Hypercasual - Arcade Idle Pack - User Manual

GET THE LATEST DOCUMENTATION

IMPORTANT NOTE!!!

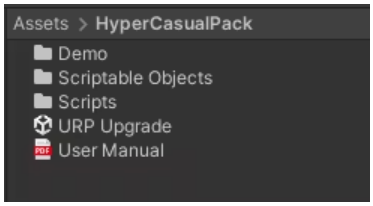
This package depends on DOTween, it's free so install it before installing this package! If you still have problems, please come to our **Discord Server** and ask questions first, if you are still having problem, mail: **thearcadebridge@gmail.com**

I hope you like the package, thanks for buying it!

Package Contents

- **Full source code**
- **Simple Save & Load System**
- **Flexible Scriptable Object Pooling System**
- **Random Level Generator**
- **Inventory**
- **Item Seller (Floating Text, Floating Image)**
- **Item Collectors (Stockpiler, Multiple Condition)**
- **Item Spawner**
- **UI resource monitoring**
- **UI Joystick**
- **Character Controller script**

URP Upgrade



You can use URP Upgrade unity package for upgrading materials to URP.

How to Start

After importing the package, open the demo scene in

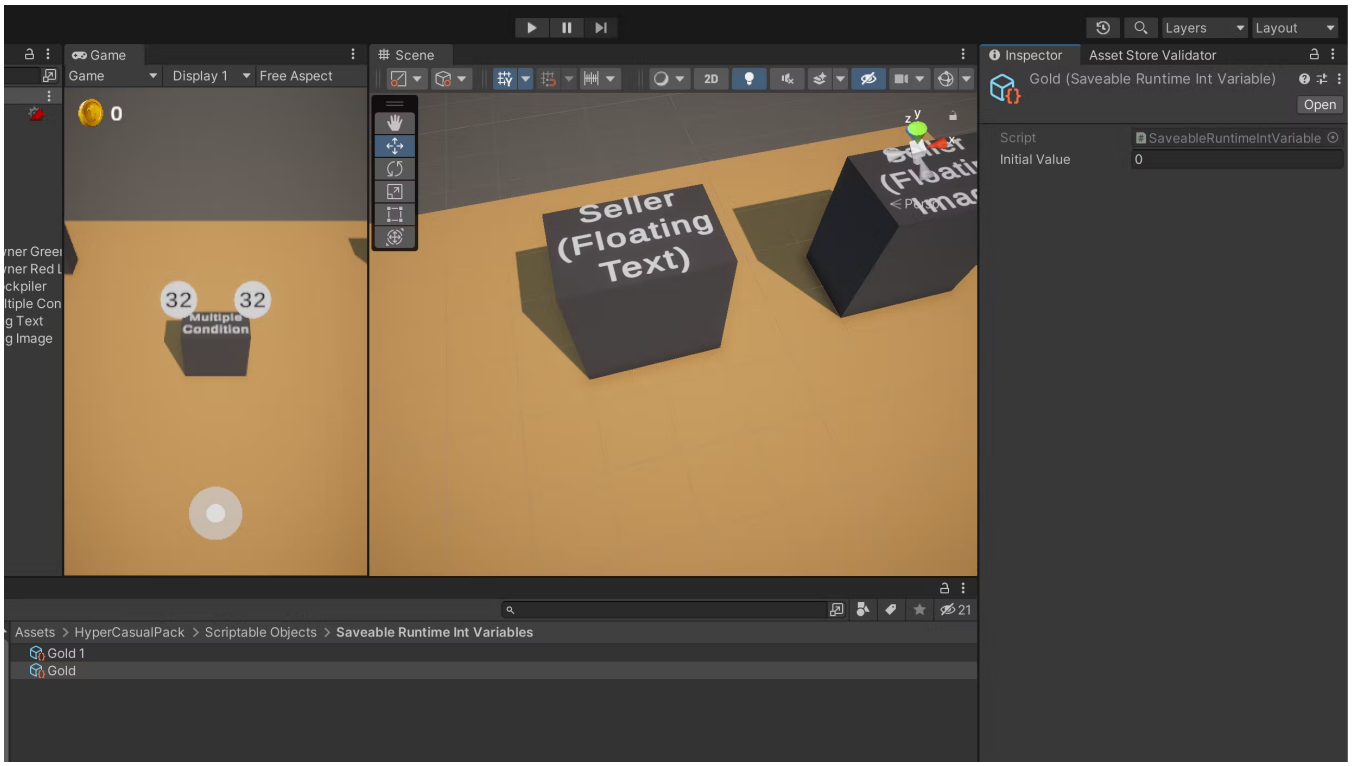
> HyperCasualPack/Demo/Demo.unity

After that you can see how Pickable Collectors, Sellers, Spawners and all those other systems work. In the next section, we'll explain what every script parameter do, so you have better understand overall. If you are still confused, you can always check the default versions of the systems in their corresponding prefabs.

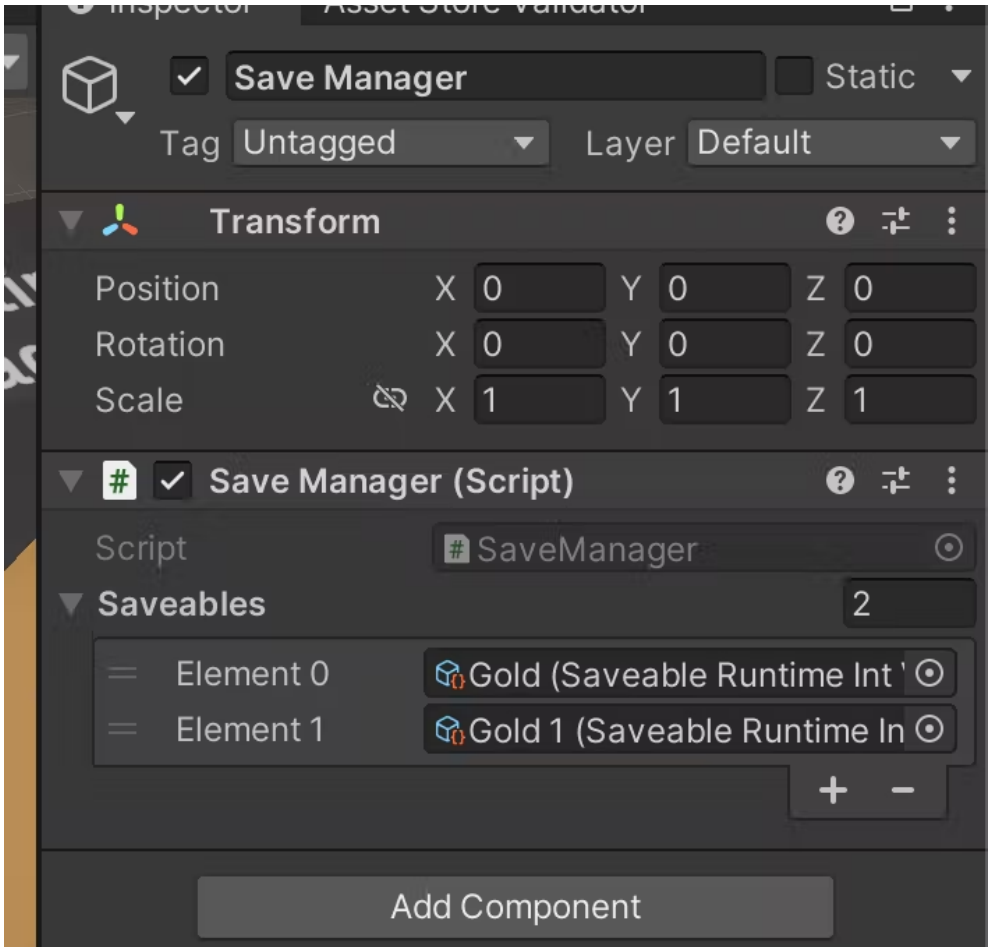
Component Explanations

Saveable Runtime Int Variable

Hypercasual Pack contains a lot of Scriptable Objects, some of them is just plain data holder but others have functionality as well. We can change initial value if we don't have any save file, if we saved our game state then this value will be overwritten by save file.

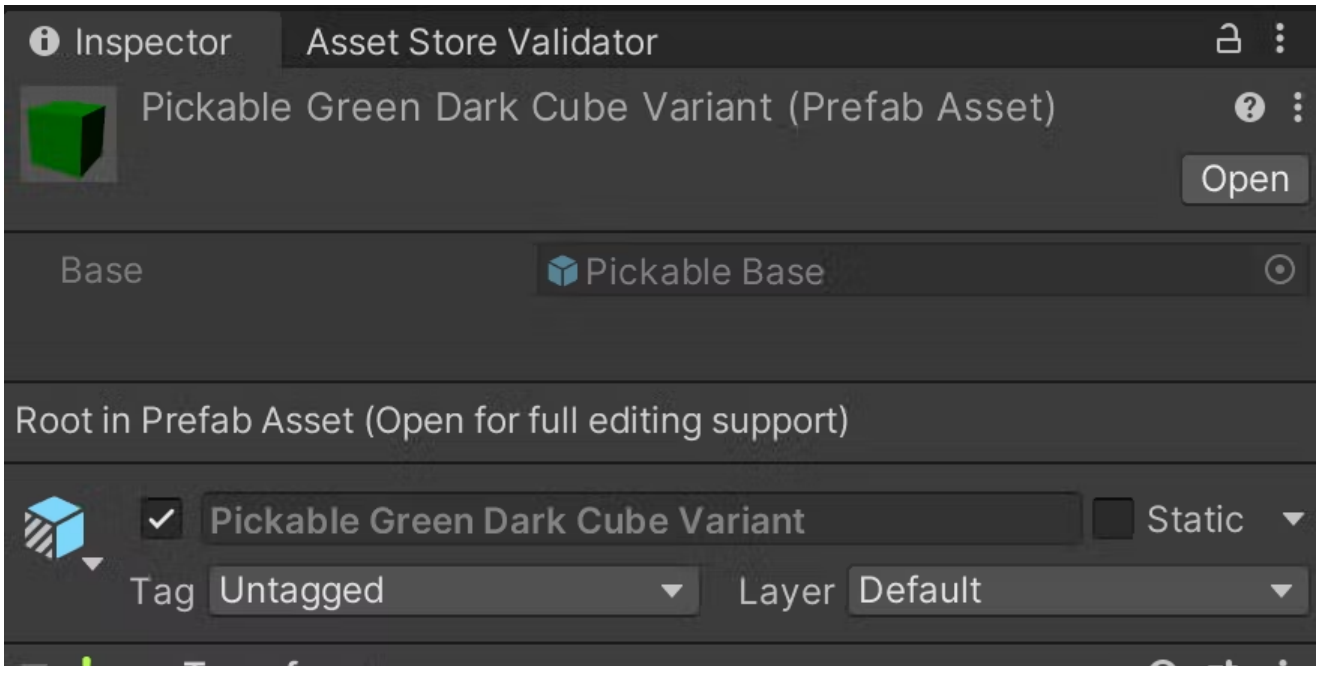


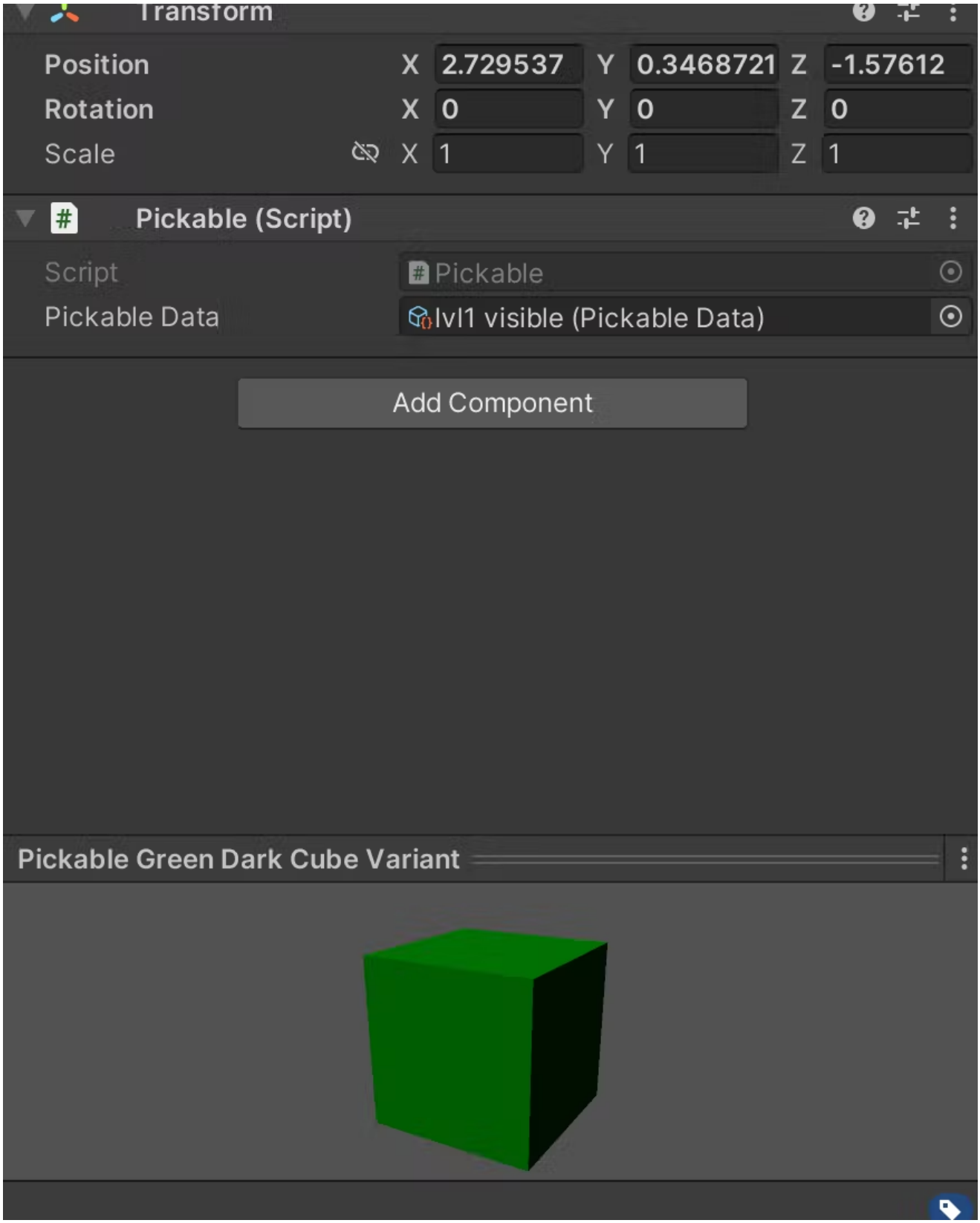
Save Manager



You need to assign Saveable Runtime Ints to the Save Manager so it can Save and Load them when game runs. Use Save Manager for saving your game states.

Pickable

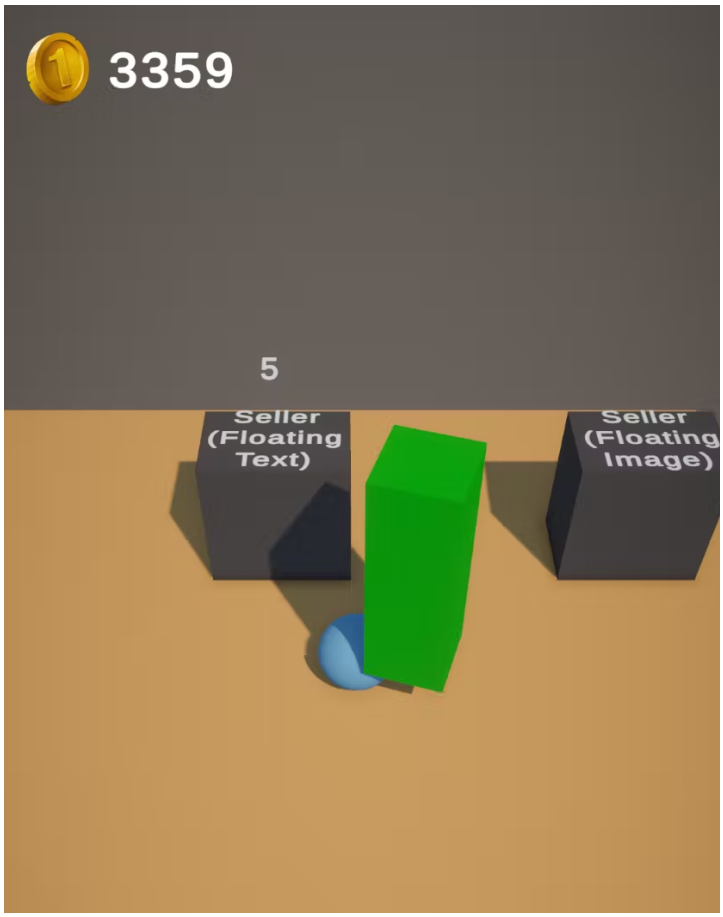


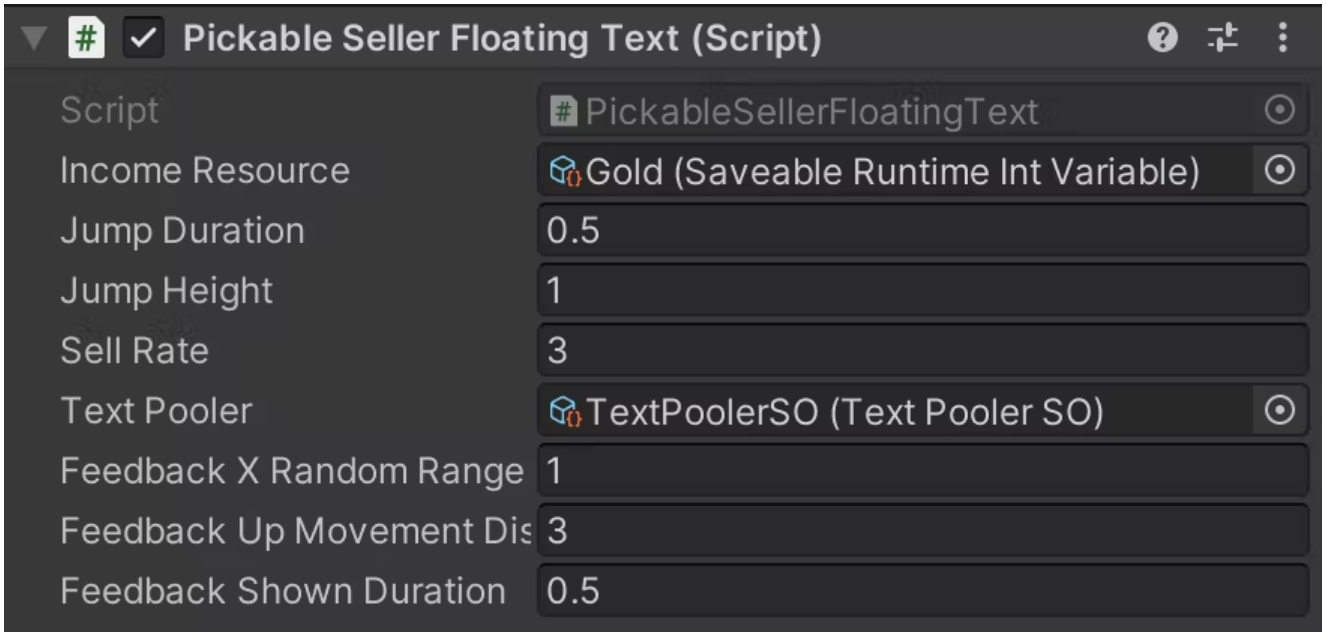


This is one of the Pickable examples, it mostly works as a tagging, but it has an important Pickable Data, which contains Sell Value and Is Visible (which sets whether it should be visible or not when we pick it).

Pickable Sellers

Floating Text

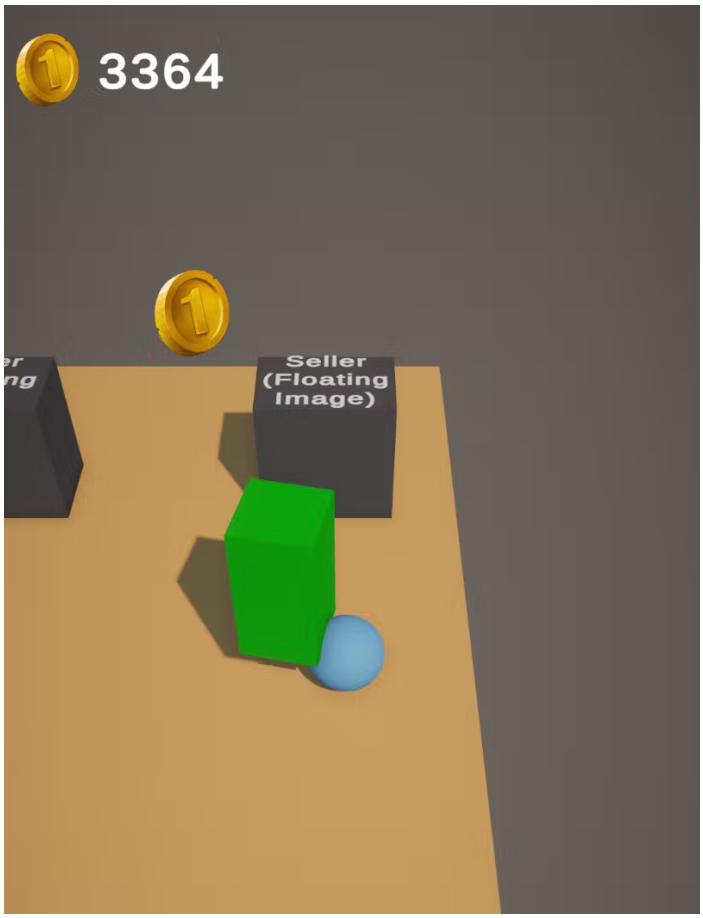


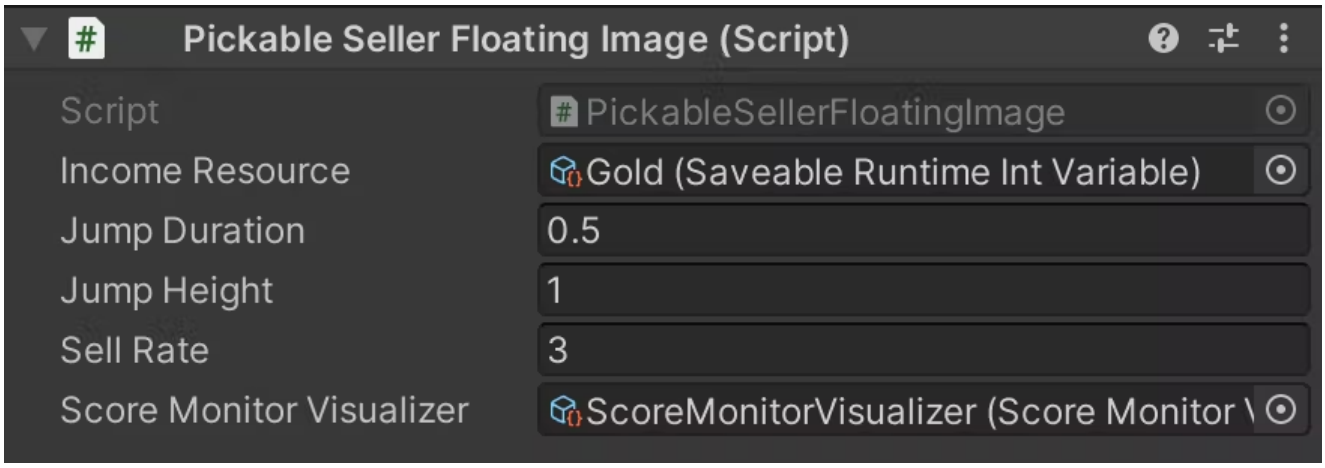


- **Income Resource:** Which int variable should we increase when we sell
- **Jump Duration:** Jump duration when we sell item

- **Jump Height:** Jump height when we sell item
- **Sell Rate:** Selling count per second
- **Text Pooler:** Which text pool to use when displaying feedback
- **Feedback X Random Range:** This value controls the random range of x of the floating text. If you set range too high, feedback text's arrive point will be too much left or right
- **Feedback Up Movement Distance:** This value controls the up movement distance, if you set it too high, feedback text will go too high
- **Feedback Shown Duration:** This value controls the shown duration of the text feedbacks

Floating Image

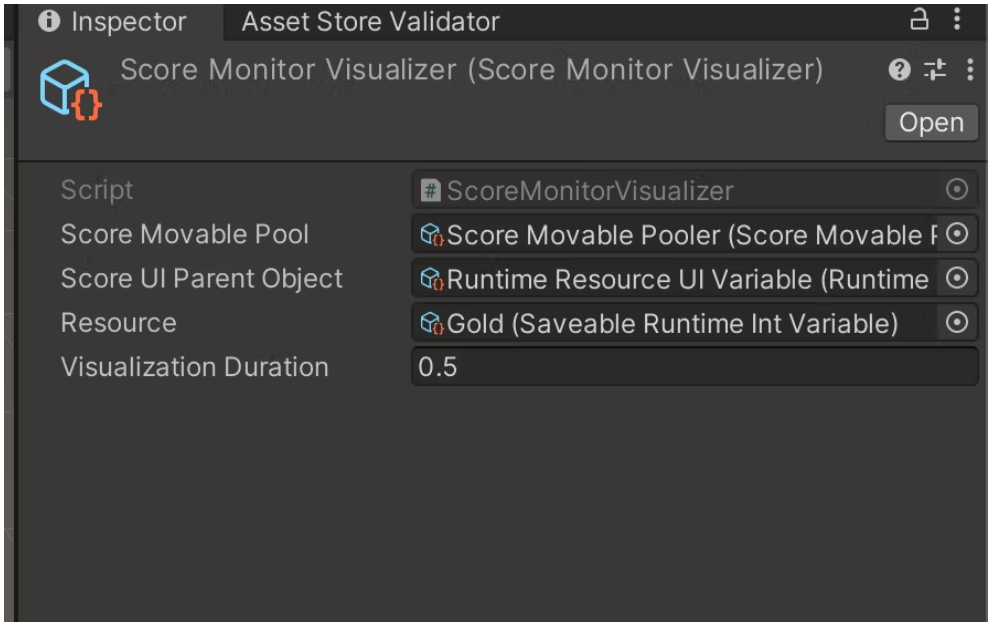




We have only **Score Monitor Visualizer** which is different from the **Floating Text** script.

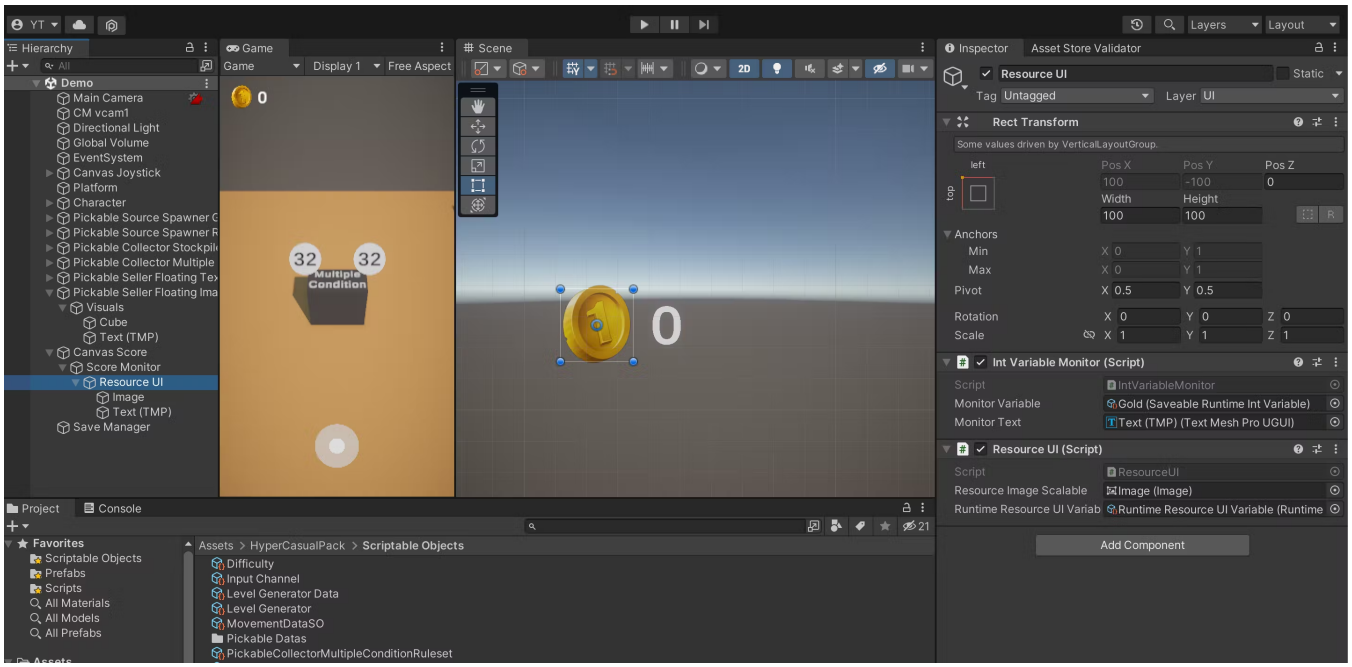
- **Score Monitor Visualizer:** Scriptable Object that is responsible for spawning images on the screen and make them move toward UI.

Score Monitor Visualizer



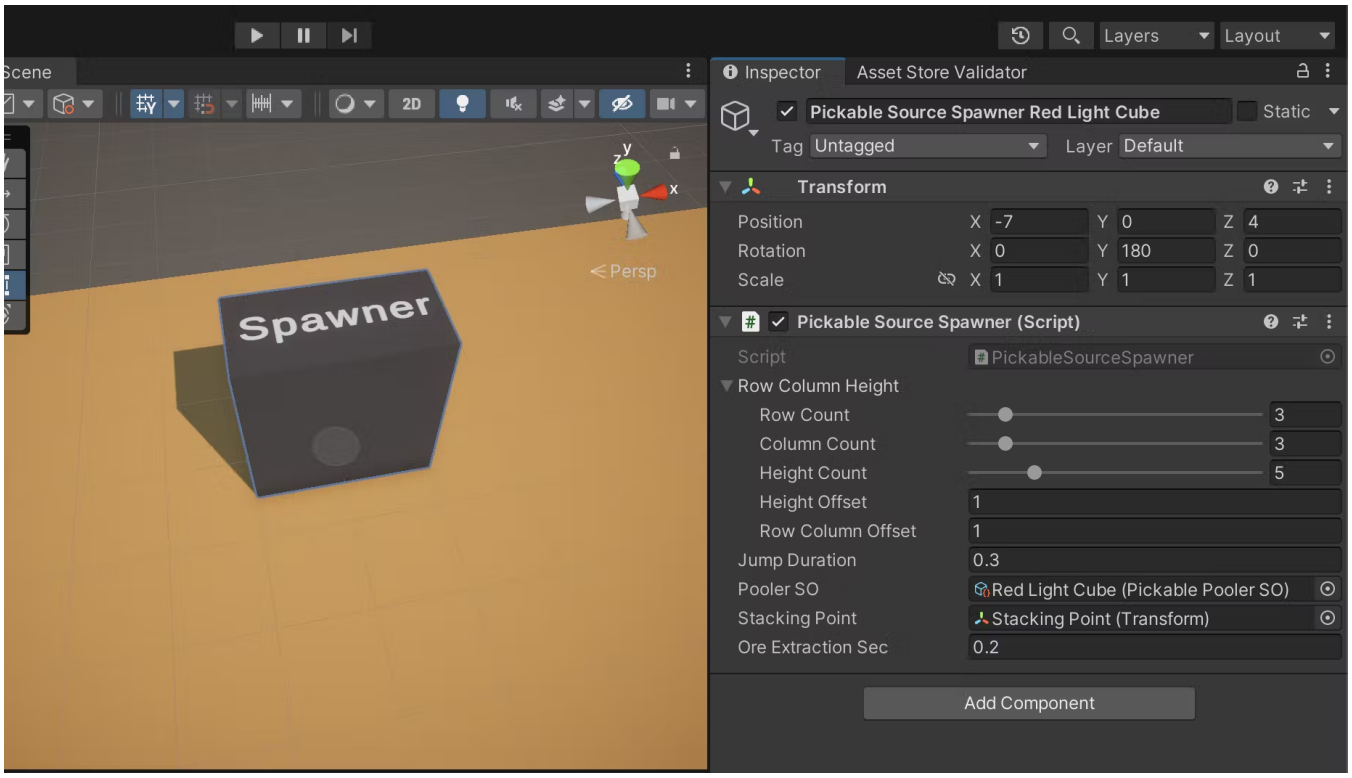
- **Score Movable Pool:** Pool for movable objects on screen. It's an implementation of the generic object pooler.
- **Score UI Parent Object:** Object to arrive. Score Movable Objects will go towards it and after that it will increase it's value.
- **Resource:** Resource to increase when it arrives to the target position.
- **Visualization Duration:** Duration it takes for visualizing the movement.

Resource UI and Int Variable Monitor



Int Variable Monitor is for updating the text based on the variable it listens to.
Resource UI is for registering the object to the scriptable object so Score Monitor Visualizer knows where spawned objects should arrive.

Pickable Source Spawner

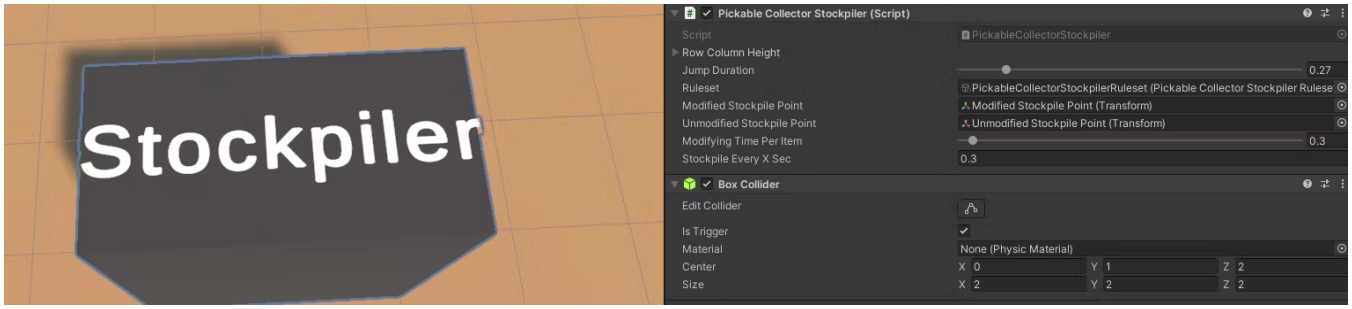


- **Row Column Height:** Properties so spawned object can jump to organized place.
- **Jump Duration:** Spawned object's jump duration.

- **Pooler SO:** Object pool to spawn from.
- **Stacking Point:** Spawned object arriving point.
- **Ore Extraction Sec:** Spawn rate, every X second.

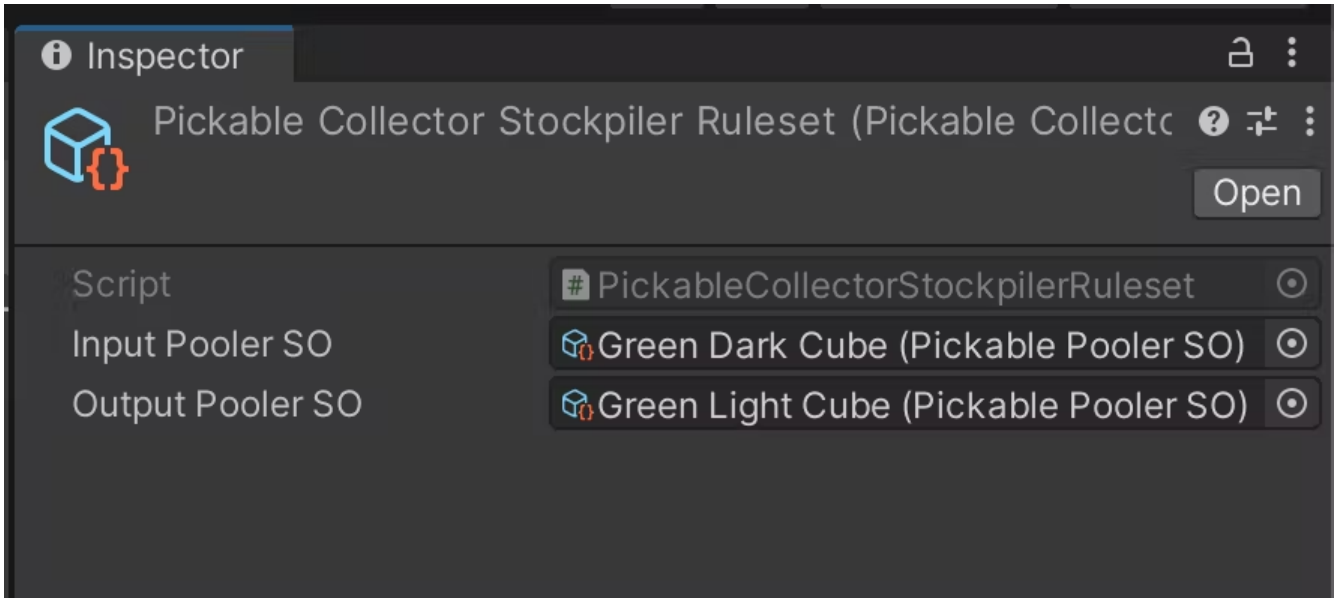
Pickable Collectors

Stockpiler



- **Jump Duration:** Time it takes in seconds to jump from Player to Unmodified Stockpile Point. It can't exceed Modifying Time Per Item.
- **Ruleset:** Contains Input Pooler which is the object it collects, and Output Pooler which is the output object it produces. If it can't find the same type as Input Pooler, it won't collect.
- **Modified Stockpile Point:** Modified items' pivot point, then position calculation happens based on the offsets from Row Column Height.
- **Unmodified Stockpile Point:** Unmodified items pivot point, they are same as Modified Stockpile Point.
- **Modifying Time Per Item:** Time it takes to modify one item (in second).
- **Stockpile Every X Sec:** Item taking speed.

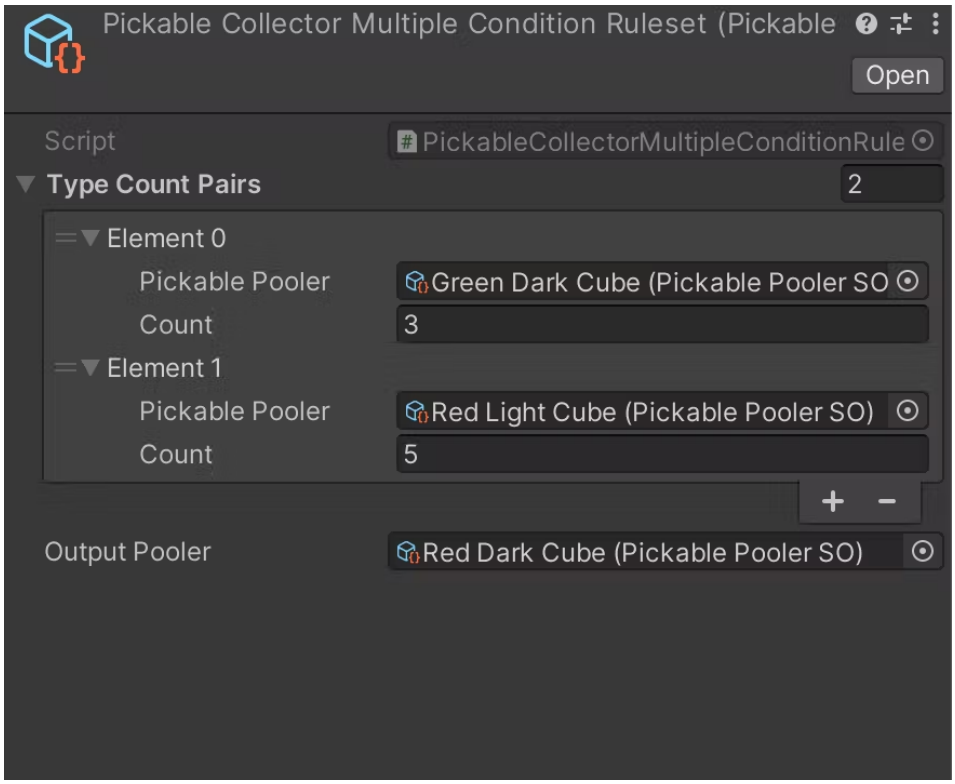
Stockpiler Ruleset



Multiple Condition

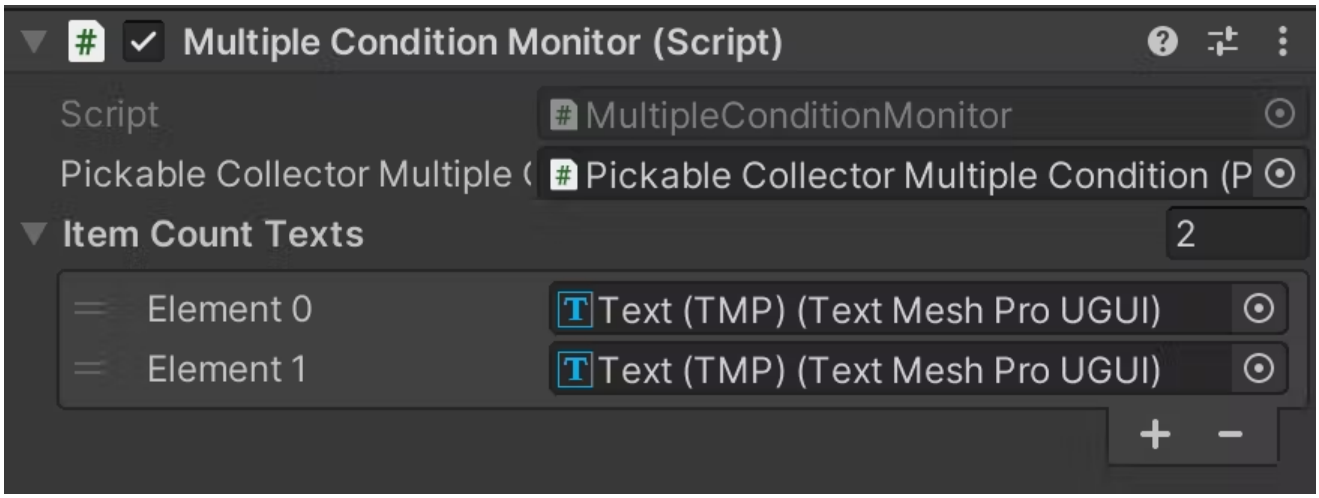
- **Jump Duration:** Time it takes in seconds to jump from player to Unmodified Stockpile Point.
- **Ruleset:** Multiple Condition ruleset which contains which items to pick from player.
- **Stockpile Every X Sec:** Item taking speed.
- **Jump Height:** Jump height from Player to Modified Stockpile Point.

Multiple Condition Ruleset



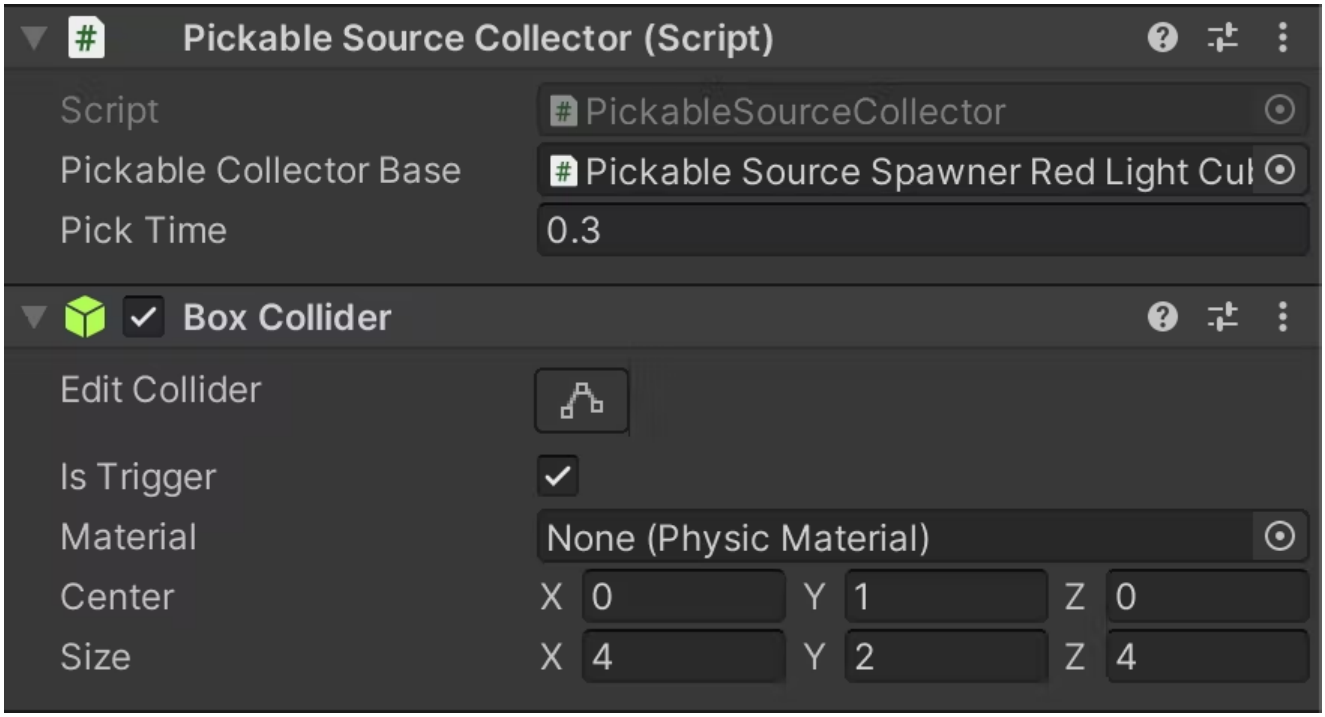
Type Count Pairs contains items, and it also contains how many of them it needs. When we fulfill conditions, object from Output Pooler will be instantiated.

Multiple Condition Monitor



This component will update the texts based on the Pickable Collector Multiple Condition it listens to. Beware that text field count needs to be the same with multiple condition ruleset's Type Count Pairs count. First element in Multiple Condition Monitor updates itself based on the first element in the Pickable Collector Multiple Condition. You can see the example usage in Demo.

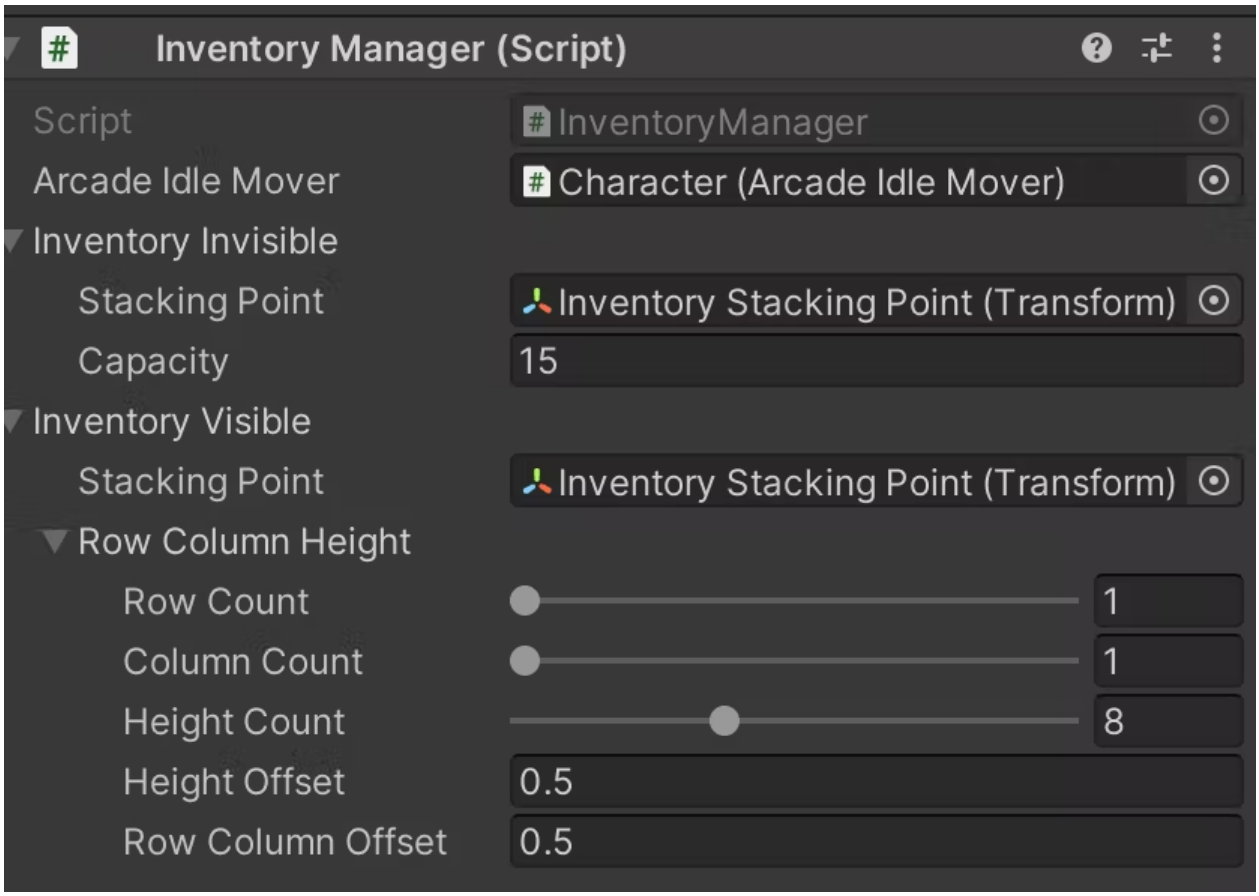
Pickable Source Collector



Pickable Source Collector works with Collider (trigger), that way it can know whether an object with inventory has entered into the trigger area.

- **Pickable Collector Base:** Source object to collect from
- **Pick Time:** Collect every X second

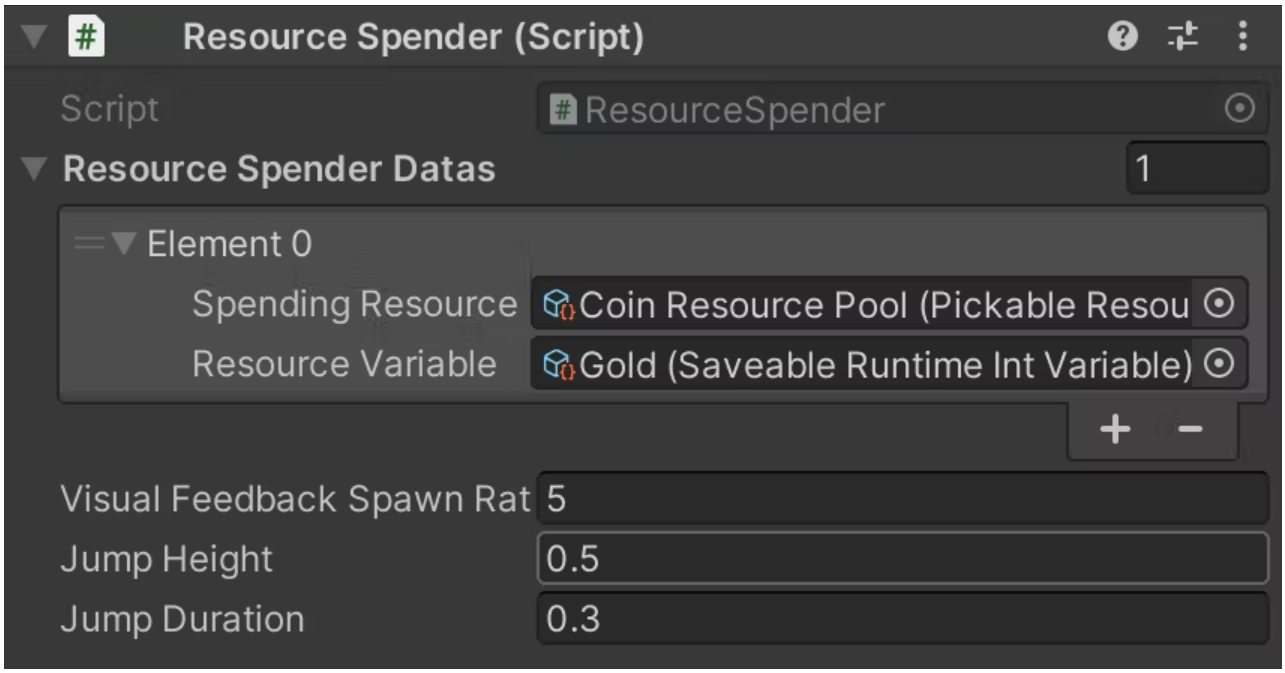
Inventory



We have 2 Inventories, Invisible and Visible one. Invisible Inventory collects objects and make them invisible when they arrive to Stacking Point while Visible Inventory uses Stacking Point as a Stacking starting point. Then depending on your Row Column Height settings, stacked objects will move to desired position.

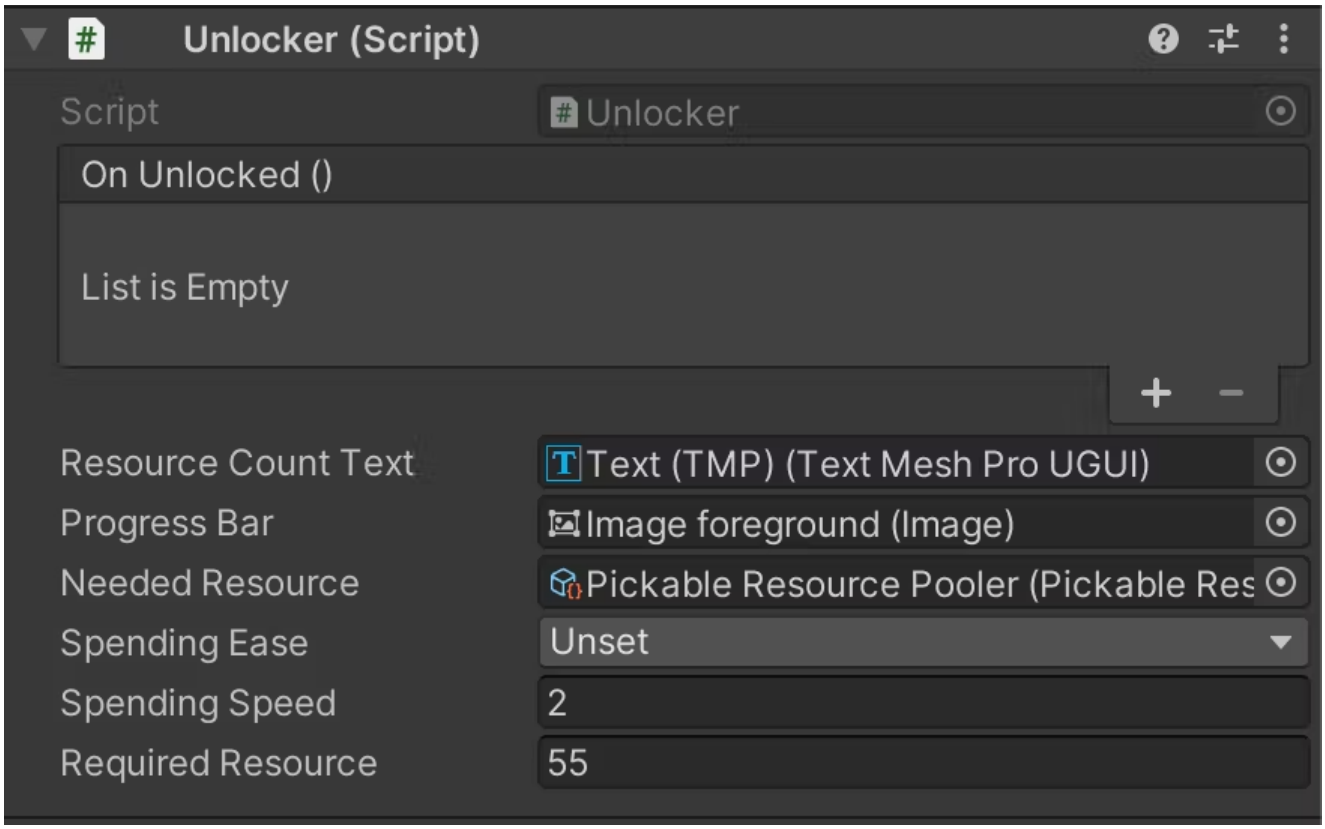
- **Arcade Idle Mover:** Dependency, so Inventory can understand whether we are interactable (it means if we stop) or not.

Resource Spender



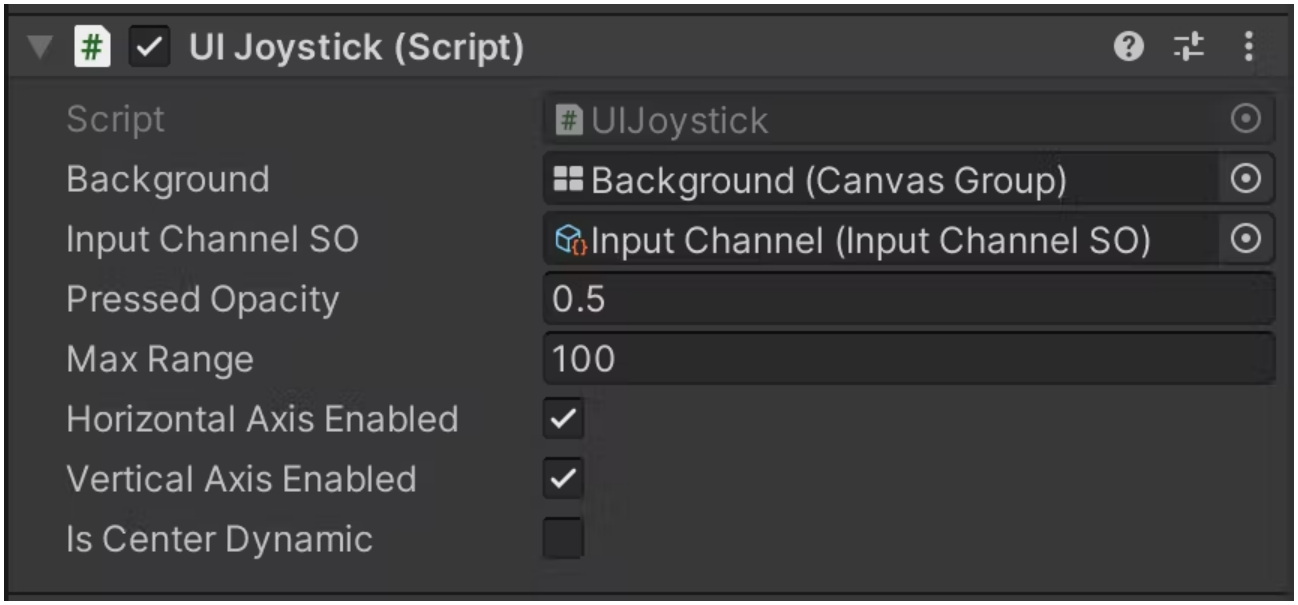
- **Resource Spender Datas:** Data that contains required variables for spending
  - **Spending Resource:** Visuals to use when spending resource
  - **Resource Variable:** Actual variable that holds your resource value
- **Visual Feedback Spawn Rate:** Every X amount, we spawn visual feedback, if you make it 1, than every amount you spend will spawn resource visual.
- **Jump Height:** Resource visuals uses this height when jumping towards target
- **Jump Duration:** Duration for jumping to the target

Unlocker



- **On Unlocked:** Event to trigger when we unlock.
- **Resource Count Text:** Text to update when we give resource.
- **Needed Resource:** Needed resource for transaction.
- **Spending Ease:** Which ease you want to use for transaction. Linear ease will pay in a steady pace where for example InQuint will increase its speed gradually.
- **Spending Speed:** This will determine the time it takes to complete transaction.
- **Required Resource:** Resource it needs to complete transaction

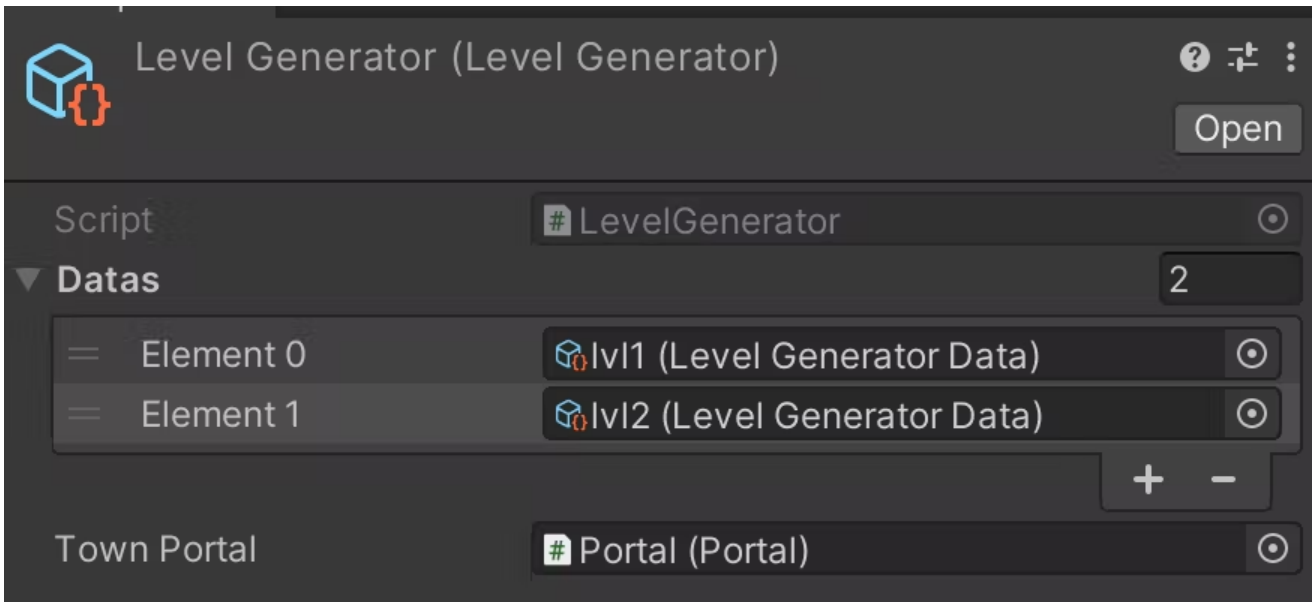
UI Joystick



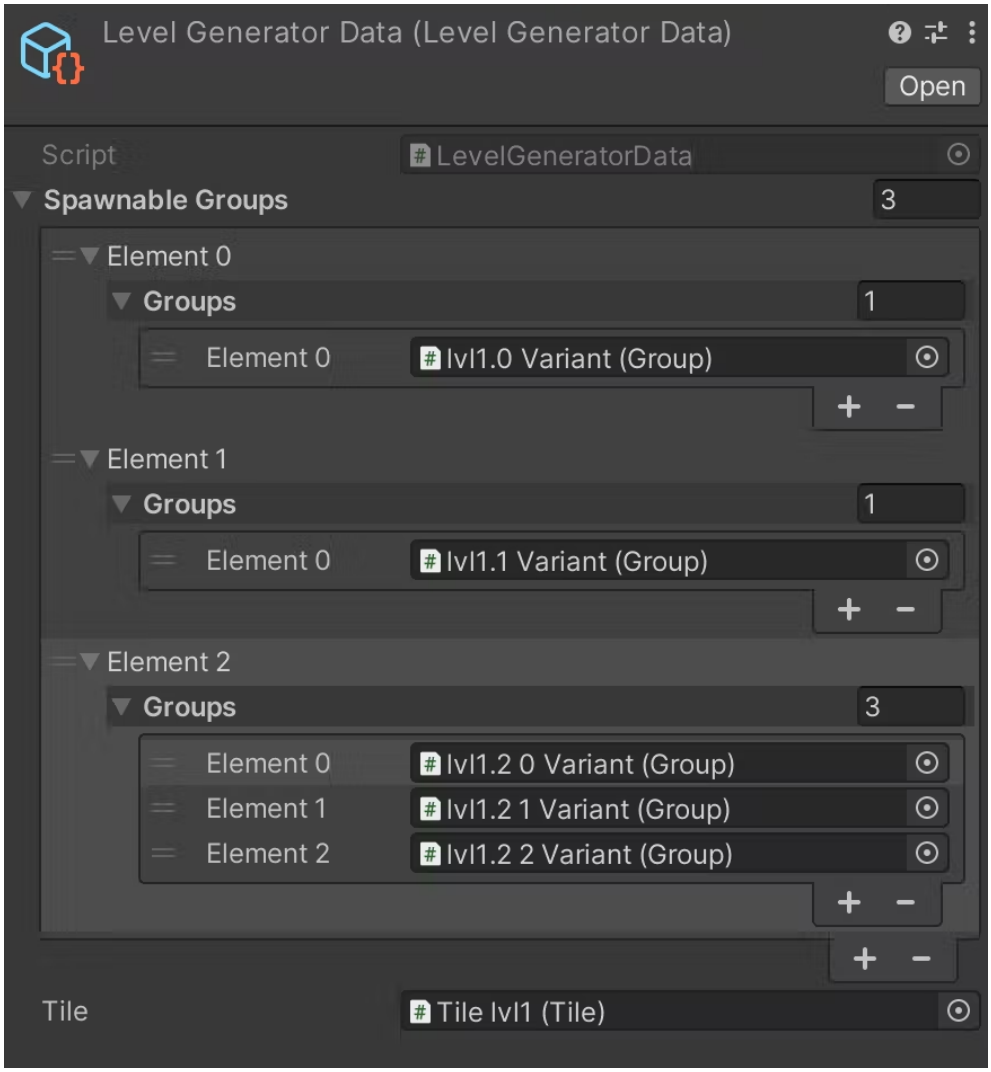
- **Background:** Background of the joystick, generally a white circle.
- **Input Channel SO:** Middle layer (like mediator pattern) where we send input values through.
- **Pressed Opacity:** Opacity level of the joystick after we pressed the joystick.
- **Max Range:** Max range of joystick dragging from its center.
- **Horizontal Axis Enabled:** Whether joystick works horizontally.
- **Vertical Axis Enabled:** Whether joystick works vertically.
- **Is Center Dynamic:** Is center of the joystick moves with the joystick.

Level Generator

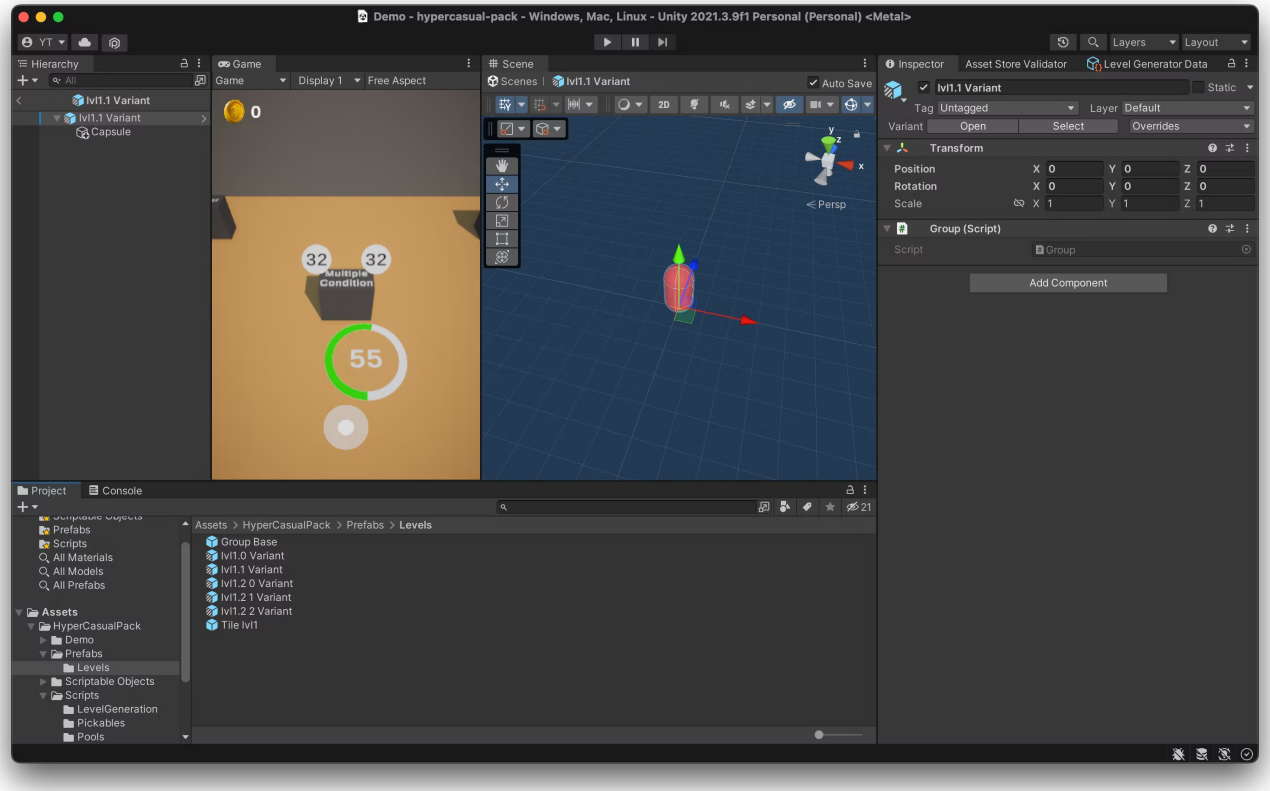
Level Generator system contains Level Generator, Level Generator Data, Group, Tile, Portal. And you need to have Level Generation Trigger for triggering new level generation. It's in there for example and you can implement your own trigger. It's pretty straightforward.



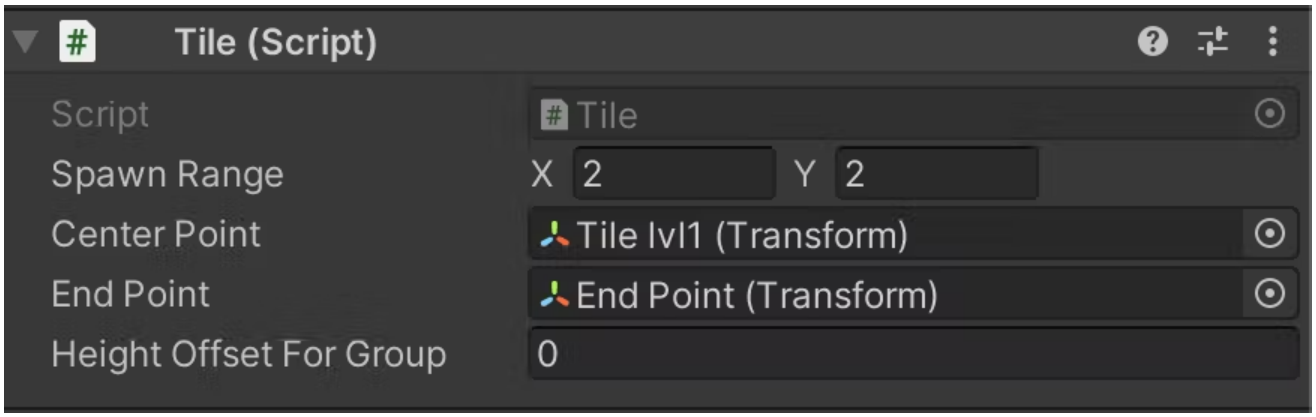
- **Level Generator:** Contains all methods for level generation. It also contains Level Generator Datas, for every level, so you can control what will be spawned for every new level.



- **Level Generator Data:** It takes Group, which you can assign to the prefabs, then you can use those to create levels. Order is important, system uses the order for difficulty. You can add variations by adding new Group to the elements so level generator spawns randomized.



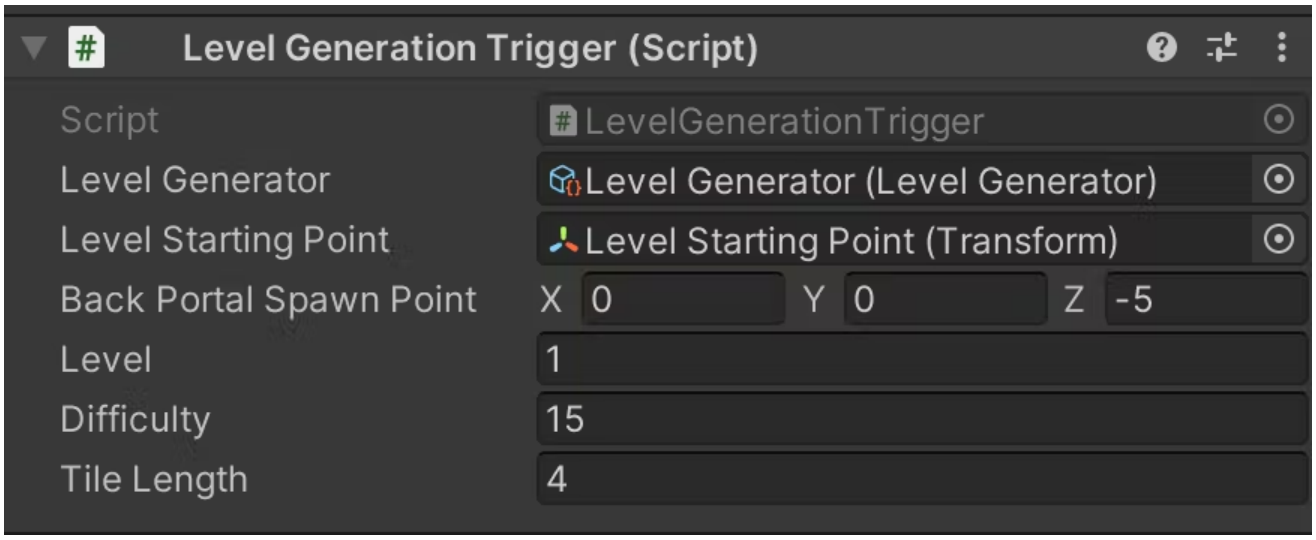
- **Group:** This works like a tagging, you add this script to the prefabs, so you can assign them to Level Generator Data.



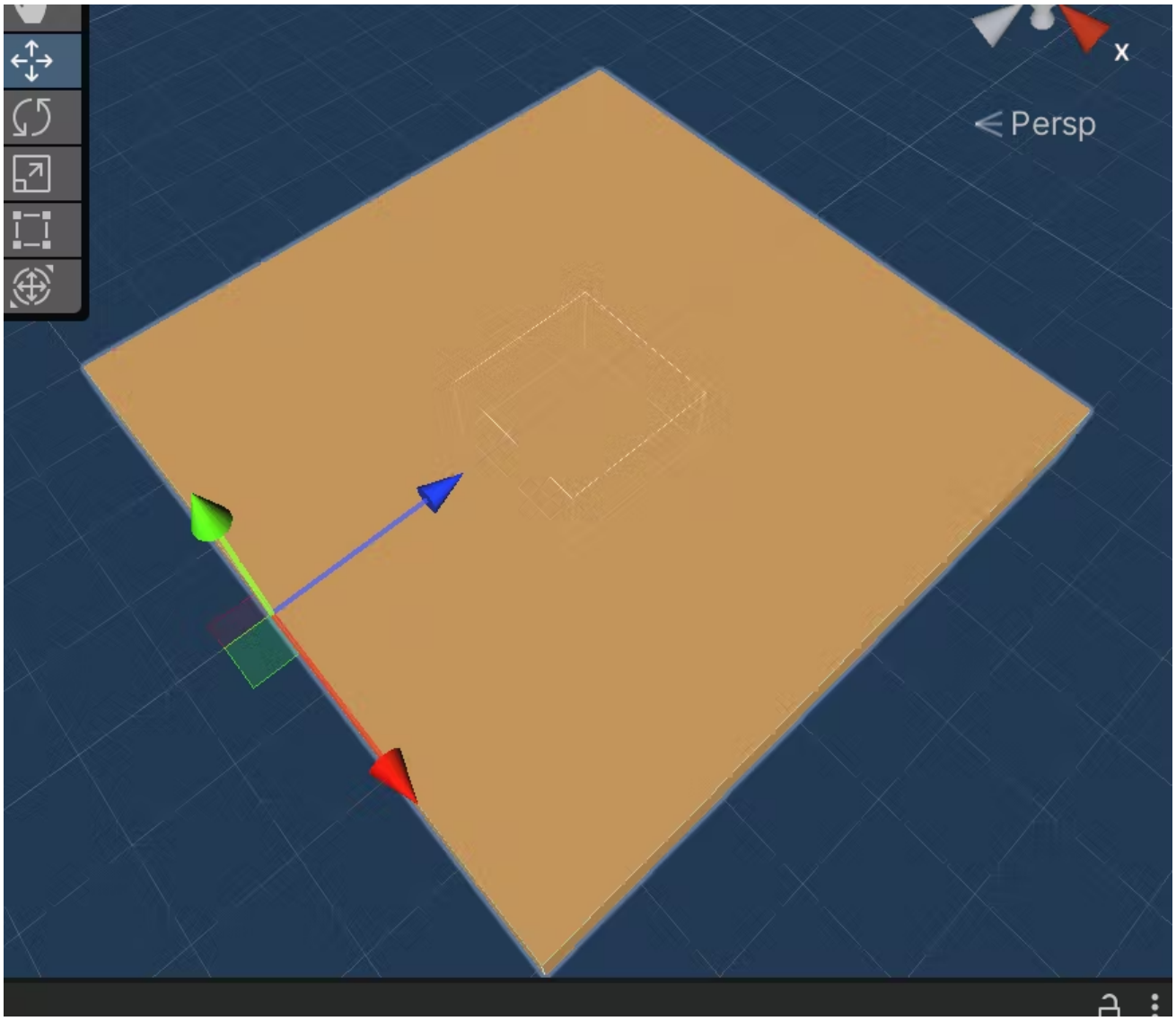
- **Tile:** This also works like a tagging. Although, it has few properties that allow you customize its behavior.

  -*Spawn Range:* Group objects will spawn based on this randomness area. If you don't want randomness, change it to 0,0
  -*Center Point:* Spawned objects will use this pivot for spawn position
  -*End Point:* Point to use when instantiating new adjacent tile, position it to the end of the platform.
  -*Height Offset For Group:* Group objects will spawn with this y offset

---



- **Level Generation Trigger:** Triggers Level Generator so it can generate new level for us.
- *Level Starting Point*: Level Generator will spawn the first tile in here. Take note that you need to offset Tile so it's pivot is on the starting point of tile like the image below.



- *Back Portal Spawn Point:* Offset to add when spawning back portal, so you can destroy the spawned level. It will position itself based on the level starting point.
- *Level:* Which level to spawn, Level Generator has Level Generator Datas for every level, first level is level 0.
- *Difficulty:* Based on the difficulty, generator will generate high numbers in the ordered list of Level Generator Data.
- *Tile Length:* Tile count to spawn. Note that as you decrease length, generator will try to spawn highest difficulty tiles, so it can fulfill your difficulty request.

Pools

```
public class PickableResourcePoolerSO : ObjectPoolerSO<PickableResource>
```

In order to create new pool, just inherit from ObjectPoolerSO with type. Don't forget to add CreateAsset attribute. Then you can create new asset by right clicking in the project window.

```
Pickable pickable = pickableResource.TakeFromPool();
```

You can get the item from pool like this.

```
pickableResource.PutBackToPool(nonPickableResource);
```

And you can put it back to pool like this.

For example, this is one of the pools in the pack. You can configure it with Max Size and Behaviour. Max size allows you to specify size for your pool, if pooled object exceeds this limit, they will going to be destroyed instead of putting back to pool. Behaviour allows you to specify the object you want to use it with pool.

,