Vertical Space Shooter

Technical Summary

This technical test was done using Unity **2019.4.14f1** the last version of Unity's **LTS** version at the moment of the start, also was sugin **Addressables** and **Text Mesh Pro**, all from Unity's package manager.
Art assets used in the project were from Kenney's free assets.

To play the game in Unity editor, the Init Scene must be loaded, as that is where the **ManagerProvider** and the **SplashLoader** logic are present.
The **SplashLoader** instantiates all the Managers within the game initialize them through the **ManagerProvider**. Each manager registered to the ManagerProvider with certain priority which is used at the moment of initialization.

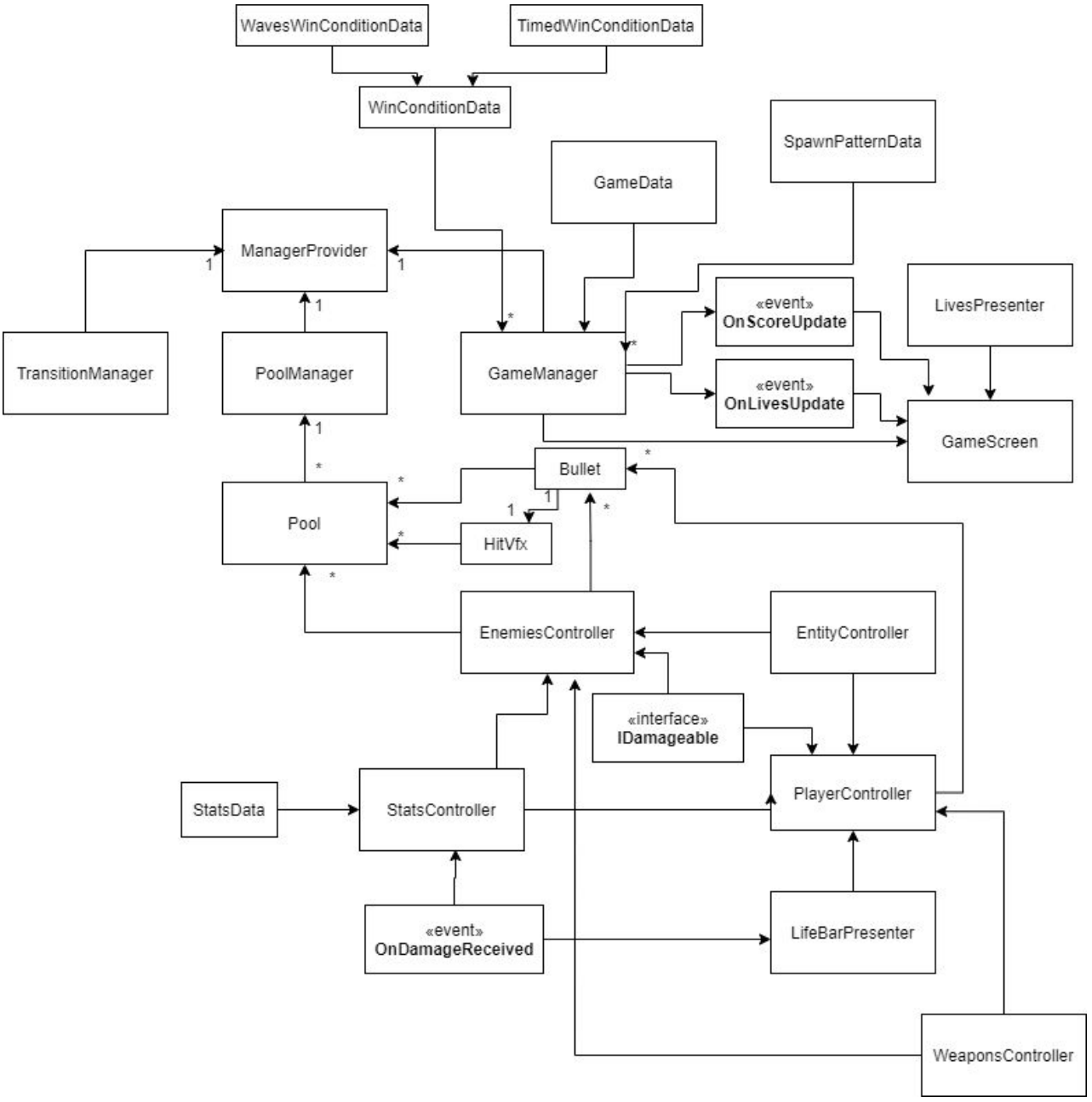
After the initialization process is done, the gameplay scene (called **Level1**) is loaded and the GameManager is responsible for the Enemies Spawn, Player Spawn and Wave control of enemies. I would like to add more encapsulation on the Wave System and make it a State Machine on its own, so the control would be more precise and easy to use but due time constraints and prioritizations i decided to keep it simply to show some gameplay.
For this demo, I just created 1 type of enemy and I was looking to add more using OOP, things that i was not able to achieve due time constraints and prioritization but the intention was in there.

I put efforts on creating the PoolSystem that allow me to use Pooling principles which are essential in the Game Development process, the objects in this Pool are mainly Enemies, Bullets and VFX. The pool is configured in the prefab called **PoolManager** that can be found in the Prefab's folder.

Bullet are using a Unity's method call **OnBecameVisible** and **OnBecameInvisible** that are being called by the engine when a GameObject is being display by a camera or not, the downside is that this doesn't work in Editor and only works on executables, this is due the Editor's cameras always looking into the gameobjects. More about this can be found in Unity's documentation (link), for all these it did some tweaks and added a Coroutine that Despawn the objects and send them back to the pool after a lifespan in seconds, which is configurable in the editor.

The GameManager is constantly checking if the game is done, for that is using the WinConditionData, an abstract ScriptableObject that check if the winCondition is meet, that ways the Conditions to win the game is very flexible and at this point, 2 are already implemented, one that check for Waves Completeds and one that check for Time Passed. This can be very escalable and I'm very happy with it.

Almost all the game data necessary for the game is configured in a ScriptableObject called **GameData** and can be accessed through the **GameManager.**


WavesWinConditionData
TimedWinConditionData
WinConditionData
SpawnPatternData
GameData
ManagerProvider
1
1
1
*
*
«event»
OnScoreUpdate
LivesPresenter
TransitionManager
PoolManager
GameManager
«event»
OnLivesUpdate
1
*
GameScreen
*
Bullet
*
1
1
*
Pool
*
HitVfx
*
*
EnemiesController
EntityController
«interface»
IDamageable
StatsData
StatsController
PlayerController
«event»
OnDamageReceived
LifeBarPresenter
WeaponsController