SimWold Tech Test

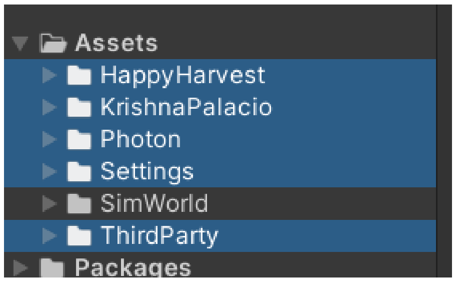
The systems

As the technical proof is incomplete there is more to talk about the thought process rather than the systems themselves.
Anyway I will explain the initial flow and what the systems were intended for even though the systems are incomplete. Just before that, I will mention the third party packages that you should not pay too much attention to as I hardly used them.

Third party assets

Each selected folder in the image is part of third party asset packs

- HappyHarvest: Unity example for a game like Strawdey Valley
- KrishnaPalacio: Sprites and animations pack for a pixel art dungeon game
- Photon: SDK for multiplayer games
- Settings: Settings folder used by HappyHarvest
- ThirdParty/SceneReference: SceneReference is a single class, a wrapper that provides the means to safely serialize Scene Asset References

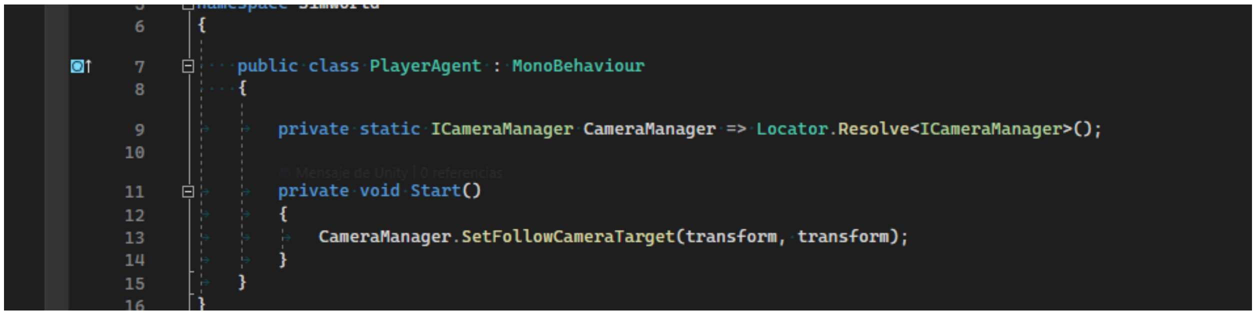


Systems: Managers and Locator

As usual, there are "Managers" classes in the game code. So far the managers that exist in the game are:

- NavigationManager: Responsible for the navigation between scenes, transition animations and loading screen.
- CamaraManager: Access point to get the camera that is currently rendering the scene, also contains logic related to switching cameras between spectator cameras (default) and character tracking cameras.
- OverlayManager: Contains logic to start, show and hide "overlays", to summarize its logic we can think that it works as a UI Stack. Basically it serves to accumulate UI.

- DialogManager: Exactly the same as the OverlayManager, but instead of being a UI Stack it is a UI Queue. Basically it serves for UI queues, useful for dialogs that sometimes come one after the other.
  The managers are initialized by the ManagersInitilizer class, and then they are registered in the Locator. The Locator is simply a dictionary of managers in memory, used to avoid abuse of the Singleton pattern in managers.



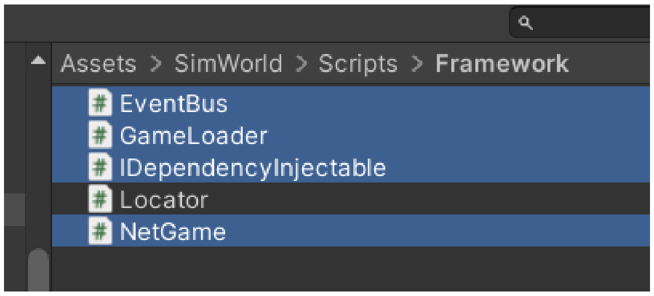
```
{
    public class PlayerAgent : MonoBehaviour
    {
        private static ICameraManager CameraManager => Locator.Resolve<ICameraManager>();

        private void Start()
        {
            CameraManager.SetFollowCameraTarget(transform, transform);
        }
    }
}
```

Systems: Others

Some scripts of the project finally did not have a very important use, and some had no use (but will have it soon). These are:

- EventBus: I'm a big fan of the EventBus, so I was planning to implement it, but so far it wasn't a priority.
- GameLoader: Simply loads the main menu when the game starts at scene 0.
- IDependencyInjectable: Currently used by managers for initialization, although it is part of good practices it is not very common to need it in videogames (at least in indie videogames).
- NetGame: When trying to make the online game it was one of the first classes that I created, it has commented code related to PhotonBR.



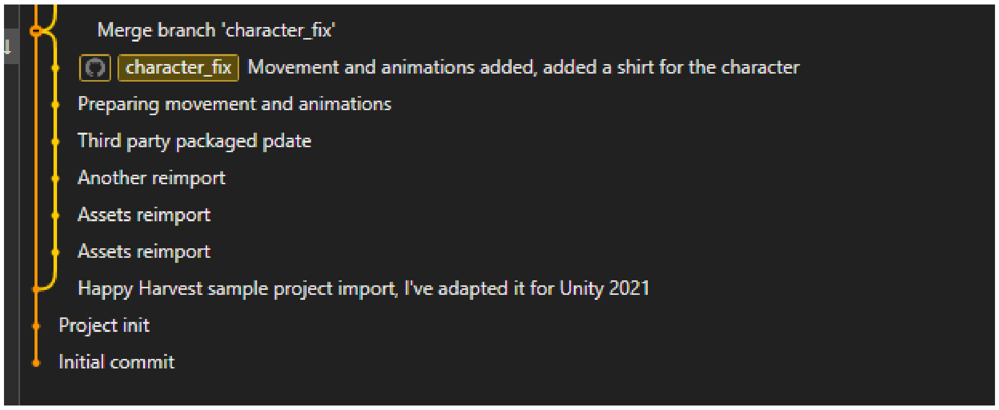
The rest of the scripts in the project are very simple and the name of their classes just explains what the scripts do.

Thought process

At the beginning of the project, considering the short time available, I preferred not to "reinvent the wheel" in terms of creating a game similar to Stradew Valley. So I started looking for resources that I could reuse for the game I was asked to create. I found the Happy Harvest Unity sample project and it

seemed perfect, the problem was that the minimum version of Unity supported by Happy Harvest is 2022.x, and you explicitly asked to use Unity 2021.3.21.
Anyway the example project was too useful so I imported it anyway to Unity 2021 and fixed the compile errors I got, since the project was prepared for Unity 2022. After the fixes everything worked fine except for one thing, the character animation and its prefab.
The character prefab used as main asset a PSD file that in Unity 2022 could be imported without problems thanks to one of the latest versions of the PSD Importer package, unfortunately Unity 2021 is not fully compatible with the latest versions of that package. So finally I had everything I needed except the most important part of the game; the character model.

I thought about alternatives... One alternative was to look for character assets with sprites that could be adapted to a SkinnedMesh for animations, another alternative was to look for character assets with animations based on a set of frames. Regardless of the alternative, I was going to have to look for environment sprites that have very similar artwork to the new character sprite artwork, so the Happy Harvest sprites were going to be of virtually no use to me.
Since it was going to take time to set up egg-based animations, I preferred to look for characters with frame-based animations and finally chose a pixel art style asset pack.
This explanation summarizes what happened in the first 10 commits of the project...

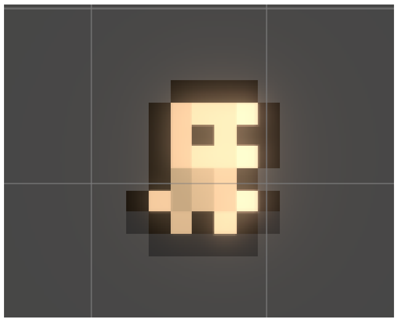


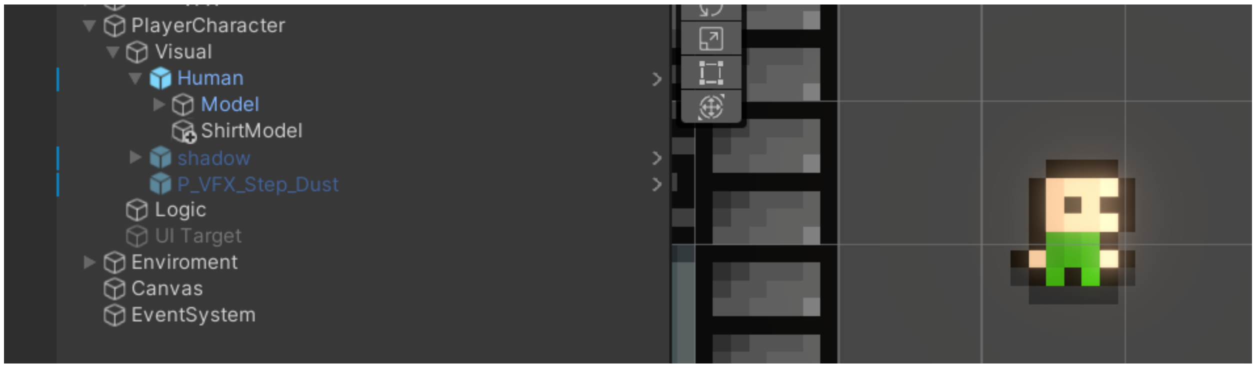
Merge branch 'character_fix'
character_fix Movement and animations added, added a shirt for the character
Preparing movement and animations
Third party packaged pdate
Another reimport
Assets reimport
Assets reimport
Happy Harvest sample project import, I've adapted it for Unity 2021
Project init
Initial commit

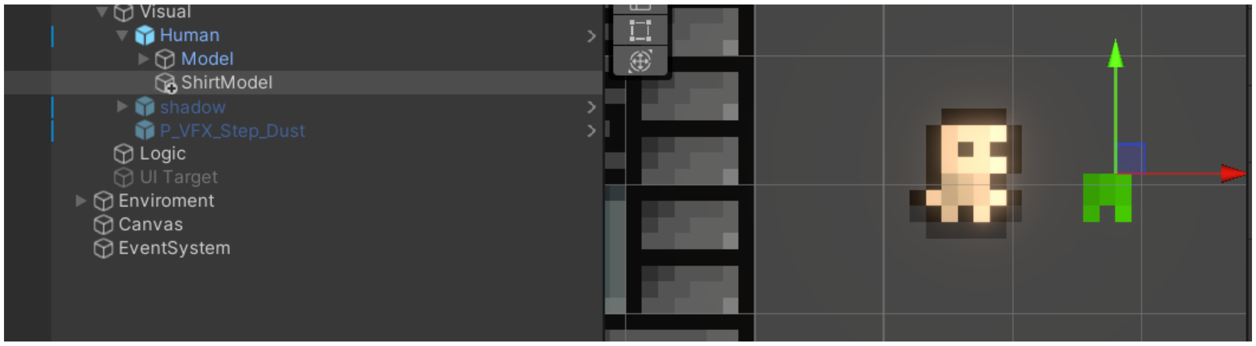
The problem with choosing the frame-based animation is that I would have problems when trying to equip sprites of other items, such as a shirt or a hat, since I would have to have a different set of frames for each item that the player could have equipped. That is, if the animation of the running character has 20 frames and the character can have equipped or unequipped a hat or a shirt, then I need different animations for the following possibilities:

- Character with no equipment
- Character with shirt
- Character with shirt and hat
- Character with shirt but without hat
- Character with hat but no shirt
  This would give a total of 5 different running animations, finally I would need 100 frames.
  The pixel art of the art is simple but I preferred not to waste time creating 80 new frames for each

animation (since the asset pack I got didn't have these animations, it only had animations of a character without equipment). So I thought of a trick to save work, instead of having a single SpriteRenderer for all the character animations, I would have additional SpriteRenderers only for the additional items (hats or shirts) that the character might have equipped, these additional SpriteRenderers would have their own animations to separate the animation of the clothes from the animation of the character's body.

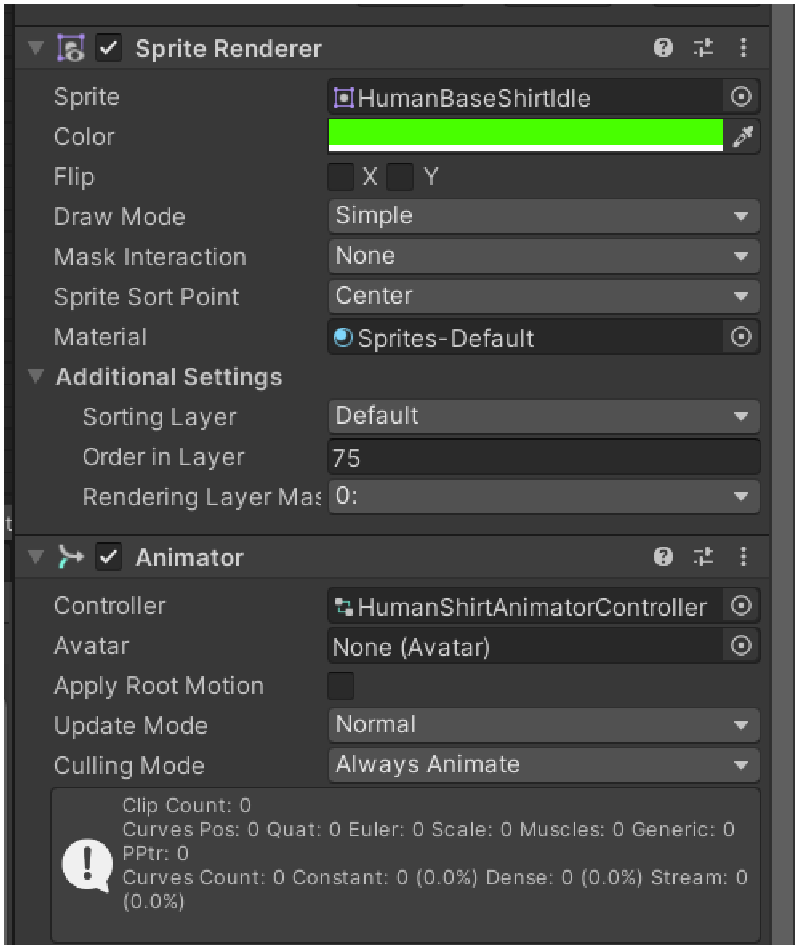






This way I was able to save a lot of work related to the animations, without complicating a lot the logic related to the character's equipment. Since if you wanted to equip a shirt you only have to activate the SpriteRenderer inside ShirtModel. And it would also be possible to have different color variations of the same item and sell them as separate items in the store.

If it would be necessary to add a shirt with a different pixel design it would be enough to create its animation and then change the Animator.Controller field in the ShirtModel object.



Sprite Renderer

Sprite	HumanBaseShirtIdle
Color	🟩
Flip	X Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	Sprites-Default

Additional Settings

Sorting Layer	Default
Order in Layer	75
Rendering Layer Mas	0:

Animator

Controller	HumanShirtAnimatorController
Avatar	None (Avatar)
Apply Root Motion	
Update Mode	Normal
Culling Mode	Always Animate

```
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0
PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0
(0.0%)
```

During the rest of the development I had no major problems, the next steps were to add Photon since I was planning to make the game online. Besides adding logic related to the UI, game flow and managers.

Personal assessment

Since I didn't complete the game on time organization is one of the things I should mention in my personal assessment, although there isn't much to mention about it.
I thought that this weekend I would have more free time and could focus only on the technical test but in the end that was not the case for reasons outside of this work. So I should have chosen a better day to start.

On the technical side, I think the choice to use frame-based animation instead of bones was not entirely correct, maybe it would work in this little technical test but it's certainly not what I would use in a real game, it's just not something that scales well over time. Bone-based animation would be the best option.