Side Scrolling Space Shooter Documentation

1. Summery

This pack provides a quality setup for a side scrolling shooter game and quick prototyping. The state of the prototype includes background music and sfx as well as examples of game polish such as detailed particle effects, animations and camera shake are included for reference.

Everything is setup and ready to use including a main menu, playable level and a game over menu.

1.1 Blueprints

This section will provide information on the main setup of the most important Blueprints within the project.

1.12 GameMode

BP_ShooterGameMode houses most of the work going on behind the scenes. Here you will find:

- Save/load functions
- The function to keep track of the game score
- A global camera shake event that can be called and passed unique values by any other Blueprint
- The IslandSpawner function to control the generation of background objects

1.13 Player

BP_Player is the main Blueprint for the player to control. It is the default pawn used for the BP_ShooterGameMode. The functionality for this Blueprint is covered in depth using the comments within. It currently has 3 different types of projectiles and more are planned to be added, each is called and managed inside of the BP_Player.

1.14 Enemy Spawner

BP_EnemySpawner controls the spawning of enemies and enemy waves. This is one of the more complex Blueprints in the project as it houses the logic to control the entire game flow. More information on this Blueprint can be found in the comments or as part of my overview video here: https://www.youtube.com/watch?v=tPV6tAY6CWY

1.2 Inherited Classes

To try and provided the greatest ease of use and efficiency for prototyping this project heavily implements inheritance between its classes. This means that any type of Blueprints which might share a number of traits can inherit those traits from a base class leaving you to only implement traits specific to your new Blueprint.

To create a new Blueprint which inherits from something else you simply right click on the base Blueprint and select “Create Child Blueprint class”.

1.21 Enemies

Each of the enemy types is a child of BP_EnemyBase and therefore inherits all of the components and variables inside of BP_EnemyBase.

To create your own new type of enemies this way, you will need to use the method in 1.2 to create a new child Blueprint. Once this is done your new enemy will have a number of variables to edit such as health, fire rate, movement speed as well as some default movement properties and damage system. All you should need to do is change the Static Mesh, tweak the values to your preference and you have a brand new enemy to add to your game.

1.22 Pickups

Similar to the enemies each type of pickup Blueprint is a child of BP_PickupBase and inherits its base properties. These include movement to make any pickup scroll across the screen and home towards the player when at a certain distance.

To create a new type of pickup simply follow the steps mentioned in the previous section and simply change your Static Mesh. The main difference here is that each unique pickup handles it’s own logic to declare that it’s been collected, to see how this is done check the “OnComponentBeginOverlap” section of any pickup for example the “BP_PickupCoin” and implement similar new functions in the BP_ShooterGameMode to add your new pickup functionalities.

1.23 Projectiles

The projectiles have been set up in two different ways to help demonstrate a couple of different but equally flexible systems for quick implementation of different types of projectiles in your game.

1.23.1 Enemy Projectile

BP_ProjectileEnemy is a single Blueprint that is passed in to be used for each type of enemy in the game. Each enemy that uses this Blueprint as a projectile then alters the scale and rotation when spawning them into the world.

Pro Tip: As well as being a great way to save time creating multiple assets for things that may not be a great point of interest to the player, this is also a performance boosting technique. Being able to call fewer unique assets into existence means that less needs to be processed during each draw call.

1.23.2 Player Projectile

Each type of player projectile uses the standard form of inheritance. Each projectile is a child of BP_ProjectilePlayer which contains variables to control the speed, damage & scale. As well as the “Projectile Movement” component to control the movement of each projectile and the functionality to detect collisions and apply damage.

To create your own projectile simply create a new child class of BP_ProjectilePlayer, change your Static Mesh and tweak the variables to your needs. Once again you have a brand new blueprint ready for use in your game after just a couple of easy steps.

1.3 Maps

The default map in the “Project Settings” should be “Menu”. This will start you on the main menu map. From here you are given the option to quit or play, pressing play will take you to the “Main” map which is where the gameplay takes place.

1.4 Art & Assets

All of the assets are free to use for commercial purposes and have been created by myself.

All of the assets have been created to be as efficient for game performance as possible. Each of the non ship assets share a single texture, “TEX_GameTexture” which has their UV maps spread across the sectioned colours.

The ships generally use two textures the orange and the blue variants, “TEX_PlayerTexture” & “TEX_EnemyTexture”. This means that for the entire project there could potentially be only 3 materials and 3 textures that need to be added to the list of draw calls.

TIP: Other textures have been provided in the images folder to provide alternate colour schemes for the ships, to apply these simply drag the texture into the slot on the material of the ship you wish to change the colour of. Any 2x2 texture will work with any variant of ship. You can alter my textures and change the colours for even more variety.

1.5 HUD

The HUD systems in the project follow the standard use of the Unreal Widget System. All of the information and setup can be found in the comments inside of each system.

1.51 Player HUD

To make sure that the player HUD is not displayed outside of gameplay (during menu screens) there is a check after the EventBeginPlay inside of BPPL checking to make sure the name of the current Map isn’t menu. This can be changed or extended to stop the HUD showing on other maps.

1.52 Menu HUD

The Menu HUD is activated in the Menu Level Blueprint. This can be found from the main interface by selecting, Blueprints -> Open Level Blueprint.