FYP Solution Architecture & Project

Dead Reckoning AI Survival (2D Game)

Supervisor:Miss.Sana Irshad

IQRA University

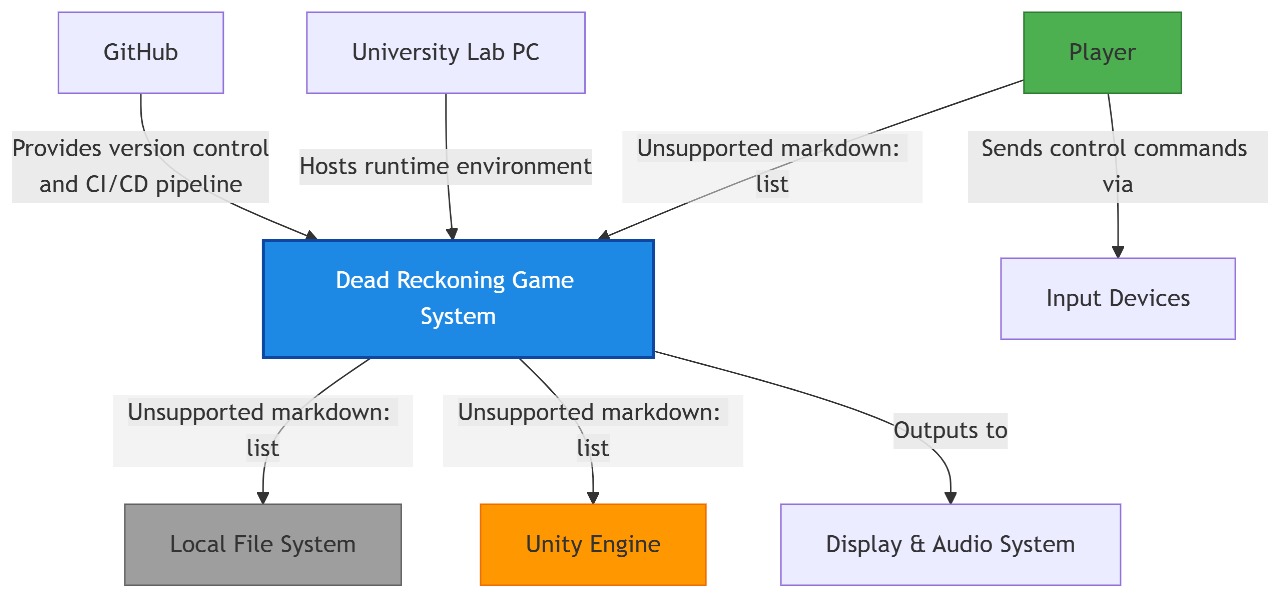
02-06-2025

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Group Members** | | | | |
| ***Sr.*** | ***Student ID*** | ***Name*** | ***Email*** | ***Mobile*** |
| 1. | IU05-0321-0096 | M. ZAIN ALAM | muhammad.14150.ac@iqra.edu.pk | 03357569497 |
| 2. | IU05-0121-0164 | MOIZ ALI KHAN | moiz.13570.ac@iqra.edu.pk | 0332-0327950 |
| 3. | IU05-0122-0261 | SYEDA AYESHA MEHMOODI | syeda.14679.ac@iqra.edu.pk | 0313-2747411 |
| 4 | IU05-0122-0231 | REHMA MUSKAN | rehma.14649.ac@iqra.edu.pk | 0317-5761993 |

1. Solution Architecture

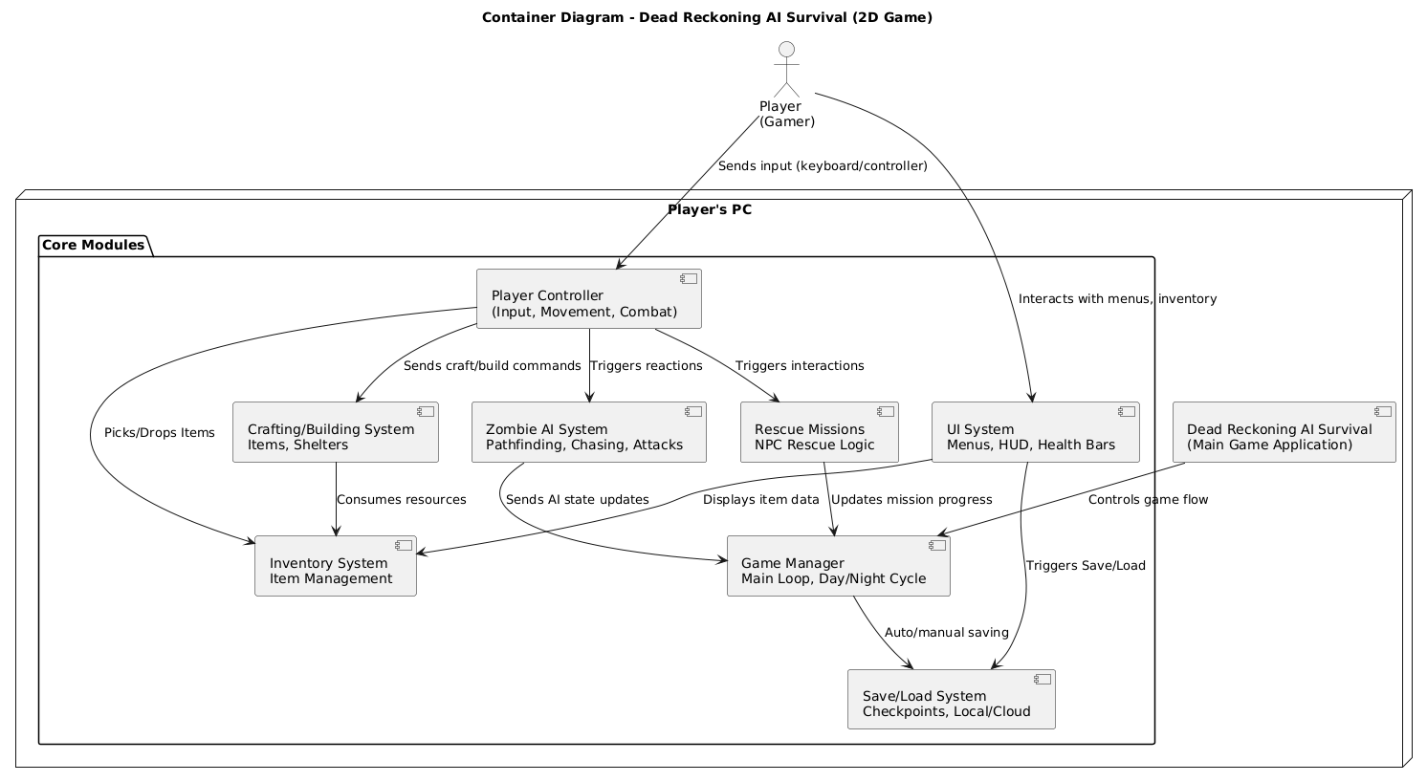
**Key Diagrams & Specifications**

* 1. ***System Context Diagram***

******

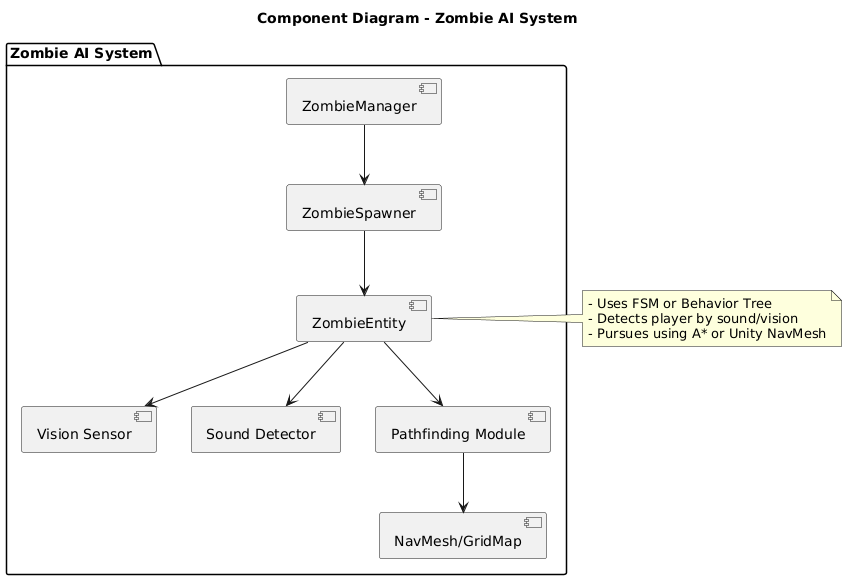
* 1. ***Decompostion***

**1. C4 Container Diagram (Game-Oriented)**

******

* Player (Actor): Represents the real-life gamer who interacts with the game through inputs and UI.
* Player's PC (Node): Hosts the main game application.
* Main Game Application (Container): The executable game that integrates all core modules.
* Core Modules:
* Player Controller: Handles player input, movement, combat mechanics.
* Zombie AI: Implements enemy logic like detection, pathfinding, and attacks.
* Game Manager: Oversees the game loop, global states like day/night, and progression.
* Inventory System: Manages collected, equipped, and craftable items.
* Crafting/Building: Allows crafting of tools, weapons, and shelter construction.
* Rescue Missions: Manages interaction and logic for rescuing NPCs.
* UI System: Displays health, inventory, and menus.
* Save/Load System: Saves and loads game progress locally or via cloud.

**2. Component Diagram (for AI System — Zoomed View)**

******

**Zombie AI System**

This module controls all zombie-related logic in the game, making the undead feel responsive and intelligent.

**ZombieManager**

* Central controller that oversees zombie behaviors across the map.
* Manages the lifecycle, updates, and AI state transitions for all zombie entities.
* Interfaces with the game manager to synchronize behavior with game events (e.g., day/night cycle).

**ZombieSpawner**

* Responsible for spawning zombie entities in valid game zones.
* May include spawn conditions based on difficulty, player location, or in-game time.

**ZombieEntity**

* Represents an individual zombie in the game world.
* Executes AI behaviors using FSM (Finite State Machine) or Behavior Tree logic.
* Integrates sensors and pathfinding to dynamically interact with the environment and the player.

**Vision Sensor**

* Allows the zombie to detect the player if within line of sight and viewing angle.
* Works in tandem with lighting and field-of-view logic.

**Sound Detector**

* Triggers zombie awareness when loud sounds occur (e.g., gunshots, footsteps).
* May simulate radius-based sound propagation for realism.

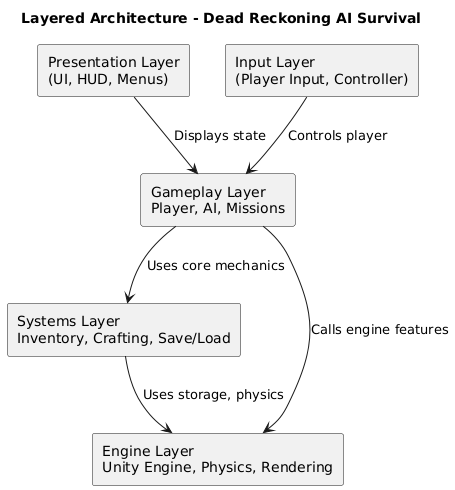
**Pathfinding Module**

* Calculates optimal paths for chasing the player or patrolling using:
  + A\* Algorithm in 2D Grid environments.
  + Unity NavMesh (if using Unity Engine) for navigation in complex maps.

**NavMesh/GridMap**

* Underlying structure that represents walkable areas in the game.
* Used by the pathfinding system to calculate valid movement paths.

**3. Layer Diagram**

******

1. Presentation Layer

* Contains all user-facing elements such as:
  + Menus, Heads-Up Display (HUD), Health Bars, Inventory UI
* Goal: Deliver game feedback and information to the player.

2. Input Layer

* Handles input from keyboard, mouse, or game controller.
* Translates raw input into player actions (movement, attack, interactions).
* Ensures responsive gameplay control.

3. Gameplay Layer

* Core game logic resides here:
  + Player control, Zombie AI behavior, Rescue Missions, Day/Night Cycle
* Coordinates player actions with world logic and game progression.

4. Systems Layer

* Functional sub-systems for:
  + Inventory Management, Crafting/Building, Saving and Loading game state.
* Provides reusable mechanics that support gameplay.

5. Engine Layer

* Base layer relying on the Unity Game Engine:
  + Handles physics simulation, animation, rendering, and low-level game loop.
  + Manages asset loading, audio, and platform-specific services.
  1. ***Data Design***

| **Data Category** | **Specific Elements** | **Storage Method** |
| --- | --- | --- |
| **Player Profile** | Health, Stamina, XP Level, Skill Points, Unlocked Abilities | PlayerPrefs (Unity) |
| **Inventory** | Weapons, Ammo, Crafting Materials, Medical Supplies, Key Items | JSON File |
| **Shelter State** | Built structures, Upgrades, Defenses, Storage Containers | Binary Save File |
| **Game Progress** | Completed missions, Rescued NPCs, Discovered locations, Story choices | JSON File |
| **World State** | Resource node status (harvested/regrown), Zombie spawn points, Time of day | Scene-specific Binary File |
| **Settings** | Key bindings, Audio levels, Graphics options, Language | PlayerPrefs (Unity) |

#### **2. Schemas & Formats**

| **Data Type** | **Format** | **Justification** |
| --- | --- | --- |
| Player Profile | **Unity PlayerPrefs** | Built-in secure storage for small key-value pairs |
| Inventory/Saves | **Compressed JSON** | Human-readable for debugging, compressible for size |
| Shelter/World State | **Binary** | Smaller size, faster loading, harder to tamper with |
| Game Configuration | **ScriptableObjects** | Unity-native format for designer-friendly editing |

#### **3. CRUD vs. Event Sourcing**

| **Approach** | **Implementation** |
| --- | --- |
| **CRUD (Primary)** | - Inventory management (add/remove items) - Player stats updates (health, XP) - Shelter construction/destruction |
| **Event Logging** | - Critical actions (zombie kills, mission completions) for analytics - Anti-cheat tracking (unusual item acquisition) - Save file versioning history |

**1 . File System Structure**

DeadReckoning\_Data/

├── Saves/

│ ├── slot1.save *# Binary game state*

│ ├── slot1\_meta.json *# Save metadata (JSON)*

│ └── backup/ *# Auto-backups*

├── Config/

│ ├── keybinds.cfg *# Input settings*

│ └── game\_settings.cfg *# Graphics/audio*

├── Inventory/

│ └── player\_inv.json *# Player inventory*

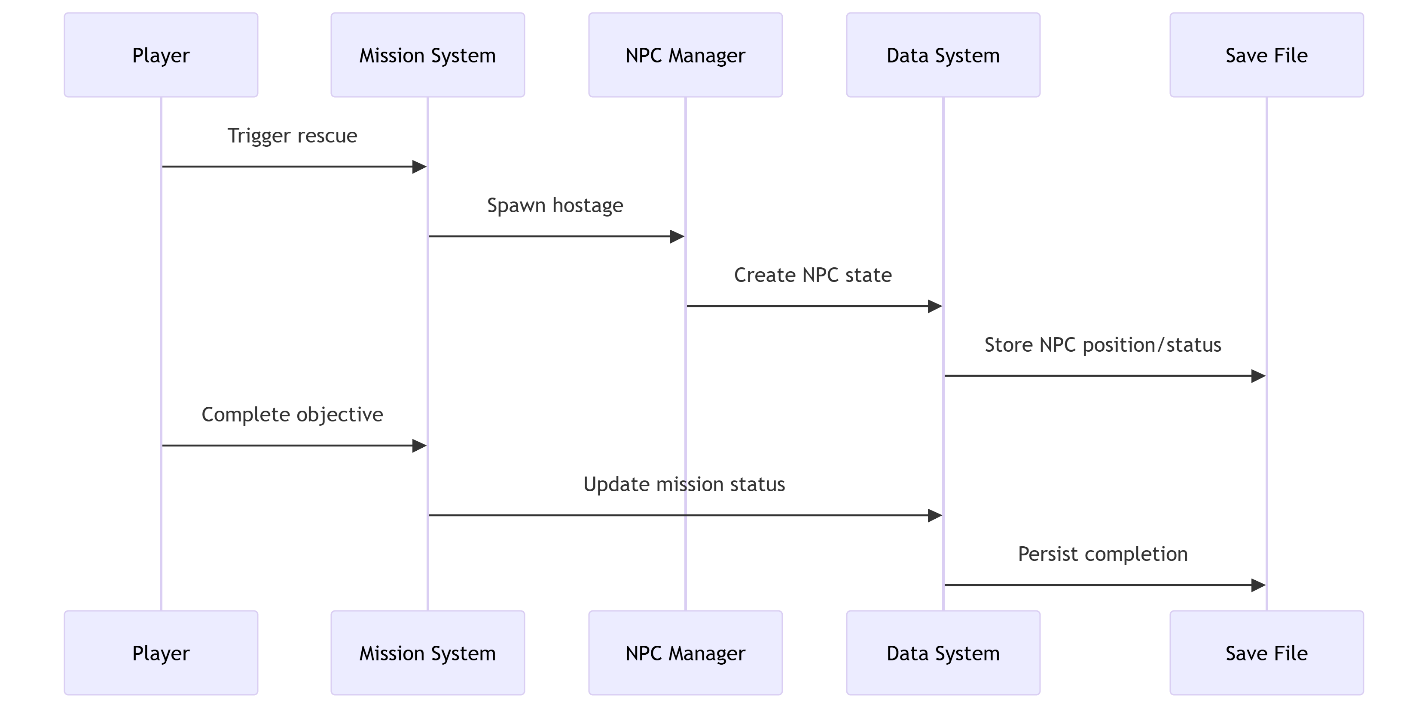
└── Logs/

├── events.log *# Critical action log*

└── debug.log *# System diagnostics*

Game-Specific:

1. **Optimize Save Files**
   * Use Unity's PlayerPrefs for small data (settings, key bindings)
   * Implement chunk-based saving for large worlds (save only changed areas)
   * Compress JSON with LZ4 before writing to disk
2. **Rescue Mission Data Flow**



o

* 1. ***Technology Stack***

**1. Programming Languages**

* **C#** (if using Unity): Ideal for game development with strong Unity support and robust libraries.

**2. Game Engines**

* **Unity** *(preferred)*:
  + Large community, plugin support, rich 2D tools.
  + Built-in AI pathfinding and physics.

**3. Graphics & Asset Tools**

* **Piskel / Aseprite / Photoshop**: For pixel art creation and animations.
* **Tiled**: For creating tile-based levels.

**4. Database / Storage**

* **JSON files** (for offline saves): Easy to serialize/deserialize player state..
* **Firebase Realtime DB or Firestore (Optional)**: For cloud save or syncing across devices.

**5. Version Control**

* **GitHub**: For team collaboration, versioning, issue tracking, and integration with CI/CD.

**6. Audio Tools**

* **Audacity**: Editing sound effects.
* **FreeSound.org**: Source for royalty-free audio assets.

**7. Methodologies**

* **Waterfall**: Clear, sequential phases: Requirement Gathering → Design → Implementation → Testing → Deployment.
* Suitable for structured academic projects with fixed scope and clear deadlines.
* Ensures that deliverables are well-defined and frozen early, avoiding scope creep.

**Technology Stack Justification (ADR Summary)**

|  |  |  |
| --- | --- | --- |
| Component | Choice | Justification |
| Game Engine | **Unity** | Familiar to students, supports 2D/AI well, lab licenses available. |
| Language | **C#** | Easy to use with Unity, readable, strong community support. |
| Database | **JSON** | Lightweight, offline support, no server setup required. |
| Cloud (Optional) | **Firebase** | Easy integration, real-time sync, secure, free tier available. |
| Graphics Tool | **Piskel/Photoshop** | Students already have experience, lightweight, efficient. |
| Version Control | **GitHub** | Enables team collaboration, transparency, and backups. |
| Methodology | **Waterfall** | Predictable planning and delivery but less flexible to design changes mid-way. |
| Audio | **Audacity/FreeSound** | Free, sufficient for indie game sound design. |

* 1. ***Cross-Cutting Concerns***

**Security Considerations**

Even though the game operates fully offline with no user accounts or network services, core security principles still apply to protect player data and ensure integrity:

**▶ Local Save Integrity**

* Game progress is serialized into local .json files.
* Data is written to Unity’s persistentDataPath, ensuring it is isolated from unauthorized access in normal OS environments.

**▶ Data Tampering Prevention**

* Save files are human-readable; to prevent exploitation or tampering (e.g., cheating):
  + Optional obfuscation or lightweight encryption (e.g., XOR or Unity’s PlayerPrefsX) can be applied.
  + File hashes can verify integrity at load time.

**▶ Secure Serialization**

* All save/load operations use Unity’s built-in JsonUtility or Newtonsoft.Json, which are resistant to common injection and deserialization vulnerabilities.
* Only trusted internal classes are deserialized to prevent unexpected object instantiation.

**▶ OWASP Top 10 Alignment (where applicable):**

| **Risk** | **Mitigation** |
| --- | --- |
| Insecure Storage | Files saved in protected app-specific paths. |
| Insecure Deserialization | Strict schema enforcement in load logic. |
| Code Exposure | Build process strips debug logs and metadata. |

**Scalability Potential**

Although scalability isn't a current concern for this **single-player** experience, future enhancement possibilities are acknowledged:

**▶ Future Cloud Extension (Optional)**

* Cloud features like leaderboard sync or rescue-mission sharing could leverage:
  + **Firebase Realtime DB or Firestore** for low-maintenance backend.
  + **Serverless functions** for processing player stats and gameplay analytics.

**▶ Edge Case Handling**

* System architecture is modular, allowing for efficient memory management:
  + Only active entities (e.g., zombies on screen) are processed.
  + Object pooling (Unity’s ObjectPool<T>) can be introduced for scalability in zombie spawns or crafting sprees.

**Observability**

Robust monitoring during development aids in debugging, performance tuning, and post-mortem analysis:

**▶ Logging**

* Debug.Log, LogWarning, and LogError are used to track significant game events.
* During testing, logs are captured and exported to local files for offline review.
* Error scenarios (e.g., missing save files) are displayed with friendly in-game messages.

**▶ Metrics & Profiling**

* Unity Profiler is used to track:
  + FPS fluctuations during large zombie swarms.
  + Memory usage during item crafting or inventory updates.
  + Garbage collection spikes.

**▶ Failure Feedback**

* Any failed serialization or file system issue is caught and logged with context.
* Non-blocking UI popups inform players of load/save problems without crashing the game.

**Maintainability & Testability**

To ensure long-term maintainability and smooth teamwork throughout the Waterfall model phases:

**▶ Modular Architecture**

* Core game features are segmented into reusable scripts:
  + PlayerController.cs, ZombieAI.cs, CraftingSystem.cs, etc.
* Follows the **single-responsibility principle** for cleaner codebase management.

**▶ Manual Dependency Management**

* Unity’s component model is used in place of formal DI tools.
* Dependencies like audio, UI managers, and save handlers are assigned through the Inspector or loaded using Singleton patterns — avoiding rigid coupling.

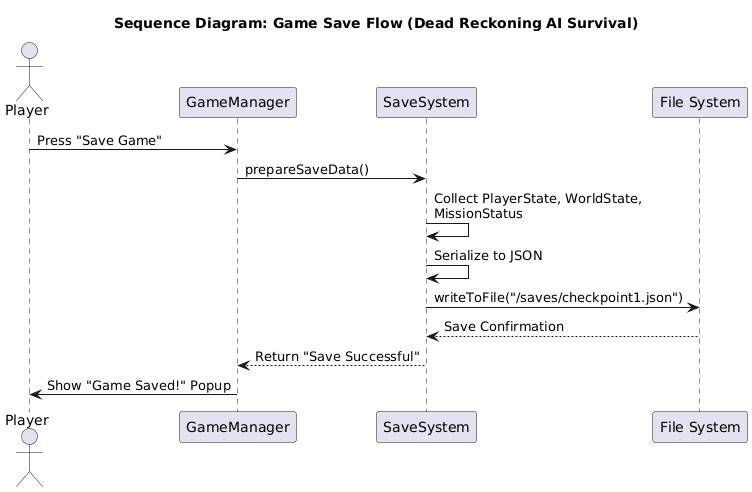
**▶ Testing Approach**

* Logic-based components (e.g., inventory, AI states) are written to be testable in Unity’s **PlayMode and EditMode test runners**.
* Boundary conditions (e.g., empty inventories, corrupted save files) are covered in pre-release builds.

**▶ Code Style and Version Control**

* C# naming conventions and inline documentation ensure clarity.
* GitHub repositories enforce structure via:
  + Branch protection rules.
  + Pull request reviews.
  + Release tagging aligned with Waterfall milestones.

**Sequence Diagram: Game Save Flow**



* 1. ***Contraints & Compliance***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category | Constraint | Type | Impact | Planned Resolution / Notes |
| Technical | Game must run on mid-spec university PCs | Hardware | Medium | Optimize performance for 2D; use Unity with efficient assets |
| Technical | No server/database access | Architecture | Low | Use local JSON files for save/load system |
| Technical | Unity Free version only | Licensing | Medium | Use Unity Personal features; avoid cloud/analytics features |
| Technical | No multiplayer or online features | Functional | Low | Focus on polished offline single-player experience |
| Financial | No budget for premium tools/assets | Budget | Medium | Use open-source, royalty-free, or student-created assets |
| Academic | Must complete within March–Dec 2025 | Time | High | Follow Waterfall phases strictly; freeze scope after design |
| Academic | Limited lab access (restricted hours) | Availability | Medium | Use GitHub and home systems to supplement lab time |
| Team Capacity | Fixed team size of 4 students with defined roles | Human Resource | Medium | Divide tasks clearly; avoid over-assignment |
| Ethical/Legal | No PII or biometric data collected | Legal | Low | Offline-only; no GDPR conflicts |
| Ethical/Content | Includes stylized combat with zombies | Ethical | Low | Cartoon-style visuals; avoid realistic gore |
| Compliance | Must align with university FYP policies | Regulatory | High | Supervisor review; documented methodology and deliverables |

**2. Project Planning Phase**

**Work Breakdown Structure (WBS)**

**1. Game Planning & Setup**

1.1 Project kickoff and role assignment

1.2 Requirements gathering and brainstorming

1.3 Game Design Document (GDD) creation

1.4 Asset/tools/repo setup (Unity, GitHub, Piskel)

**2. Core Gameplay Mechanics**

2.1 Implement basic player movement

2.2 Add jumping, crouching, and attack controls

2.3 Integrate basic animation states

2.4 Create base HUD (health, stamina)

**3. Inventory System**

3.1 Design inventory data structure

3.2 Implement item pickup/drop

3.3 Build inventory UI interface

3.4 Test item interaction with world

**4. Crafting & Building System**

4.1 Define crafting recipes and UI

4.2 Implement item combination logic

4.3 Implement shelter placement system

4.4 Integrate with inventory (consume materials)

**5. Zombie AI System**

5.1 Implement detection via sound/vision

5.2 Develop chasing & attacking logic

5.3 Integrate AI with NavMesh/Pathfinding

5.4 Test multiple zombie behaviors

**6. Rescue Missions**

6.1 Design NPC interaction system

6.2 Build mission start/completion flow

6.3 Reward system (XP, resources)

6.4 Test branching outcomes

**7. Save/Load Functionality**

7.1 Implement local save system

7.2 Integrate save triggers (checkpoints, beds)

7.3 Optional: Cloud sync with Dropbox

7.4 Test consistency of saved state

**8. Game Manager & World Logic**

8.1 Create central GameManager object

8.2 Implement day-night cycle and event triggers

8.3 Link modules through GameManager

8.4 Test dynamic world changes (e.g., night increases zombie aggression)

**9. UI/UX Design**

9.1 Create wireframes/mockups (menus, inventory, crafting)

9.2 Implement main menu, pause, and options

9.3 Connect UI to underlying systems

9.4 Polish animations, sounds, transitions

**10. Audio Integration**

10.1 Add background music

10.2 Add SFX: footsteps, attacks, crafting, etc.

10.3 Volume control and audio settings in UI

10.4 Test audio timing and feedback

**11. Level Design**

11.1 Design terrain layout and tilemaps

11.2 Populate game world with interactables

11.3 Place zombies, items, missions

11.4 Build progression-based world regions

**12. Testing & Optimization**

12.1 Perform module-by-module testing

12.2 Fix critical gameplay bugs

12.3 Optimize performance (collisions, loading)

12.4 Test across different systems

**13. Documentation & Reporting**

13.1 Prepare weekly sprint summaries

13.2 Create technical documentation (code, systems)

13.3 Update GDD with real implementation notes

13.4 Write final report

**14. Demo Preparation & Finalization**

14.1 Prepare gameplay walkthrough video

14.2 Build and package final executable

14.3 Conduct internal dry-run presentation

14.4 Final FYP presentation and submission

**Estimation Table (Top-Level Tasks Only)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Task | Optimistic (O) | Most Likely (M) | Pessimistic (P) | Time Estimate (TE hrs) | Estimated Weeks |
| 1 | Game Design & Setup | 10 | 14 | 20 | 14.3 | 0.18 |
| 2 | Core Gameplay Mechanics | 20 | 28 | 40 | 29.3 | 0.37 |
| 3 | Inventory System | 15 | 22 | 30 | 22.3 | 0.28 |
| 4 | Crafting & Building System | 18 | 24 | 32 | 24.3 | 0.30 |
| 5 | Zombie AI System | 24 | 30 | 40 | 30.7 | 0.38 |
| 6 | Rescue Missions | 16 | 20 | 28 | 20.7 | 0.26 |
| 7 | Save/Load Functionality | 12 | 16 | 25 | 17.0 | 0.21 |
| 8 | Game Manager & Logic | 16 | 22 | 30 | 22.3 | 0.28 |
| 9 | UI/UX Design | 18 | 24 | 30 | 24.0 | 0.30 |
| 10 | Audio Integration | 10 | 14 | 20 | 14.3 | 0.18 |
| 11 | Level Design | 16 | 20 | 30 | 20.7 | 0.26 |
| 12 | Testing & Optimization | 24 | 28 | 36 | 28.0 | 0.35 |
| 13 | Documentation & Reports | 18 | 22 | 30 | 22.3 | 0.28 |
| 14 | Demo & Final Prep | 12 | 16 | 24 | 16.7 | 0.21 |

**Milestone Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Milestone ID | Title | Description | Target Week | Deliverable(s) | Acceptance Criteria |
| M1 | Core Gameplay Prototype | Implement base game mechanics and movement system | Week 4 | Character movement, jump, attack mechanics | Player can move, jump, and perform attack actions |
| M2 | Zombie AI & Map Integration | Add enemy logic, zombie detection, and basic tilemap-based level design | Week 6 | Zombie patrol, attack AI, and one test map | Player and zombie interaction works with proper collision |
| M3 | Inventory + Crafting System | Build UI-based inventory and implement item collection and crafting logic | Week 8 | Inventory panel + working crafting system | Items can be picked, combined, and used |
| M4 | UI/UX System + Dialogue | Integrate health bar, menus, and dialogue choice system | Week 10 | Fully functional UI and basic choice system | UI is visible, responsive; one dialogue branch implemented |
| M5 | Save/Load Game State | Ability to store and restore player state, position, and inventory | Week 11 | Save/load working with test scenarios | Save file persists and restores full state accurately |
| M6 | Art, Sound & Polish Pass | Final game assets and background music/SFX integration | Week 12 | Finalized sprites, audio for events | Visuals and sound meet final game aesthetic |
| M7 | QA Testing & Bug Fixing Sprint | Run full test cases and resolve critical gameplay and UI bugs | Week 13 | Bug report summary and changelog | All major bugs fixed; game stable in all areas |
| M8 | Final Build & Presentation Prep | Create game trailer, gameplay walkthrough, and prepare for supervisor demo | Week 14-15 | Playable .exe, 2-minute trailer, slide deck | Build runs smoothly; presentation assets ready |

**Team Members**

* **Moiz** – AI, Game Logic, Combat
* **Rehma** – Movement, Crafting, Level Design
* **Ayesha** – UI/UX, Rescue, Save System
* **Zain** – Documentation, Coordination

**RACI Table (Key Tasks Only)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Moiz | Rehma | Ayesha | Zain |
| Project Kickoff | Responsible | Responsible | Responsible | Accountable |
| GDD Finalization | Informed | Consulted | Consulted | Responsible & Accountable |
| Player Movement | Informed | Responsible & Accountable | Consulted | Informed |
| Inventory System | Informed | Responsible & Accountable | Consulted | Informed |
| Crafting System | Informed | Responsible & Accountable | Consulted | Informed |
| Zombie AI (Detection, Path) | Responsible & Accountable | Informed | Consulted | Informed |
| Combat System | Responsible & Accountable | Consulted | Informed | Informed |
| Rescue System | Informed | Consulted | Responsible & Accountable | Informed |
| UI Design & Integration | Informed | Consulted | Responsible & Accountable | Informed |
| Save/Load Functionality | Informed | Consulted | Responsible & Accountable | Informed |
| Day/Night Cycle | Responsible & Accountable | Consulted | Informed | Informed |
| Game Manager Integration | Responsible & Accountable | Informed | Consulted | Consulted |
| Level Design | Consulted | Responsible & Accountable | Responsible & Accountable | Informed |
| Mid-Term Demo | Responsible & Accountable | Responsible | Responsible | Responsible |
| Final Testing | Responsible & Accountable | Responsible | Responsible | Consulted |
| Report Writing & Submission | Informed | Informed | Consulted | Responsible & Accountable |
| Final Presentation | Responsible | Responsible | Responsible | Accountable |

**Risk Register Table**

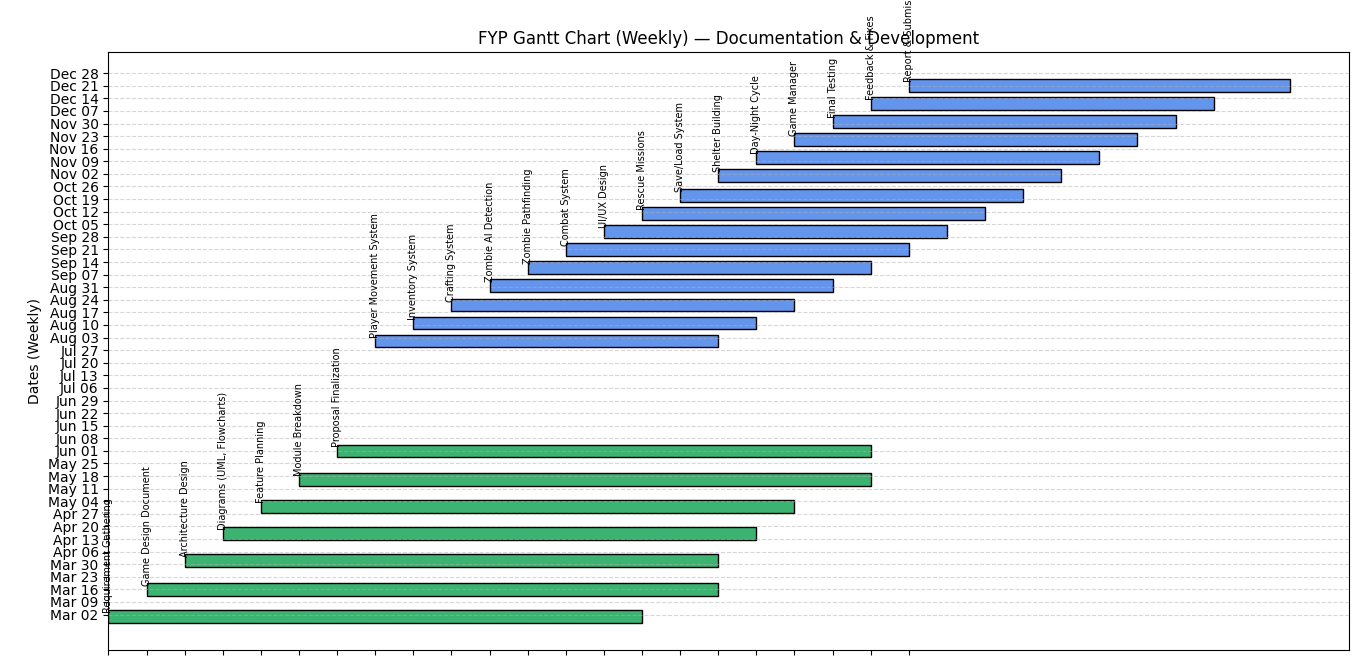
**TECHNICAL**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Risk Title | Likelihood | Impact | Severity | Mitigation Strategy | Contingency Plan |
| T1 | Platform/OS Updates | Medium | High | High | Regular compatibility testing after major OS updates. | Delay affected release; release platform-specific patch later. |
| T2 | Unity Engine Bugs or Deprecation | Medium | High | High | Lock Unity version early; track Unity changelogs closely. | Rollback to last stable version or switch to alternative feature/workaround. |
| T3 | Third-Party Plugin Failure | Medium | High | High | Maintain local backups of plugins; avoid critical reliance. | Replace plugin with an in-house or alternative solution. |
| T4 | Hardware Incompatibility Reports | Low | High | Medium | Collect system logs during playtests; identify patterns early. | Release device-specific patches or block unsupported devices. |
| T5 | Copyright Violation Notice | Low | Critical | Critical | Use only CC-BY/MIT licensed assets; modify by at least 30%; maintain license logs. | 48-hour asset replacement protocol; consult university legal advisor if escalated. |

**NON-TECHNICAL**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Risk Title | Likelihood | Impact | Severity | Mitigation Strategy | Contingency Plan |
| N1 | A team member may become unavailable in the future due to unforeseen commitments | 2 | 5 | 10 | Distribute knowledge across roles and keep backup assignments | Adjust timeline or workload among available team members |
| N2 | The supervisor might be unavailable during a critical decision-making phase | 2 | 4 | 8 | Book meetings early and share progress regularly in documentation | Proceed with agreed plan and get feedback later if needed |
| N3 | Misunderstanding of project scope could occur between team and supervisor | 3 | 4 | 12 | Keep a clear requirement log and get supervisor confirmation | Re-align through a formal review meeting or update scope |

**Gantt Chart Table**



**Communication Plan**

**Meeting Cadence**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Frequency | Purpose | Platform |
| Internal Team Meeting | Weekly (Every Tuesday and Thursday) | Discuss weekly progress, blockers, and task planning | In-person / Zoom |
| Supervisor Update Meeting | Weekly (as per availability) | Review technical progress and get academic feedback | In-person |
| Sprint Retrospective | Bi-weekly | Review completed tasks, feedback, and improvements | In-person / Zoom |

**Communication Channels**

|  |  |
| --- | --- |
| Platform | Usage |
| WhatsApp Group | Daily communication for quick updates and coordination |
| Google Drive | Document sharing, storage, and collaborative editing |
| GitHub | Version control, code collaboration, and issue tracking |
| Email | Formal communication and weekly supervisor updates |

**Supervisor Updates**

* The team will conduct a **weekly in-person meeting** with the project supervisor to discuss:
  + Tasks completed during the week
  + Current challenges or roadblocks faced
  + Goals and tasks planned for the upcoming week
* These regular updates will help ensure:
  + Continuous feedback and guidance
  + Transparent documentation of progress for final reporting

**Code Review Policy**

* All code will be committed to a shared **GitHub repository**
* Each new feature or bug fix must:
  + Be pushed via a **separate branch**
  + Undergo **peer review** before merging to the main branch
  + Include **descriptive commit messages** for clarity
* Major merges will be discussed in the weekly team meeting