Side Scrolling Shooter Project – Technical Documentation

Contents

[GeneralManagement 3](#_Toc176435573)

[GameManager 3](#_Toc176435574)

[Dependency Injection 3](#_Toc176435575)

[GameSceneInstaller 3](#_Toc176435576)

[UserInput 3](#_Toc176435577)

[SSSActionMap 3](#_Toc176435578)

[InputLogger 3](#_Toc176435579)

[Cameras 4](#_Toc176435580)

[CameraTracker 4](#_Toc176435581)

[Characters 4](#_Toc176435582)

[Character 4](#_Toc176435583)

[HealthComponent 4](#_Toc176435584)

[CollisionHandler 4](#_Toc176435585)

[CollisionDetector 4](#_Toc176435586)

[NPC 4](#_Toc176435587)

[NPCBrain 4](#_Toc176435588)

[Enemy 5](#_Toc176435589)

[EnemyBrain 5](#_Toc176435590)

[DetectionArea 5](#_Toc176435591)

[StaticEnemy 5](#_Toc176435592)

[StaticEnemyBrain 5](#_Toc176435593)

[DynamicEnemy 5](#_Toc176435594)

[DynamicEnemyBrain 5](#_Toc176435595)

[Player 5](#_Toc176435596)

[MovementComponent 5](#_Toc176435597)

[Mover 5](#_Toc176435598)

[Jumper 6](#_Toc176435599)

[Croucher 6](#_Toc176435600)

[CombatComponent 6](#_Toc176435601)

[BehaviorTrees 6](#_Toc176435602)

[External Behavior Trees 6](#_Toc176435603)

[Abstracts 6](#_Toc176435604)

[SetSharedDependencies 6](#_Toc176435605)

[CustomSharedVariables 6](#_Toc176435606)

[NPCAction 7](#_Toc176435607)

[NPCConditional 7](#_Toc176435608)

[DataSO 7](#_Toc176435609)

[AmmoData 7](#_Toc176435610)

[BulletData 7](#_Toc176435611)

[CharacterData 7](#_Toc176435612)

[NPCData 7](#_Toc176435613)

[PlayerData 8](#_Toc176435614)

[GrenadeData 8](#_Toc176435615)

[MeleeWeaponData 8](#_Toc176435616)

[RangedWeaponData 8](#_Toc176435617)

[Weapons 8](#_Toc176435618)

[Weapon 8](#_Toc176435619)

[MeleeWeapon 8](#_Toc176435620)

[RangedWeapon 8](#_Toc176435621)

[Grenade 8](#_Toc176435622)

[ExplosionArea 9](#_Toc176435623)

[Ammo 9](#_Toc176435624)

[Bullet 9](#_Toc176435625)

[RiffleBullet 9](#_Toc176435626)

[Enums 9](#_Toc176435627)

[Signals 9](#_Toc176435628)

[UI 10](#_Toc176435629)

[GameSceneUIManager 10](#_Toc176435630)

[Static Canvas 10](#_Toc176435631)

[DynamicCanvas 10](#_Toc176435632)

[DynamicUIElement 10](#_Toc176435633)

[DamagePresenter 10](#_Toc176435634)

[DamageTable 11](#_Toc176435635)

[Utilities 11](#_Toc176435636)

[CustomLogger 11](#_Toc176435637)

[LayerParameters 11](#_Toc176435638)

[AnimatorParameters 11](#_Toc176435639)

[Tags 11](#_Toc176435640)

[Extensions 11](#_Toc176435641)

[MonoBehaviourPool<T> 11](#_Toc176435642)

[Serializable Dictionary Scripts 12](#_Toc176435643)

[ReactiveProperty<TValue, TOwner> 12](#_Toc176435644)

[Current Problems 13](#_Toc176435645)

# GeneralManagement

## GameManager

GameManager is a MonoBehaviour class that is responsible for overall project management. Currently it is responsible for quitting the game only.

# Dependency Injection

The project makes an extensive use of Zenject library for dependency injection pattern implementation. Some scripts also implement Zenject interfaces (ITickable, IInitializable etc.) to avoid inheritance from MonoBehaviour for efficiency reasons.

## GameSceneInstaller

GameSceneInstaller is an heir to MonoInstaller class that contains dependencies that are distributed to applicants when a game scene is loaded.

# UserInput

The project is underpinned by the Unity New Input System. The old input system is disabled.

## SSSActionMap

SSSActionMap is a Unity auto-generated script for SSSActionMap.inputactions file. SSSActionMap is automatically regenerated if SSSActionMap.inputactions is changed.

## InputLogger

InputLogger is a class that creates an instance of SSSActionMap and processes, relays its events and reads some values. There can be only one instance of InputLogger in a scene which is ensured by GameSceneInstaller. All other classes must get a reference to InputLogger rather than trying to create and read SSSActionMap directly.

# Cameras

The project uses Cinemachine which is an official free Unity asset designed to simplify development of camera movements. The camera automatically follows the Player.

MainCamera renders everything except UI. UICamera renders nothing, but UI. UICamera is overlaid over MainCamera and parented to it.

## CameraTracker

CameraTracker is a MonoBehaviour script that contains references to all Cameras in the game scene. Its function is to detect when MainCamera changes its position and invoke a corresponding event. The event is used by DynamicCanvas to recalculate DynamicUIElements’ positions.

# Characters

## Character

Character is an abstract MonoBehavior class that both Player and NPCs in the game inherit from. It contains logic and components that are common for both Player and NPCs, such as HealthComponent, CollisionHandler etc.

## HealthComponent

HealthComponent is a Character’s dependency that is responsible for everything that happens to its health, like taking damage and healing. If the Character dies, HealthComponent invokes the corresponding event.

## CollisionHandler

CollisionHandler is a class that analyzes and handles collision callbacks.

## CollisionDetector

CollisionDetector is a MonoBehavior script that must be attached to the same gameobject that has a collider on it. It listens to Unity collision callbacks and relays them to Character’s CollisionHandler. Usually this component is added to SpriteRenderer that is a child gameobject to Character itself.

## NPC

NPC is an abstract heir to Character which contains common logic and variables for all Characters, except for Player itself.

## NPCBrain

NPCBrain is an abstract class that is responsible for decision making for an NPC. Every NPC must have an NPCBrain.

## Enemy

Enemy is an abstract heir to NPC that is in general supposed to be hostile to Player.

## EnemyBrain

EnemyBrain is an abstact heir to NPCBrain. All variants of Brains for Enemies must inherit from this script.

## DetectionArea

DetectionArea is a MonoBehaviour enemy specific script that is responsible for detecting Player. When Player enters DetectionArea, it sets the corresponding property in EnemyBrain to Player, otherwise, it sets it to null.

## StaticEnemy

StaticEnemy is a concrete Enemy class that does not move on its own, but can be moved by Unity physics.

## StaticEnemyBrain

StaticEnemyBrain is an EnemyBrain created specifically for StaticEnemy.

## DynamicEnemy

DynamicEnemy is a concrete Enemy class that can move on its own. Once it detects the Player with its DetectionArea it pursues him until it is close enough to attack.

## DynamicEnemyBrain

DynamicEnemyBrain is a concrete EnemyBrain created specifically for DynamicEnemy. It contains pursuit logic. For example, it defines when the DynamicEnemy must jump or how close it must approach the Player to attack.

## Player

Player is a Character script that represents the player in the game. It has multiple dependencies which control separate parts of its behaviour.

## MovementComponent

MovementComponent is an abstract class that holds common fields for all Player’s dependencies that are responsible for its movement.

## Mover

Mover derives from MovementComponent and is responsible for Player moving around when it is grounded and not jumping.

## Jumper

Jumper is another heir to MovementComponent that allows Player to jump up, jump down from platforms and enables some control in midair.

## Croucher

Croucher is Player’s dependency that is responsible for crouching and standing up after being crouched. When Player is crouched, other movements are disabled.

## CombatComponent

CombatComponent is a concrete type that handles all logic related to equipping and using weapons.

# BehaviorTrees

The project makes an extensive use of BehaviorDesigner asset made to create complex AI behavior. The link to the documentation: <https://opsive.com/support/documentation/behavior-designer/overview/> . There are also tutorial videos in YouTube.

## External Behavior Trees

An external behavior tree is a number of nodes that are exported from the parent tree and can be reused in other trees by adding BehaviourTreeReference nodes.

The project widely exploits these external trees to avoid duplication. For example, attack trees can be reused by both friendly and hostile npcs.

All behavior trees are stored in \_Scripts/BehaviourTrees/ExternalTrees.

## Abstracts

Abstracts is a folder in \_Scripts/BehaviorTrees. It is comprised from abstract classes, utility classes and Actions which are too general to place them elsewhere.

## SetSharedDependencies

SetSharedDependencies is a class which inherits from Action class of Behavior Designer. Since many custom nodes, created specifically for the current project, require a reference to an NPC or its heir, the main purpose of this class is to find and set a reference to the NPC and its variations.

## CustomSharedVariables

Behavior Designer allows sharing dependencies among multiple nodes and behavior trees. However, in order to do this such variable must be created as a shared one. Common data types are converted to shared variables by default. However, in order to share an NPC reference among multiple nodes, a respective heir of SharedVariable class must first be created.

CustomSharedVariable is a declaration script for all custom SharedVariables that are defined in the project.

## NPCAction

NPCAction is an heir to Action class of BehaviorDesigner. It is main purpose is preventing duplication of the code related to receiving a reference to the NPC in custom Actions. NPCAction also contains references to all components the NPC has, so they can be referenced directly omitting the NPC itself.

Multiple heirs to this class can be created, each of them can contain a reference to the target class inheriting from NPC, such as EnemyAction, AllyAction etc.

## NPCConditional

NPCConditional is an heir to Conditional class of BehaviorDesigner. It is main purpose is preventing duplication of the code related to receiving a reference to the NPC in custom Conditional. NPCConditional also contains references to all components the NPC has, so these aspects can be referenced directly omitting the NPC itself.

Multiple heirs to this class can be created, each of them can contain a reference to the target class inheriting from Humanoid, such as EnemyConditional, AllyConditional etc.

# DataSO

SO stands for ScriptableObject. This folder/assembly must include all SOs created for the project.

## AmmoData

AmmoData is an abstract heir to ScriptableObject that contains all properties common for ammunition.

## BulletData

BulletData is a concrete heir to AmmoData that consists of bullet specific properties.

## CharacterData

CharacterData is an abstract heir to ScriptableObject which contains common properties for all Characters in the game.

## NPCData

NPCData is an abstract heir to CharacterData that is comprised of properties general to all NPCs in the project.

## PlayerData

PlayerData derives from CharacterData and adds some properties specific to Player.

## GrenadeData

GrenadeData is an heir to ScriptableObject that contains all properties for Grenades.

## MeleeWeaponData

MeleeWeaponData is an heir to ScriptableObject that contains all properties for MeleeWeapons.

## RangedWeaponData

RangedWeaponData is an heir to ScriptableObject that contains all properties for RangedWeapons.

# Weapons

## Weapon

Weapon is an abstract class that contains common variables and methods for all weapons in the project.

## MeleeWeapon

MeleeWeapon is an abstract heir to Weapon which consists of variables and methods specific to all melee weapons in the project, such Sword etc.

MeleeWeapon has a melee weapon specific Block method which blocks incoming damage.

## RangedWeapon

RangedWeapon is an abstract heir to Weapon which consists of variables and methods specific to all ranged weapons in the project, such AssaultRifle etc.

RangedWeapon has a ranged weapon specific Aim method which points the weapon towards a world position.

## Grenade

Grenade is an abstract heir to Weapon which consists of variables and methods specific to all grenades in the project, such FragGrenade etc.

Grenade also has an Aim method, but unlike that of RangedWeapon a Character aim with a Grenade only when it holds it and not throwing, while RangedWeapon is pointed constantly to a world position.

Grenade also has a Pointer which is a MeshRenderer and Transform shaped into a line. It is updated to indicate the throw direction and force.

A Character is unable to throw another Grenade until the thrown one remains unexploded. After a Character throws a Grenade its sprite child object gets unparented, it is done so that the child object was unaffected to the parent’s physics. (The parent is effected by Character’s movements).

Note: You must look for an ExplosionArea when trying to process a collision coming from an explosion. And it is important to remember that when a Grenade’s sprite is thrown and exploding it remains unparented from the actual Grenade. So it is impossible to find the Grenade component by calling TryGetComponent() on Transform.parent.

## ExplosionArea

ExplosionArea is a MonoBehaviour script parented to Grenade’s sprite. It requires a CircleCollider in the trigger mode and its primary function is expanding. ExplosionArea does not deal damage to whatever it touches. But it plays explosion effects.

# Ammo

## Bullet

Bullet is an abstract MonoBehaviour which indicates all bullets existing in the game. This class is responsible for bullets’ flying. It pushes the rigidbody forward until it hits a collider or the flying time is up.

## RiffleBullet

RifleBullet is a concrete Bullet made for rifles.

# Enums

The Enums folder and the corresponding assembly definition have been created to eliminate cyclic references among assembly definitions. All enums must be put into this folder.

# Signals

The project is underpinned by SignalBus that is implemented in Zenject asset. SignalBus works with C# classes that act as signals. Usually each signal must be implemented as a separate C# class.

# UI

## GameSceneUIManager

GameSceneUIManager is a MonoBehaviour script that is responsible for initializing and regulating all UI in the game scene.

## Static Canvas

Static Canvas is a usual canvas renamed to Static. It holds all UI elements that do not move on canvas in runtime.

## DynamicCanvas

Unlike Static Canvas, DynamicCanvas is a class attached to a canvas. This class holds all DynamicUIElements that are supposed to change their positions on the canvas in runtime. The reason for this separation is that a whole canvas is redrawn every time when any detail on it changes.

DynamicCanvas hides DynamicUIElements when they are out of screen boundaries and reenables them if they get back to the screen.

DynamicCanvas has a few MovementTypes that characterize how DynamicUIElement behaves on the canvas.

FixedToWorldPosition: DynamicUIElement is replaced on DynamicCanvas in the way it is fixed to the world position assigned to it.

FixedOnDynamicCanvas: DynamicUIElement will keep its position on the canvas which it receives when placed on the canvas for the first time. It is useful for elements that must not move on the DynamicCanvas when enabled, but can change their position when they are reenabled (like pop-up menus or hints).

FixedOnXAxis: DynamicUIElement is tied to the x coordinate of its world position, but its movements on y axis remain unconstrained. This mode is used for DamageTables that must stay still on x, but be allowed to float on y.

## DynamicUIElement

DynamicUIElement is a base abstract class for all UI elements that must be placed on DynamicCanvas. Be mindful that a DynamicUIElement must invoke TurnedOff action when you are turning it off so that DynamicCanvas could properly remove it.

## DamagePresenter

DamagePresenter is a MonoBehaviour script that receives DamagedSignals and calculates proper world positions for DamageTables that depict inflicted damage. DamagePresenter places DamageTables in the shape of a wave over damaged Characters. It also chooses the correct color for DamageTables depending on the amount of inflicted damage.

DamagePresenter makes use of MonoBehaviourPool to source DamageTables and from SerializableDictionary to link damage and its color.

## DamageTable

DamageTable is a MonoBehavior script and a prefab that has a text field. After it has been drawn, it floats up and vanishes. DamageTable implements IMonoBehaviuorPoolElement and makes use of DOTween asset and UniTask to float and disappear.

# Utilities

## CustomLogger

CustomLogger is a static class that can log standard string messages from any other class. This class must be used throughout the project to log messages meant for developers.

## LayerParameters

LayerParameters is a static class that includes all layer names in the project for other scripts to reference them. It is added to minimize risks related to misspelling layer names in other scripts.

The static constructor of LayerParameters ensures that registered names exist among the layers in the project.

## AnimatorParameters

AnimatorParameters is a static class that holds all parameter names used by animators. It is added to minimize risks of misspelling animator parameters in specific scripts.

## Tags

Tags is a static class that holds all tags used in the project. It is added to minimize risks of misspelling tags in specific scripts.

## Extensions

Extensions is a static class that holds useful extension methods for different types.

## MonoBehaviourPool<T>

MonoBehaviourPool is a utility class that simplifies controls over MonoBehaviours that get spawned and turned off regularly, eliminates code duplication among different classes. In most cases UI classes such as DescriptionWindow and StateIndicationSystem make use off of MonoBehaviourPools.

In order to be initialized inside the MonoBehaviourPool, T must implement IMonoBehaviourPoolElement that requires implementation of public GameObject {get} property, since game objects in a MonoBehaviourPool<T> are manipulated through their activation and deactivation.

When instantiated, a gameObject is supposed to call its Awake() method automatically. However, there were some errors when GameObject remained null since Awake() was not called after instantiation in a pool. To fix this issue, IMonoBehaviourPoolElement must also implement public Awake() method which will be called by a pool if GameObject of its new T element is null. This guarantees that any object in a pool will be constructed properly.

## Serializable Dictionary Scripts

Serializable Dictionary scripts are all the scripts added to the folder Utilities/SerializableDictionary. They are used to solve the problem of dictionary serialization in Unity.

SerializableDictionary<TKey,TValue> is a generic class, that holds all the logic related dictionary with some custom modifications from the author. The link to the source: <https://forum.unity.com/threads/finally-a-serializable-dictionary-for-unity-extracted-from-system-collections-generic.335797/> . According to the author, he took the standard dictionary script out of System.Collections and removed/added some lines of code.

DictionaryDrawer<TK,TV> is a generic class responsible for visualization of a SerializableDictionary<TKey, TValue> in UnityEditor. The author says that DictionaryDrawer makes impossible setting reference types as keys without prior modifications to the script. However, I have managed to solve this problem by making the script create new instances of TK if it is of a reference type.

Class CustomDictionaries is a general C# script where specific instances of the two previous types must be created for a specific project. There are also some comments in it containing useful advice on how exactly custom SerializableDictionary heirs must be created and worked with.

## ReactiveProperty<TValue, TOwner>

ReactiveProperty<TValue, TOwner> is a generic class created to fire events automatically when TValue changes. TOwner is added mostly, because in our experience subscribers need to know what script exactly TValue belongs. The set accessor for TValue property is left public for no other choice, but if another script tries to change TValue, then it can be easily tracked from TOwner script which declares its custom ReactiveProperty.

# Current Problems

1. BehaviorDesigner which is widely used in the project, is not a free asset. Currently a pirate version is used. Before the release it is recommended to be the asset officially to avoid legal claims (its price is below 100 USD and it can be purchased in the Unity Asset Store).