Subject: Test Assignment for Unity Developer Position at Estoty

I am pleased to submit the completed test assignment for the Unity Developer position. Here is what I implemented in the project:

Architecture

- Since the project doesn't require general dependencies (like CurrencyService, ProgressService, etc.), I kept everything within a single "Gameplay" scene. This scene contains Zenject's SceneContext, which resolves the necessary dependencies. After that, the LevelLoader takes control and spawns the level prefab.
- Levels are prefabs that are spawned by the LevelLoader. This approach allows for easy destruction and creation of any level as needed.
- Once the level prefab is created, the LevelEntryPoint script initializes the game: it instantiates the player, links the MVC components and starts the spawners. Centralizing the initialization code in one place is highly beneficial, as it allows us to inject logic between any steps, such as showing ads before the player spawns or letting the player select an initial ability.

Spawners

The project contains two types of spawners: enemy and bonus.

- **Enemy Spawners:** The Enemies GameObject uses ILevelMapService to spawn enemies. This service provides the map data, which, in this case, includes the camera frustum and arena size.
- **Bonus Spawners:** Under the "Props" GameObject, bonus spawners are extensions of the DropOnDeathComponent, which, as the name suggests, drops items upon character death. To resolve the lag issue caused by too many items spawning at once, I implemented an object pool. The DropOnDeathComponent has a "Use Pool" option—if unchecked, it will spawn bonuses independently.

UI

- Given the simplicity of the UI, I used a Screen Space - Overlay canvas and an orthographic camera that follows a specified target (in this case the player). However, it can easily be adjusted to follow any transform. The camera follows the player using the FollowTargetComponent.
- In the LevelEntryPoint you will find the MVC linkage. It requires three types of interfaces: IView, IController, and IModel. The IView is represented by the HUD GameObject, the IController is dynamically created and can be changed in the SceneContext MonoInstaller bindings, and the IModel is the spawned player.

Gameplay

- Each gameplay element is a component (a MonoBehaviour class) that can be added to or removed from the character prefab. This makes it easy to adjust character behavior even during runtime.
- Player movement is handled by the MovementComponent, which uses the SimpleInput system. It takes MovementData (such as acceleration) to calculate and apply movement velocity to the attached Rigidbody2D.
- Enemy movement is controlled by the ChaseTransformMovementComponent.
- You can easily understand a character's behavior by examining the components attached to its root.
- The player can use both melee and ranged weapons—choose whichever suits your needs.

P.S.

I did not spend much time fine-tuning the colliders, as I decided to focus more on the gameplay elements. While some colliders might look a bit strange, they still function as intended.