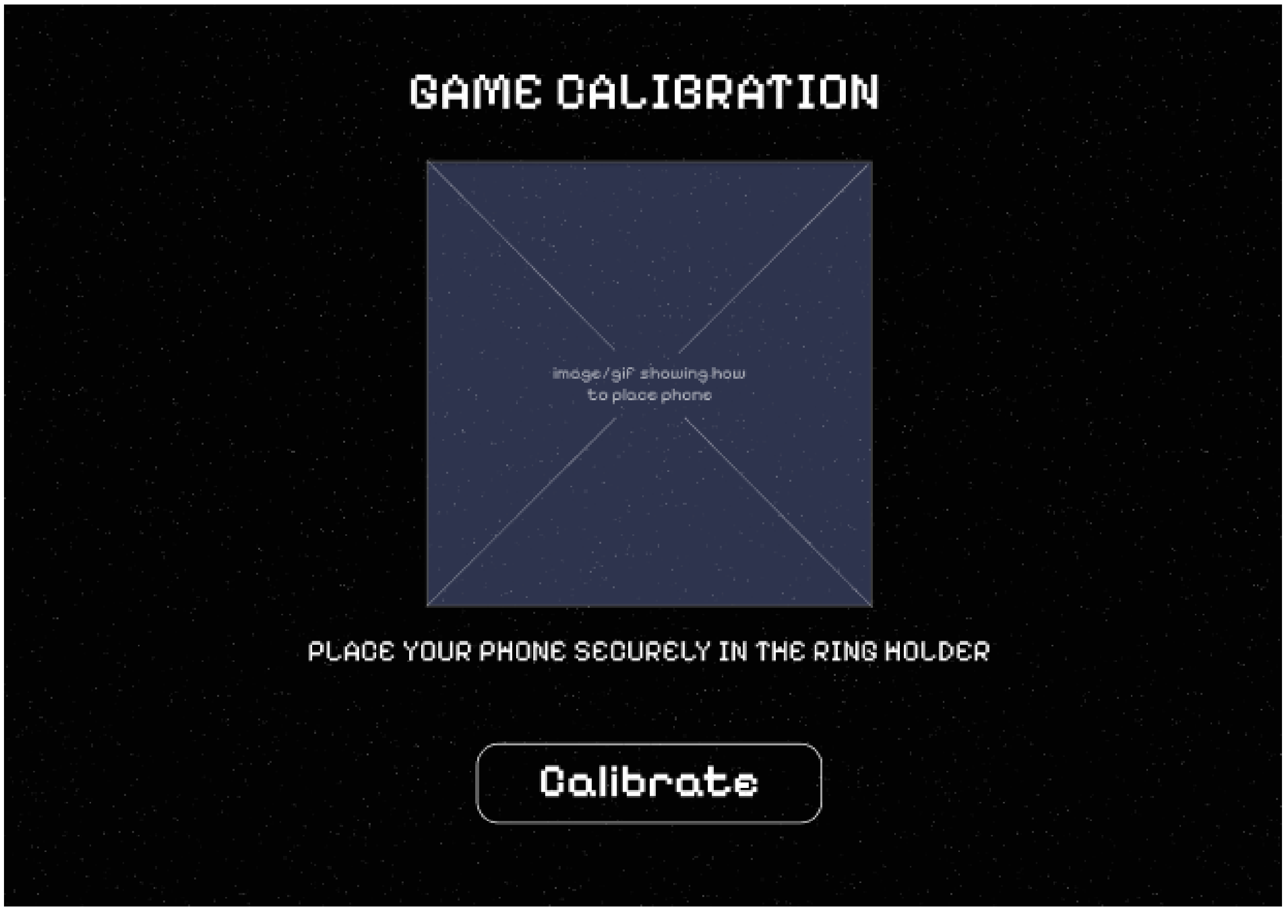
6.2.4 Options Screen



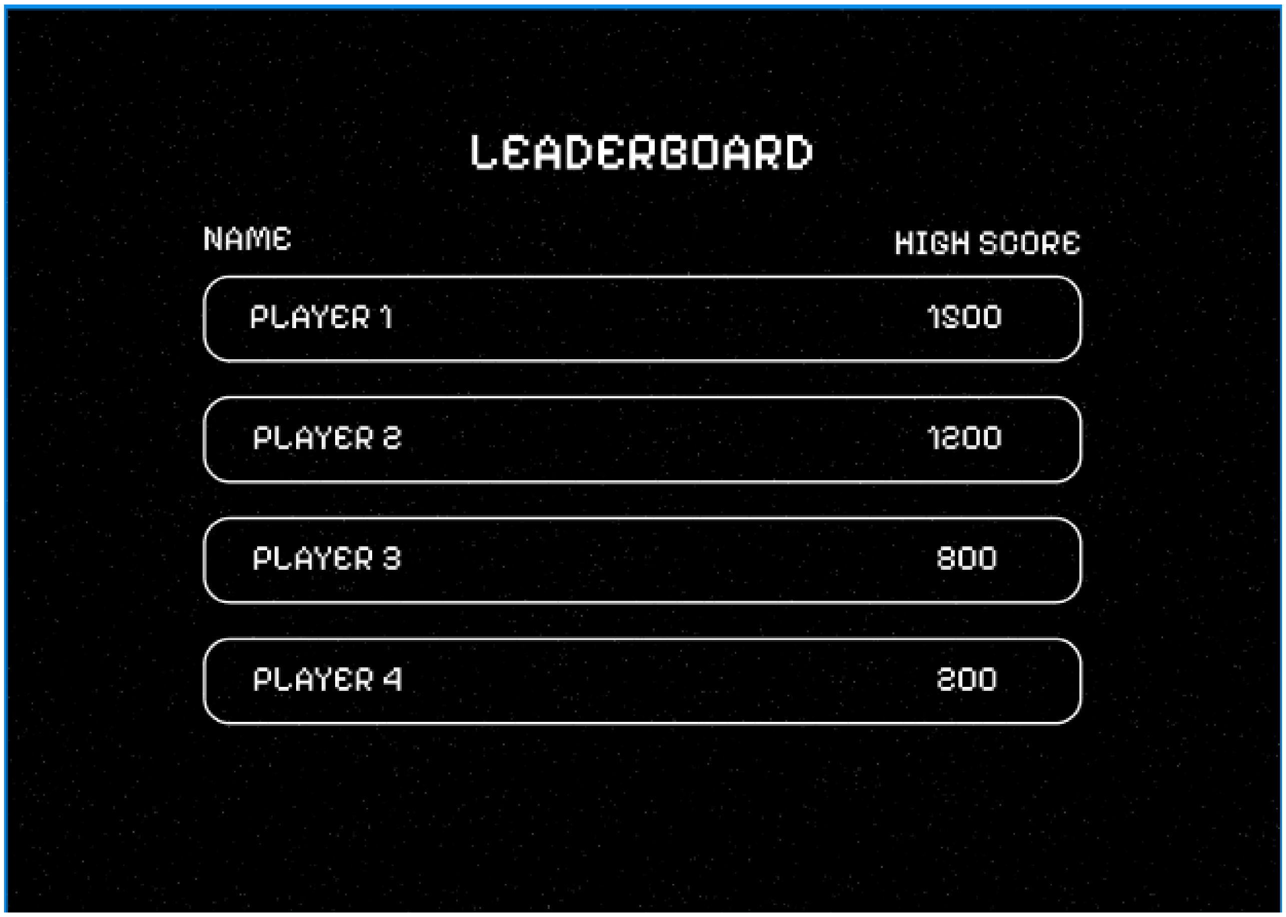
6.2.5 Accessibility Screen



6.2.6 Calibration Screen


GAME CALIBRATION
image/gif showing how to place phone
PLACE YOUR PHONE SECURELY IN THE RING HOLDER
Calibrate

6.2.7 Leaderboard Screen



LEADERBOARD

NAME	HIGH SCORE
PLAYER 1	1900
PLAYER 2	1200
PLAYER 3	800
PLAYER 4	200

6.3 Screen Objects and Actions

6.3.1 Gameplay Screen

Object: Spaceship

Actions: Moves based on device tilt, follows star paths, stabilizes during "Hold Steady Challenge."

Object: Star Patterns (Constellation Paths)

Actions: Guide movement, change color when successfully followed, disappear when completed.

Object: Health Icon

Actions: Has a maximum count of 3 and decreases when player fails to complete the level.

Object: Stability Bar

Actions: Has a maximum count of 3 and decreases when player fails to complete the level.

6.3.2 Home Screen

Object: Start Button

Actions: Switches to Gameplay Screen.

Object: Level Button

Actions: Dropdown menu showing easy, medium and hard difficulty levels

Object: Map Button

Actions: Switches to the Custom Map Screen

Object: Login Button

Actions: Switches to Google Authentication

Object: Info Button

Actions: Shows a link to the user guide

Object: Audio Button

Actions: Mute/Unmute game audio.

Object: Leaderboard Button

Actions: Shows local leaderboard scores.

Object: Calibration Button

Actions: Switches to game calibration screen.

Object: Accessibility Button

Actions: Switches to Accessibility Menu

Object: Settings Button

Actions: Switches to Settings Menu

6.3.3 Options Screen

Object: Tilt Sensitivity Bar

Actions: Adjust spaceship navigation sensitivity.

Object: Music Bar

Actions: Increase or decrease music level.

Object: Sound Bar

Actions: Increase or decrease in-game sound effects.

Object: Connection Mode

Actions: Select USB/WiFi Connection.

6.3.4 **Accessibility Menu**

Object: High Contrast option.

Actions: On/Off Button

Object: Help Button

Actions: Link to user guide

Object: Play Tutorial option.

Actions: Switches to tutorial gameplay screen.

6.3.4 **Calibration Screen**

Object: Calibrate Button.

Actions: Simulate subjective pivot values from the user using HyperIMU.

6.3.4 **Leaderboard Screen**

Name: Shows Player Name

High Score: Shows High Score.

7.0 REQUIREMENTS MATRIX

Functional Requirement	System Component	Design Rationale
FR1.1: User Account Creation	User Registration Module	Implements secure user authentication and account management
FR1.2: User Login	User Authentication Component	Handles credential verification and session management
FR1.3: Password Recovery	User Management Subsystem	Provides secure password reset functionality
FR2.1: Device Setup Instructions	User Interface Module	Displays step-by-step guidance for device placement and calibration
FR2.2: Sensor Calibration	Sensor Data Acquisition Module	Aligns device sensors to ensure accurate motion tracking
FR2.3: Recalibration Option	User Interface & Sensor Calibration Module	Allows users to adjust sensor alignment
FR3.1: Motion-Based Controls	Game Control Interface	Translates device tilt into spacecraft navigation
FR3.2: Real-Time Motion Tracking	Data Processing & Interpretation Module	Processes sensor data for precise movement control
FR3.3: Therapeutic Movement Patterns	Game Control Interface & Exercise Pattern Generator	Creates and manages customizable exercise paths

FR4.1: Progress Tracking	Data Management Subsystem	Records and stores local leaderboard data.
FR5.1: Therapeutic Mode	Game Mode Management Component	Implements specialized rehabilitation-focused gameplay
FR5.2: Exercise Customization	Exercise Configuration Module	Allows therapists to adjust exercise parameters
FR5.3: Mode Switching	Game Mode Management Component	Enables transitioning between tutorial and casual modes
FR6.1: Sensor Data Streaming	Data Transmission Module	Facilitates real-time motion data transfer
FR6.2: Connection Support (USB/WiFi)	Communication Layer	Ensures flexible data transmission options
FR6.3: Synchronization Troubleshooting	User Feedback System	Provides connection status and error handling
FR7.1: Accessible UI Design	User Interface Module	Creates large, readable interface elements
FR7.2: Accessibility Features	Accessibility Support Module	Implements audio and visual assistance features
FR8.1: Tutorial Mode	User Guidance System	Provides interactive onboarding and instructions
FR8.2: In-Game Help	Help and Support Module	Offers comprehensive troubleshooting resources

FR8.3: Support Contact	User Interface & Support Integration	Enables communication with technical support

8.0 APPENDICES

8.1 Glossary

Term	**Definition**
IMU (Inertial Measurement Unit)	A sensor module that includes an accelerometer, gyroscope, and magnetometer for motion tracking.
UDP (User Datagram Protocol)	A communication protocol used for fast, connectionless data transmission.
Tilt Control	The game mechanic where users control the spaceship by tilting their mobile device.
WebGL (Web Graphics Library)	A JavaScript API used to render graphics in Unity WebGL applications.
HyperIMU	A mobile application that streams motion sensor data via USB or WiFi.
Therapeutic Mode	A gameplay mode that provides structured motor skill exercises for rehabilitation.

8.2 References

- **IEEE Std 1016-2009** - IEEE Standard for Software Design Descriptions.
- **Unity WebGL Documentation** - Unity Manual - WebGL.