Vertical Scroller

Technical summary

The technical test was implemented with Unity 2018.4.18f1 using Addressables and Text Mesh Pro from Unity's package manager.
Art and Audio assets used in the project came from Kenney's free assets.

To play the game in Unity editor, the BootScene must be loaded as here is where the **ManagerProvider** and **GameBoot** logic are present and the **EventManager** is initialized.
The ManagerProvider makes sure to initialize all of the manager services that are registered to it. It works as a Service Provider.

After the bootup process is done, the gameplay scene is loaded. Here the **SceneController** starts the **WaveSequencer,** that is setup to use the Level1 wave configuration. The player ship prefab is instantiated and the WaveSequencer requests spawn points from the **SpawnerManager** to spawn certain types of enemies at a given time according to the level configuration.

The wave sequencer is set up to cycle the Waves from the level config for demonstration purposes.

The Player keyboard input are WASD or Arrow keys for movement, Space key for shooting laser.

Some classes, like the **GameManager, Character** and **GameHUD,** implement the IEventListener interface and register themselves in the EventManager as a Listener for a specific event. When an event is triggered this is propagated through all of registered listeners.

Object pooling is used to reuse and recycle bullets and enemies. The **DynamicObjectPool** class is in charge of this. A gameobject is passed to create a configurable starting amount of instances of it. After that an object can be requested from the pool and if all are being used the pool instantiates a new one and adds it to the pool.

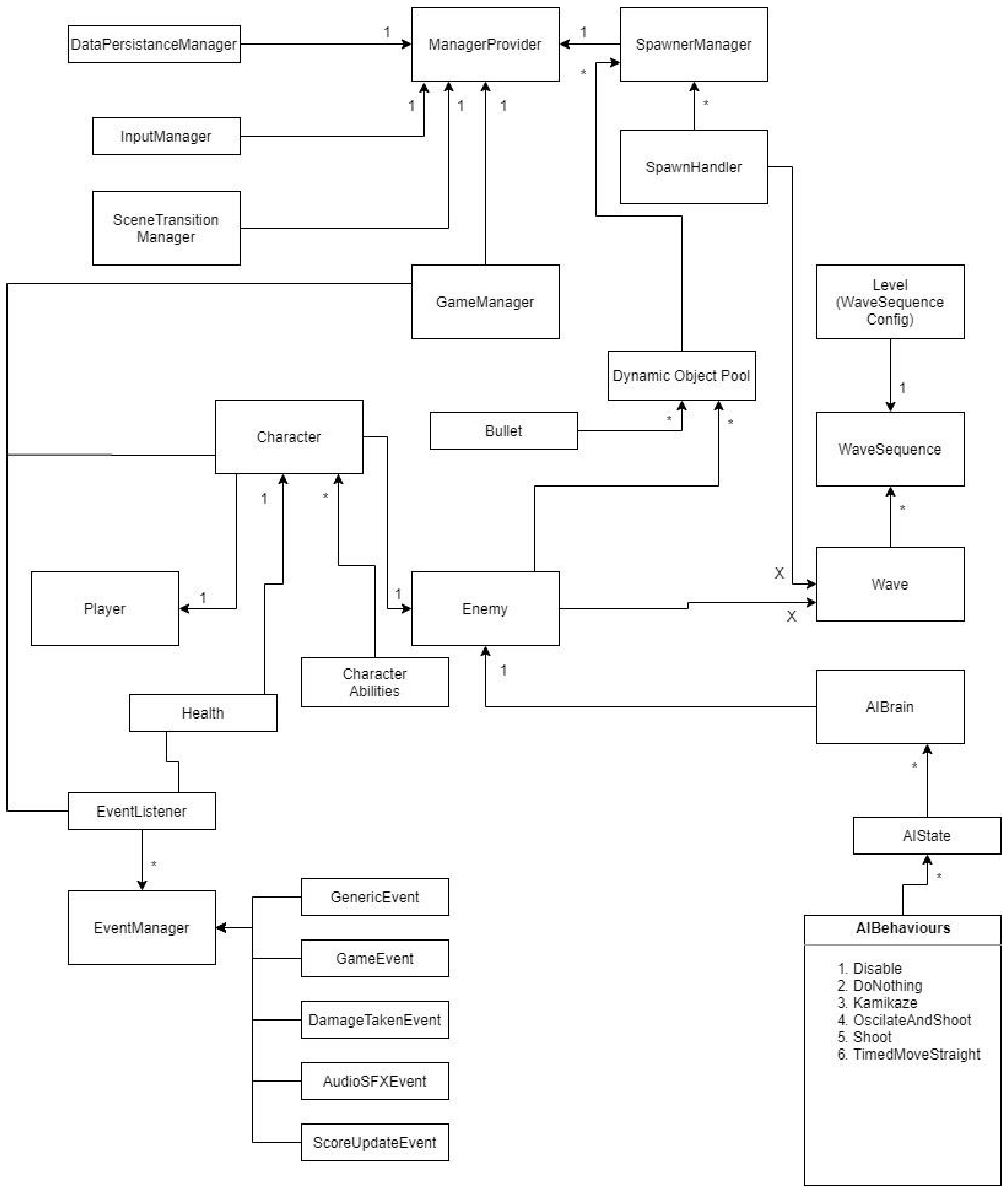
State machine is used in several types of implementation from quick and simplistic to more robust. The **AIBrain** contains **AIStates** that executes a set of specific behaviours and can transition between to other states. Currently there are 3 types of enemies with different configured behaviours. A kamikaze type, a light ship that shoots, and a heavy ship that remains until killed shoot at its target. The Wave Sequencer uses a static implementation using delegate methods and the current active state is a reference to a method.

Data persistence is done by writing a file serialized in binary format. Currently storing the player highscore.

A **winning condition** can be set up and customized by editing **MainGameConfig.** Currently there are 2 types of winning conditions besides “None” that are “Score” and “Waves Survived”.

<u>Score</u>: Set the win condition to count the player score, after reaching the amount to win the win screen is shown
<u>Wave Survived</u>: Set the win condition to count waves completed, these are triggered by an event, and after reaching amount to win the win screen is shown.

UML diagram


DataPersistanceManager
1
ManagerProvider
1
SpawnerManager
*
InputManager
1
1
1
*
SpawnHandler
SceneTransition Manager
GameManager
Level (WaveSequence Config)
Dynamic Object Pool
1
Character
Bullet
*
*
WaveSequence
1
*
*
Player
1
1
Enemy
X
Wave
X
Character Abilities
1
Health
AIBrain
*
EventListener
AIState
*
*
EventManager
GenericEvent
GameEvent
DamageTakenEvent
AudioSFXEvent
ScoreUpdateEvent
AIBehaviours
1. Disable
2. DoNothing
3. Kamikaze
4. OscilateAndShoot
5. Shoot
6. TimedMoveStraight