AstroStreet AR

Software Design Document (SDD)

Assignment 14

CS 3338 – Software Engineering Tools

California State University, Los Angeles

Group 2

Royce Jamerson
Khalid Jamil
Michael Lieng
Ricardo Ibanez

Date: December 2, 2025

Contents

1 Version Description **1**

2 Introduction **1**
2.1 Purpose 1
2.2 System Overview 1
2.3 Intended Audience 1
2.4 Document Overview 2

3 System Architecture **2**
3.1 Architecture Overview 2
3.2 Major Components 3
3.3 Component Interactions 5

4 User Interface **5**
4.1 Overview 5
4.2 Main Menu 6
4.3 AR Gameplay Screen 6
4.4 Pause Menu 7
4.5 Game Over Screen 7
4.6 High Scores Screen 8
4.7 Settings Screen 8

5 Database Design and Explanation **8**
5.1 Overview 8
5.2 HighScore Entity 9
5.3 Setting Entity 9
5.4 Relationships and Data Flow 10

6 Glossary **10**

7 References **11**

1 Version Description

This section tracks the revision history of the *AstroStreet AR* Software Design Document (SDD) for Group 2 in CS 3338 – Software Engineering Tools.

Version	Description	Date
1.0	Initial version of the Software Design Document for *AstroStreet AR*, prepared for Assignment 14.	December 2, 2025

Future revisions of this document can be recorded by adding new rows to the table with updated version numbers, dates, and brief descriptions of the changes.

2 Introduction

2.1 Purpose

The purpose of this Software Design Document (SDD) is to describe the overall design and internal structure of *AstroStreet AR*, a mobile augmented reality (AR) shooter game developed by Group 2 for CS 3338 – Software Engineering Tools at California State University, Los Angeles. This document translates the high-level requirements of the project into a concrete design that can guide implementation, testing, and future maintenance.

2.2 System Overview

AstroStreet AR is a smartphone game in which players go outside, use their phone's camera, and see virtual asteroids and alien ships overlaid on the real world. The player rotates their body and phone to aim and taps on the screen to fire projectiles and destroy incoming targets. The game tracks score, player health or shields, and increasing difficulty as more enemies appear over time.

At a high level, the system consists of:

- A mobile client that renders the AR scene, handles user input, manages game logic, and displays the user interface.
- AR services provided by the underlying platform (e.g., ARCore/ARKit via an engine such as Unity) to track device position and orientation and to overlay virtual objects on the camera feed.
- Local data storage to persist high scores and basic player settings on the device.

2.3 Intended Audience

This document is intended for:

- Members of Group 2, who will use the design as a blueprint for implementing *AstroStreet AR*.

- The course instructor and teaching staff, who will review the design as part of Assignment 14.
- Future developers or maintainers who may extend the game's features, port it to new platforms, or integrate additional services (such as online leaderboards).

2.4 Document Overview

The remainder of this SDD is organized as follows:

- **System Architecture** describes the major components of *AstroStreet AR*, their responsibilities, and how they interact.
- **User Interface** presents the planned screens and HUD elements, including the main menu, AR gameplay view, and score displays.
- **Database Design and Explanation** outlines the data structures used to store high scores and game settings, and explains how the application accesses this data.
- **Glossary** defines key technical terms and acronyms used throughout the document.
- **References** lists external resources, documentation, and tools consulted while designing *AstroStreet AR*.

3 System Architecture

3.1 Architecture Overview

The architecture of *AstroStreet AR* is organized around a single mobile client application that runs on a smartphone and communicates with platform-specific AR services. Within the client, the game is decomposed into modular components that manage AR tracking, game state, enemy behavior, player input, user interface, and local data storage.

At a high level, the system consists of the following layers:

- **Presentation Layer**: Responsible for rendering the camera feed, drawing virtual objects (asteroids, alien ships, projectiles), and displaying the heads-up display (HUD) and menus.
- **Game Logic Layer**: Manages the core game loop, including spawning enemies, updating the player state, handling collisions, tracking score, and determining win/lose conditions.
- **AR and Device Services Layer**: Interfaces with the underlying AR framework (e.g., ARCore/ARKit via Unity) to obtain camera images, device pose (position and orientation), and other sensor data.
- **Data Layer**: Handles reading and writing persistent data, such as high scores and basic settings, on the device.

Figure 1 conceptually illustrates the relationships between the major components.

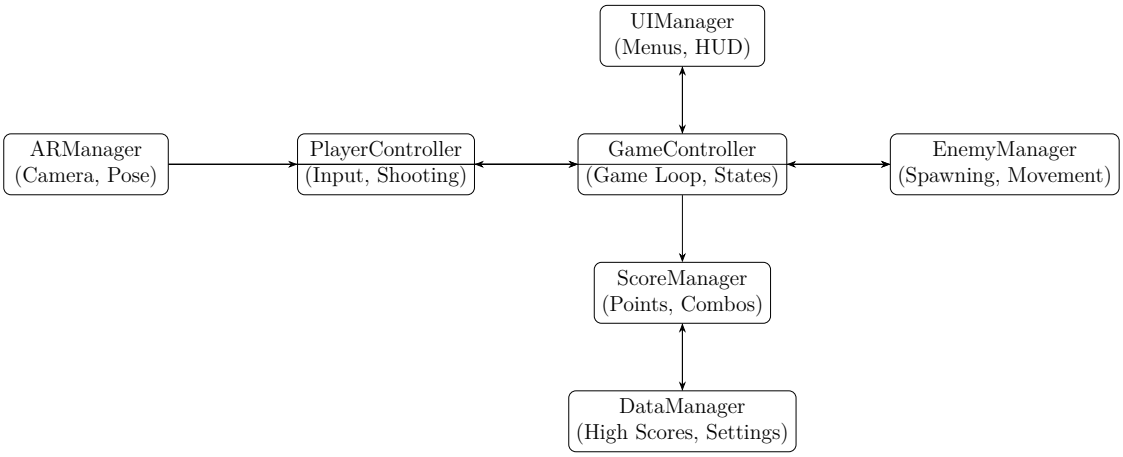

Figure 1: High-level component architecture of *AstroStreet AR*.

3.2 Major Components

The main components of the client application are described below.

3.2.1 ARManager

The **ARManager** is responsible for:

- Initializing the AR session when the game starts.
- Accessing the device camera feed and sensor data from the AR framework.
- Providing the current device pose (position and orientation) so that virtual objects can be placed consistently in the player's view.

Other components (such as the **GameController** and **EnemyManager**) rely on the **ARManager** to ensure that objects are rendered relative to the player's real-world orientation.

3.2.2 GameController

The **GameController** coordinates the overall game loop:

- Starting and stopping gameplay sessions.
- Updating the game state each frame (time remaining, player health or shields, active enemies, and projectiles).
- Notifying other components when the game transitions between states (e.g., from "Main Menu" to "In Game" to "Game Over").

It serves as the central point of control, invoking the **EnemyManager**, **PlayerController**, and **UIManager** as needed during each update cycle.

3.2.3 EnemyManager

The **EnemyManager** handles all enemy-related behavior:

- Spawning asteroids and alien ships according to the current difficulty level.
- Updating enemy movement and checking when enemies become too close to the player.
- Informing the **GameController** when enemies are destroyed or when they collide with the player.

The **EnemyManager** uses AR pose information from the **ARManager** to decide where to place enemies in the player's field of view.

3.2.4 PlayerController

The **PlayerController** interprets user input and updates the player's actions:

- Reading touch input (e.g., tapping the screen to shoot).
- Aligning the virtual crosshair or reticle with the player's current viewing direction.
- Requesting the creation of projectiles when the player fires and reporting hits back to the **GameController**.

It acts as the bridge between physical device movements/touches and in-game actions.

3.2.5 UIManager

The **UIManager** manages the user interface and HUD:

- Displaying the main menu, pause menu, and game over screen.
- Rendering HUD elements during gameplay, such as the score, health or shield bar, time remaining, and any active power-ups.
- Receiving navigation commands from the user (e.g., starting a new game, returning to the main menu).

The **UIManager** communicates with the **GameController** and **ScoreManager** to update visual information.

3.2.6 ScoreManager

The **ScoreManager** tracks and updates the player's score:

- Awarding points when enemies are destroyed.
- Applying score multipliers or bonuses for streaks.
- Exposing the current score to the **UIManager** for display during gameplay.

At the end of a game session, the **ScoreManager** collaborates with the **DataManager** to save high scores.

3.2.7 DataManager

The **DataManager** is responsible for:

- Persisting high scores and basic settings (such as sound preferences and control sensitivity) to local storage on the device.
- Loading saved data when the application starts.
- Providing a simple interface for other components to read and write data without dealing directly with the storage mechanism.

3.3 Component Interactions

During a typical gameplay session, the components interact as follows:

1. The user selects "Start Game" from the main menu. The **UIManager** notifies the **GameController** to begin a new session.
2. The **GameController** initializes the game state and requests the **ARManager** to start the AR session.
3. The **EnemyManager** begins spawning enemies in positions derived from the AR pose provided by the **ARManager**.
4. Each frame, the **PlayerController** reads user input (screen taps) and creates projectiles along the player's current viewing direction.
5. The **GameController** updates all active entities, checks for collisions between projectiles and enemies, and adjusts the player's health or shields if enemies get too close.
6. The **ScoreManager** updates the score when enemies are destroyed, and the **UIManager** refreshes the HUD to show the latest score and health values.
7. When the game ends (e.g., the player's health reaches zero), the **GameController** notifies the **ScoreManager** and **DataManager** to save the final score, then instructs the **UIManager** to display the game over screen.

This modular architecture is intended to keep responsibilities clearly separated, making the system easier to implement, test, and extend with future features such as additional enemy types, power-ups, or online leaderboards.

4 User Interface

4.1 Overview

The user interface (UI) of *AstroStreet AR* is designed to be simple and usable outdoors while the player is holding a smartphone. Navigation is primarily tap-based, and the most important information (score, health, and basic controls) is always visible during gameplay.

The main UI elements include:

- A **main menu** for starting the game and accessing other screens.
- An **AR gameplay view** that overlays enemies and HUD elements on top of the camera feed.
- A **pause menu** for temporarily stopping gameplay.
- A **game over screen** that summarizes the player's performance.
- A **high scores** screen.
- A **settings** screen for basic configuration options.

4.2 Main Menu

When the application launches, the user is shown the main menu screen. This screen contains:

- The game title: **AstroStreet AR**.
- A brief subtitle or tagline (e.g., "Aim at the sky and blast incoming asteroids!").
- A set of buttons:
  - **Start Game** – begins a new AR gameplay session.
  - **High Scores** – opens the high scores screen.
  - **Settings** – opens the settings screen.
  - **Exit** (optional on platforms that support it) – closes the application.

The main menu is displayed on top of a static or subtly animated background image related to outer space or the night sky. From this screen, the player can reach all other major UI flows.

4.3 AR Gameplay Screen

The AR gameplay screen is the core of the user experience. It uses the device camera feed as the background and overlays virtual game elements on top.

The key elements of this screen are:

- **Camera View**: The live camera feed fills the entire screen to show the player's real-world surroundings.
- **Crosshair or Reticle**: A small graphical reticle is centered on the screen and indicates where the player's shots are aimed.
- **Score Display**: The current score is shown in the upper-left corner of the screen.
- **Health or Shield Bar**: The player's remaining health or shield is represented as a bar or set of segments in the upper-right corner.

- **Timer** (if used): If the game mode uses a time limit, the remaining time is displayed near the top of the screen.
- **Pause Button**: A small pause icon appears in one corner (e.g., top-right), allowing the user to open the pause menu.

To interact with this screen, the user:

- Physically rotates or tilts the phone to aim the reticle at enemies that are rendered in the AR view.
- Taps the screen to fire projectiles in the direction of the reticle.

Enemies (asteroids and alien ships), projectiles, explosions, and power-ups are all rendered as virtual objects anchored in the AR scene.

4.4 Pause Menu

The pause menu appears when the user taps the pause button during gameplay. When the game is paused, the AR scene is dimmed or frozen, and a small overlay panel is shown with the following options:

- **Resume** – returns to the AR gameplay screen and continues the session.
- **Settings** – opens a subset of settings (for example, sound on/off and sensitivity) without leaving the session.
- **Quit to Main Menu** – ends the current game and returns to the main menu.

4.5 Game Over Screen

When the player's health reaches zero or the game session ends for another reason (such as a time limit), the game over screen is shown.

This screen contains:

- A prominent "Game Over" title.
- The player's final score.
- An indication if a new high score was achieved.
- Buttons for:
  - **Play Again** – start a new game session.
  - **Return to Main Menu** – go back to the main menu.

4.6 High Scores Screen

The high scores screen displays a list of the top scores recorded on the device. A simple layout might include:

- A title such as "High Scores".
- A table or vertical list showing rank, score value, and date achieved.
- A button to return to the main menu.

4.7 Settings Screen

The settings screen allows the user to adjust basic configuration options for *AstroStreet AR*. Possible settings include:

- **Sound**: toggle sound effects on or off.
- **Control Sensitivity**: adjust how quickly the reticle responds to device movement.
- **Brightness or UI Contrast**: optional adjustments to help make the HUD more readable outdoors.

The settings screen is accessible from both the main menu and the pause menu, and includes a button to return to the previous screen.

Overall, the user interface is designed to minimize on-screen clutter so that players can clearly see both the real world and the virtual objects they are interacting with while playing *AstroStreet AR*.

5 Database Design and Explanation

5.1 Overview

AstroStreet AR uses a small amount of persistent data to store player high scores and basic configuration settings. In a mobile implementation, this data can be stored using a lightweight mechanism such as a local database (e.g., SQLite), key–value storage, or the game engine's built-in persistence system. This section describes the logical database design that the **DataManager** component works with, independent of the specific storage technology.

The primary data entities are:

- **HighScore**: represents a single recorded high score.
- **Setting**: represents a user-adjustable configuration, such as sound or control sensitivity.

5.2 HighScore Entity

The **HighScore** entity stores the results of completed game sessions. Each record corresponds to one high score achieved by the player.

Field	Type	Description
HighScoreID	Integer	Unique identifier for the high score entry.
PlayerName	Text	Optional name or initials provided by the player (e.g., "RJ").
Score	Integer	Final score achieved at the end of a game session.
DateAchieved	DateTime	Date and time when the score was recorded.

In a simple single-player implementation, `PlayerName` can be optional or default to a generic value (such as "Player"). The **DataManager** is responsible for:

- Inserting a new **HighScore** record when a game ends.
- Retrieving the top N scores ordered by `Score` (and optionally by `DateAchieved`) for display on the High Scores screen.
- Deleting or pruning old scores if a maximum number of stored records is desired.

5.3 Setting Entity

The **Setting** entity stores simple key–value pairs that represent user preferences for the game.

Field	Type	Description
SettingKey	Text	Unique key name for the setting (e.g., ``soundEnabled'', ``sensitivity'').
SettingValue	Text	String representation of the value (e.g., ``true'', ``0.8''). This can be parsed into the appropriate type (boolean, number, etc.) by the **DataManager**.

Examples of settings that might be stored include:

- `soundEnabled`: whether sound effects are on or off.
- `sensitivity`: a numeric value controlling how quickly the reticle responds to device movement.
- `uiBrightness`: an optional value controlling HUD brightness or contrast.

The **DataManager** provides simple operations to:

- Read a setting by its `SettingKey`, returning a default value if it has not been set yet.
- Write or update a setting when the user changes options in the Settings screen.

5.4 Relationships and Data Flow

The logical relationship between the entities is straightforward:

- Each **HighScore** record is independent and does not reference other records.
- Each **Setting** record is identified by a unique `SettingKey`, forming a key–value store.

During normal operation:

1. At application startup, the **DataManager** loads existing **Setting** records and applies them to the game (for example, enabling or disabling sound).
2. When the user finishes a game, the **ScoreManager** passes the final score to the **DataManager**, which creates a new **HighScore** record with the score and current date.
3. When the user opens the High Scores screen, the **UIManager** requests the list of top scores from the **DataManager** and displays them.
4. When the user changes a setting in the Settings screen, the **UIManager** calls the **DataManager** to update the corresponding **Setting** record.

This simple database design is sufficient for the current scope of *AstroStreet AR* and can be extended in the future (for example, to support multiple named players or to synchronize scores with an online leaderboard) without significantly changing the rest of the system architecture.

6 Glossary

This section defines key terms and acronyms used throughout the *AstroStreet AR* Software Design Document.

Term	Definition
AR (Augmented Reality)	A technology that overlays virtual objects (such as asteroids and alien ships) onto a live view of the real world, typically using a camera and device sensors.
UI (User Interface)	The visual elements and interactive controls that the player uses to navigate the application, such as buttons, menus, and icons.
HUD (Heads-Up Display)	On-screen information shown during gameplay, including the score, health or shield bar, timer, and other status indicators.
FPS (Frames Per Second)	The number of times per second that the game updates and redraws the screen; higher FPS generally results in smoother gameplay.
API (Application Programming Interface)	A defined set of functions or endpoints that one software component can use to communicate with another, such as an AR framework or a web service.
ARCore / ARKit	Platform-specific AR frameworks provided by Google (ARCore) and Apple (ARKit) that supply camera images, device pose, and tracking capabilities to mobile applications.
Component	A logical part of the system with a specific responsibility, such as **GameController**, **EnemyManager**, or **DataManager**.
Entity	A logical representation of a data object stored by the system, such as a **HighScore** or **Setting**.
Game Session	A single continuous playthrough that starts when the player begins a game and ends when the player loses, quits, or returns to the main menu.
Persistent Data	Information that is saved to storage so that it remains available after the application is closed, such as high scores and user settings.

7 References

This section lists external resources, tools, and documentation that inspired or informed the design of *AstroStreet AR*.

References

[1] Unity Technologies, *Unity Manual: AR Foundation*. Available at: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation

[2] Google, *ARCore Developer Guide*. Available at: https://developers.google.com/ar

[3] Apple Inc., *ARKit Overview*. Available at: https://developer.apple.com/augmented-reality

[4] Jason Gregory, *Game Engine Architecture*. A K Peters/CRC Press, 2nd edition, 2014.

[5] Leslie Lamport, *LaTeX: A Document Preparation System*. Addison-Wesley, 2nd edition, 1994.