SURVIVORS TEST

Introduction

To start with, I have done every task requested in the test. However, I regret not doing some extra touches that I wanted to do, like adding audio and more visual effects, due to time constraints. I was only able to work on the test for almost 20 hours in total.

All the values are customizable in Scriptable Objects, but they are not optimized or balanced. Feel free to play around with them.

It's not as polished as I would have liked, but I did what I could in the time I had. I hope you like it! 😊

Player

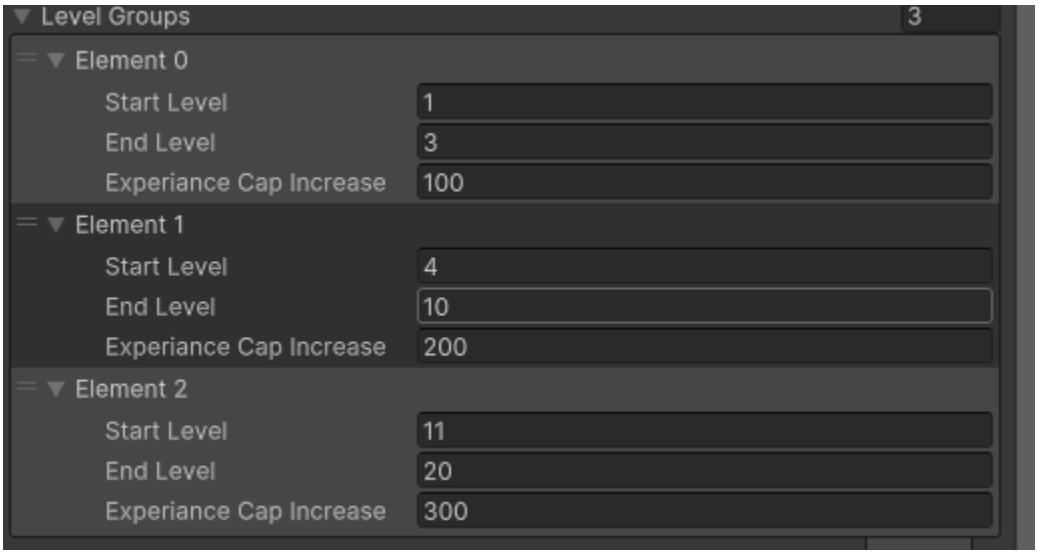
For the player movement I implemented a Joystick which in the hierarchy can be found under JoystickCanvas. I opted for a separate canvas for optimization reasons, because the joystick UI element is moved constantly and that refreshes the whole canvas every time so it made no sense to refresh all the other UI elements by moving the joystick. The script is called Joystick.cs

Animation and sprite changes for the player are handled by the PlayerAnimator.cs, attached to the Player game object.

Player movement is handled in PlayerController.cs, also attached to the Player GO.

Player stats are changed in the PlayerStats.cs. This script holds logic to the player's stats, power ups and leveling system. The levels are grouped into Level Groups which can be customized in the inspector from the Player GO. The purpose of the groups is to keep one experience cap for multiple levels. In the below image, there are three groups and it means the levels between 1 and 3 each have an experience cap of 100, the levels between 4 and 10 a cap of 200 and so one.

IMPORTANT!! When adding groups make sure the StartLevel of the new group is one higher than the EndLevel of the previous group.



The player stats are set and customized in the Player.cs ScriptableObject. All scriptable objects are found under the ScriptableObjects folder.

Weapon

I made a system that allows equipping multiple weapons but since it wasn't required I only worked with one.

The script that handles the weapon mechanic is called WeaponController.cs and it is attached to the Hand Left child object of the Player GO. It does the initialization of the weapon, the attack logic, the pick up logic and the enemy handling logic.

For the one weapon in the game, stats are set from the MachineGun SO, but the asset creation menu allows creation of multiple weapons.

For the bullets I opted for **object pooling** as they move fast and are very dynamic.

Enemies

There are 3 types of enemies in the game: Grunt, Zombie and Ogre. The prefabs are found under Prefabs/Enemies. Because they didn't have any particularities as per the instructions, I did not use inheritance as it wasn't necessary. There are two scripts called EnemyStats.cs and EnemyController.cs attached to all prefabs. The varying stats are being set in scriptable objects which can be created from the menu under ScriptableObjects/Enemies/Enemy. It has data such as damage, speed, probability of spawning and more.

Enemy Spawning

The enemy spawning logic happens in the EnemySpawner.cs attached to the EnemySpawner GO in the Gameplay scene.

I split the spawning into two sections: Wave spawning and Enemy spawning.

The Enemy Spawner holds a list of the total waves configured. Each Wave gets its configuration from a scriptable object. Each Wave SO holds a time to wait before the wave is spawned. Each wave can be configured with what enemies to spawn and how many.

For the enemy spawning I did not do object pooling. It can be done but it would have taken more time with the current spawning system that I set up and I didn't have any more time.

Loot

SO for the loot can be created from the asset menu under ScriptableObjects/Items. I've already made three such objects for Health, Exp and Ammo and their drop probability can be configured in the SO.

A script called ItemDrop.cs, attached to each enemy, handles the logic for dropping an item upon enemy death, based on probability.

For the items I have used **inheritance** because they have common logic as well as specific logic. All the item scripts inherit from one parent script called Collectible. The parent script handles the flying towards the player and the autodestruct logic. The CollectItem method is being overridden in each individual item's cript

GameManager.cs and UIManager.cs

The GameManager.cs handles the game logic and it contains a simple state machine with two states (Gameplay and GameOver). I usually do this because you can also have states like Paused, Item Selection, Cutscene etc.

It does the game over logic and keeps track of time and score.

The UIManager.cs handles everything UI related, like HUD updates and showing the stats screen at the end.

Upgrades

I have made a total of five upgrades, all of which can be found in the ScriptableObjects folder, under PowerUps. They are also configurable and more can be created. I am using inheritance here also, from an abstract scriptable object class.

Note! Because it wasn't specified in the test, I did not implement a timer for these power ups that activate at the end of each level up. So currently, they are permanent.

DI

For the DI I kept it simple. One scene context with one MonoInstaller, which binds classes from existing objects in the scene. For classes that required one dependency I used field injection, for classed that require more (I have no more than two), I used Method injection:

```
/// <summary>
/// inject references to Joystick and UIManager
/// </summary>
/// <param name="newJoystick"></param>
/// <param name="newUIManager"></param>
[Inject]
public void Construct(Joystick newJoystick, UIManager newUIManager)
{
    this.joystick = newJoystick;
    this.uiManager = newUIManager;

}
```