Specifications

Starlit Overwatch

February 4 2025

Ziyuan Wang, Rui-jia Meng, Jaden Hum, Jiaqi Yang, Fardin Democri, Yueqiao Zhang, Ahmad Mirza

Noname

Contents

1. Background

1.1 Motivation

Why is this being developed?

- ***Starlit Overwatch*** is being developed to offer a modernized tactical, turn-based strategy experience inspired by classic games like XCOM: UFO Defense (1994).
- The goal is to expand on the classic formula rather than replicate it, introducing robust resource management, weather-influenced combat, and a global map for strategic planning.

What problem is it trying to make better?

Existing tactical games often rely on limited feature sets, with little exploration of dynamic role systems or meaningful environmental effects. **Starlit Overwatch** seeks to address these shortcomings by:

1. **Expanding Resource Management**
    - Players manage critical resources such as **money**, **tech points**, and **raw materials**.
    - These resources impact **infrastructure upgrades**, **technology research**, and **soldier development**, creating a direct link between base-building and mission readiness.
    - Strategic decision-making is reinforced across a dynamic, global map.

2. **Introducing Dynamic Role Systems**

- Soldiers specialize in distinct roles, including **combat**, **medic**, **tech**, and **scout**.
- Each specialization offers **unique buffs and weaknesses**, evolving through a robust **technology tree** that unlocks progressively stronger capabilities.
- The system encourages adaptability, ensuring soldiers play meaningful roles within single-player campaigns.

3. **Enhancing Environmental Effects**
   - **Weather-influenced combat mechanics** add an extra layer of depth.
   - Players must adapt strategies based on changing environmental factors, making each mission unique.
   - Campaigns set in **varied terrains** (e.g., forests, deserts, urban zones) ensure that challenges remain dynamic and strategically engaging.

- **Another problem:** Some tactical games lack a meaningful narrative that ties missions together, making progression feel disjointed.

- **How our game solves it:**
  - **Integrated Storyline**: Missions are connected by a cohesive narrative that provides context and raises the stakes.
  - **Branching Outcomes**: Player decisions during missions can alter future objectives and faction relationships, ensuring each campaign feels unique.

Who primarily asked for it, and why?

- A community of turn-based strategy enthusiasts looking for a fresh spin on classic gameplay. They want a story-driven tactical experience that demands deeper planning, diverse soldier roles, and global resource management.

Other purpose-related information to keep in mind?

- Development will focus on a **smaller, more manageable world** (e.g., a single continent or singular island/village) to streamline gameplay, allowing for more intricate, mission-based storytelling.

1.2 Stakeholders

Type	Name	Coordinates	Success Measure
Owner	Alex Hamilton	Phone: 444-444-4000 Email: alex@overwatchdev.com	*"If the game meets sales targets, has a steady community, and garners positive feedback, we'll know it's successful."*
Manager	Priya Kapoor	Phone: 444-444-4111 Email: priya@overwatchdev.com	*"If development milestones are on time, under budget, and we maintain consistent quality, I'll be satisfied."*
End User (Employee)	Jamie Miller	Internal Dev Team Email: jamie@overwatchdev.com	*"If the game is stable, easy to patch, and encourages ongoing engagement through robust content, it's a success."*
End User (Customer)	Strategy Gamers	(Online Community) **Game Forums**: Reddit communities, official developer forums, and Steam discussion boards. **Social Media**: Twitter, Facebook, Instagram, and dedicated Discord servers.	*"If the gameplay is challenging, replayable, and offers fresh mechanics like weather-driven tactics and role variety, we'll keep playing."*
Indirect User	Gaming Press	Various gaming news outlets **IGN** **GameSpot** **PC Gamer**	*"If the game stands out from other turnbased titles and garners positive reviews and buzz, we'll consider it a hit."*

2. Overview

2.1 Usage

Owner:

1. When: The owner can access the game to play and ensure the functionality of the game whenever he wants.
2. How: The owner launches the game through the game launcher or a desktop shortcut after installation.

Manager:

1. When: During project milestone reviews.
2. How: Launches the game through an IDE (IntelliJ, Visual Studio etc.) and pulls the latest version through version control (Git).

Developers:

1. When: During Development phases.
2. How: Launches the game through an IDE (IntelliJ, Visual Studio etc.) and pulls the latest version through version control (Git).

Player (end user):

1. When: In their free time player uses the game as a pass time activity.
2. How: Player installs the game on their computer and launches the game using the game launcher or a desktop shortcut.

Indirect Users (Gaming Press):

1. When: Anytime they want to report about the game.
2. How: They install the game on their computer and launches the game using the game launcher or a desktop shortcut.

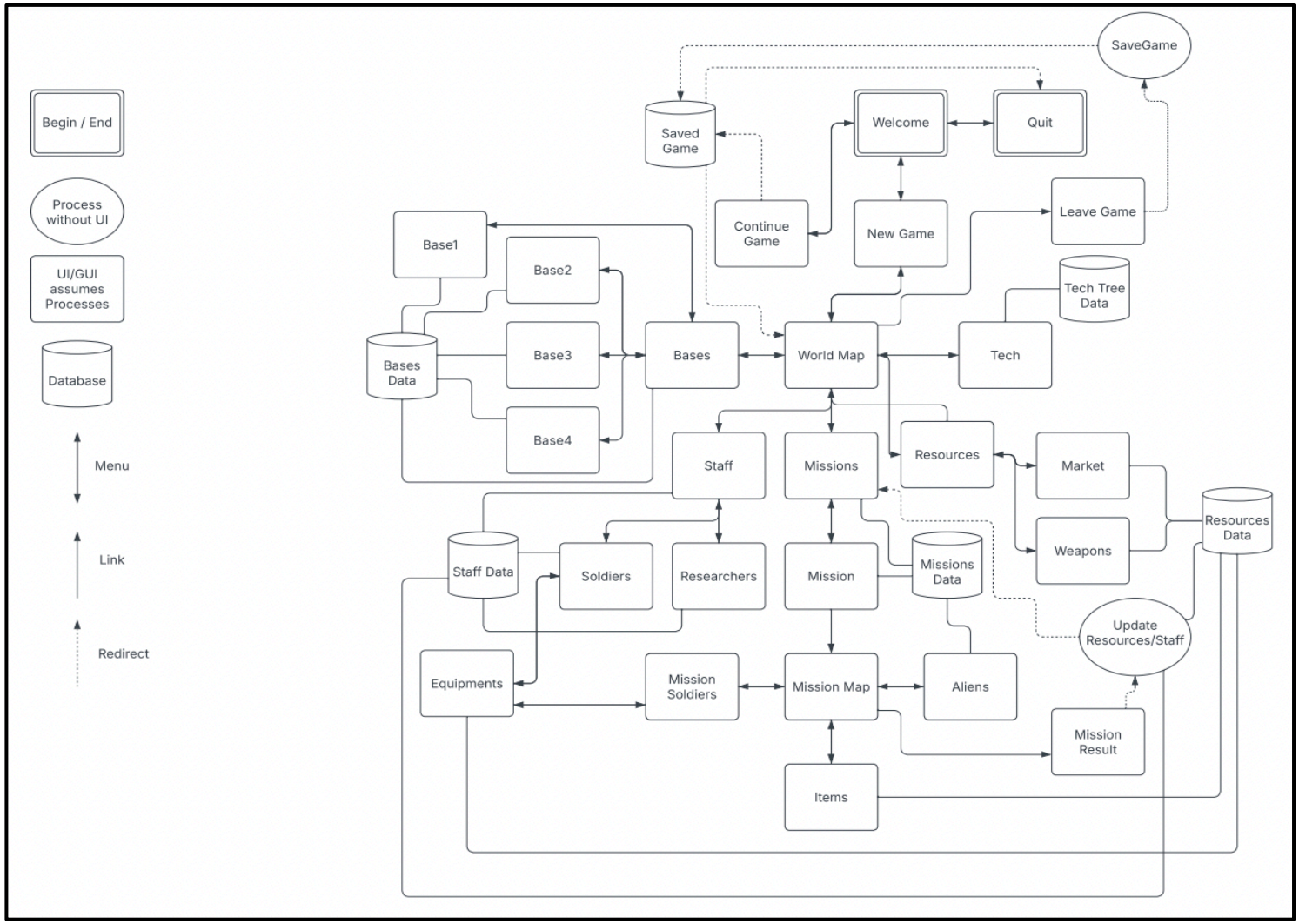
2.2 Design

Development Timeline

Expected Development Start: February 7 th

Expected Development End: April 29 th

Storyboard Diagram



Storyboard Explanation

Welcome:

Welcome page, player can interact with:

1. New Game to start a new game
2. Continue Game to continue the previous game
    a. Connected to saved game database where the saved game is stored and loaded locally
3. Quit to exit game

World Map

The main screen of the game where all other features are accessible through:

1. **Tech**
    a. Connected to Tech Tree database where the technology is stored.
    b. Players can modify, and upgrade take using this feature.

2. **Bases**
    a. Players can choose Base 1,2,3,4 which allows the player to visualize and modify each base.
    b. Connected to Bases Data database where information from each base is stored.

3. **Staff**

    a. Players can choose Soldiers where all soldiers are and can interact with each soldier.

        i. Equipment

            1. Players can visualize and modify soldiers' equipments.

    b. Players can choose Researchers where all researchers are and add or remove researchers.

    c. Connected to Staff database where all information regarding the staff is stored.

4. **Resources**

    a. Players can visualize and interact with all resources acquired.

    b. Weapons:

        i. Players can visualize and utilize the weapons acquired.

    c. Market:

        i. Players can use the market to Buy/Sell resources.

    d. Connected to Resources database where all information about the resources is stored:

5. Missions

a. Players can visualize available missions and choose a mission

b. Mission

i. Player enters the mission map where the mission will take place

1. **Aliens**

   a. The enemies that the player can visualize and interact with

   b. Connected to Mission Data database where all information regarding the mission and the aliens is stored

2. **Items**

   a. Players can visualize their items and use them during the mission

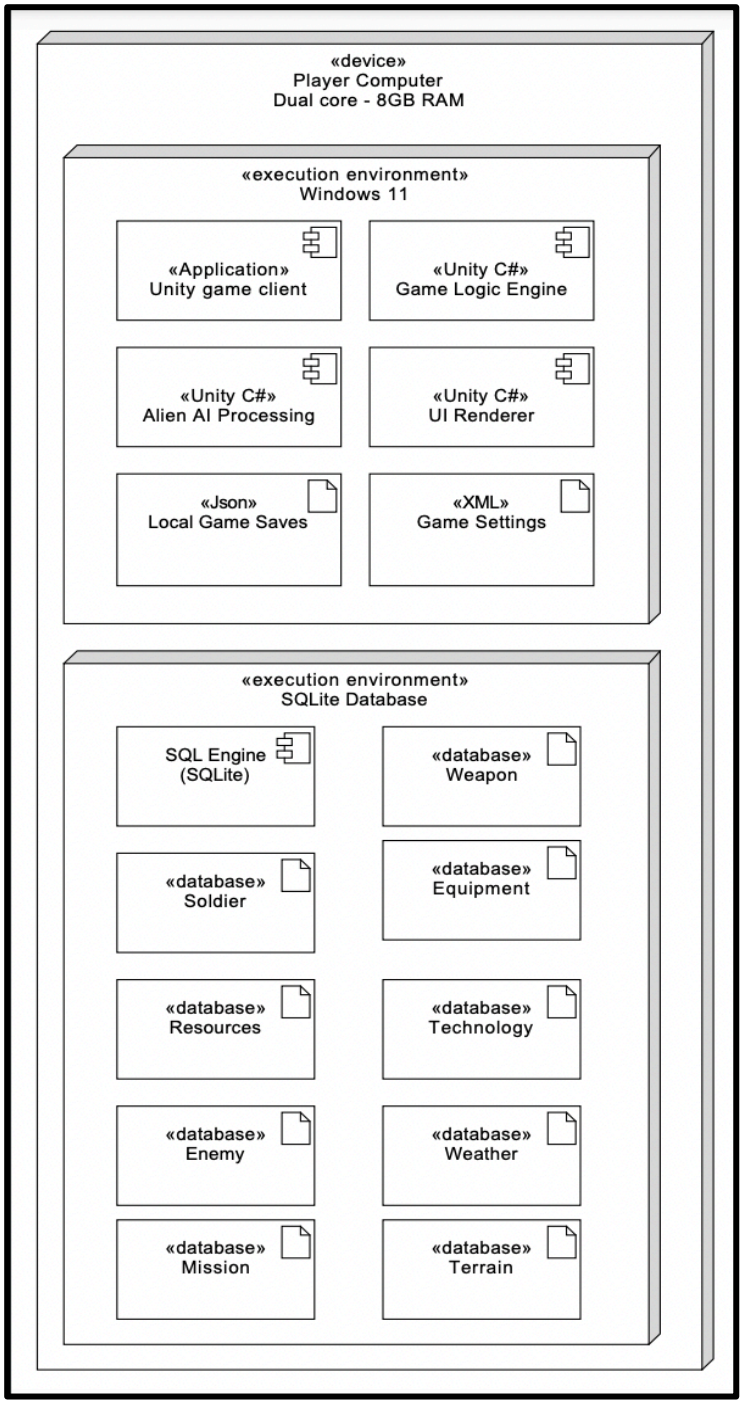
   b. Connected to the resources database

3. **Mission soldiers**

   a. Players can visualize and interact with soldiers chosen for the current mission

   b. Connected to the equipments database

2.3 Deployment Diagram



Deployment Diagram Explanation

1. Overall Architecture

This game is a single-player game, running on Windows 11, with all data stored locally. There is no Game Server.

SQLite is used as the local database to store core game data, such as soldiers, missions, weapons, technology, and resources.

JSON and XML are used for local file storage:

JSON is used for game saves.

XML is used for game settings (e.g., volume controls, key bindings).

All game logic, including combat, UI rendering, and AI processing, is handled entirely within the Unity Game Client.

2. Key Components

a) Windows 11 <<execution environment>>

Unity Game Client: The main executable of the game, containing all core logic.

Game Logic Engine (Unity C#): Handles combat mechanics, mission progression, and leveling systems.

Alien AI Processing (Unity C#): Controls enemy AI behavior.

UI Renderer (Unity C#): Manages user interface rendering.

Local File Storage:

JSON is used for saving and loading game progress.

XML is used for game settings configuration.

b) SQLite Database <<execution environment>>

SQL Engine (SQLite): Handles database queries.

Stores core game data:

Soldiers, missions, resources, weapons, and technology.

This Deployment Diagram ensures that all essential game components and data are managed efficiently on the player's local machine, allowing for full offline functionality while keeping structured and unstructured data properly separated.

3. Definitions

3.1 Branding

Look and Feel

The design of *Starlit Overwatch* is crafted to emphasize its tactical and strategic nature while maintaining a clean, focused aesthetic. To provide a less distracting background, the game primarily employs a mostly black backdrop across its screens. This choice not only enhances the sense of stealth and precision but also ensures that the player's attention remains on the critical elements of the interface.

At the beginning, a black camouflage background is subtly introduced, blending military-inspired visuals with a minimalist approach. This understated design avoids overwhelming the player, allowing for a more immersive and focused experience. As the game progresses, key sections are highlighted with distinct colors, each serving a specific purpose while maintaining a cohesive and simple theme.

The transition from dark, simple backgrounds to brighter visuals for mission screens is designed to make navigation easy and keep the experience engaging. By prioritizing clarity and functionality, the design of *Starlit Overwatch* creates an environment that draws players into the tactical gameplay while keeping distractions to a minimum.

Color Palette

Each screen is assigned a unique color theme while a black background provides cohesion and contrast:

- Black Background with Camouflage: Creates a sense of focus, stealth, and cohesion across most screens.
- Green: Featured in the world map to signify exploration, planning, and mission success.
- Orange: Dominates the menu screen to evoke warmth and focus for strategic decisions.
- Brown and Orange Combo: Found in the bases section to convey ruggedness and emphasize construction and management aspects.
- Red: Used sparingly for critical decision points, such as "Quit" or "Exit" buttons, ensuring immediate visibility.
- White: Provides clarity and ensures all text and icons remain legible.

Logo

The Starlit Overwatch logo is a bold visual representation of the game's tactical nature:

Elements:

1. **Weaponry and Armor Protection:** The soldier holding a rifle embodies readiness, defense, and tactical precision.

2. **Font:** The bold, clean typography reflects the game's modern and forward-thinking design, emphasizing clarity and strength.

3. **Overall Design:** The logo uses metallic gradients and a shield-like composition to convey strength, strategy, and resilience.

Justification

The branding choices effectively mirror the themes of Starlit Overwatch, ensuring that every design element enhances the player's experience and reinforces the game's core mechanics:

1. **Immersive and Tactical Experience**
   - The black background and camouflage patterns create an atmosphere of stealth and precision, reflecting the game's strategic nature.
   - Distinct colors for each screen help guide the player, making key areas like the world map, base management, and menus easily identifiable while maintaining a cohesive look.
2. **Clear and Functional Design**
   - The use of high-contrast text and icons ensures readability across different screens.
   - A clean and focused aesthetic keeps distractions minimal, allowing players to concentrate on strategy and decision-making.

3. **Strong Visual Identity**

    - The logo, with its representation of weaponry and armor protection, immediately communicates the game's tactical and military focus.
    - The bold typography reinforces clarity and strength, creating a modern and professional feel.

4. **Strategic Use of Color**

    - Green for the world map emphasizes exploration and mission success.
    - Orange and brown in base management screens create a sense of stability and construction.
    - Red is used sparingly to highlight critical actions such as "Quit" or "Exit," ensuring players quickly recognize important decisions.

Scenario 1:

For Scenario 1, I will be displaying the branding by showing mock-ups of the user interface flow from the **Welcome** screen (beginning) to the **New Game** selection. From there, the user will navigate to the **World Map**, which consists of four main sections: **Base, Staff, Mission, and Tech**. The player will then access **Base**, The player can choose between the bases created, selecting from **Base 1**, **Base 2**, **Base 3**, or **Base 4**. This will showcase the game's clear progression while keeping a consistent look and feel across all screens.

1

PLAY

2

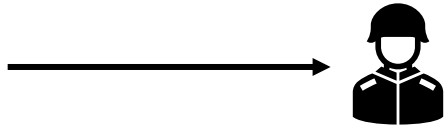
QUIT

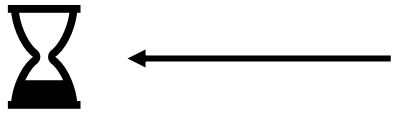
3

User selects “PLAY”

MENU

STARLIT OVERWATCH

NEW GAME  4

5  LOAD GAME

EXIT 6

User selects “NEW GAME”

WORLD MAP

STARLIT OVERWATCH

MISSIONS 7

BASE 8

STAFF 9

TECH 10

EXIT 11

User selects “BASES”

BASES

STARLIT OVERWATCH

12 BASE 1 13

New Base 14

Button Option 1 15

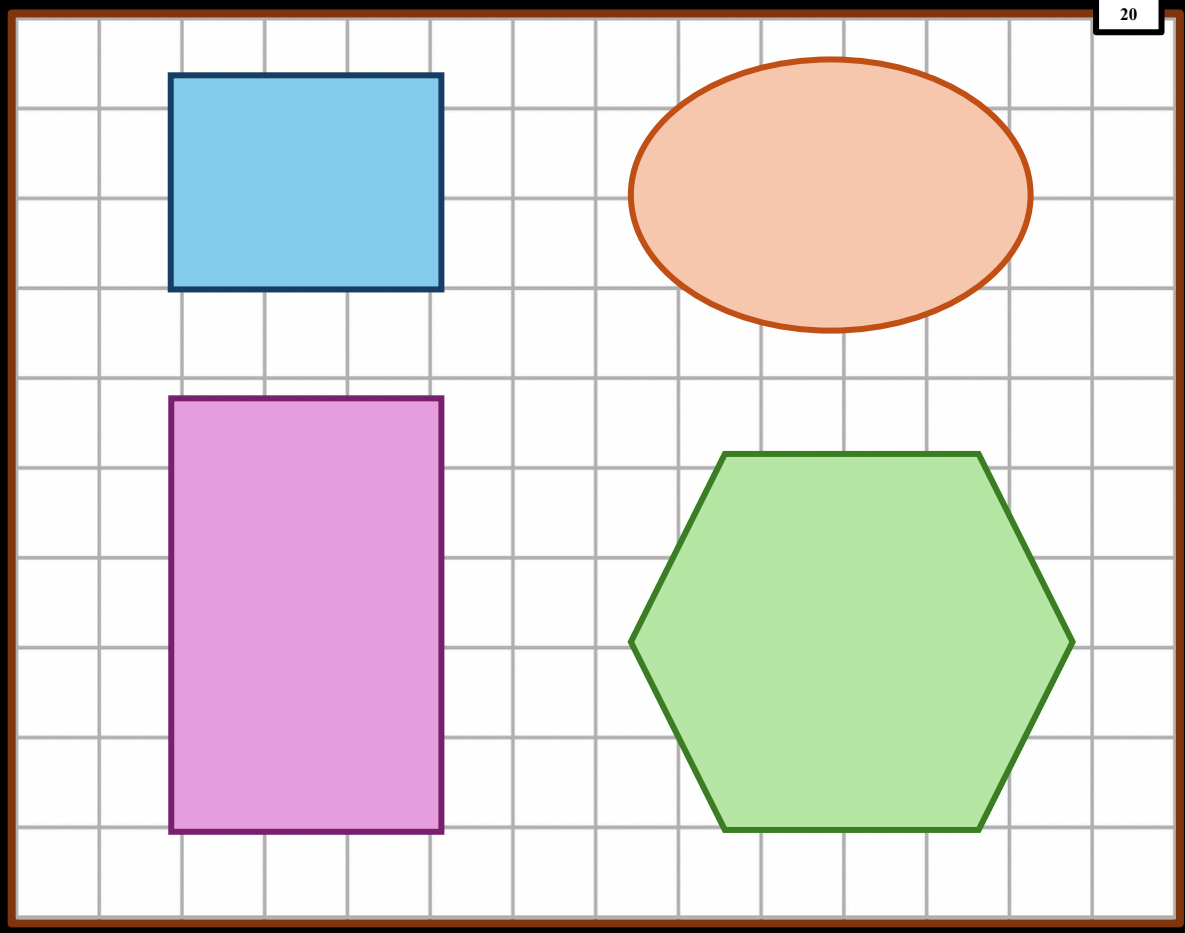
Button Option 2 16

Button Option 3 17

Button Option 4 18

EXIT 19

20



User selects Arrow Button

BASES

STARLIT OVERWATCH

BASE 2

New Base

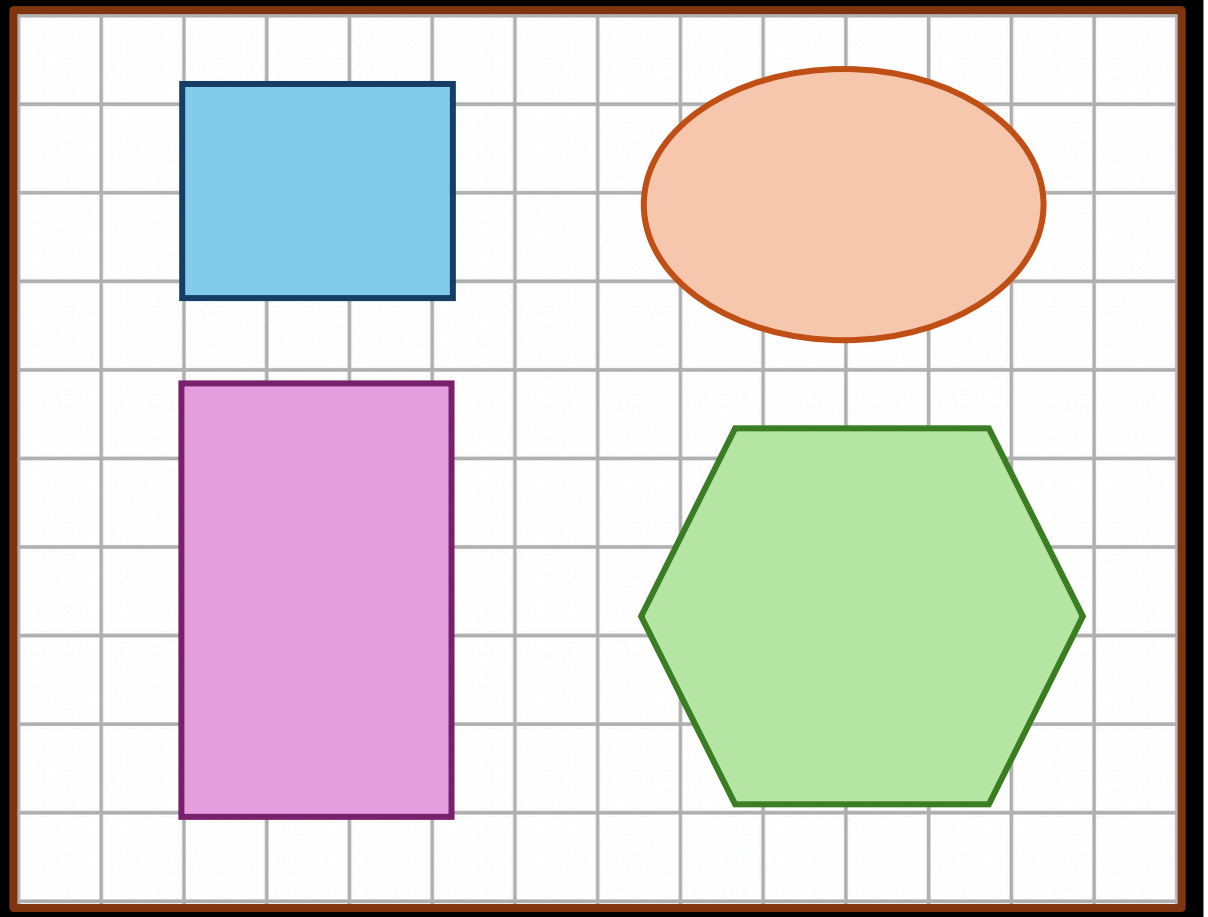
Button Option 1

Button Option 2

Button Option 3

Button Option 4

EXIT



Element Functionality - Branding and User Flow Mock-ups for Scenario 1			
Element Number	**Element**	**Description**	**Functionality**
1	Game Title	Displays the title of the game	Static display of the game title
2	Play Button	Starts a new game session	Confirms the user's decision to start the game session
3	Quit Button	Closes the game application	Confirms the user's decision to exit the application
4	New Game Button	Opens a new game session	Initiates a new game session for the user
5	Load Game Button	Loads a previously saved game	Loads a saved game session for the user
6	Exit Button (Menu)	Exits the menu screen	Exits the current menu screen
7	Missions Button	Accesses missions available to the player	Navigates to the missions available for the player
8	Base Button	Navigates to the base management screen	Redirects the user to the base management menu
9	Staff Button	Manages troop staff information	Provides access to manage troop staff details
10	Tech Button	Accesses technology upgrades	Allows the user to access technology upgrades
11	Exit Button (World Map)	Exits the world map screen	Exits the current world map screen
12	Base Navigation (Left)	Navigates to the previous base in the list	Navigates to the previous base in the list
13	Base Navigation (Right)	Navigates to the next base in the list	Navigates to the next base in the list
14	New Base Button	Creates a new base	Creates a new base for management
15	Button Option 1	Triggers option 1 functionality	Executes the first action option
16	Button Option 2	Triggers option 2 functionality	Executes the second action option
17	Button Option 3	Triggers option 3 functionality	Executes the third action option
18	Button Option 4	Triggers option 4 functionality	Executes the fourth action option
19	Exit Button (Base Menu)	Exits the base menu screen	Exits the base menu screen
20	Grid Shapes	Graphical representation of base components	Displays graphical components of the base

Scenario 2:

For Scenario 2, I will be displaying the branding by showing mock-ups of the user interface flow from the **Welcome** screen (beginning) directly to the **Quit** option. This path provides a simple exit option while ensuring that all user interactions align with the game's visual and thematic design.

1

PLAY

2

QUIT

3

User selects “QUIT”


SESSION
TERMINATED

Element Functionality - Branding and User Flow Mock-ups for Scenario 2			
Element Number	**Element**	**Description**	**Functionality**
1	Game Title	Displays the title of the game	Static display of the game title
2	Play Button	Starts a new game session	Confirms the user's decision to start the game session
3	Quit Button	Closes the game application	Confirms the user's decision to exit the application
4	Session Terminated Message	Indicates that the session has ended	Informs the user that the game session is terminated

3.2 Required Technologies

Hardware

- Seven Developer Workstations
    - Each workstation can run Windows, macOS, or Linux (as Unity and Visual Studio Code are cross-platform).
- Database Server (Local or Cloud)
    - A dedicated machine or cloud instance for hosting the MySQL database.
    - If a local server is used, it should be on a stable LAN for optimal performance.

Software

- **Unity Engine (with C#)**
    - Free (Personal Edition) version suitable for small teams, offering all necessary features for development.
    - C# is fully integrated into Unity for scripting game mechanics.
- **Visual Studio Code (or Equivalent IDE)**
    - Free, lightweight, and available on multiple operating systems.

- Easy to set up and configure for Unity and C# projects.

SQLite Database

- Lightweight, file-based database suitable for local applications and small projects.
- Free and open-source, requiring no server setup, making it ideal for development and game-related data storage.
- Easy to integrate with applications, as it stores data in a single .db file and supports standard SQL queries.
- Can be used locally on a personal machine or shared via file-based storage solutions.

3.3 Project and System Constraints

- **Budget Constraints**

  The team operates with no dedicated budget for software or additional hardware. Therefore, all chosen solutions (Unity, MySQL) must be free or open source to keep costs at zero.

- **Equipment Constraints**

  Each of the seven developers already has a workstation (Windows/macOS/Linux).

  No additional servers or specialized hardware can be purchased, so the existing local server or a low-cost cloud instance for MySQL must suffice.

- **Technology Constraints**

  The project must be developed using Unity with C#, as it meets the free/open-source requirement and is cross-platform.

  MySQL will be used for database needs, hosted either locally or on a cloud service that offers a free tier.

  Discord is the main communication platform for collaboration and coordination among team members.

3.4 Client Expectation

Must-Have Features

- Tactical Turn-Based Gameplay: Control squads of specialized soldiers (combat, medic, tech, or scout).
- Resource Management: Collect and use money, tech points, and raw materials for infrastructure, soldier development, and upgrades.
- Single-Player Campaign: Varied missions, weather effects, and a global map for strategic planning.
- Base Building & Staffing: Establish bases, hire/train staff, and manage technological improvements.

Expectations / Wants

- User-Friendly Interface: Clear menus, tooltips, and guidance.
- Immersive World Building: Engaging lore and environmental storytelling.
- Modular Upgrades: Flexible tech tree and soldier progression systems.
- Scalability: Capacity to add or refine features without major changes.

Assumptions (Success/Failure)

- Success means strong user feedback, solid game delivery, few bugs, and effective resource/staff management.

- Failure means unclear gameplay, frequent crashes, or missions that cannot be completed.

Project Scope - "Won't" and "Will"

- **Won't:**

  Include multiplayer modes (co-op or PvP).

  Require paid add-ons for core gameplay.

- **Will:**

  Deliver a single-player experience centered on turn-based combat, strategic resource use, and base operations.

  Offer enough content for a fully engaging campaign.

3.5 Statements on Impact, Ethics, and Performance

- **Impact:**

  Starlit Overwatch aims to enhance turn-based strategy by integrating robust resource management, a global strategic layer, and weather-influenced combat. This approach encourages thoughtful play and strengthens the genre's appeal to both veterans and newcomers.

- **Ethics:**

  The game will ensure fair gameplay by avoiding exploitative monetization or pay-to-win mechanics. Character roles and soldier development will be thoughtfully designed to maintain balance and representation, avoiding any insensitive content or stereotypes.

- **Performance:**

  The development team commits to optimizing the game for a stable frame rate on standard hardware.

  MySQL will be optimized to manage game data efficiently with minimal overhead.

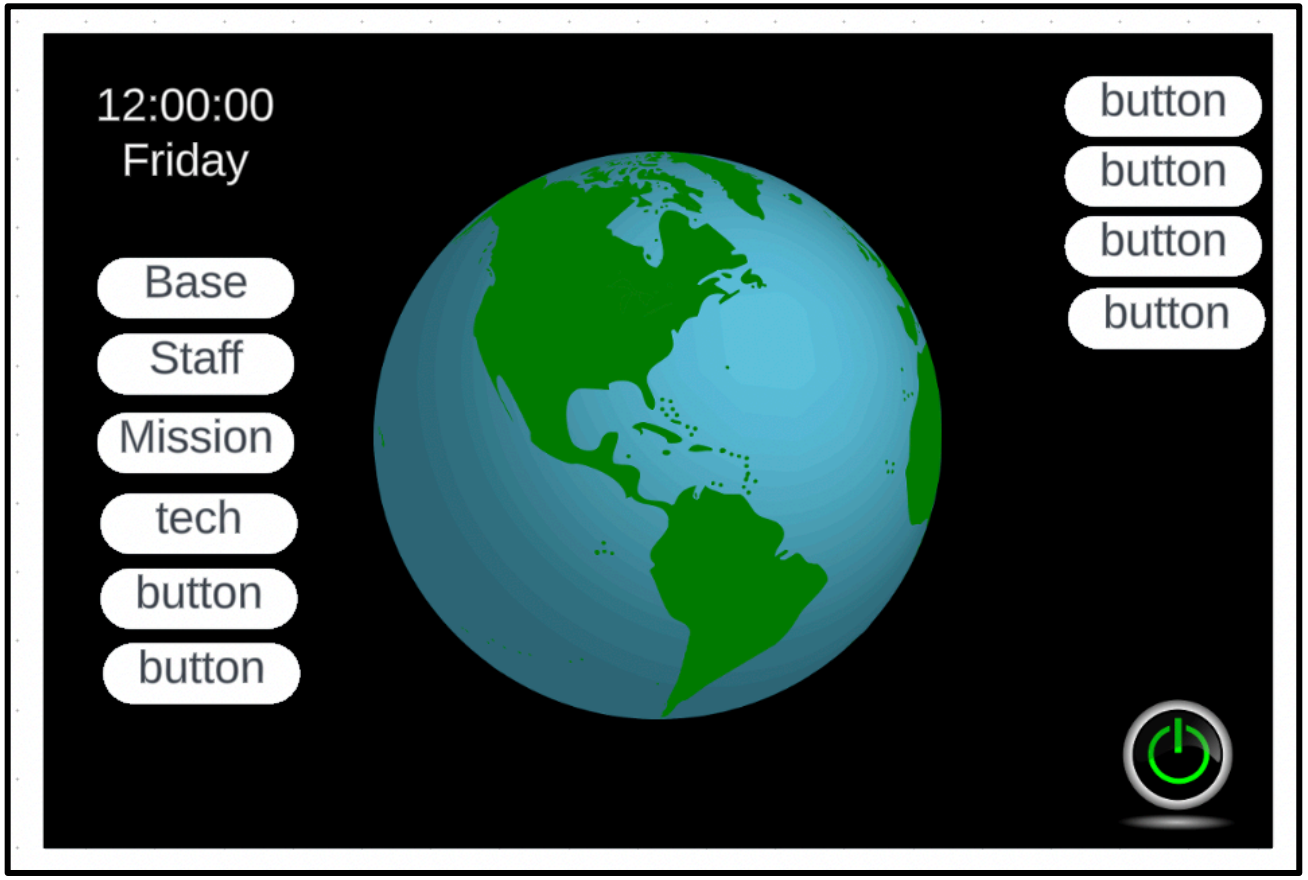
  Unity’s profiling tools will be used to reduce load times and ensure smooth transitions between missions.

4. Screen Layouts

4.1 Screen Mock-ups

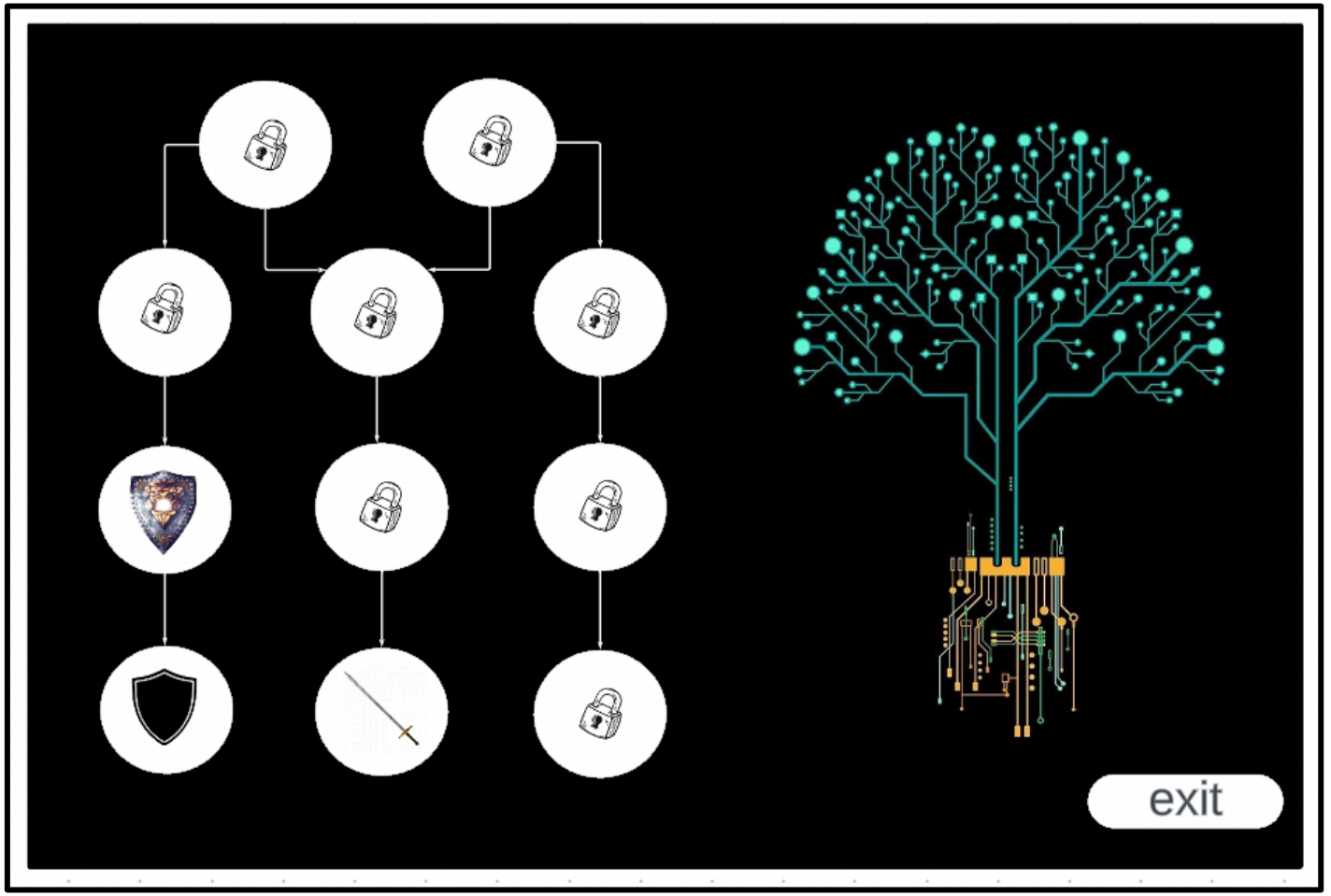
Mock-Up 1 - Main Menu Screen – World Map

- The main menu screen displays a globe in the center, surrounded by labeled buttons on the left and right for navigation and functionality, alongside a digital clock and day display in the top-left corner. A power icon is located at the bottom-right corner for potential exit or shutdown functionality.



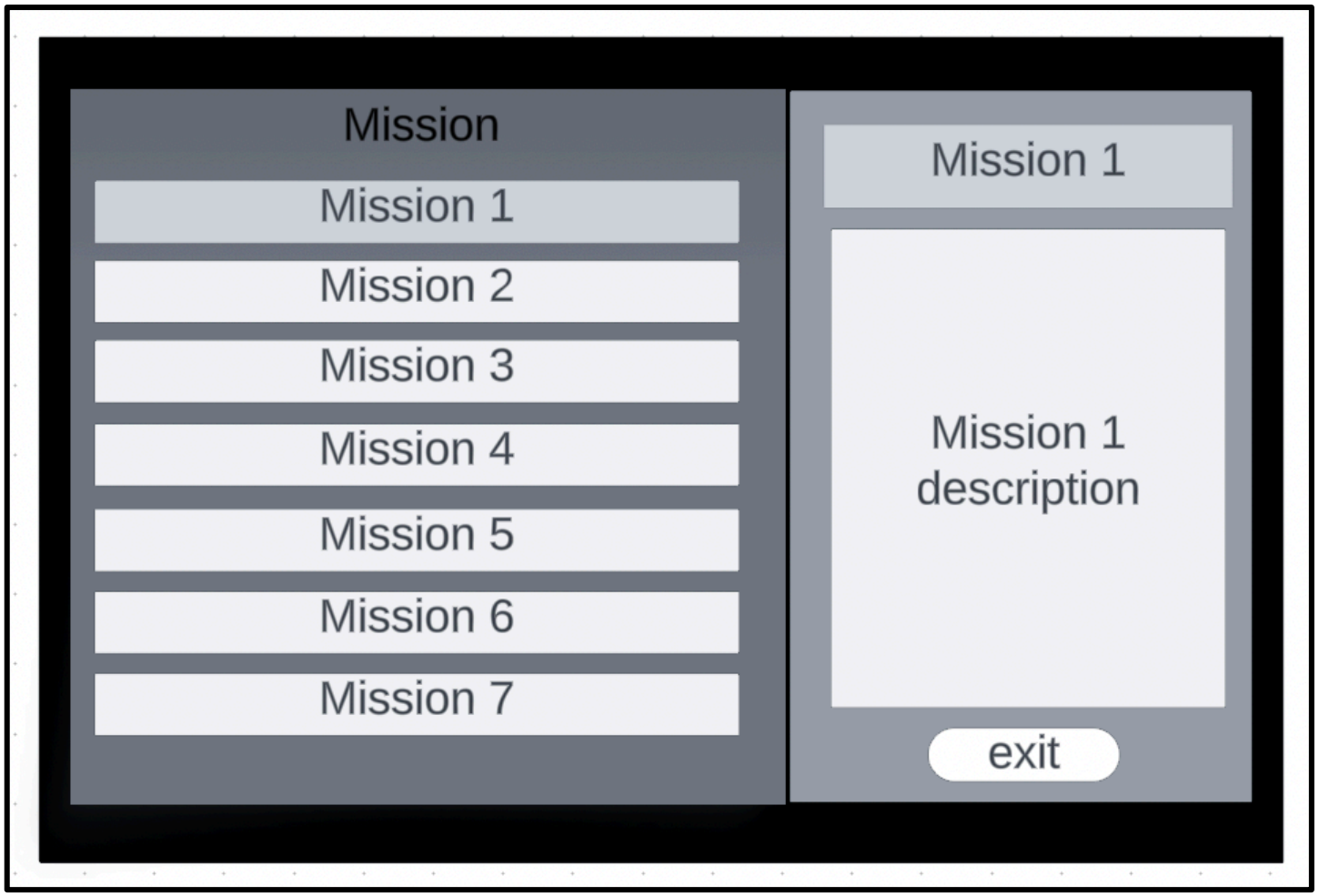
Mock-Up 2 - Tech Tree Screen

- The Tech page (Tech Tree) screen presents a branching diagram on the left, showcasing a progression of locked and unlocked items like shields and swords, representing advancements or upgrades. On the right, a visually striking circuit-tree graphic symbolizes technological growth, with an "exit" button at the bottom-right for navigation.



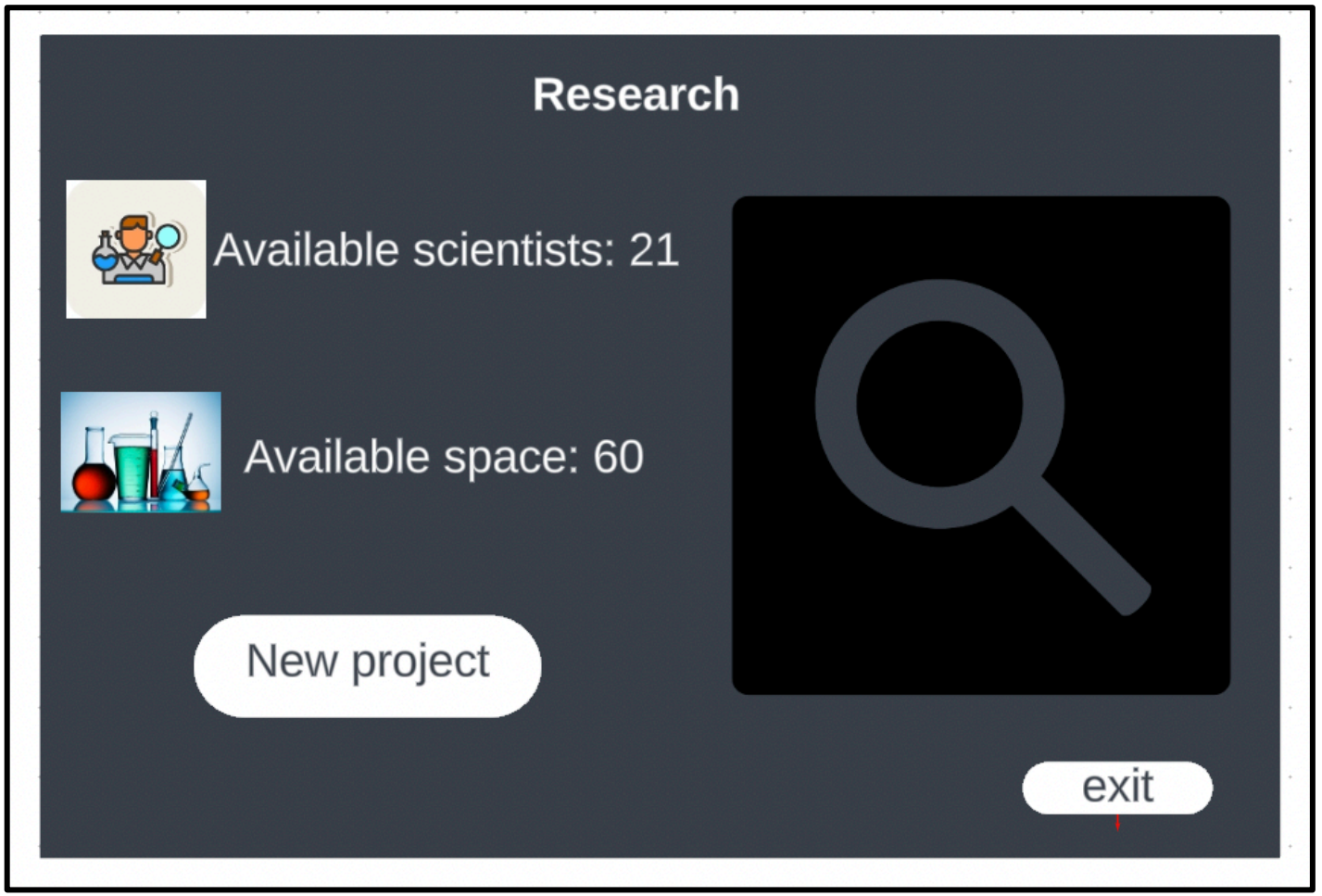
Mock-Up 3 - Mission Screen

- The Mission screen displays a list of missions on the left panel, allowing users to select individual missions for details. The right panel shows the title and description of the selected mission, with an "exit" button at the bottom-right for navigation.



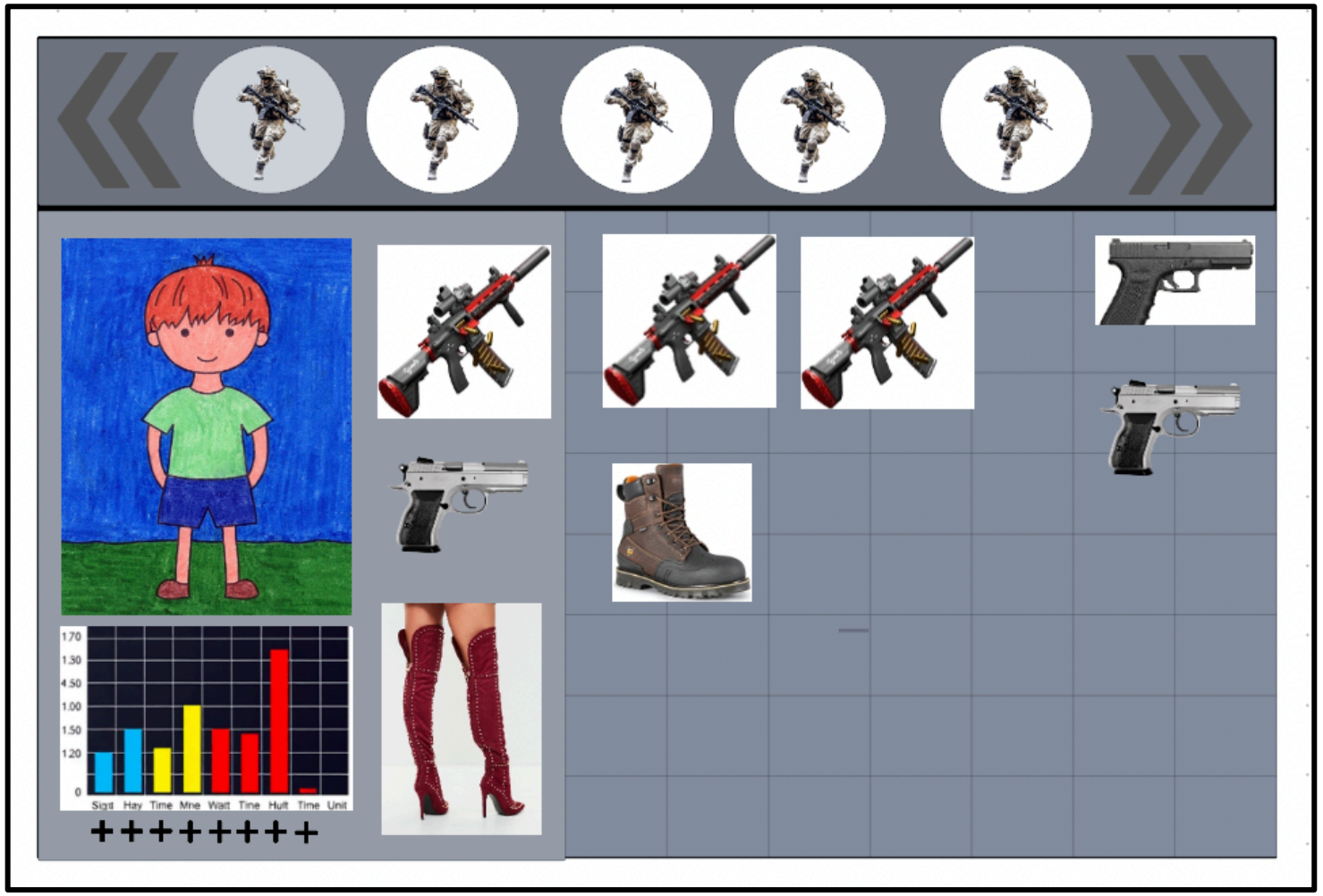
Mock-up 4 - Research Screen

- The Research screen provides an overview of current resources, displaying the number of "Available scientists" as 21 and "Available space" as 60 (both numbers are arbitrary placeholders). A "New project" button is included for initiating research tasks, and a magnifying glass icon on the right could represent search or detailed exploration, with an "exit" button at the bottom-right for leaving the screen.



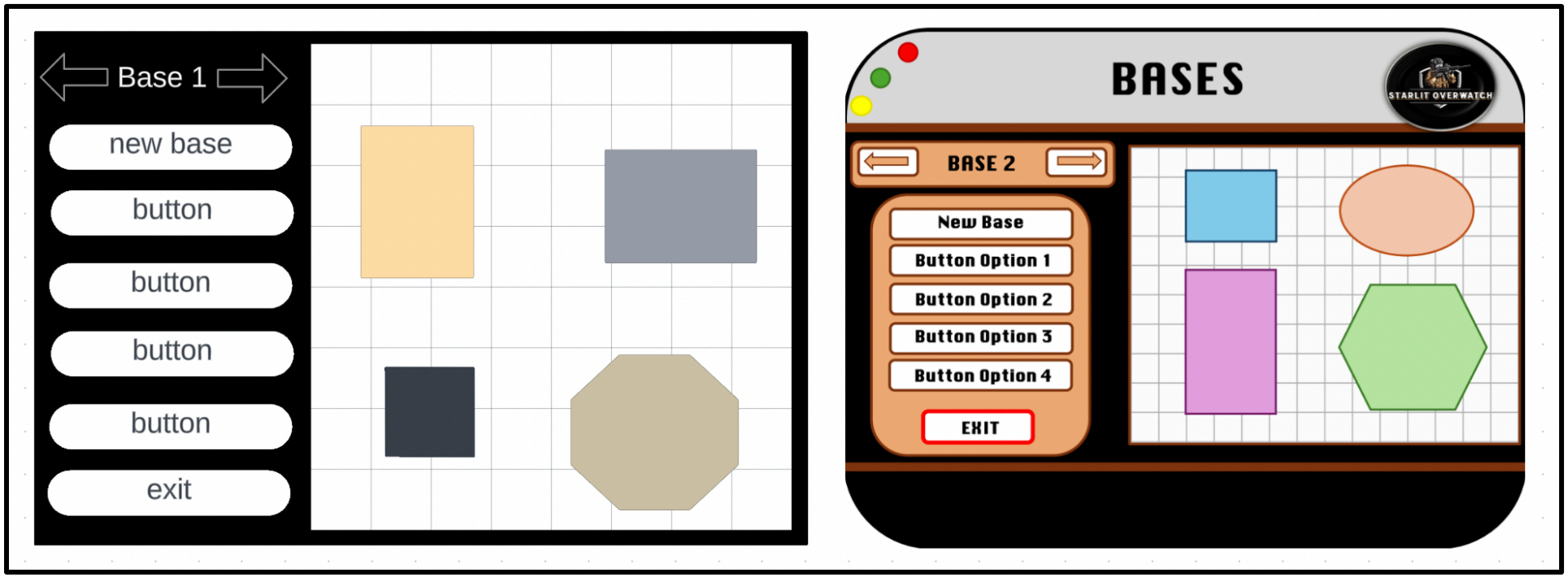
Mock up 5 - Inventory and Character Customization Screen

- The Inventory and Character Customization screen displays character options at the top and a grid below for managing items like weapons, boots, and stats. It allows users to customize and equip their character efficiently.



Mock-up 6 - Bases Management Screen

- The left image represents the layout of the base management screen, focusing on functionality and button arrangement. The right image shows the detailed design, including styled buttons, labeled sections, colorful geometric representations of bases, and a polished interface.
- The layout allows users to navigate between bases, create new ones, and manage their structures through the buttons on the left. The grid visually represents base components, making it easy to organize and customize layouts.



4.2 Specialty screens

Mock-up 1 - Welcome Screen

- This is the welcome screen for a game titled "Starlit Overwatch." It features options to start the game by pressing "Play" (green button) or exit the game using "Quit" (red button), set against a camo-styled background.

Mock-up 2 - Warning Screen

- This is a confirmation dialog screen asking the user if they are sure they want to exit the game, warning that any unsaved progress will be lost. It provides two options: "EXIT" to confirm exiting and "CANCEL" to stay in the game, both styled with red text on red buttons.

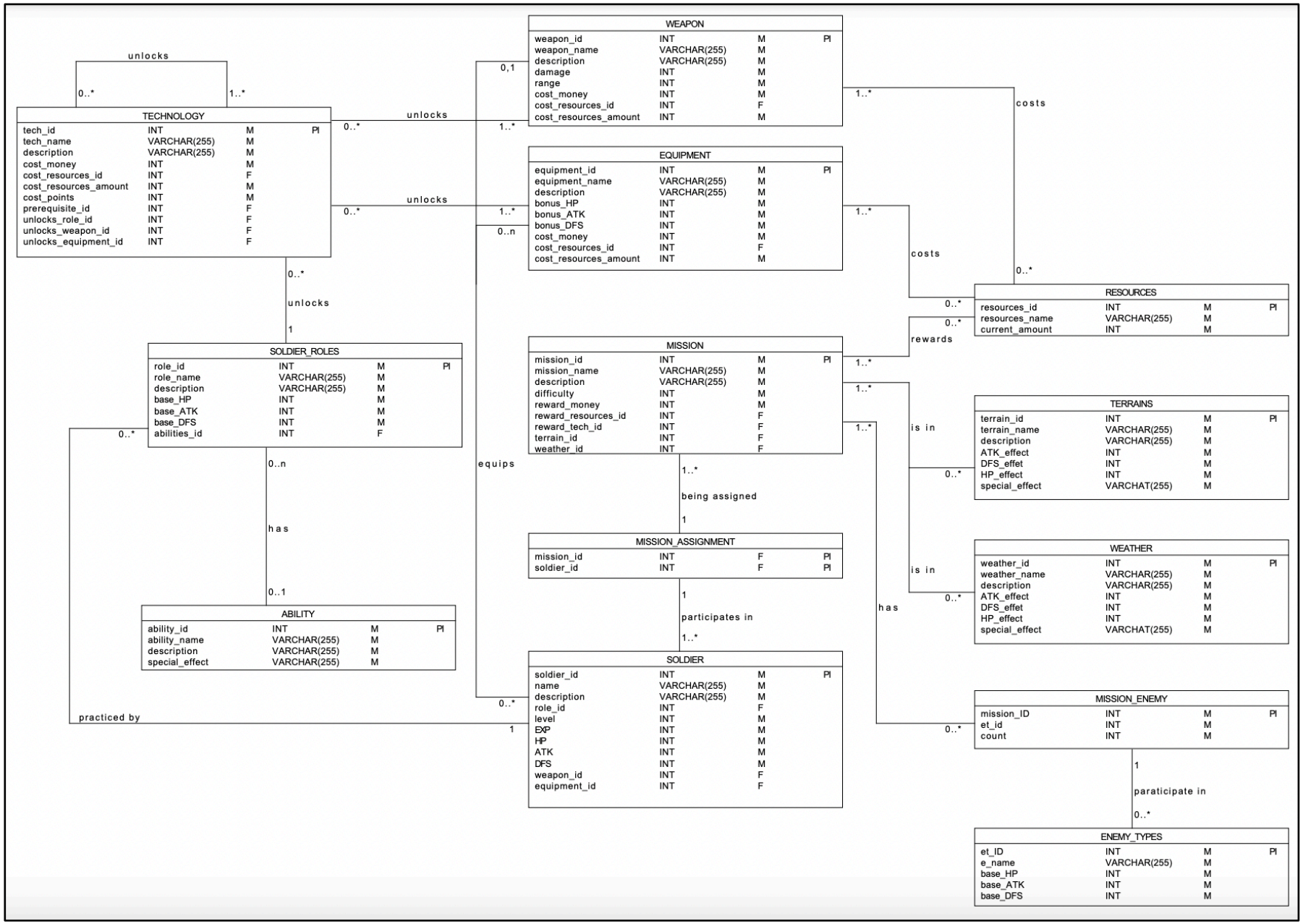
Mock-up 3 - Termination Screen

- This screen displays a message indicating that the session has ended, with the text "SESSION TERMINATED" prominently shown in bold green letters with a purple outline. The background features a camouflage-style design, emphasizing a military or tactical theme.

5. Schemas and Queries

5.1 Defining file structures

5.1.1 Database Diagram



5.1.2 Database Table Structures

TECHNOLOGY Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
tech_id	INT	(none)	Primary Key	NOT NULL, AUTO-INCREMENT
tech_name	VARCHAR	255	Member Key	NOT NULL
description	VARCHAR	255	Member Key	NOT NULL
cost_money	INT	(none)	Member Key	NOT NULL
cost_memory	INT	(none)	Member Key	NOT NULL
cost_resources_id	INT	(none)	Foreign Key	REFERENCES resources(resources_id), can be NULL
cost_resources_amount	INT	(none)	Member Key	NOT NULL
cost_points	INT	(none)	Member Key	NOT NULL
prerequisite_id	INT	(none)	Foreign Key	Self-reference: REFERENCES technology(tech_id), can be NULL
unlocks_role_id	INT	(none)	Foreign Key	REFERENCES soldier_roles(role_id), can be NULL
unlocks_weapon_id	INT	(none)	Foreign Key	REFERENCES weapon(weapon_id), can be NULL
unlocks_equipment_id	INT	(none)	Foreign Key	REFERENCES equipment(equipment_id), can be NULL

SOLDIER ROLES Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
role_id	INT	(none)	Primary Key	NOT NULL, AUTO-INCREMENT
role_name	VARCHAR	255	Member Key	NOT NULL
description	VARCHAR	255	Member Key	NOT NULL
base_HP	INT	(none)	Member Key	NOT NULL
base_ATK	INT	(none)	Member Key	NOT NULL
base_DPS	INT	(none)	Member Key	NOT NULL
abilities_id	INT	(none)	Foreign Key	REFERENCES ability(ability_id); design may vary (could be a link

ABILITY Database table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
ability_id	INT	(none)	Primary Key	NOT NULL, AUTO-INCREMENT
ability_name	VARCHAR	255	Member Key	NOT NULL
description	VARCHAR	255	Member Key	NOT NULL
special_effect	VARCHAR	255	Member Key	NOT NULL

SOLDIER Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
soldier_id	INT	(none)	Primary Key	NOT NULL, AUTO-INCREMENT
name	VARCHAR	255	Member Key	NOT NULL
description	VARCHAR	255	Member Key	NOT NULL
role_id	INT	(none)	Foreign Key	NOT NULL, REFERENCES soldier_roles(role_id)
level	INT	(none)	Member Key	NOT NULL
XP	INT	(none)	Member Key	NOT NULL
HP	INT	(none)	Member Key	NOT NULL
ATK	INT	(none)	Member Key	NOT NULL
DPS	INT	(none)	Member Key	NOT NULL
weapon_id	INT	(none)	Foreign Key	REFERENCES weapon(weapon_id), can be NULL
equipment_id	INT	(none)	Foreign Key	REFERENCES equipment(equipment_id), can be NULL
building_id	INT	(none)	Foreign Key	REFERENCES building(building_id), can be NULL

MISSION Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
mission_id	INT	(none)	Primary Key	NOT NULL, AUTO-INCREMENT
mission_name	VARCHAR	255	Member Key	NOT NULL
description	VARCHAR	255	Member Key	NOT NULL
difficulty	INT	(none)	Member Key	NOT NULL
reward_money	INT	(none)	Member Key	NOT NULL
reward_resources_id	INT	(none)	Foreign Key	REFERENCES resources(resources_id), can be NULL
reward_tech_id	INT	(none)	Foreign Key	REFERENCES technology(tech_id), can be NULL
terrain_id	INT	(none)	Foreign Key	REFERENCES terrains(terrain_id), can be NULL
weather_id	INT	(none)	Foreign Key	REFERENCES weather(weather_id), can be NULL

MISSION ASSIGNMENT Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
mission_id	INT	(none)	Primary Key	NOT NULL, REFERENCES mission(mission_id)
soldier_id	INT	(none)	Primary Key	NOT NULL, REFERENCES soldier(soldier_id)

MISSION ENEMY Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
mission_id	INT	(none)	Primary Key	NOT NULL, REFERENCES mission(mission_id)
et_id	INT	(none)	Primary Key	NOT NULL, REFERENCES enemy_types(et_id)
count	INT	(none)	No	NOT NULL

ENEMY TYPES Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
et_id	INT	(none)	Primary Key	NOT NULL, AUTO_INCREMENT
e_name	VARCHAR	255	Member Key	NOT NULL
description	VARCHAR	255	Member Key	NOT NULL
base_HP	INT	(none)	Member Key	NOT NULL
base_ATK	INT	(none)	Member Key	NOT NULL
base_DPS	INT	(none)	Member Key	NOT NULL

RESOURCES Database table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
resources_id	INT	(none)	Primary Key	NOT NULL, AUTO_INCREMENT
resources_name	VARCHAR	255	Member Key	NOT NULL
current_amount	INT	(none)	Member Key	NOT NULL

WEAPON Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
weapon_id	INT	(none)	Primary Key	NOT NULL, AUTO_INCREMENT
weapon_name	VARCHAR	255	Member Key	NOT NULL
description	VARCHAR	255	Member Key	NOT NULL
damage	INT	(none)	Member Key	NOT NULL
range	INT	(none)	Member Key	NOT NULL
cost_money	INT	(none)	Member Key	NOT NULL
cost_resources_id	INT	(none)	Foreign Key	REFERENCES resources(resources_id), can be NULL
cost_resources_amount	INT	(none)	Member Key	NOT NULL

EQUIPMENT Database table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
equipment_id	INT	(none)	Primary Key	NOT NULL, AUTO_INCREMENT
equipment_name	VARCHAR	255	Member Key	NOT NULL
description	VARCHAR	255	Member Key	NOT NULL
bonus_HP	INT	(none)	Member Key	NOT NULL
bonus_ATK	INT	(none)	Member Key	NOT NULL
bonus_DPS	INT	(none)	Member Key	NOT NULL
cost_money	INT	(none)	Member Key	NOT NULL
cost_resources_id	INT	(none)	Foreign Key	REFERENCES resources(resources_id), can be NULL
cost_resources_amount	INT	(none)	Member Key	NOT NULL

TERRAIN Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
terrain_id	INT		Yes	NOT NULL, AUTO_INCREMENT
terrain_name	VARCHAR	255	No	NOT NULL
description	VARCHAR	255	No	NOT NULL
ATK_effect	INT		No	NOT NULL
DPS_effect	INT		No	NOT NULL
HP_effect	INT		No	NOT NULL
special_effect	VARCHAR	255	No	NOT NULL

WEATHER Database Table				
Field Name	**Data Type**	**Length**	**Key/Primary/Sort**	**Constraints**
weather_id	INT		Yes	NOT NULL, AUTO_INCREMENT
weather_name	VARCHAR	255	No	NOT NULL
description	VARCHAR	255	No	NOT NULL
ATK_effect	INT		No	NOT NULL
DPS_effect	INT		No	NOT NULL
HP_effect	INT		No	NOT NULL
special_effect	VARCHAR	255	No	NOT NULL

5.1.3 Formatted text files

---

1. TECHNOLOGY

- **Primary Key:** tech_id
- ***Attributes:***
  - tech_name: Name of the technology.
  - description: Description of the technology.
  - cost_money: Monetary cost to unlock.
  - cost_resources_id: Resources required (linked to RESOURCES).
  - cost_resources_amount: Amount of resources required.
  - cost_points: Points needed to unlock.
  - prerequisite_id: References another technology as a prerequisite.
  - unlocks_role_id, unlocks_weapon_id, unlocks_equipment_id: Indicates what this technology unlocks.

- ***Relationships:***

  ⇒ prerequisite_id is a self-referential foreign key. This implements the tech-tree structure.

  ⇒ cost_resources_id links to RESOURCES. It is a many-to-many relationship.

  ⇒ Unlocks_weapon_id, unlocks_role_id, unlocks_equipment_id and unlocks_building_id link to WEAPON, SOLDIER_ROLES, EQUIPMENT and INFRASTRUCTURE through a many-to-many relationship.

---

2. SOLDIER

- **Primary Key:** soldier_id
- ***Attributes:***
  - name: Name of the soldier.
  - faction: Soldier's faction.
  - level, XP, HP, ATK, DPS: Soldier's Stats.
  - weapon_id, equipment_id: References to assigned weapon and equipment.
  - Role_id: References to the soldier's practiced role.

- ***Relationships:***

  ⇒ weapon_id links to WEAPON. It's a many-to-one relationship.

  ⇒ equipment_id links to EQUIPMENT. It's a many-to-many relationship.

  ⇒ Building_id links to INFRASTRUCTURE. It's a many-to-many relationship

  ⇒ Role_id links to EQUIPMENT. It's a many-to-one relationship.

3. SOLDIER ROLES

- **Primary Key:** role_id
- ***Attributes:***
  - role_name: Name of the role.
  - description: Description of the role.
  - base_HP, base_ATK, base_DPS: Base stats for this role.
  - abilities_id: References abilities assigned to this role.

- ***Relationships:***

  ⇒ abilities_id links to ABILITY. It's a one-to-many relationship.

---

4. ABILITY

- **Primary Key:** ability_id
- **Attributes:**
  - ability_name: Name of the ability.
  - description: Description of the ability.
  - special_effect: Any special effect.

---

5. WEAPON

- **Primary Key:** weapon_id
- ***Attributes:***
  - weapon_name: Name of the weapon.
  - description: Weapon details.
  - damage, cost_money, cost_resources_id, cost_resources_amount: Cost details.
- ***Relationships:***

  ⇒ cost_resources_id links to RESOURCES. It's a many-to-many relationship.

---

6. EQUIPMENT

- **Primary Key:** equipment_id
- ***Attributes:***
  - equipment_name: Name of the equipment.
  - description: Equipment details.
  - bonus_ATK, bonus_DPS: Additional stats.
  - cost_money, cost_resources_id, cost_resources_amount: Cost details.

- ***Relationships:***

  ⇒ cost_resources_id links to RESOURCES. It's a many-to-many relationship.

---

7. INFRASTRUCTURE

- **Primary Key:** building_id
- **Attributes:**
  - building_name: Name of the infrastructure.
  - description: Infrastructure details.
  - level: The level of the infrastructure
  - capacity: The capacity of the infrastructure
  - cost_money, cost_resources_id, cost_resources_amount: Cost details.
- ***Relationships:***
  - cost_resources_id links to RESOURCES. It's a many-to-many relationship.

---

8. RESOURCES

- **Primary Key:** resources_id
- ***Attributes:***
  - resources_name: Name of the resource.
  - current_amount: Available quantity.

---

9. MISSION

- **Primary Key:** mission_id
- ***Attributes:***
  - mission_name: Name of the mission.
  - description: Mission details.
  - difficulty: Difficulty level.
  - reward_money, reward_resources_id, reward_amount: Rewards.
  - terrain_id, weather_id: Environmental factors.

- ***Relationships:***

  ⇒ reward_resources_id links to RESOURCES. It's a many-to-many relationship.

⇒ terrain_id links to TERRAIN. It's a many-to-many relationship.

⇒ weather_id links to WEATHER. It's a many-to-many relationship.

---

10. MISSION ASSIGNMENT

- **Primary Keys:** mission_id, soldier_id
- ***Purpose:***
  - Tracks which soldiers are assigned to which missions.

- ***Relationships:***

⇒ mission_id links to MISSION. It's a one-to-many relationship.

⇒ soldier_id links to SOLDIER. It's a one-to-many relationship.

---

11. MISSION ENEMY

- **Primary Keys:** mission_id, et_id
- ***Attributes:***
  - count: Number of enemies in the mission.

- ***Relationships:***
  - et_id links to ENEMY_TYPES. It's a one-to-many relationship.

---

12. ENEMY TYPES

- **Primary Key:** et_ID
- **Attributes:**
  - et_name: Enemy name.
  - HP, base_ATK, base_DPS: Enemy stats.

---

13. TERRAIN

- **Primary Key:** terrain_id
- **Attributes:**
  - terrain_name: Terrain type.
  - ATK_effect, DPS_effect, HP_effect: Modifiers affecting combat.

---

14. WEATHER

- **Primary Key:** weather_id
- **Attributes:**
  - weather_name: Type of weather.
  - ATK_effect, DPS_effect, HP_effect: Modifiers affecting combat.

Key Relationships in the Database

1. Missions are influenced by terrain and weather.
2. Soldiers have roles, weapons, and equipment.
3. Abilities are linked to soldier roles.
4. Technology unlocks new technologies, weapons, equipment, and roles.
5. Missions reward technology, resources and money.
6. Mission enemies are categorized into types.
7. Special effects in abilities, terrains, and weather can be expressed in a JSON format text file.

5.1.4 Defined Operations

Queries Examples.

Here are **10 examples SQL queries**:

---

1. Retrieve all soldiers with their assigned weapons and equipment

```
SELECT s.soldier_id, s.name, s.level, s.XP, w.weapon_name, e.equipment_name
FROM SOLDIER s
LEFT JOIN WEAPON w ON s.weapon_id = w.weapon_id
LEFT JOIN EQUIPMENT e ON s.equipment_id = e.equipment_id;
```

---

2. Get all missions with their difficulty level and rewards

```
SELECT mission_name, difficulty, reward_money, resources_name, reward_amount
FROM MISSION m
LEFT JOIN RESOURCES r ON m.reward_resources_id = r.resources_id;
```

---

3. List all weapons that cost more than 500 money

```
SELECT weapon_name, cost_money
FROM WEAPON
WHERE cost_money > 500;
```

---

4. Find soldiers assigned to a specific mission (e.g., mission_id = 2)

```
SELECT s.soldier_id, s.name, m.mission_name
FROM MISSION_ASSIGNMENT ma
JOIN SOLDIER s ON ma.soldier_id = s.soldier_id
JOIN MISSION m ON ma.mission_id = m.mission_id
WHERE ma.mission_id = 2;
```

---

5. Retrieve abilities and the soldier roles that have them

```
SELECT sr.role_name, a.ability_name, a.description
FROM SOLDIER_ROLES sr
JOIN ABILITY a ON sr.abilities_id = a.ability_id;
```

---

6. Count the number of soldiers in each role

```
SELECT sr.role_name, COUNT(s.soldier_id) AS total_soldiers
FROM SOLDIER s
JOIN SOLDIER_ROLES sr ON s.role_id = sr.role_id
GROUP BY sr.role_name;
```

---

7. Show all missions with their respective terrain and weather effects

```
SELECT m.mission_name, t.terrain_name, t.ATK_effect, w.weather_name, w.HP_effect
FROM MISSION m
LEFT JOIN TERRAINS t ON m.terrain_id = t.terrain_id
LEFT JOIN WEATHER w ON m.weather_id = w.weather_id;
```

---

8. Find all technologies that unlock new weapons

```
SELECT t.tech_name, w.weapon_name
FROM TECHNOLOGY t
JOIN WEAPON w ON t.unlocks_weapon_id = w.weapon_id;
```

---

9. Retrieve all enemies appearing in a specific mission (e.g., mission_id = 5)

```
SELECT m.mission_name, et.et_name, me.count
FROM MISSION_ENEMY me
JOIN ENEMY_TYPES et ON me.et_id = et.et_ID
JOIN MISSION m ON me.mission_id = m.mission_id
WHERE me.mission_id = 5;
```

---

10. Get the total number of available resources

```
SELECT resources_name, SUM(current_amount) AS total_amount
FROM RESOURCES
GROUP BY resources_name;
```

These queries cover different parts of the schema, including **soldiers, missions, equipment, technologies, and resources**.

5.2 Remote API definitions

API Examples.

Here are **three examples of remote API definitions** based on the database schema. These APIs use **JSON format** for data exchange.

---

1. API to Retrieve Soldier Information

Endpoint: GET /api/soldiers/{soldier_id}
Description: Fetch details of a specific soldier, including their role, weapon, and equipment.

Request Example:

GET /api/soldiers/12

Response Example:


```
{
 "soldier_id": 12,
 "name": "John Doe",
 "level": 5,
 "XP": 1200,
 "HP": 250,
 "ATK": 75,
 "DPS": 30,
 "role": {
  "role_id": 3,
  "role_name": "Sniper",
  "base_HP": 200,
  "base_ATK": 85
 },
 "weapon": {
```

```
    "weapon_id": 7,
    "weapon_name": "Long-range Rifle",
    "damage": 95
    "range": 5
  },
  "equipment": {
    "equipment_id": 2,
    "equipment_name": "Kevlar Armor",
    "bonus_HP": 50
  }
}
```


---

2. API to Assign a Soldier to a Mission

Endpoint: POST /api/missions/assign
Description: Assign a soldier to a mission.

Request Example:


```
{
  "mission_id": 8,
  "soldier_id": 12
}
```


Response Example:


```
{
  "message": "Soldier successfully assigned to mission.",
  "mission_id": 8,
  "soldier_id": 12
}
```


---

3. API to Retrieve Mission Details

Endpoint: GET /api/missions/{mission_id}
Description: Fetch details of a specific mission, including terrain, weather, and enemies.

Request Example: GET /api/missions/5

Response Example:


```
{
 "mission_id": 5,
 "mission_name": "Operation Desert Storm",
 "difficulty": "Hard",
 "reward_money": 1500,
 "reward_resources": {
  "resources_id": 3,
  "resources_name": "Steel",
  "reward_amount": 50
 },
 "terrain": {
  "terrain_id": 2,
  "terrain_name": "Desert",
  "ATK_effect": -10,
  "DPS_effect": -5
 },
 "weather": {
  "weather_id": 1,
  "weather_name": "Sandstorm",
  "HP_effect": -15
 },
 "enemies": [
  {
   "et_ID": 7,
   "et_name": "Rebel Sniper",
```

```
    "count": 3
   },
   {
    "et_ID": 4,
    "et_name": "Tank",
    "count": 1
   }
  ]
}
```

6. Important Sections

6.1 Table of Contents of Critical Sections

1. Missions

- **Critical Section Name:** Tactical Mission Timer for Auto-Failure
- **Subsection:** missions-critical-section-1
- **Purpose:** Introduces a turn-based timer for specific tactical missions. The timer adds
- urgency by enforcing mission failure if the player does not complete objectives within the set limit.

2. Staff

- **Critical Section Name:** Soldier Recovery
- **Subsection:** staff-critical-section-1
- **Purpose:** Tracks soldier downtime after missions to ensure they cannot be overused or assigned before fully recovering, maintaining game balance and realism.

6.2 Critical Sections

1. **Critical Section:** Tactical Mission Timer for Auto-Failure

   - **Location in Program**: **Missions → Tactical Gameplay**
   - **Description of Its Importance & Purpose:**
     - This timer adds urgency and tension to specific tactical missions, encouraging strategic planning and decision-making under pressure.
     - If the timer expires before the objective is completed, the mission automatically fails, resulting in penalties such as resource loss, morale reduction, or soldier casualties.
     - The timer prevents prolonged or exploitative playstyles where players delay indefinitely during tactical missions.

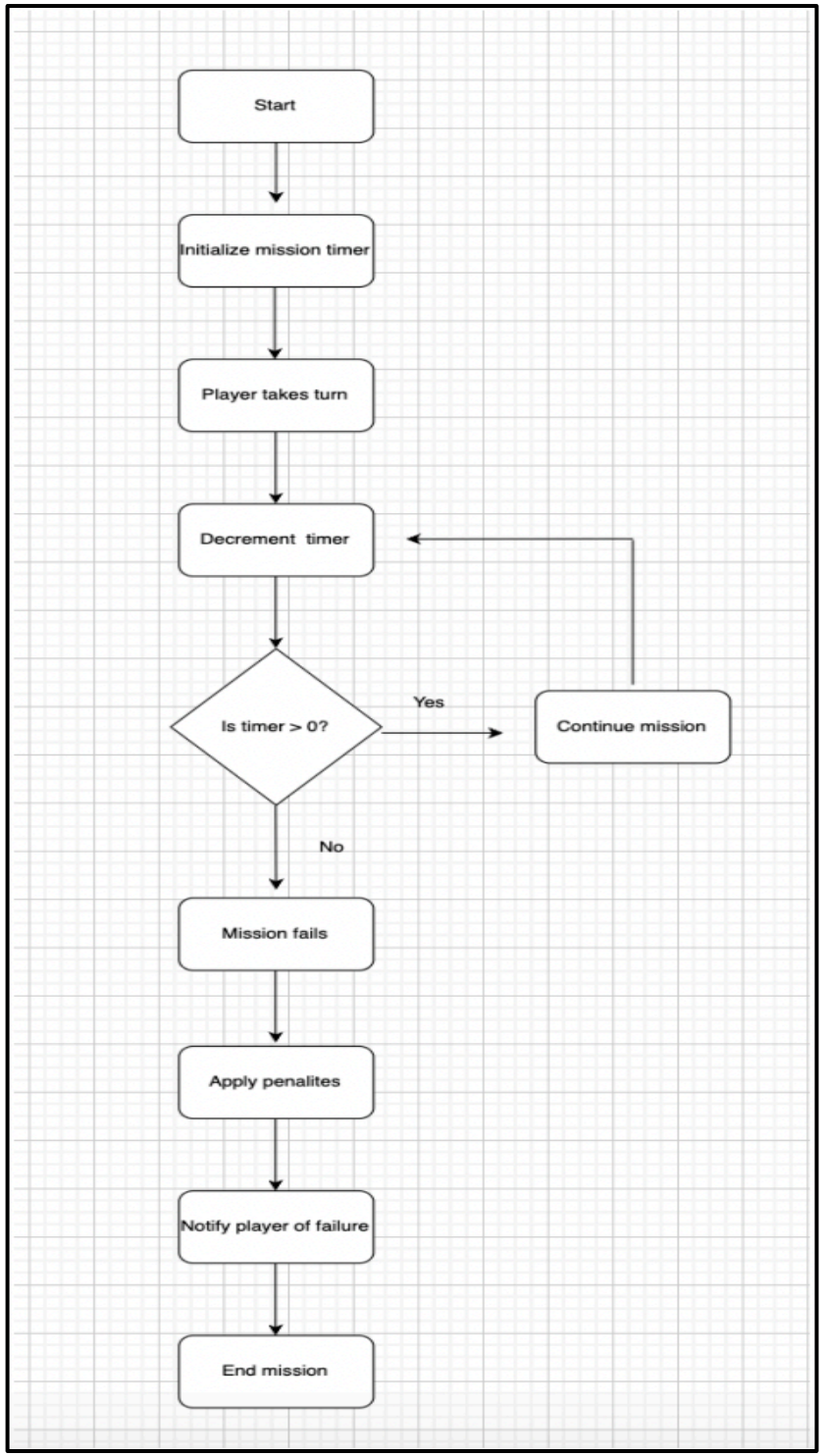
   - **Proposed Solution in Words:**
     1. **Timer Start**: Begin the countdown timer when the tactical mission starts.
     2. **Turn-Based Countdown**: Decrement the timer at the end of each player turn.
     3. **Check Timer at End of Turn:**
        a. If the timer reaches zero, trigger mission failure logic.
        b. If the objective is completed, stop the timer and mark the mission as a success.
     4. **Failure Logic:**

a. Apply penalties to resources, morale, or soldiers (e.g., soldiers are “captured” or lost if the timer expires).

b. End the mission and notify the player.

5. **UI Integration**: Display the timer prominently in the mission interface to keep players aware of the remaining time.

Flowchart 1


Start
Initialize mission timer
Player takes turn
Decrement timer
Is timer > 0?
Yes
Continue mission
No
Mission fails
Apply penalites
Notify player of failure
End mission

- **Explanation of Flowchart 1:**

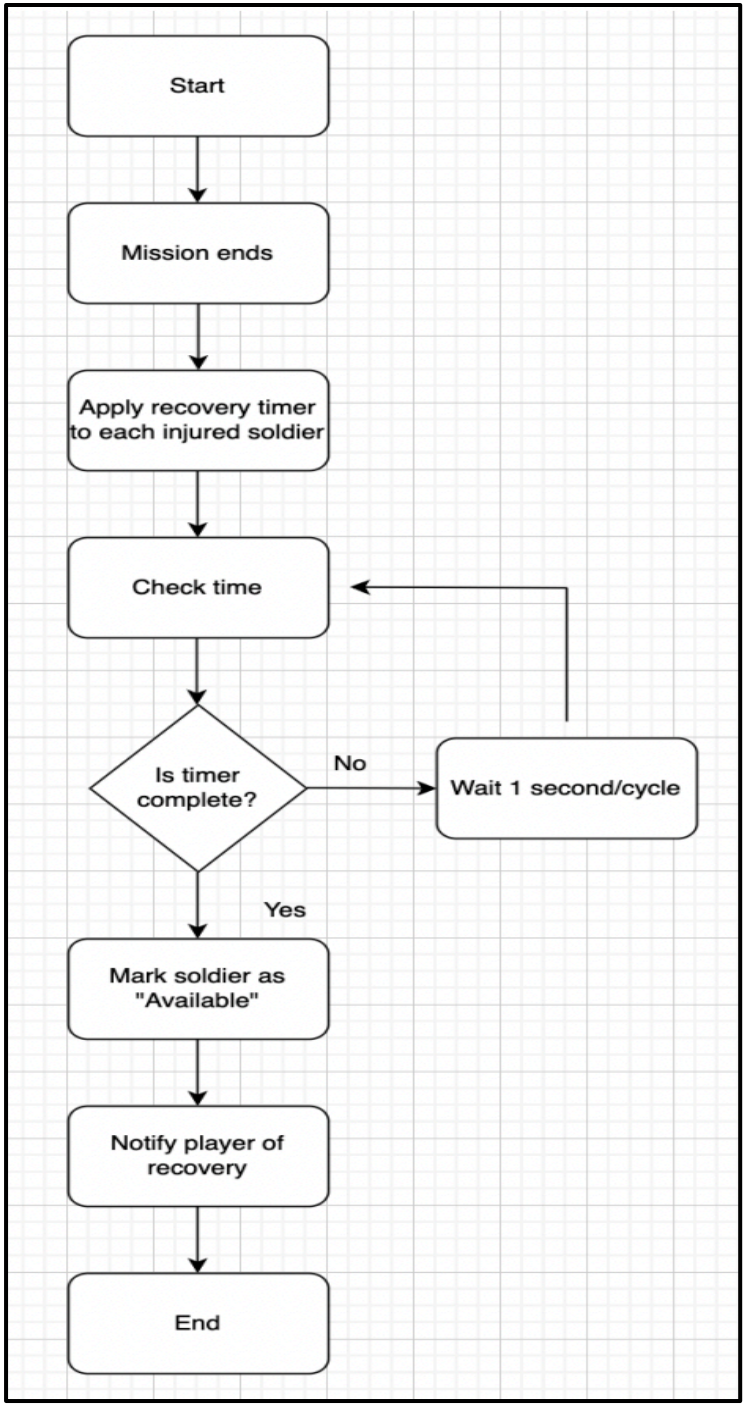
1. **Start Tactical Mission:** The process begins when the player deploys soldiers to a mission.
2. **Initialize Mission Timer:** The timer is set based on mission-specific constraints (e.g., "Complete within 10 turns").
3. **Player Takes Turn:** The player performs their actions during their turn.
4. **Decrement Timer:** At the end of the player's turn, reduce the timer by one.
5. **Check Timer:**
    a. If the timer is greater than zero, the mission continues, and the player takes another turn.
    b. If the timer reaches zero, proceed to mission failure logic.
6. **Mission Fails:** Trigger the mission failure state due to time expiration.
7. **Apply Penalties:**
    a. Deduct resources
    b. Lower morale
    c. Mark soldiers as injured or killed if applicable
8. **Notify Player of Failure:** Provide feedback on why the mission failed (e.g., "Time expired. Soldiers lost!").
9. **End Mission:** The tactical phase finishes.

Critical Section: Soldier Recovery

- **Location in Program: Staff → Recovery Timer**
- **Description of Its Importance & Purpose:**
    - Soldiers are a vital resource in the game, and their availability directly affects the player's ability to complete missions. After each mission, soldiers need time to recover health, morale, or other attributes before they can be reassigned.
    - Without this system, players could overuse injured soldiers, breaking immersion and game balance. Proper recovery management adds depth and realism, forcing players to plan strategically.
    - This feature prevents gameplay exploits while maintaining progression fairness.

- **Proposed Solution in Words:**
    1. **Start Recovery Timer:** When a mission ends, assign a recovery timer to each participating soldier based on mission difficulty, injuries, or fatigue levels.
    2. **Track Recovery Progress:** Periodically check the timer to update the soldier's recovery status.
    3. **Mark as Available:** Once the recovery timer expires, update the soldier's status to "Available," allowing them to b| reassigned.
    4. **Notify Player:** Inform the player when soldiers are fully recovered, ensuring they can efficiently manage their roster.
    5. **UI Integration:** Display recovery timers for all injured soldiers in the staff management screen.

Flowchart 2


Start
Mission ends
Apply recovery timer to each injured soldier
Check time
Is timer complete?
No
Wait 1 second/cycle
Yes
Mark soldier as "Available"
Notify player of recovery
End

Explanation of Flowchart 2:

1. **Start:** The process begins as soon as a mission ends, triggering the recovery logic for soldiers who participated in the mission.
2. **Mission Ends:** At this step, the game identifies that a mission has concluded, and it prepares to evaluate the soldiers involved.
3. **Apply Recovery Timer to Each Injured Soldier:** The system evaluates all soldiers who participated in the mission. For each soldier marked as injured, a recovery timer is assigned. The recovery timer is based on the severity of the injuries (e.g., minor injuries may take less time to heal, while severe injuries take longer).
4. **Check Time:** The system checks the status of the recovery timer for each injured soldier to ensure that the soldiers' statuses are updated as their recovery progresses.
5. **Decision:** Is Timer Complete? -

    a. If the recovery timer for a soldier has not yet expired: The system waits 1 second or 1 cycle before looping back to check again.

    b. If the timer has expired: The soldier is marked as "Available" for redeployment.
6. **Wait:** The system waits for a short interval (e.g., 1 second in real-time) or a single game cycle before checking the timer again. This prevents the system from continuously running checks, reducing unnecessary computational load.
7. **Mark Soldier as "Available":** Once the recovery timer expires, the system updates the soldier's status to "Available." The soldier is now ready for redeployment in future missions.
8. **Notify Player of Recovery:** The player receives a notification indicating that the soldier has fully recovered. This step ensures the player remains informed and can make strategic decisions about their roster.

9. **End:** The process concludes when all injured soldiers have either fully recovered, or their timers are still active and awaiting completion.

7. Effort Estimation

7.1 Optimal Workflow Graph

Development Environment

Unity/C#

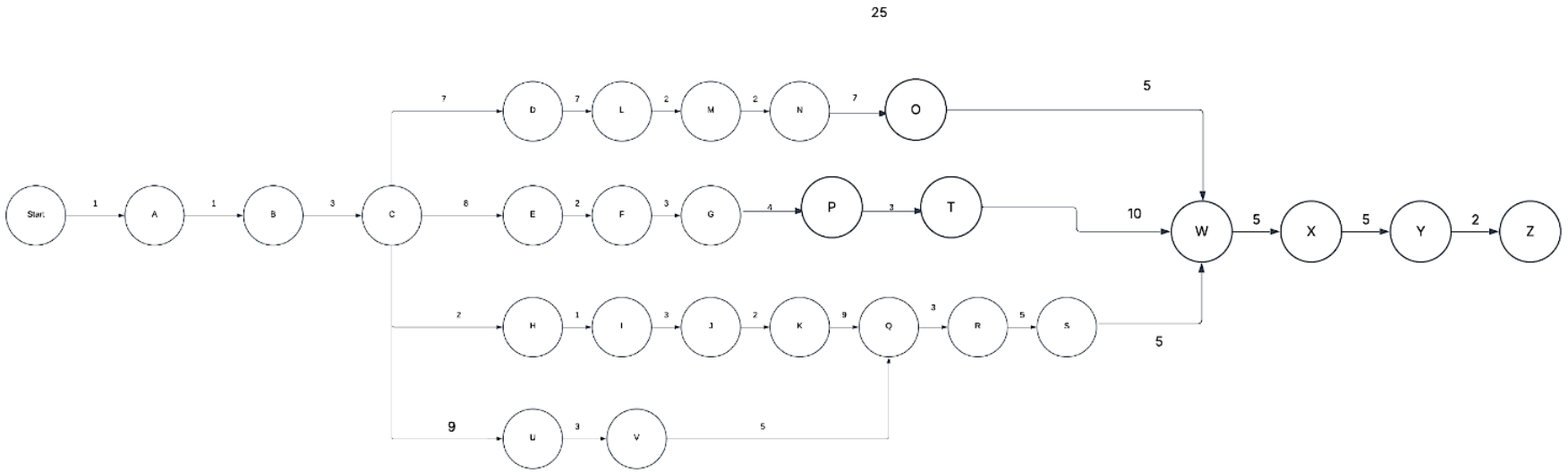
The unity game engine is both simple, free and effective. Both can be easily installed and are compatible over multiple operating systems. They can be easily coded using the Visual Code IDE.

Database Server

SQLite

It is a free and open-source system that can be used and setup easily. The system can be run locally with no extra cost. If that is not possible, a cloud server with SQL could be setup with minimal costs as well although less ideal.

Activity Network Diagram with Shortest Path Analysis



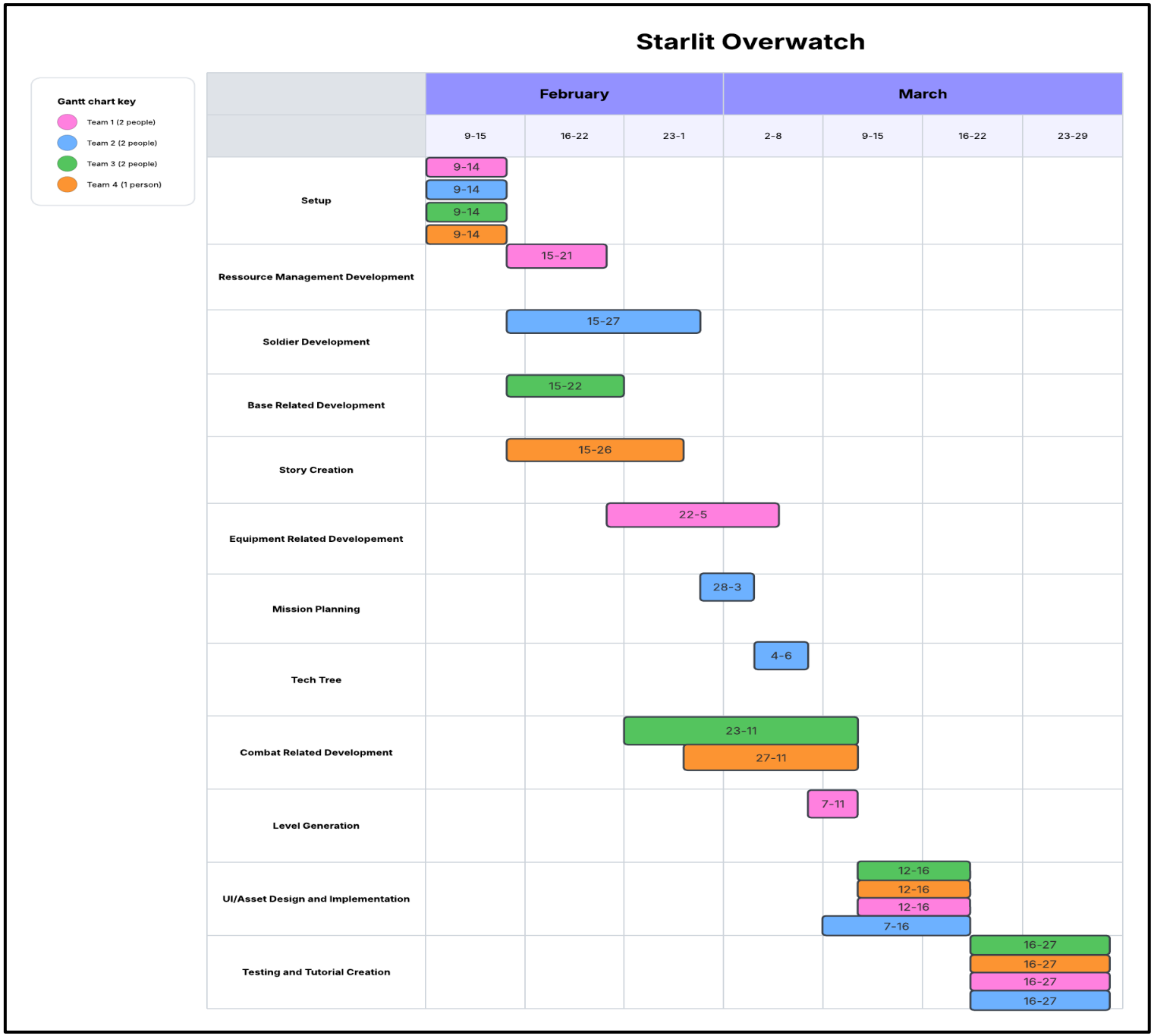
Shortest Path and Time Estimation

The shortest path is defined by Start-A-B-C-D-L-M-N-O-W-X-Y-Z and is 47 days long. With padding and the challenges of working with a large team. It is estimated that the project would take around 65 days in total to complete while including padding and break periods.

7.2 Effort Estimation Table

Effort Estimation Table		
Nodes	**Description**	**Dependency**
A	Development Environment Setup (Unity)	None
B	Database setup	None
C	Basic Systems Setup	Environment Ready
D	Basic resource management development	Environment Ready
E	Soldier management development	Environment Ready
F	Soldier inventory option	E
G	Soldier type creation	E
H	Base screen development	None
I	Base structure placement function	Environment Ready
J	Implementation of different structures	H
K	Structure upgrade system	H
L	Equipment function development	Environment Ready
M	Creating equipment	L
N	Adding unique equipment passives	L
O	Level generation	Environment Ready
P	Mission planning function	Environment Ready
Q	Combat infrastructure development	Environment Ready
R	Turn-base combat	Q
S	Enemy AI function	Q
T	Tech tree development	Environment Ready
U	Story/Narrative	None
V	Story Campaign	U
W	UI/Assets Design and Implementation	Most features are in place
X	Unit testing	All previous development
Y	Play testing	All previous development
Z	Creating Tutorial	All previous development

7.3 Gantt Chart



Starlit Overwatch Task and Team Timeline

This Gantt chart, titled "Starlit Overwatch," outlines a project schedule spanning February and March, detailing tasks assigned to different teams. It shows overlapping timelines for key activities such as development, story creation, mission planning, and testing, with color-coded sections representing team contributions. The chart highlights sequential dependencies and resource allocation across multiple tasks, providing a clear visual of the project's workflow and deadlines.

Most steps in the Gantt chart are the ones describe in the effort estimation graph. A lot of steps were condensed in the Gantt chart to keep it at a reasonable size. For example, instead of Soldier Management development, soldier inventory option and soldier type creation, soldier development is indicated on the Gantt chart. For more details, refer to the table on page 1.