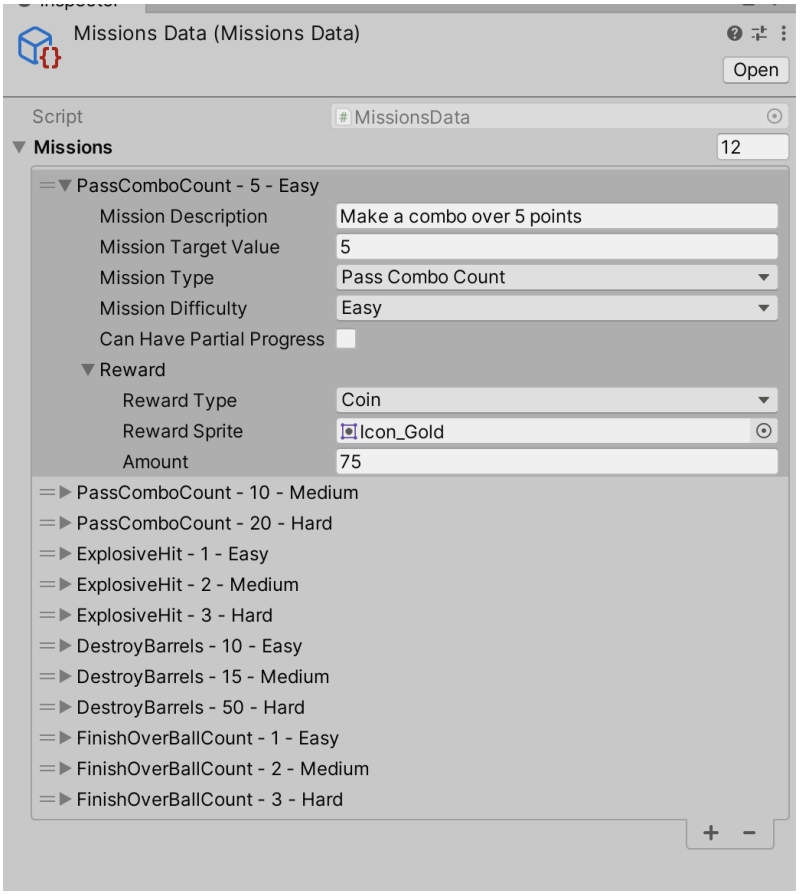
Aykut Yılmaz - Case Study

1- To optimise draw calls I enabled barrel material's GPU instancing. That saved us a lot of draw calls thanks to the batching feature.

In the end in frame debugger it reduced from around 200 to 35 draw calls.

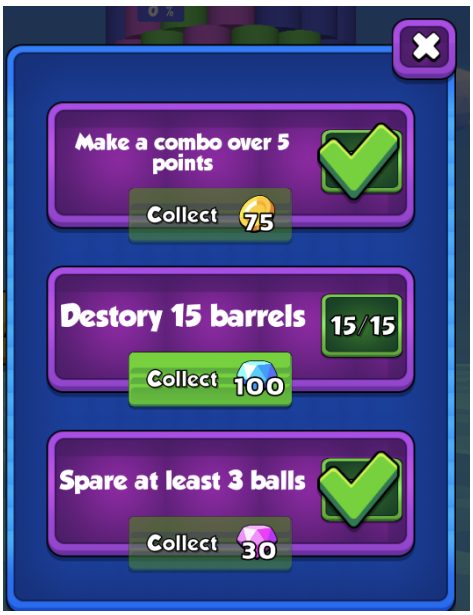
- For the object pooling of the barrels, I created a generic ObjectPool.cs class. By using this class it's easy to create any kind of pool in the future. I created 2 different pools for regular tower tiles and special tower tiles. I used a factory pattern and created TowerTileFactory.cs for managing the pools we have. Factory and pools stay in the scene even after scene reload, so we use pooled barrels during the gameplay even in different levels.

2- For missions I created a scriptable object to easily manage all the mission's data in the game. By using this data object you can add new missions or update the existing ones.



MissionManager.cs uses that data and initialises the UI. There are 4 different types of missions right now and they are assigned randomly (1 from each difficulty) on the game start. While you play the game, the mission manager is updated if a mission related event has happened.

Once you complete a mission you will see in the mission UI, the reward is collectable. Simply click the collect button and get your reward. When you collect all 3 rewards, new missions will be assigned.



Mission system is connected with RemoteConfig via MISSIONS_ENABLED parameter. When it's disabled,players won't see any mission button in the UI.

In my system it's pretty easy to add new kinds of missions. We only need to add MissionType enum and trigger the event of that mission in the code when it happens.

Also adding rewards is pretty easy. Currently there are 3 types of rewards. RewardType is held in an enum, so adding a new type into there would be enough for adding new kinds of rewards.