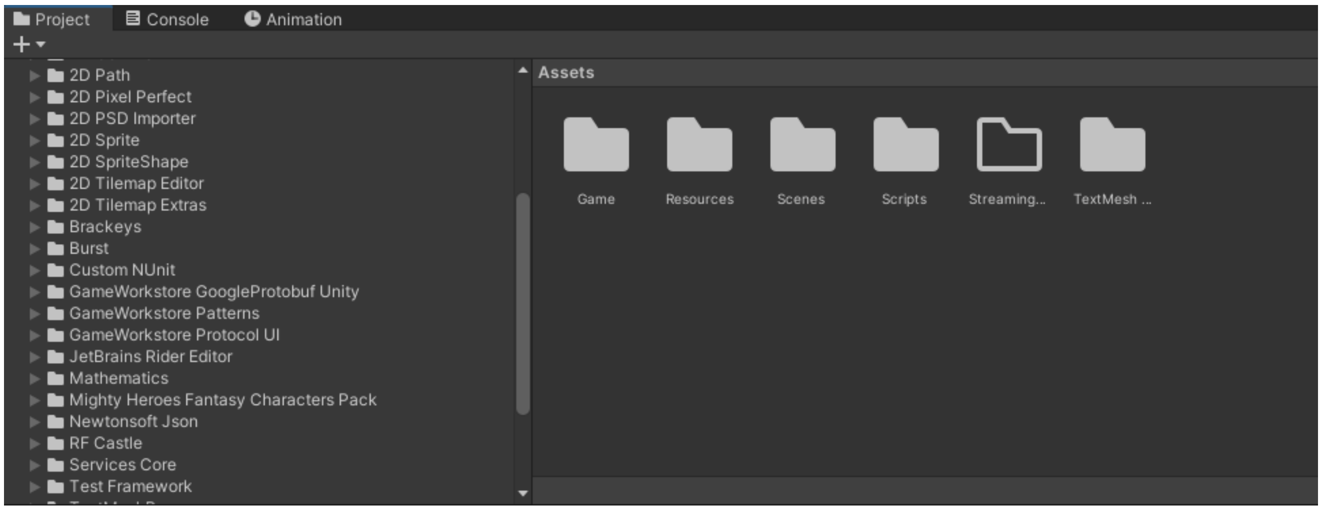
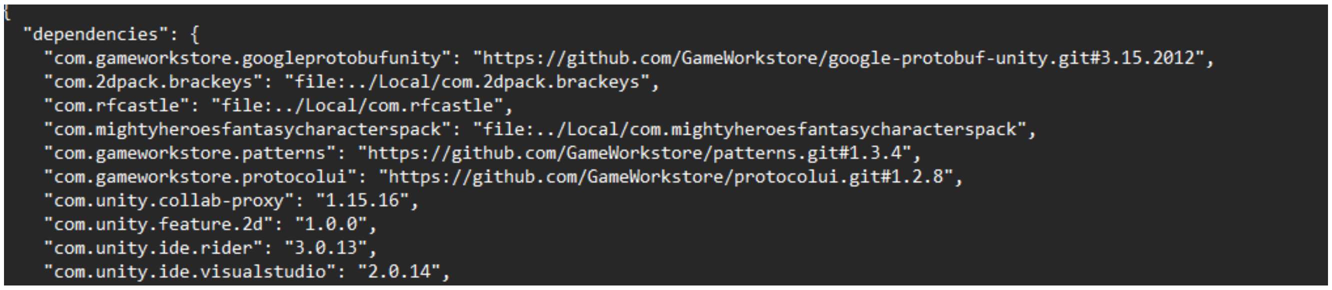
To start the test production, the first thing I did was analyze what needed to be done and create a sequence for executing the necessary tasks.

The initial step in the project is to import all the packages that will be needed to complete it. For this, I prefer to use a technique different from simply placing the files in the "Assets" folder of Unity, as over time, this can make the project disorganized. That's why I convert the packages into libraries, similar to how it's done with Unity's internal APIs.



To do this, I directly use the .package file from Unity to simulate that my assets are internal packages.



```
{
  "dependencies": {
    "com.gameworkstore.googleprotobufunity": "https://github.com/GameWorkstore/google-protobuf-unity.git#3.15.2012",
    "com.2dpack.brackeys": "file:../Local/com.2dpack.brackeys",
    "com.rfcastle": "file:../Local/com.rfcastle",
    "com.mightyheroesfantasycharacterspack": "file:../Local/com.mightyheroesfantasycharacterspack",
    "com.gameworkstore.patterns": "https://github.com/GameWorkstore/patterns.git#1.3.4",
    "com.gameworkstore.protocolui": "https://github.com/GameWorkstore/protocolui.git#1.2.8",
    "com.unity.collab-proxy": "1.15.16",
    "com.unity.feature.2d": "1.0.0",
    "com.unity.ide.rider": "3.0.13",
    "com.unity.ide.visualstudio": "2.0.14",
```

The packages I used were:

- GameWorkstore/google-protobuf-unity: Unity NPM version of Google Protobuf! This repository applies the same license terms as the original version. (github.com)

(This is an excellent plugin to use Google Protobufs in the Unity project. This is very useful when you need to create game saves, as it is not only extremely lightweight but also can be read by any programming language (in case there's a backend in operation). Additionally, it's automatically converted when you need to send it to Firebase.)

- Free 2D Mega Pack | 2D | Unity Asset Store
(Art package, used solely for creating the shop vendor.)

- Rogue Fantasy Castle | 2D Environments | Unity Asset Store
(Art package used for assembling map tiles.)

- Mighty Heroes (Rogue) 2D Fantasy Characters Pack | 2D Characters | Unity Asset Store
(Art package used for character artwork.)

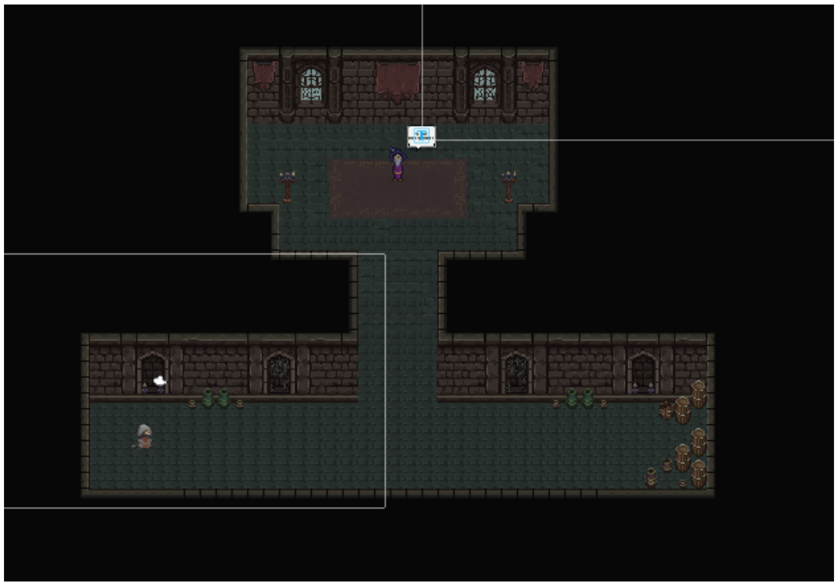
- GameWorkstore/patterns: Patterns Library for Unity Projects or C# projects (github.com)

(This is a plugin with utilities for Unity, providing a set of action systems, arrays, and other efficient and optimized functions beyond the standard Unity offerings. I often use this package extensively.)

GameWorkstore/protocolui: Protocol UI is a small framework to work with multiple screens on a easy and organized way and archive layered panels on Unity. (github.com)

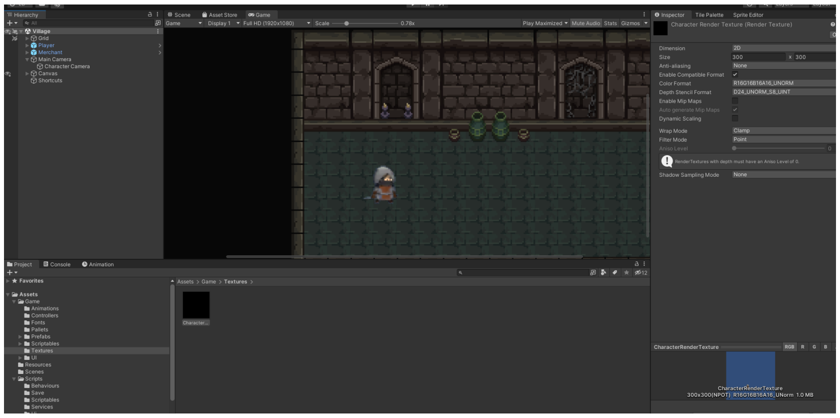
(This package is a UI assembly workflow.)

After configuring the project, I set up a small scene so that I could begin implementing programming.



At the beginning of the project, I decided to use the character that was provided to me, which was this character:

However, it stood out significantly from the environment due to not being pixel art. Hence, I devised a small "scheme." To give it a "pixel art" appearance, I employ two cameras in-game. One camera renders the entire scene except for the character, and the other one renders only the character. This camera doesn't have an active view in the scene; its view is directed to a texture that modifies its perspective to a lower resolution than the actual scene, allowing me to edit the character's resolution.



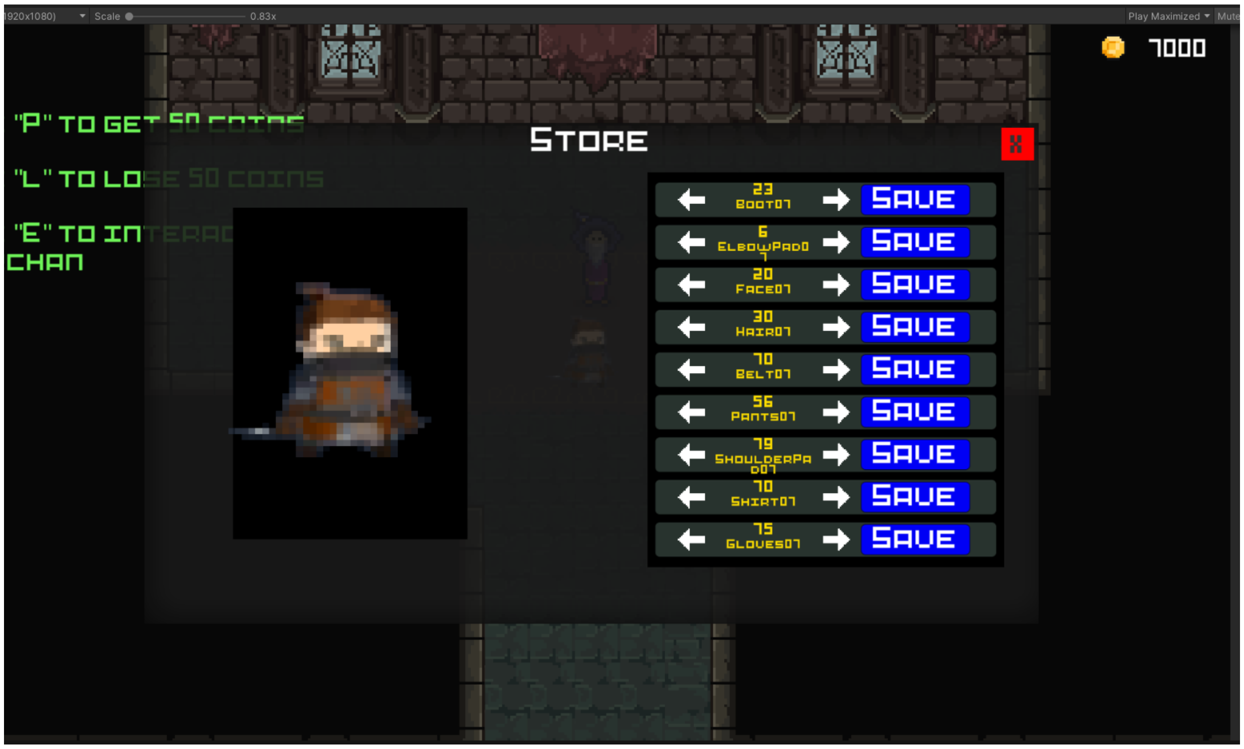
Character Render Texture (Render Texture)

Dimension 2D
Size 300 x 300
Anti-aliasing None
Enable Compatible Format ✓
Color Format R16G16B16A16_UNORM
Depth Stencil Format D24_UNORM_S8_UINT
Enable Mip Maps
Auto generate Mip Maps ✓
Dynamic Scaling
Wrap Mode Clamp
Filter Mode Point
Aniso Level 0
RenderTextures with depth must have an Aniso Level of 0.
Shadow Sampling Mode None

CharacterRenderTexture
300x300(NPOT) R16G16B16A16_UNorm 1.0 MB

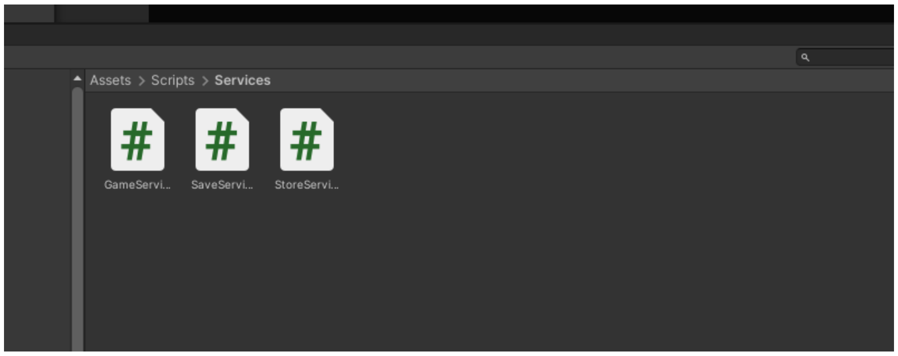
With the character and the map ready, I proceeded to implement character movement, which is quite straightforward. I adjust the character's physical velocity and move them through the scene using player inputs along with a speed modifier.

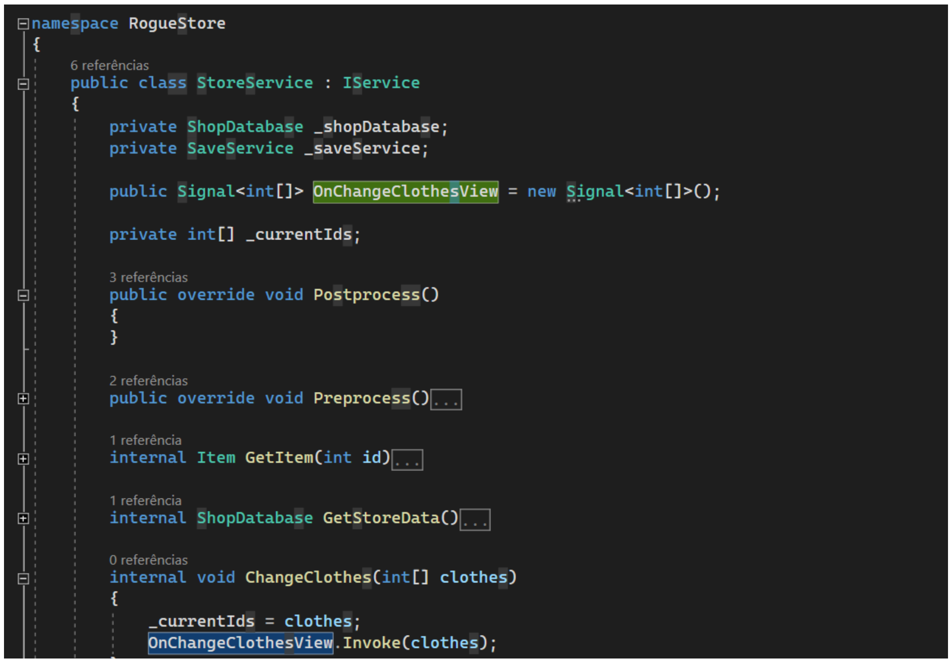
Interaction with the shopkeeper occurs through a check around the character for objects that are interactive and inherit from the abstract class "Interactable." If there's something interactable nearby, I trigger the interaction, which in this case involves opening the shop.



In addition to implementing the shop functionality, I also introduced a save system in the game. When the player purchases a skin, it's saved in a Protobuf format that gets deserialized into PlayerPrefs. This way, players can change and purchase skins for all available body parts (a total of 9).

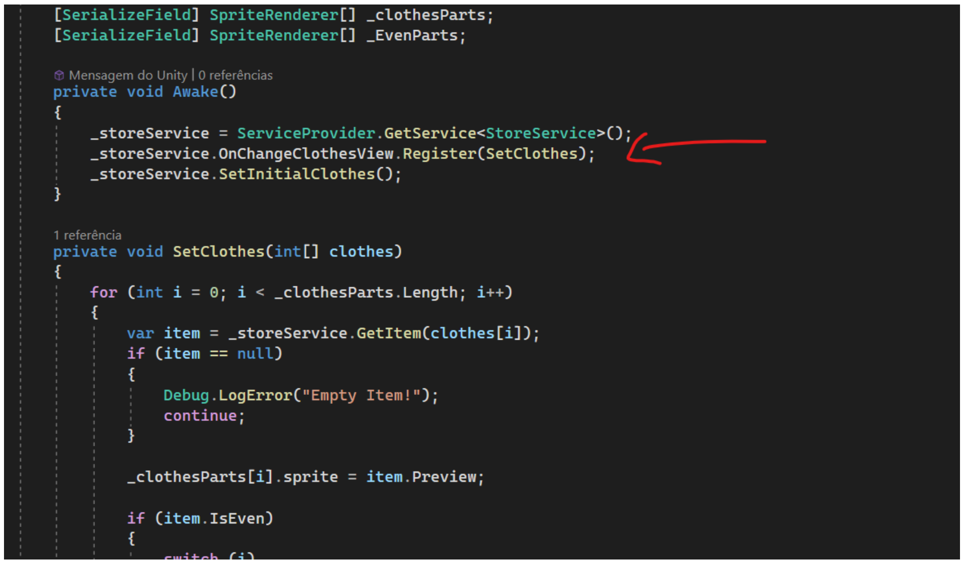
To make this system work, I use services, which are classes that can be called even before Unity's Awake method is triggered. These services can create actions that can be listened to by several other classes within the project.





```
namespace RogueStore
{
    6 referências
    public class StoreService : IService
    {
        private ShopDatabase _shopDatabase;
        private SaveService _saveService;

        public Signal<int[]> OnChangeClothesView = new Signal<int[]>();

        private int[] _currentIds;

        3 referências
        public override void Postprocess()
        {
        }

        2 referências
        public override void Preprocess()...

        1 referência
        internal Item GetItem(int id)...

        1 referência
        internal ShopDatabase GetStoreData()...

        0 referências
        internal void ChangeClothes(int[] clothes)
        {
            _currentIds = clothes;
            OnChangeClothesView.Invoke(clothes);
```

The StoreService is a prime example of this. It notifies all other classes when the player changes clothes in the shop using signals. It sends an array of hashes containing the modifications made to the player's clothing. For instance, the "CharacterClothesSetup" class registers a function, waiting for a call to place the clothing on the player when needed.



```
    [SerializeField] SpriteRenderer[] _clothesParts;
    [SerializeField] SpriteRenderer[] _EvenParts;

    Mensagem do Unity | 0 referências
    private void Awake()
    {
        _storeService = ServiceProvider.GetService<StoreService>();
        _storeService.OnChangeClothesView.Register(SetClothes);
        _storeService.SetInitialClothes();
    }

    1 referência
    private void SetClothes(int[] clothes)
    {
        for (int i = 0; i < _clothesParts.Length; i++)
        {
            var item = _storeService.GetItem(clothes[i]);
            if (item == null)
            {
                Debug.LogError("Empty Item!");
                continue;
            }

            _clothesParts[i].sprite = item.Preview;

            if (item.IsEven)
            {
```

My final thoughts on the project are that I really enjoyed creating it to test my speed and project organization skills, and to enhance my workflow. I believe I've made solid implementations, but I didn't have enough time to polish and make it truly "visually appealing." Nevertheless, I hope I've been able to convey my approach to working on the project.

I'm grateful for the opportunity and look forward to any updates!

Lucas Brum