Table of Contents

5. System Features

5.1 Feature 1: Player Movement
5.2 Feature 2: Character Customization (OutfitChanger)
5.3 Feature 3: Math Question System (QuestionManager)
5.4 Feature 4: User Interface and Feedback (UIhandler)
5.5 Feature 5: Audio Management (AudioManager)
5.6 Feature 6: Level Progression

6. System Architecture and Module Description *(To be added)*

6.1 System Architecture Overview
6.2 Module 1: PlayerMovement
6.3 Module 2: OutfitChanger
6.4 Module 3: QuestionManager
6.5 Module 4: UIhandler
6.6 Module 5: AudioManager
6.7 Module 6: Level Management

7. Testing Plan *(To be added)*

7.1 Unit Testing
7.2 Integration Testing
7.3 System Testing
7.4 User Acceptance Testing (UAT)

8. Diagrams

8.1 Use Case Diagram

1. Introduction

1.1 Purpose

The primary objective of this project is to conceptualize, design, and develop a **Unity-based 2D game** that features a dynamic and interactive **character customization system** along with enjoyable gameplay elements. The game empowers players to modify and personalize their in-game avatars by altering various visual components such as outfits, body parts, and accessories. Through this feature, the project aims to enhance user engagement, immersion, and creative involvement in the gameplay environment.

In addition to offering a fun and interactive player experience, this project also serves as a practical application of essential game development principles, including **sprite management, UI-based interactions, animation state transitions, modular coding practices, and structured game architecture design within Unity**.

This documentation presents a systematic and comprehensive outline of the system's requirements, design components, architecture, and functional behavior. It acts as a reference guide that will support an organized development process and ensure that the final output meets quality standards, usability expectations, and performance goals.

1.2 Scope

The scope of this project encompasses the creation and implementation of the following components:

- A **2D Unity-based video game** integrating character customization as a core gameplay feature.
- Functional modules that allow players to cycle through outfit and appearance options, including **previous, next, and random selection features**.
- Gameplay mechanics seamlessly combined with character customization elements to ensure a cohesive player experience.
- A visually appealing and **user-friendly interface**, designed to provide smooth navigation for outfit selection and overall game interaction.
- Thorough testing processes focused on validating game performance, usability, functionality, and user experience across supported platforms such as **desktop and mobile devices**.

The current version of the game is designed exclusively as a **single-player experience**, emphasizing customization, creativity, and basic game progression. Advanced functionalities such as multiplayer systems, high-level artificial intelligence opponents, online services, or cloud-based features are **not included within the scope** of this project and may be considered for future enhancements.

1.3 Definitions, Acronyms, and Abbreviations

- **Unity:**
  A widely used cross-platform game engine designed for developing both 2D and 3D interactive applications, offering powerful tools for graphics, animation, physics, and scripting.
- **Sprite:**
  A 2D graphical object or bitmap image used in game environments to represent characters, props, UI icons, and other visual elements.
- **Customization Module:**
  A dedicated system within the game that enables users to modify character appearance by selecting different outfits, accessories, and body components.
- **Prefab:**
  A reusable object template in Unity that encapsulates a GameObject and its associated components, allowing developers to efficiently instantiate and manage consistent design elements throughout the game.
- **HUD (Heads-Up Display):**
  A graphical interface layer displayed on-screen during gameplay to present real-time information such as character appearance options, gameplay indicators, and control instructions.

1.4 References

- **Unity Official Documentation:**
  Provides complete technical guidance for Unity engine usage, scripting, UI systems, and development workflows.
  https://docs.unity3d.com/

- **C# Language Documentation — Microsoft Learn:**
  Reference material for C# syntax, programming concepts, and best practices used for scripting in Unity.
  https://learn.microsoft.com/en-us/dotnet/csharp/

- **Unity Learning Resources & Game Design Patterns:**
  Platform offering official tutorials, learning paths, and best-practice patterns for building efficient and scalable game systems.
  https://unity.com/learn

- **Previous SRS on Math Explorer Game:**
  Reviewed for formatting structure and documentation reference to maintain uniformity in project reporting.

1.5 Overview

This document provides a comprehensive description of the system and its development approach, including the functional and non-functional requirements, architectural design, core modules, and the testing strategy for the Unity-based game. A central component of the project is the **Outfit Changer system**, which allows players to customize their characters by cycling through various sprite options or generating randomized outfit selections. This customization functionality will be seamlessly integrated into the main gameplay loop, enhancing player engagement and personalization.

The system has been designed with scalability and modularity as key principles, ensuring that future enhancements—such as additional outfit options, new animated character states, expanded gameplay environments, reward features, and interactive progression systems—can be incorporated with minimal changes to the existing architecture. This forward-compatible structure supports long-term expansion and efficient maintenance, making the project adaptable to evolving design needs and gameplay requirements.

---

2. Overall Description

2.1 Product Perspective

This project is developed as an independent, **standalone 2D Unity-based game**, intended to offer an immersive and interactive learning and gameplay experience. The system follows a **modular architecture**, where each feature is structured as a separate component, ensuring flexibility, maintainability, and ease of future enhancements. Each module operates independently while working in coordination with others to deliver a seamless gameplay flow.

The major system modules include:

- **PlayerMovement Module:**
  Responsible for managing character motion within the game world, including walking, jumping, and responding to physics-based interactions. It ensures smooth and responsive character control for an intuitive player experience.

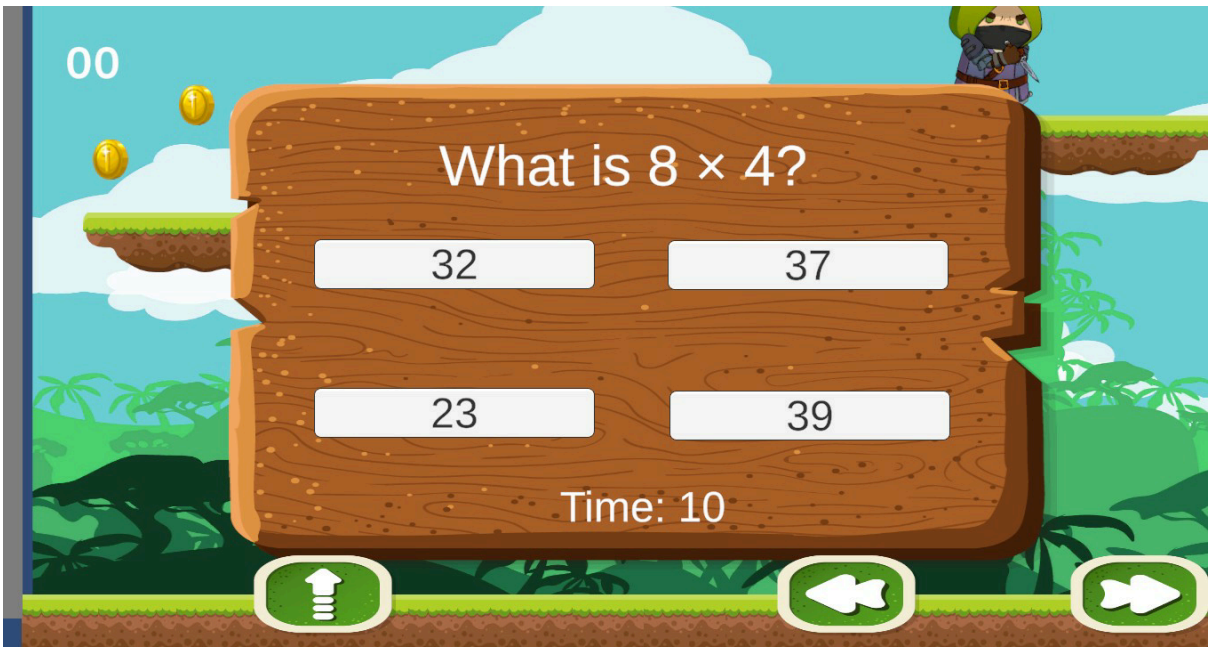
- **OutfitChanger Module:**
  Allows players to customize their character by selecting different outfits, body parts, or accessories using **Next, Previous, and Randomize** controls. This module enhances player identity, creativity, and engagement by offering personalized avatar appearance.

- **QuestionManager Module:**
  Generates and displays math questions as part of gameplay challenges. This module includes a timer, answer validation logic, and scoring mechanism to promote learning, focus, and performance tracking.

- **UIHandler Module:**
  Manages all user interface interactions including menus, in-game HUD elements, pop-up dialogs, player feedback (e.g., win or fail messages), and transitions between scenes or levels. It ensures an intuitive and organized visual experience.

- **AudioManager Module:**
  Controls and plays background music and sound effects based on player actions and game events. This module enhances immersion and reinforces feedback through audio cues.

Collectively, these modules form a structured and scalable framework, enabling smooth expansion such as new levels, advanced AI interactions, multiplayer capability, or additional character customization options in future versions.



2.2 Product Functions

The system provides a series of core functions to support gameplay, learning interaction, and user engagement. The major functions include:

Character Customization

- Enables players to personalize their character by cycling through various outfit sprites and accessories.
- Provides a random outfit selection feature for enhanced fun and engagement.
- Offers real-time visual updates on the character to reflect customization changes immediately.

Player Movement

- Allows the character to move left/right and jump within a 2D environment.
- Responds to user input via keyboard or touchscreen controls.
- Includes collision detection for interactive objects such as question boxes, enemies, and environmental boundaries.

Math Question System

- Generates randomized math problems covering addition, subtraction, multiplication, and division.
- Presents multiple-choice answers and validates player responses.
- Integrates a question-timer feature and displays appropriate feedback for correct and incorrect answers.
- Updates player score and maintains progression based on performance.

User Interface & Feedback

- Displays interactive question panels, win/lose state screens, and level-transition dialogs.
- Provides real-time score updates, progress indicators, and HUD elements.
- Ensures smooth and intuitive menu navigation for starting, pausing, or exiting gameplay.

Audio & Effects

- Plays background music continuously to maintain engagement.
- Triggers sound effects for key events such as outfit changes, correct/incorrect answers, jumps, and game outcomes.
- Enhances user experience with responsive audio cues tied to player actions.
    -

2.3 User Classes and Characteristics

Primary Users

- Children aged **6 to 10 years** for the educational version of the game.
- Casual players of all ages for the general entertainment version.

Characteristics:

- Limited prior gaming experience expected.
- Require simple and intuitive control schemes (keyboard, mouse, or touch).
- Prefer visually appealing and interactive interfaces with clear feedback mechanisms.
- Motivated by fun, reward-based progression, and character customization.

Secondary Users

- **Teachers, parents, or evaluators** who may observe gameplay for educational value or player learning outcomes.

Characteristics:

- Focus on the game's educational impact, usability, and player engagement.
- Require clarity in learning objectives and progression feedback.

2.4 Operating Environment

The game is developed as a standalone 2D Unity application and is intended to run on multiple platforms. The operating environment includes:

- **Platform:**
    - Primary: Windows PC
    - Optional: Android/iOS (when exported for mobile platforms)
- **Unity Engine Version:**
  Unity 2021 or later (2D mode enabled)
- **Hardware Requirements:**
  Standard desktop or laptop capable of running Unity-built games, including basic GPU support for 2D graphics rendering.
- **Software Dependencies:**
    - Unity 2D modules and physics engine
    - TextMeshPro for UI text rendering
    - Standard Unity UI components for menus, buttons, and HUD

2.5 Design and Implementation Constraints

The development and execution of the system is subject to the following constraints:

- The game must operate smoothly with no noticeable lag or frame drops.
- Only 2D assets and sprite-based animation are used; 3D models and rendering are not supported.
- Character customization is restricted to predefined sprite variations (no dynamic outfit generation).
- The math system currently supports only basic arithmetic operations (Addition, Subtraction, Multiplication, Division).
- Audio files (background music and sound effects) must be royalty-free or properly licensed.

2.6 Assumptions and Dependencies

The game design is based on the following assumptions and dependencies:

- Users will have access to standard input devices such as a keyboard, mouse, or touchscreen for controls.
- The game is designed to operate offline; no internet connection is required.
- Gameplay logic relies on Unity's built-in physics engine and animation framework.
- Future system expansion (e.g., additional outfits, advanced math difficulty, multi-level world, reward systems) is planned, but the current version assumes a single-player mode and basic difficulty settings.

3. Specific Requirements

3.1 Functional Requirements

These describe what the system **must do**:

3.1.1 Player Movement

- The player character must move **left or right** using keyboard or on-screen controls.
- The player character must be able to **jump**, only when grounded.
- Animations must reflect movement speed and direction.
- Collision detection must be enabled for **ground, enemies, and question boxes**.

3.1.2 Character Customization (OutfitChanger)

- Players can cycle through outfits using **Next** and **Prev** buttons.
- Players can randomize the outfit with a **Randomize** button.
- Current outfit must be **visually updated immediately** on the character sprite.
- The system must **prevent index overflow/underflow** in sprite lists.

3.1.3 Math Question System (QuestionManager)

- The system must generate **random math problems**: Addition, Subtraction, Multiplication, Division.
- Each question must have **4 multiple-choice answers**, with one correct answer.
- A **timer (default 30 seconds)** is displayed for each question.
- If the timer runs out, the question is marked **incorrect**, and feedback is displayed.
- Correct answers increment **player score**; incorrect answers do not.
- Feedback (win/lose) must be shown using **UIhandler panels**.

3.1.4 UI and Feedback (UIhandler)

- Display **question panel** when a player interacts with a question box.
- Show **Win/Lose images** based on correctness of answer.
- Display **score** and **timer** during gameplay.
- Allow **Restart Level** and **Next Level** functionality.
- Ensure all panels are **visible and responsive** on supported platforms.

3.1.5 Audio Management (AudioManager)

- Background music must **loop continuously** during gameplay.
- Sound effects must play for the following events:
    - Correct answer
    - Wrong answer
    - Player collision with objects
    - Level transitions
- Audio playback must not interfere with game performance.

3.1.6 Level Progression

- The game must have **at least 5 levels**.
- Completing a level (solving required questions) allows the player to **advance to the next level**.
- Level progression must be **persistent only for current session** (no cloud save in current version).

3.2 Non-Functional Requirements

These define how the system should behave:

3.2.1 Performance

- The game must run at **minimum 30 FPS** on target desktop devices.
- All interactions, including outfit changes and question panels, must have **no noticeable lag**.

3.2.2 Usability

- The interface must be **intuitive for children aged 6–10**.
- Buttons, text, and panels must be **clearly visible** and **responsive**.

- Instructions must be **easy to understand** without prior guidance.

3.2.3 Reliability

- The game must **not crash** during normal gameplay.
- Player score and question results must **update correctly** every session.

3.2.4 Maintainability

- Code must be **modular and well-documented** for future enhancements.
- Each module (PlayerMovement, OutfitChanger, QuestionManager, UIhandler, AudioManager) should be **independent** and reusable.

3.2.5 Security

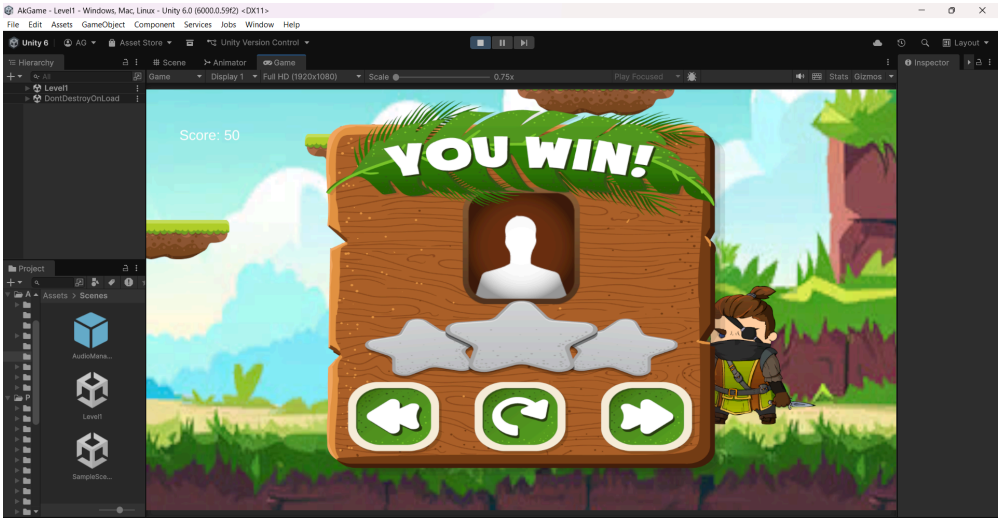
- Minimal security is required since the game is offline.
- Game data (score, level progress) must **remain consistent during the session**.

3.2.6 Portability

- The game must run on **Windows desktop**; optional support for Android/iOS in future versions.
- Must be compatible with **Unity 2D engine version 2021 or later**.

3.3 Other Requirements

- Support for **TextMeshProUGUI** for clear text rendering.
- All audio files must be **royalty-free or licensed**.
- All sprites and animations must maintain **2D consistency** (no 3D objects in current version).



4. Interface Requirements

4.1 User Interfaces

4.1.1 Main Menu

- Buttons for:
  - **Start Game** → Launch gameplay scene.
  - **Character Customization** → Open outfit selection panel.
  - **Settings** → Audio, volume control, etc.
  - **Exit** → Close the game.

- Visual elements:
  - Background image or animated scene.
  - Logo and title prominently displayed.
- Navigation must be **intuitive for children aged 6–10**.

4.1.2 Gameplay HUD

- Display **player score**.
- Display **timer** during question-solving.
- Optionally display **current level/number of remaining questions**.
- HUD elements must **not obstruct the main game view**.

4.1.3 Character Customization Panel

- Buttons:

    - **Next / Prev Outfit** → Cycle through sprite options.
    - **Randomize Outfit** → Randomly change character appearance.
- Visual preview of **current character sprite**.
- User input should reflect **instant changes** on the sprite.

4.1.4 Question Panel

- Display **question text** using TextMeshProUGUI.
- Display **four answer buttons**, clearly labeled.
- Highlight selected answer and provide **feedback** (correct/incorrect).
- Display **timer** counting down the available time to answer.

4.1.5 Level Dialog / Feedback

- Win/Lose images indicating success or failure.
- Buttons for:
    - **Next Level** → Progress to next level if correct.
    - **Restart Level** → Retry the current level.
- Panel should **appear above the gameplay scene** without obstructing previous information until acknowledged.

4.2 Hardware Interfaces

- Input devices:
    - **Keyboard** → Arrow keys (movement), Space (jump), Enter/Click (menu).
    - **Mouse / Touchscreen** → Click/tap buttons for outfit changes and question answers.

- Display:
    - Minimum resolution: 1280x720 (HD) for clear visibility of sprites and UI elements.

- Optional:
    - Speakers / headphones for background music and SFX.

4.3 Software Interfaces

- **Unity Engine (2D)** → Manages rendering, physics, animations, and input.
- **TextMeshProUGUI** → For high-quality text display.

- **AudioManager** → Plays sound effects and background music.
- **SceneManager** → Handles level transitions and scene loading.
- **Rigidbody2D & Collider2D** → Player physics and collision detection.

4.4 Communication Interfaces

- **Internal Module Communication**:
  - `PlayerMovement` → triggers collision detection for `QuestionManager`.
  - `QuestionManager` → calls `UIhandler` to show feedback panels.
  - `UIhandler` → may trigger `SceneManager` for level restart/next.
  - `OutfitChanger` → updates `SpriteRenderer` for real-time customization.
- No **external network communication** required in the current version (offline).

4.5 Interface Design Considerations

- All **UI panels** must be responsive and **scale correctly** for different screen resolutions.
- Buttons should provide **visual feedback** (hover, click, selection).
- Colors and fonts must be **child-friendly and readable**.
- Panels should **animate in/out smoothly** to maintain engagement.
- Audio cues should **synchronize with UI events** (button clicks, correct/wrong answers).

5. System Features

5.1 Feature 1: Player Movement

Description:

Allows the player character to move left/right and jump within the 2D game environment. Integrates with animations to reflect movement.

Functional Details:

- Move left or right using keyboard arrows or on-screen controls.

- Jump only when grounded.
- Collisions with ground, obstacles, question boxes, and enemies are detected.
- Animations change according to movement speed and direction.

Inputs:

- Keyboard input: Arrow keys (← →) and Space (Jump)
- Touch input: On-screen buttons for left, right, and jump

Outputs:

- Updated player position
- Animation changes
- Trigger collision events

Priority: High

5.2 Feature 2: Character Customization (OutfitChanger)

Description:
Enables players to personalize their character by changing outfits and appearance.

Functional Details:

- Cycle through outfits using **Next** and **Prev** buttons.
- Randomize outfit using **Randomize** button.
- Updates character sprite in real-time.

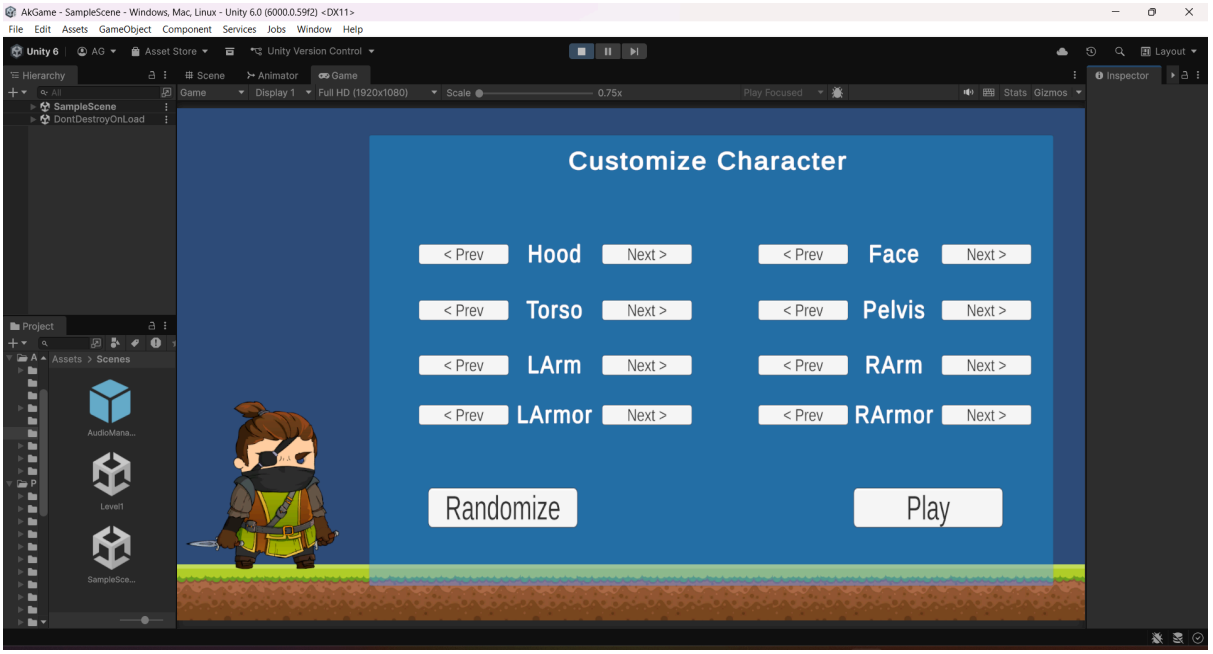
Inputs:

- Button click / tap events (Next, Prev, Randomize)

Outputs:

- Updated character sprite on the main screen

Priority: Medium



5.3 Feature 3: Math Question System (QuestionManager)

Description:

Generates math questions for players to solve with multiple-choice answers and scoring mechanism.

Functional Details:

- Randomly generate addition, subtraction, multiplication, and division questions.
- Display four answer buttons with one correct answer.
- Countdown timer for each question (default 30 seconds).
- Provide feedback for correct/incorrect answers.
- Increment score for correct answers.

Inputs:

- Player answer selection via button click or tap
- Timer countdown

Outputs:

- Correct/Incorrect feedback
- Updated score
- Trigger UIhandler feedback panel

Priority: High

5.4 Feature 4: User Interface and Feedback (UIhandler)

Description:
Manages all in-game UI panels, including question display, feedback, and level transitions.

Functional Details:

- Show question panels and timer during gameplay.
- Display win/lose feedback images after each question.
- Provide buttons for **Restart Level** and **Next Level**.
- Display score and progression indicators.

Inputs:

- Signals from `QuestionManager` (correct/wrong answers)
- Player input (button click for restart/next)

Outputs:

- Updated UI panels
- Visual feedback (win/lose images)
- Level progression actions

Priority: High

5.5 Feature 5: Audio Management (AudioManager)

Description:
Controls background music and sound effects throughout gameplay.

Functional Details:

- Play looped background music.
- Play SFX for: correct/wrong answers, collision events, level completion.
- Ensure audio playback does not interfere with game performance.

Inputs:

- Event triggers from `QuestionManager` and gameplay events

Outputs:

- Background music
- Sound effects

Priority: Medium

5.6 Feature 6: Level Progression

Description:
Manages advancement between multiple game levels.

Functional Details:

- Allow players to progress to the next level after completing a set of questions.
- Enable level restart if the player fails.
- Load corresponding level scene in Unity.

Inputs:

- Signals from `UIhandler` (Next Level / Restart Level button)

Outputs:

- Scene transition in Unity
- Reset or update gameplay state

Priority: High

---

6. System Architecture and Module Description

6.1 System Architecture Overview

- **Architecture Type:** Modular 2D Unity game architecture with component-based design.
- **Description:** The system consists of **independent modules** (PlayerMovement, OutfitChanger, QuestionManager, UIhandler, AudioManager) that communicate through events and method calls.
- **Key Components:**
    1. **PlayerMovement Module:** Handles character physics, input, and animations.
    2. **OutfitChanger Module:** Manages character customization (sprites, outfits).
    3. **QuestionManager Module:** Generates math questions, validates answers, tracks score.

4. **UIhandler Module:** Manages UI panels, feedback, and level transitions.
5. **AudioManager Module:** Controls background music and SFX.
6. **Level Manager (Optional):** Handles level progression, scene loading, and reset.

Diagram Suggestion:

- A **block diagram** showing each module as a separate box with arrows indicating interactions:
    - PlayerMovement → QuestionManager (collision triggers question)
    - QuestionManager → UIhandler (feedback panel)
    - UIhandler → SceneManager (restart/next level)
    - OutfitChanger → PlayerMovement/Character (sprite updates)
    - AudioManager → all modules (plays event sounds)

6.2 Module 1: PlayerMovement

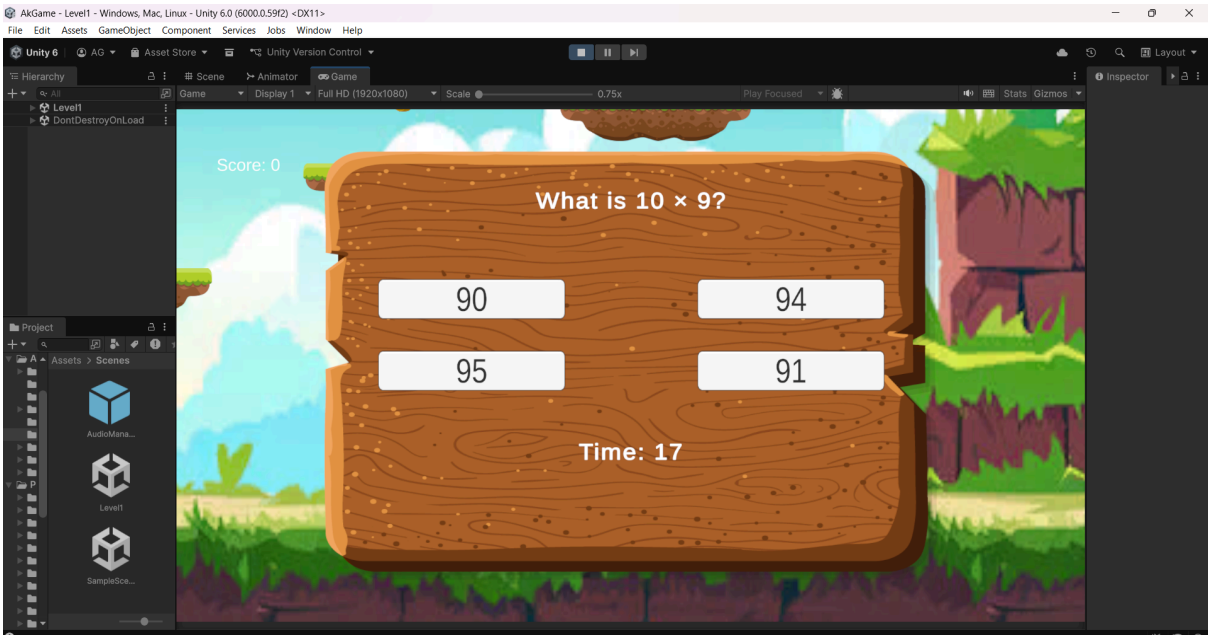
- **Responsibilities:**
    - Move character horizontally and vertically (jump).
    - Animate character based on speed/direction.
    - Detect collisions with ground, obstacles, enemies, and question boxes.
- **Inputs:** Keyboard/Touchscreen input.
- **Outputs:** Updated Rigidbody2D position, animation changes, collision events.
- **Dependencies:** Unity Rigidbody2D, Animator, Collider2D.

6.3 Module 2: OutfitChanger

- **Responsibilities:**
  - Allow player to change character outfits (Next/Prev/Random).
  - Update SpriteRenderer with selected sprite.
- **Inputs:** Button clicks or touch events.
- **Outputs:** Updated character sprite displayed in real-time.
- **Dependencies:** SpriteRenderer, List of sprites.

6.4 Module 3: QuestionManager

- **Responsibilities:**
  - Generate random math questions (Addition, Subtraction, Multiplication, Division).
  - Display questions and answer options.
  - Validate answers and update score.
  - Trigger feedback via UIhandler.
- **Inputs:** Player answer selection, timer events.
- **Outputs:** Correct/incorrect feedback, score updates, event triggers for UIhandler.
- **Dependencies:** UIhandler, ScoreManager, Unity UI elements (TextMeshProUGUI, Buttons).



6.5 Module 4: UIhandler

- **Responsibilities:**

    - Display question panels, win/lose feedback, HUD, and menus.
    - Handle Restart Level and Next Level actions.
- **Inputs:** Signals from QuestionManager (correct/wrong), player button input.
- **Outputs:** Visible UI panels, scene transitions, HUD updates.
- **Dependencies:** Unity UI system, SceneManager.

6.6 Module 5: AudioManager

- **Responsibilities:**
    - Play background music and loop it.
    - Play sound effects for events like correct/wrong answers or collisions.
- **Inputs:** Event triggers from other modules.
- **Outputs:** Audio playback (music/SFX).
- **Dependencies:** Unity AudioSource, AudioClip.

6.7 Module 6: Level Management

- **Responsibilities:**
    - Load next level scene or restart current level.
    - Track player progress during the session.
- **Inputs:** Signals from UIhandler (Next Level / Restart Level).
- **Outputs:** Scene transition, reset of gameplay variables.
- **Dependencies:** Unity SceneManager.

6.8 Module Interaction Summary

Source Module	Target Module	Purpose/Trigger
PlayerMovement	QuestionManager	Collision with question box
QuestionManager	UIhandler	Display win/lose panel
UIhandler	SceneManager	Restart/Next level
OutfitChanger	PlayerCharacter	Update character sprite

AudioManager	All Modules	Play SFX for events

---

7. Testing Plan

7.1 Unit Testing

Purpose: Verify that individual modules function correctly on their own.

Module	Test Cases	Expected Outcome
PlayerMovement	Move left/right, jump, collision detection	Character moves accurately; collisions detected properly
OutfitChanger	Next/Prev/Random outfit selection	Character sprite updates correctly; no index out-of-bounds errors
QuestionManager	Generate math questions, validate answers, timer	Correct answers recognized, wrong answers penalized, timer works
UIhandler	Show panels, restart level, next level	UI displays correctly; buttons trigger appropriate actions
AudioManager	Play background music, SFX	Music loops; SFX play at correct events without lag

---

7.2 Integration Testing

Purpose: Ensure that different modules interact correctly.

Interaction	Test Case	Expected Outcome

PlayerMovement → QuestionManager	Player collides with question box	Question panel appears; timer starts
QuestionManager → UIhandler	Player selects answer	Correct/Incorrect feedback displayed; score updated
UIhandler → SceneManager	Player clicks Next Level	Next scene loads correctly; player position resets
OutfitChanger → PlayerCharacter	Player customizes outfit	Sprite updates in real-time without affecting movement
AudioManager → All Modules	Events triggered	Appropriate sound effect plays without disrupting game

---

7.3 System Testing

Purpose: Test the complete game as a whole to ensure end-to-end functionality.

Test Scenarios:

1. Start the game → Main menu loads → Player can navigate menu options.
2. Begin gameplay → Player can move, jump, and interact with objects.
3. Encounter question box → Question panel appears → Timer starts → Score updates correctly.
4. Customize character → Outfit changes reflected immediately.
5. Complete a level → Win/Lose panel displays correctly → Next level or restart works.
6. Audio plays continuously → Correct SFX for events triggered.
7. Game handles invalid inputs gracefully → No crashes or unexpected behavior.

7.4 User Acceptance Testing (UAT)

Purpose: Validate the game with the target audience (children aged 6–10) to ensure usability and engagement.

Activities:

- Observe how children navigate the game interface.
- Check if math questions are **understandable and age-appropriate**.
- Gather feedback on character customization, visuals, and audio experience.
- Note any difficulties or confusion during gameplay.

Acceptance Criteria:

- Game is **intuitive and easy to use**.
- Math problems are solvable within the given timer.
- Game provides **positive reinforcement** for correct answers.
- Player enjoys **level progression and rewards**.
- No critical bugs or crashes during testing.

Tools: Observation, Surveys/Feedback Forms, Video Recording of gameplay sessions

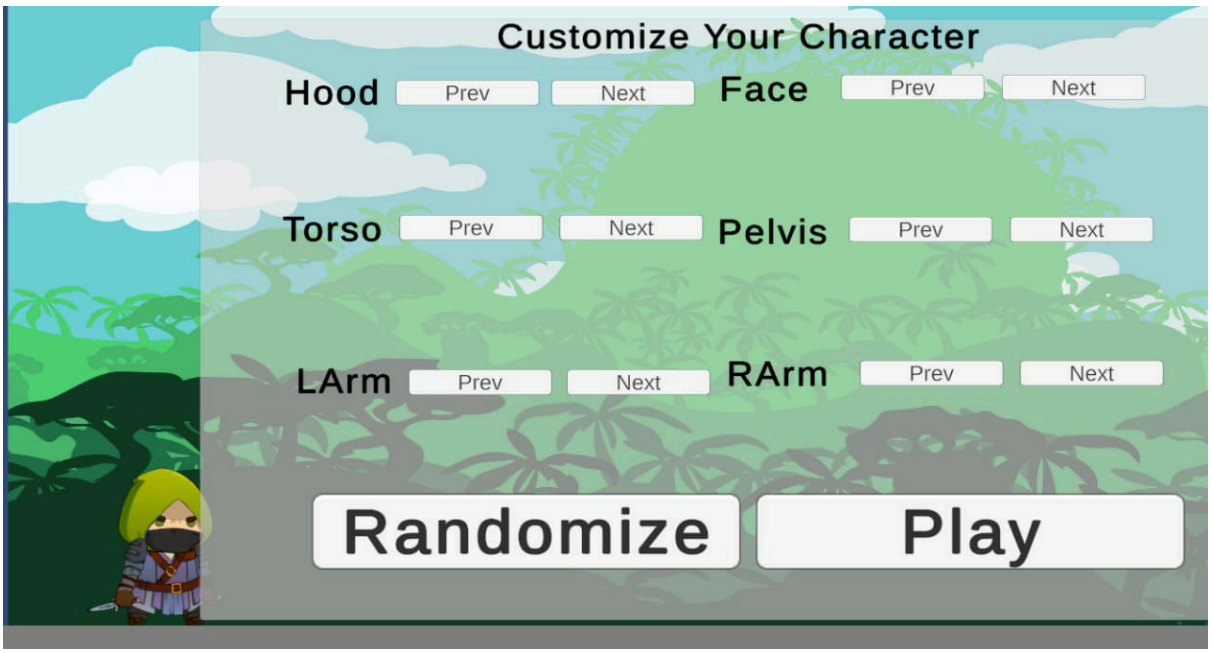
---

7.5 Testing Schedule

Phase	Duration	Activities
Unit Testing	2 Days	Test individual modules
Integration Testing	1 week	Test module interactions
System Testing	2 Days	Test full game functionality
User Acceptance Testing	1 week	Target audience testing and feedback collection

---

OUTPUT:



8 Diagrams

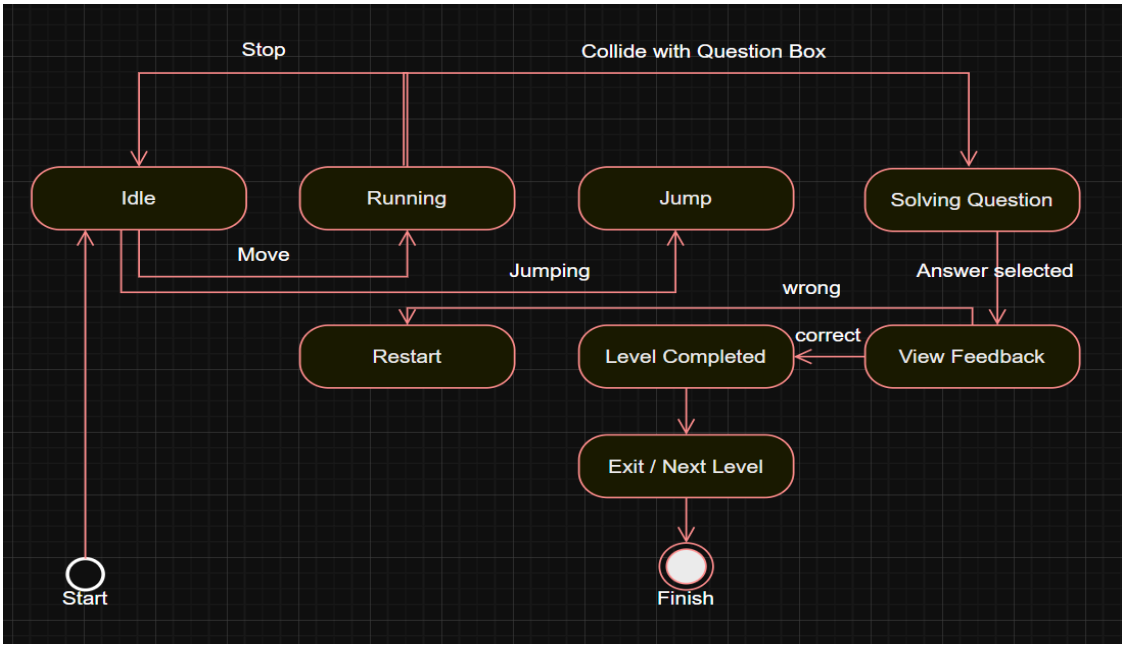
8.1 State Diagram

States:

1. Idle
2. Moving
3. Jumping
4. Colliding with Question Box
5. Solving Question
6. Viewing Feedback (Win/Lose)
7. Level Completed / Restart

Transitions:

- Idle → Moving → Idle (on keyboard input)
- Idle / Moving → Jumping → Idle (on space press, grounded)
- Moving → Colliding with Question Box → Solving Question
- Solving Question → Viewing Feedback (based on answer correctness)
- Viewing Feedback → Level Completed → Next Level / Restart Level → Idle



8.2 Class Diagram

Relationships

- `PlayerMovement`
  - Handles **movement/jumping** → interacts with environment.
  - Triggers `QuestionManager` when colliding with question objects.
- `OutfitChanger`
  - Independent, used for **sprite customization**.

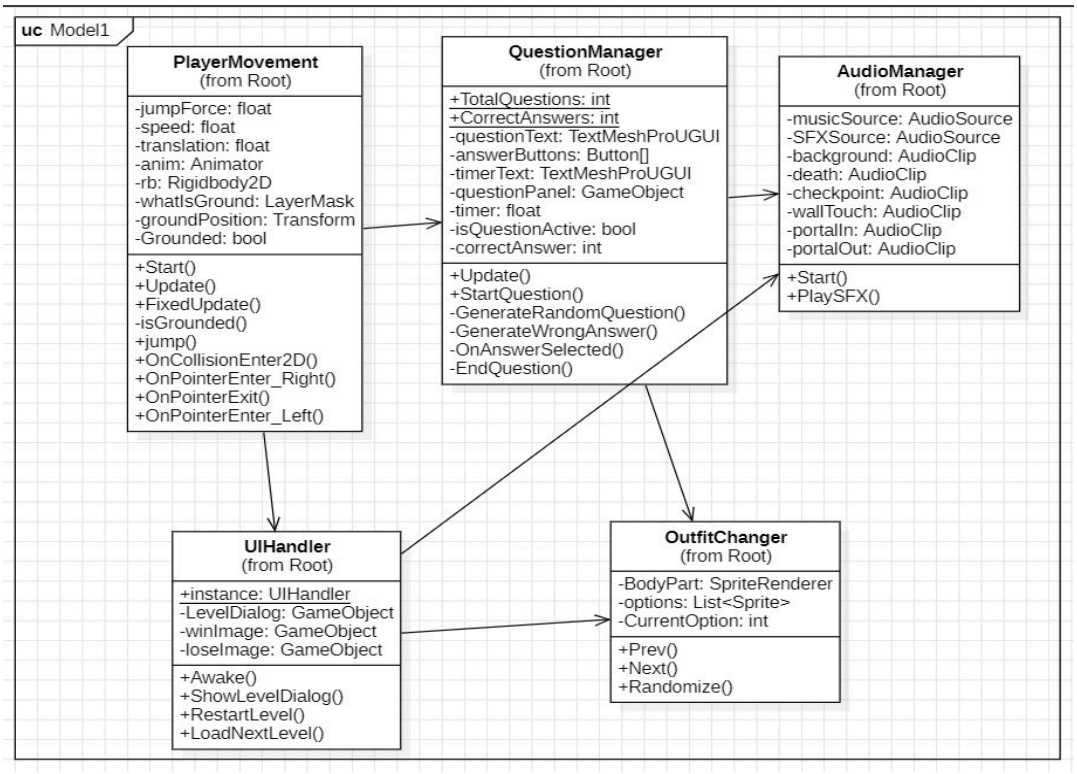
- `QuestionManager`
  - Uses **`UIhandler`** → to show win/lose dialogs.
  - Uses **`AudioManager`** → to play correct/wrong SFX.
  - Updates **`ScoreManager`** when questions are answered.

- `UIhandler`
  - Singleton. Controls dialogs, restart, and scene transitions.

- `AudioManager`

- Plays background music (on `Start`) and triggered SFX from others.

- `ScoreManager`
  - Provides scoring system used by `QuestionManager` and displayed in UI.



Use Case Diagram

Actor:

- **Player** → interacts with the game.

Use Cases:

1. Move Character (Left/Right/Jump)
2. Customize Character (Next/Prev/Random Outfit)
3. Solve Math Question (Answer multiple-choice)

4. View Feedback (Win/Lose panel)
5. Progress Level (Next Level / Restart Level)

Relationships:

- Player → interacts with all use cases.
- Solve Math Question → triggers View Feedback.
- Progress Level → depends on Solve Math Question result.