Unity Developer Trial Task

Pierre NOËL

Platform Editor

During this test, my first approach was to go for the environment scene where the users can:

- Navigate in the 3D environment
- Select on a list of assets preloaded
- Apply transformations on placed items
- Undo any transformation actions
- Save their creations

Navigate in the 3D environment

I decided to not rely on a third-party plugin to manage the inputs (on a device and in the editor). I created the ***InputTouch*** static class to allow me to simulate the touches directly on the editor game view and avoid any different behaviors from editor to device. In other words: Simulating the pinch rotate/zoom with two fingers (by using left control) and not relying on the mouse scroller.

Selecting assets data

Inspired by Struckd, I decided to go for a shortcut list of assets that the user can access quickly from the bottom of the screen. The “+” button allows the user to select from a more complete list of assets that can be toggled and displayed on the shortcut list.

The data (***AssetData***) represents a single asset with a prefab and a linked string id. It is loaded through ***IAssetLoaderBehaviour*** which the implementation can be changed for another usage (Addressable, direct download, streaming assets, etc.)

AssetData are grouped by DataSet (***AssetDataListScriptableObject***). For instance: Dataset of trees, of animals, of rocks, etc.

Applying transformations on items

The selection of items is handled by ***ObjectSelectionTransform*** which has 3 states: No selection, Data selected ready to spawn and item selected.

When an item is selected, ***SelectionTransformHelper*** is handling Position, Rotation and Scale. Placing an item can be done by dragging with one touch, rotation and scale can be performed by using the UI and dragging the finger on the screen. The UI remains on top of the item.

Undo actions

Each transform function is applied with an ***IAction*** that is stacked on ***ActionRequestHandler***. ***IAction*** can be undone and unstacked through the undo button. All actions created for this test:

- **CreateAction**
- **DeleteAction** (Which is a soft delete action that can be undone)
- **TranslateAction**
- **RotateAction**
- **ScaleAction**

Saving the scene

The scene is saving all items as JSON objects along with their position, rotation, scale, assetId. The scene has an additional name and file path saved to ease the reading and UI display. **DataManager** is saving/loading a specific scene when the scene starts. Native **JSONUtility** from Unity was used for JSON serialization.

Passing data between scenes

Playerprefs were used to pass data from one to another and avoid using DontDestroyOnLoad Gameobject (done on **AppManager**)

Creating basic assets

An editor window was created to automate the generation of Dataset to be loaded on the environment. It can be accessed with *Tools > PrefabCreator*.
A list of models or prefabs can be provided, the default prefab to use the ***RuntimeItem*** behavior, and finally, a name for the data set.

The development of this feature is not optimal and fully efficient, but it allows me to quickly create a few data sets of data to use in the app without creating every single prefab.

3rd party assets

In this test I used 2 assets from the Asset Store to help me complete the task:

- Low Poly ultimate Pack (3D model of all assets)
- Safe Area Helper (Simple tool to deal with notchs on devices)

Grass texture from textures.com and some UI sprite from flaticon.

Conclusion and general thoughts

The whole development of this test took me around 20 hours. I spent a lot of time focusing on the navigation and controls at the beginning to have something that can be tested on every feature addition.

Some features that would be missing and be considered a priority if this project was going to scale :

- Create a setting scriptable object to handle all the single float/Vector2/Vector3 settings that are serialized on the inspector.
- Handle an async loading of the scene to avoid frame freezes with big scenes.
- Use a pooling system for any instantiation regarding UI.
- Improve rotation and position to have the option to snap to vertices.
- Improve colliders of items to be more accurate than Boxcollider for raycasts and placement.
- Improve position placement to add the ability to choose on the Y-Axis.
- Miscellaneous features customizations (Change ground floor, post-processing effects, AR)