

## Software Design Description Document (Architecture Section)

### 6.1 Introduction

Akemi Isles is a 2D narrative-driven adventure game built in Unity (C#), where players explore an island, collect and use items, and interact with NPCs to progress through the story and save the island from disaster. This document presents the architecture and detailed design for the software of Akemi Isles. The project performs a cohesive gameplay system by organizing key components - such as item collection, dialogue management, and story progression - under a unified game manager which coordinates all interactions and maintains the global game state.

The following sections describe the system's objectives, interfaces, and high level architecture.

#### 6.1.1 System Objectives

The objective of this application is to provide a narrative-driven player experience where exploration and item usage advance the story. By having a centralized Game Manager that manages game state consistently, the application can track player progress, inventory, and story triggers. The seamless interaction between components such as the inventory manager, dialogue system, and in-game objects (doors, NPCs, items) is possible because of the Game Manager. With all of this implemented, Akemi Isles offers an intuitive 2D user interface for collecting, viewing, and using items.

#### 6.1.2 Hardware, Software, and Human Interfaces

Details of each are explained individually below.

#### 6.1.2.1 Unity Game Engine Interface

The base system of Akemi Isles interfaces primarily with Unity Engine Version 6000.0.31f1 using the C# scripting API. Unity acts as the framework for rendering, physics, scene management, and asset handling. All game logic (item interaction, player control, dialogue progression, etc.) is implemented in C# scripts that interface with Unity's Monobehavior lifecycle methods (Start(), Update(), OnTriggerEnter2D(), etc.). Unity's built in SceneManager is used for scene transitions with the game manager.

#### 6.1.2.2 Input Hardware Interface

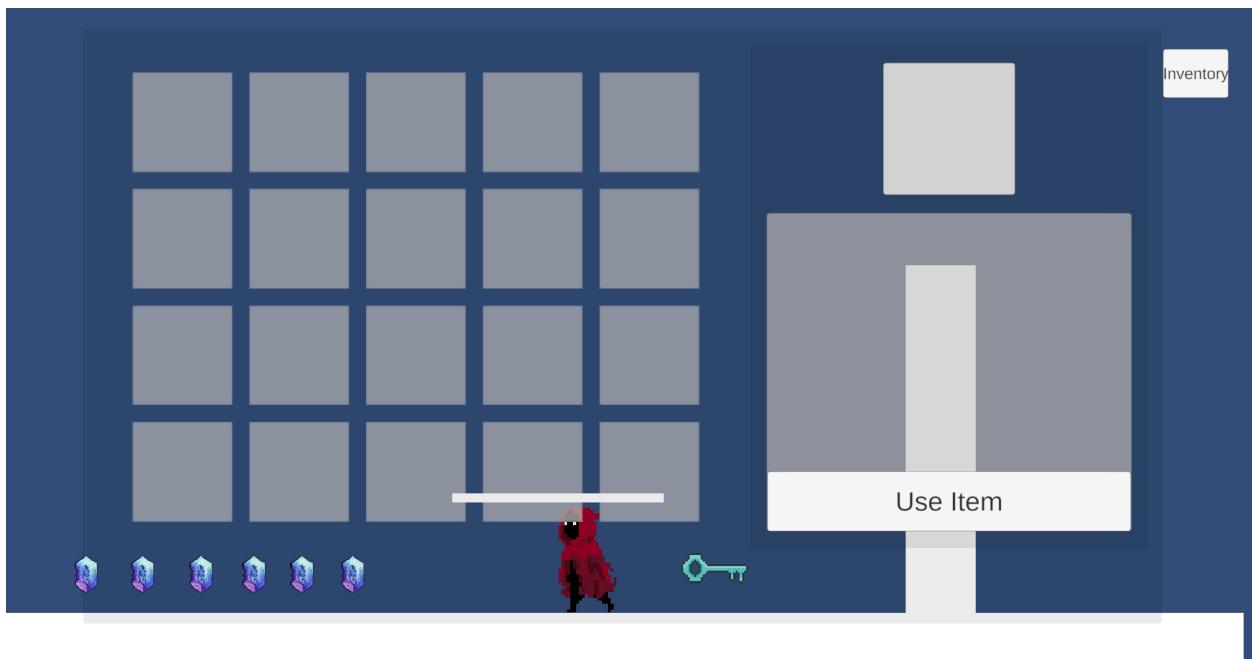
The game interfaces with standard keyboard and mouse input hardware.

- Keyboard: used for player movement (WASD or arrow keys + spacebar)
- Mouse: used for UI interactions, such as selecting items in the inventory or clicking dialogue options. All input is handled through Unity's Input System Package, using event-based input actions for better flexibility and rebind support.

#### 6.1.2.3 Graphic User Interface

The user interface (UI) is implemented using Unity's Canvas System and TextMeshPro for rendering text. The main GUI elements include:

- Inventory Menu: displays collected items, item descriptions, and allows item usage
- Dialogue Box: shows NPC dialogue lines, name tags, and dialogue options
- Interaction Prompts: appear when the player is near interactable objects
- Pause Menu: Allows players to resume, quit, or access settings



#### 6.1.2.4 File Interface

As of now, Akemi Isles does not implement persistent save data. However, the system architecture allows future implementation using Unity's PlayerPrefs API or JSON-based file serialization for local saves. No external databases or online storage are used.

#### 6.1.2.5 Third-Party Libraries and Assets

The following third-party assets are integrated into the system:

- TextMeshPro (Unity built-in) - used for high-quality text rendering throughout the UI and dialogue system
- Cinemachine (Unity package) - provides smooth camera control and tracking for the player
- 2D Sprite Assets (imported PNGs) - used for characters, environment, and item sprites

#### 6.1.2.6 Game Manager Interface

The GameManager singleton coordinates:

- Player Controller (movement, interactions)
- Inventory Manager (item collection / usage)
- Dialogue Manager (NPC dialogue progression)
- Scene Controller (scene and story changes)

#### 6.1.2.7 Hardware and Platform Interface

The game is designed to run on Windows 10+ systems as well as MacOS and supports both keyboard and mouse input devices. The game runs in 2D with a resolution of 1920x1080 by default and supports fullscreen or windowed play.

### 6.2 Architectural Design

Akemi Isles uses a modular architecture centered around a single persistent Game Manager that coordinates all gameplay systems. This approach allows individual subsystems (inventory, dialogue, player control) to remain independent while still communicating through shared state variables and events.

#### 6.2.1 Major Software Components

Component	Description
Game Manager	The central controller that maintains global game state, tracks quest progress, and triggers story events based on collected items or completed objectives.
Player Controller	Handles player movement, interaction input, and collision detection in the 2D world
Inventory Manager	Manages the player's collected items, displays the inventory UI, and handles item use logic (ex: open door with key)
Item System	Defines collectible objects through Scriptable Objects, storing

	data such as name, sprite, and effects on player stats or world objects.
Dialogue System	Manages NPC conversations, tracks dialogue branches, and adapts responses based on the current game state provided by the Game Manager.
World Interaction System	Includes triggers (e.g., doors, chests, switches) that respond to player actions or specific items being used.
UI System	Displays inventory, dialogue boxes, and other on-screen feedback. Communicates closely with both the Game Manager and Inventory Manager.

### 6.2.2 Major Software Interactions

The following describes typical communication and control flow between components during gameplay events:

- 1) The **Player Controller** detects nearby interactable objects and passes the interaction request to the **Game Manager**.
- 2) The **Game Manager** queries the **Inventory Manager** to verify whether the player has the required item.
- 3) If conditions are met, the **Game Manager** triggers an event in the **World Interaction System** (e.g., opening a door, updating NPC state).

- 4) The **Dialogue System** uses data from the **Game Manager** to determine which dialogue lines or branches are available.
- 5) The **UI System** displays updated information, such as removed or newly acquired items, changed dialogue, or new objectives.
- 6) Future save/load features will serialize the **Game Manager's** state for persistence between play sessions.

### 6.2.3 Architectural Design Diagrams

#### 6.2.3.1 Use Case Diagram

Purpose: Illustrates all major ways the player interacts with the game system.

Description: The system boundary is represented by a rectangle labeled “Akemi Isles.” Within it, oval-shaped nodes represent use cases such as “Collect Item,” “Open Inventory,” “Talk to NPC,” “Use Item,” and “Trigger Story Event.” The external Player connects to each use case, representing user-initiated actions. This diagram emphasizes how gameplay actions flow through the main systems (inventory, dialogue, and story progression).

# Akemi Isles Game

## Use Cases

- move character
- collect item
- open inventory
- talk to NPC
- solve puzzle
- progress story
- view dialogue choices
- pause / resume game



Player

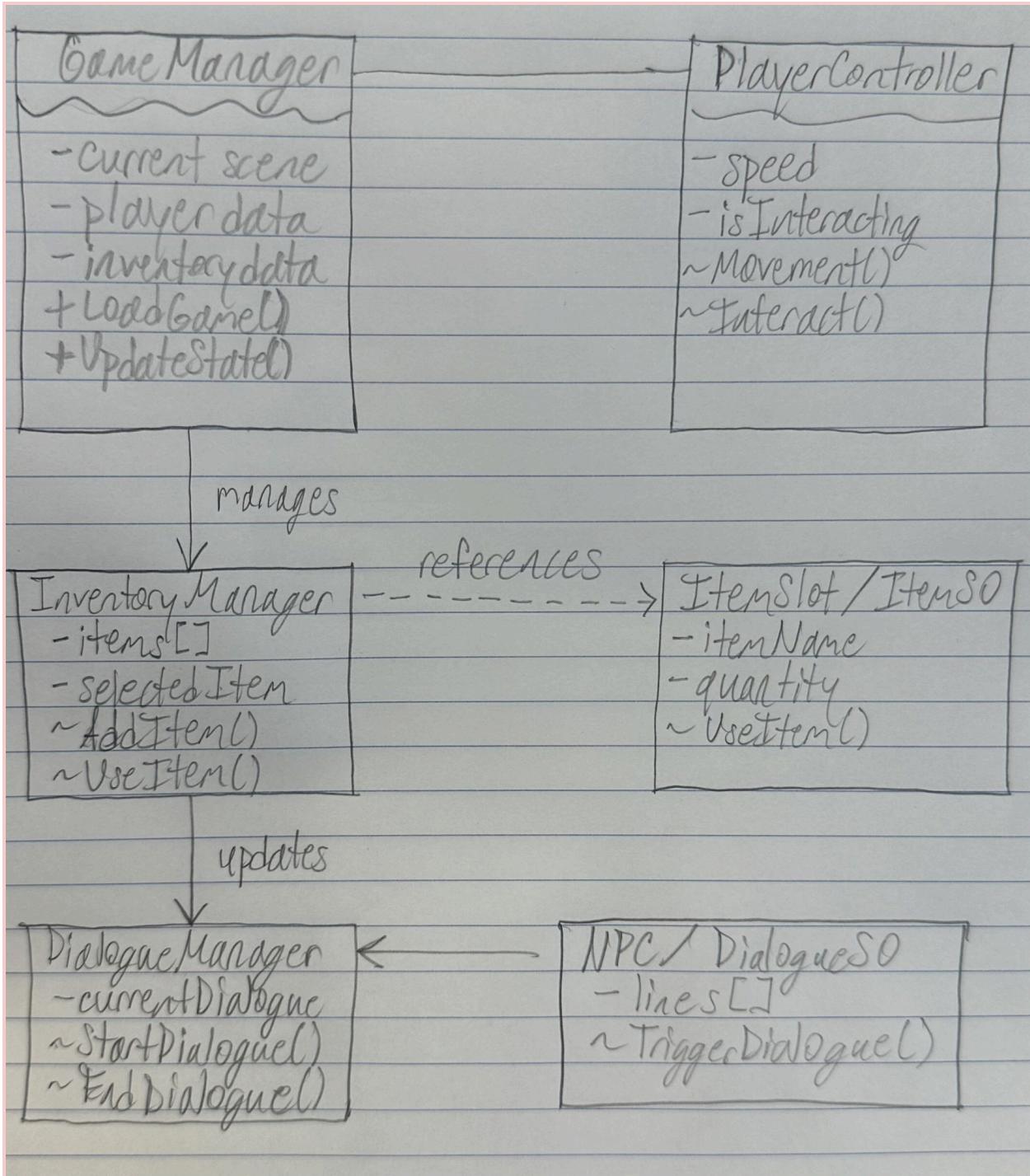
### 6.2.3.2 Top-Level Class Diagram

Purpose: Shows the relationships between the main classes and systems in the game.

Description: The GameManager is the central class responsible for maintaining the global state and coordinating interactions. It holds references to major subsystems:

- PlayerController – handles player input and movement.
- InventoryManager – manages a collection of ItemSlot objects that store ItemSO data.
- DialogueManager – controls NPC dialogues and scene-triggered conversations.

Classes interact via public methods and shared events. The structure enforces modularity, making each system independently maintainable while remaining coordinated by the GameManager.



### 6.2.3.3 Component / Deployment Diagram

Purpose: Displays how different systems and components interact at runtime.

Description: The GameManager acts as the hub of communication between gameplay systems.

- Player Controller, Inventory Manager, Dialogue System, and Scene Controller connect to the GameManager.
- Item System supplies data and scripts for items that can be picked up or used.
- Input System sends keyboard and mouse events to the Player and UI elements.
- UI Layer displays menus and dialogue boxes, updating in response to data changes in the GameManager.

This architecture ensures loose coupling between components and allows the

GameManager to maintain consistent global state across all scenes.

