First and foremost, I would like to thank you for this opportunity. I saw this as a chance to accomplish two goals simultaneously. For this project, I leveraged scripts from a previously developed proprietary framework, the **Cerberus Framework (CF)**, to streamline development and enhance the project's architectural integrity. The reusable components provided by CF served as a solid foundation, allowing for the efficient integration of core functionalities and significantly reducing development time.

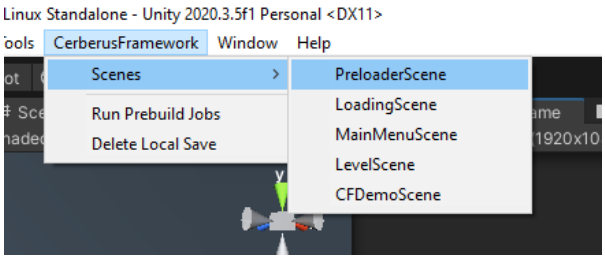
I kindly ask you to consider this case study in two distinct parts:

- **Cerberus Framework** as a demonstration of my approach to **System Design**
- **Scopely Case Study** as the focus for **Code Review**.

Cerberus Framework:

https://www.kaanekin.com/cerberus-framework

Boot Scene: Preloader Scene



UI is placed for 1920x1080 landscape resolution

Used 3rd Party Assets:

LeanGUI for Joystick control
https://assetstore.unity.com/packages/tools/gui/lean-gui-72138

Layer Lab GUI Pro - Casual Game
https://assetstore.unity.com/packages/2d/gui/gui-pro-casual-game-176695

DOTween
https://assetstore.unity.com/packages/tools/animation/dotween-hotween-v2-27676

Unity Object Pool (from Unity 2021)
https://github.com/Unity-Technologies/UnityCsReference/tree/master/Runtime/Export/ObjectPool

Summary

As reflected in the Git commit logs, the initial step in the project involved importing the **Cerberus Framework (CF)** and adapting portions of the codebase from **C# 9** to **C# 8** to ensure compatibility. Additionally, I incorporated Unity's object pooling system by manually importing the relevant classes from Unity's official repository: UnityCsReference – ObjectPool.

Following this, I proceeded to design the overall architecture of the game. Adhering to **SOLID principles** and the **Model-View-Controller (MVC)** design pattern, I established a foundational Character class, from which I derived the Base, Turret, and Enemy classes.

In support of modularity and reusability, I also introduced a Component class, which serves as the parent for Life, Attack, and Movement components. These components can be flexibly attached to character entities, enabling a modular system that allows for dynamic composition of behavior and functionality across different game elements.

With these systems in place, I was able to implement the core game mechanics in a clean and organized manner. Once the architectural foundation was completed, the remainder of the development proceeded smoothly, and I was able to finalize the project without any major issues.

The project provides a modular framework for managing characters, built around the MVC pattern to clearly separate character behavior, data, and visual representation. This architecture prioritizes modularity and extensibility, making it straightforward to introduce new character types, behaviors, and weapon systems with minimal code modification.

Project Overview

The game manages its initialization and progression through a structured flow of scenes, ensuring that managers are properly configured before gameplay begins. This process is divided into four scenes: **Preloader Scene**, **Loading Scene**, **Main Scene**, and **Level Scene**.

However, upon completing the initial implementation, the **Loading Scene** was deemed unnecessary, as the game is currently a demo with minimal resource requirements. Despite this, the Loading Scene remains a valuable component for future development, as it can facilitate asset management and optimize performance as the game expands.

The **Preloader Scene** is the first scene loaded when the game launches. Its primary function is to initialize the core managers of the game, ensuring the foundational systems are prepared for subsequent operations.

Loading Scene can be invoked between scene transitions at any point in the game. During these transitions, the Loading Scene is displayed while the target scene completes its loading process. Once the new scene is fully loaded and ready, the Loading Scene is closed, providing a seamless and visually consistent experience for the player.

Once the **Loading Scene** completes its operations, control transitions to the **Main Scene**, which functions as the central hub for player interactions. The Main Scene serves as the first interactive environment for the player, ensuring a smooth and intuitive transition from the initialization phase to active gameplay.

Managers are global components initialized at the start of the game. They remain active throughout the entire gameplay experience, providing essential core services and features that support the game's foundational operations.

Systems, on the other hand, are localized components that operate within specific gameplay phases. Unlike Managers, Systems are activated and deactivated based on the game's progression. These systems handle tasks such as enemy spawning, level transitions, and in-game events, ensuring seamless and responsive gameplay tailored to the player's current experience.

Level configuration:

1. Define Level Data

- LevelData.cs stores level information:
  - WaveData: A list of WaveConfig for enemy waves.
  - BaseConfig: Configuration for the player's base.

2. Configure Waves

- WaveConfig defines each wave:
  - EnemyConfigCountPair: List of enemy types (EnemyConfig) and counts.
  - SpawnDelay: Time before wave starts.
  - SpawnInterval: Time between enemy spawns.

3. Configure Enemies

- EnemyConfig.cs (and CharacterConfig.cs) sets enemy properties:
  - EnemyType: Type of enemy.
  - Health, MoveSpeed.
  - WeaponConfig: Enemy's weapon.
  - RewardedCoin: Coins given for killing.

4. Configure Weapons

- WeaponConfig.cs, MeleeWeaponConfig.cs, RangedWeaponConfig.cs define weapon behavior:
  - Base properties (damage, cooldown, range).
  - Ranged weapons also have bullet properties.

Reflections on the Project

What I Added:

Turret costs now dynamically increase with each purchase. The freeze effect applied is time-limited. Weapon systems have configurable cooldown periods between attacks and defined effective ranges. Finally, the interval between enemy spawns in each wave is now adjustable.

What Else Could I Have Done (With More Time):

Given more development time, several enhancements could have significantly improved the gameplay experience. Enemies could have been refined to only deal damage to the base once upon entering before being disposed of, preventing continuous damage while inside. The game could have been made more immediately engaging by starting upon the first player input, instead of automatically. Visually, the health bars could have been significantly improved with better UI design.

Bugs I Noticed:

An exception occurs when a turret is selected at the moment the game ends.

The RTS camera's current behaviour issues likely arise from its configuration settings.