The Stroke of Midnight / Player

Architecture/Design Document

**Table of Contents**

1 Introduction.. 3

2 Design Goals.. 3

3 System Behavior.. 3

4 Logical View... 4

4.1 High-Level Design (Architecture) 4

4.2 Mid-Level Design. 5

4.3 Detailed Class Design. 6

5 Process View... 7

6 Use Case View... 11

Change History

**Version:** 0.1

**Modifier:** Omer Kocar and Zoe Purcell

**Date:** 03 / 09 / 2021

**Description of Change:** Module Design Document started.

**Version:** 0.2

**Modifier:** Zoe Purcell

**Date:** 03 / 14 / 2021

**Description of Change:** Sequence Diagrams added

**Version:** 0.3

**Modifier:** Zoe Purcell

**Date:** 03 / 15 / 2021

**Description of Change:** System Behaviour and Design goals improved, Added Sequence Diagram for Lighter logic

**Version:** 0.4

**Modifier:** Zoe Purcell

**Date:** 03 / 16 / 2021

**Description of Change:** Document finished. Mid-Level and UML updated after feedback.

# **1 Introduction**

This document describes the architecture and design for The Stroke of Midnight, a game being developed by Memento Game Studios. The Stroke of Midnight is a singleplayer psychological horror game where the player breaks into an art gallery on a dare, which turns into a fight for his life as he attempts to escape.

The purpose of this document is to describe the architecture and design of the Player Module application in a way that addresses the interests and concerns of all major stakeholders. For this application the major stakeholders are:

* Developers – they want an architecture that will minimize complexity and development effort.
* Project Manager – the project manager is responsible for assigning tasks and coordinating development work. He or she wants an architecture that divides the system into components of roughly equal size and complexity that can be developed simultaneously with minimal dependencies. For this to happen, the modules need well-defined interfaces. Also, because most individuals specialize in a particular skill or technology, modules should be designed around specific expertise. For example, all UI logic might be encapsulated in one module. Another might have all game logic.
* Maintenance Programmers – they want assurance that the system will be easy to evolve and maintain on into the future.

# **2 Design Goals**

The design priorities for the Player system are:

* The design should minimize complexity and development effort.
* The design should allow Designers to easily have access to the properties of the player and player movement.
* The design should simplify Player interactions with other items and level elements.

# **3 System Behavior**

The Player Module is built from a single CharacterBase parent that both the Player and AI inherit from.This allows all Characters to have the needed features such as their HealthComponent and death behaviour. The Player should be able to perform a number of tasks, such as sprint, drain stamina, pull out a light, teleport via Portals and update their spawn location with Checkpoints.

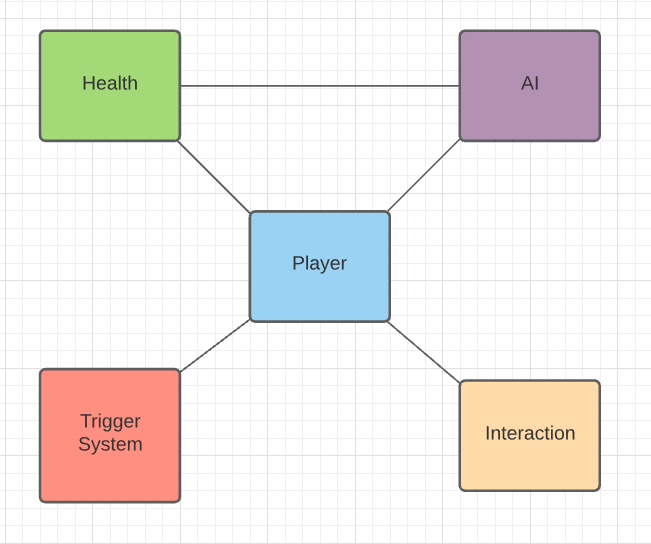
# **4 Logical View**

The logical view describes the main functional components of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

In this section the modules of the system are first expressed in terms of high level components (architecture) and progressively refined into more detailed components and eventually classes with specific attributes and operations.

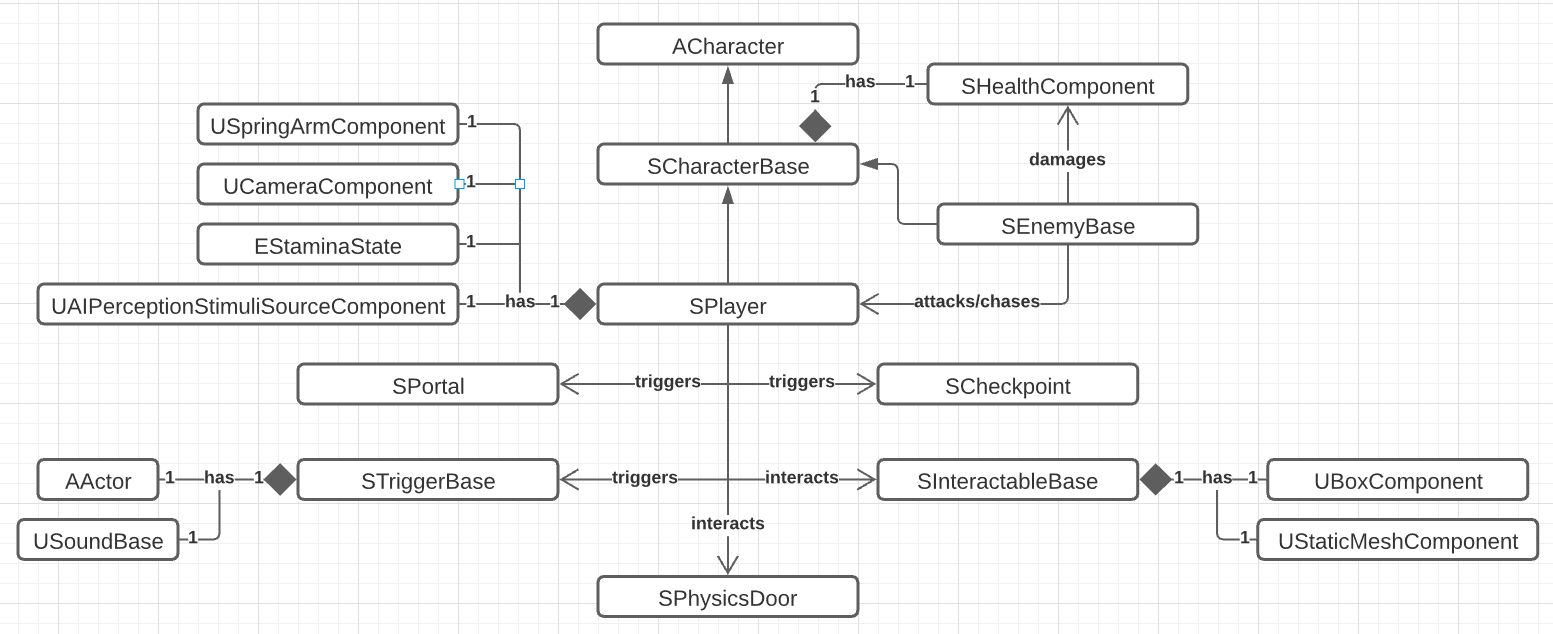
## **4.1 High-Level Design (Architecture of the Entire system)**

The high-level view or architecture consists of 5 major components:

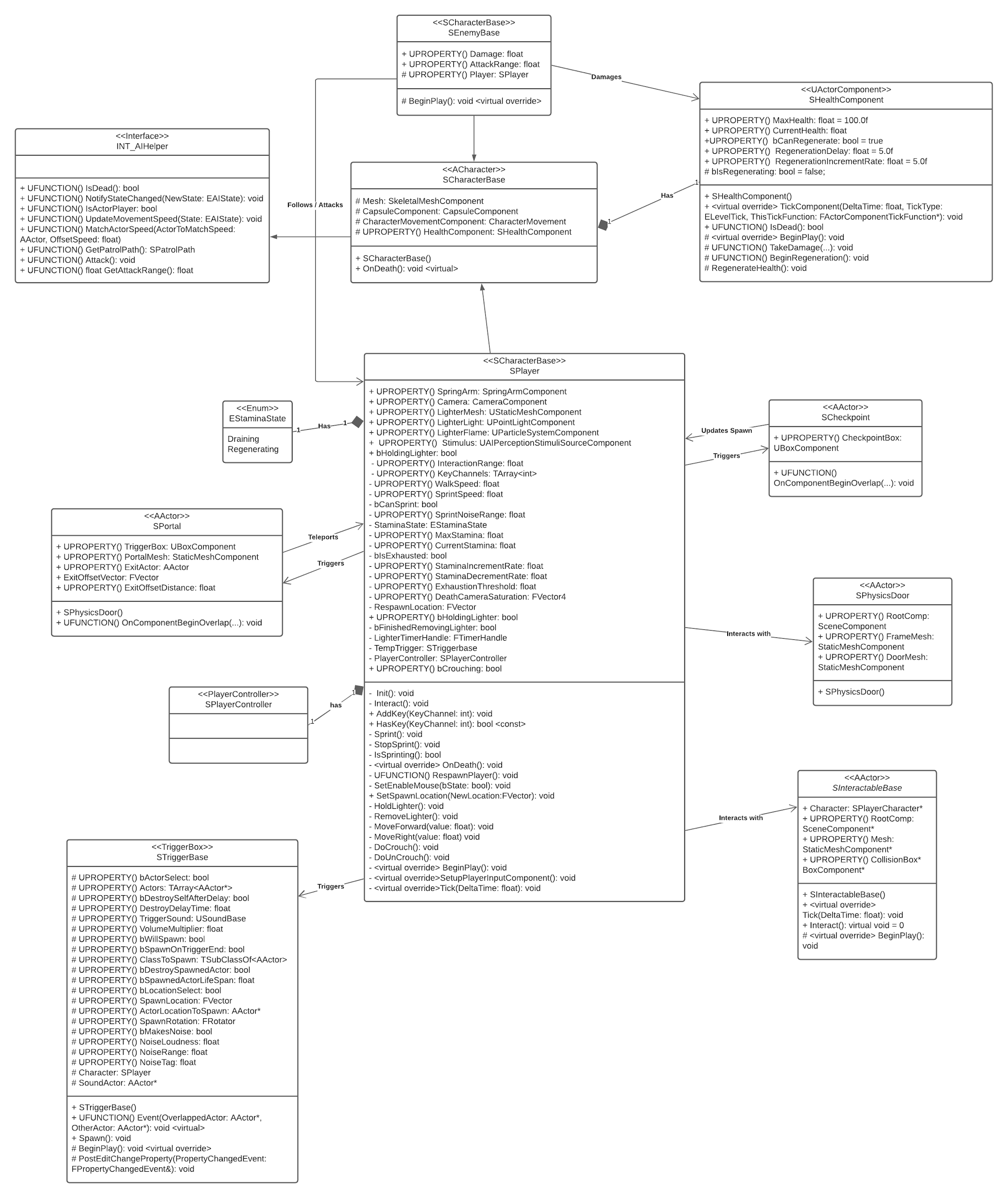


* Player System is the main system, which consists of a controlled character that takes in user input.
* Event Trigger Module holds all the unique event triggers and their effects on the Player character.
* AI System is used for all enemy behaviour.
* Health System handles how Characters take damage/receive health.
* Interaction System handles different objects that can be picked up, turned on, or interacted with by the player.

## **4.2 Mid-Level Design of the Player Module**



## **4.3 Detailed Class Design of the Player Module**

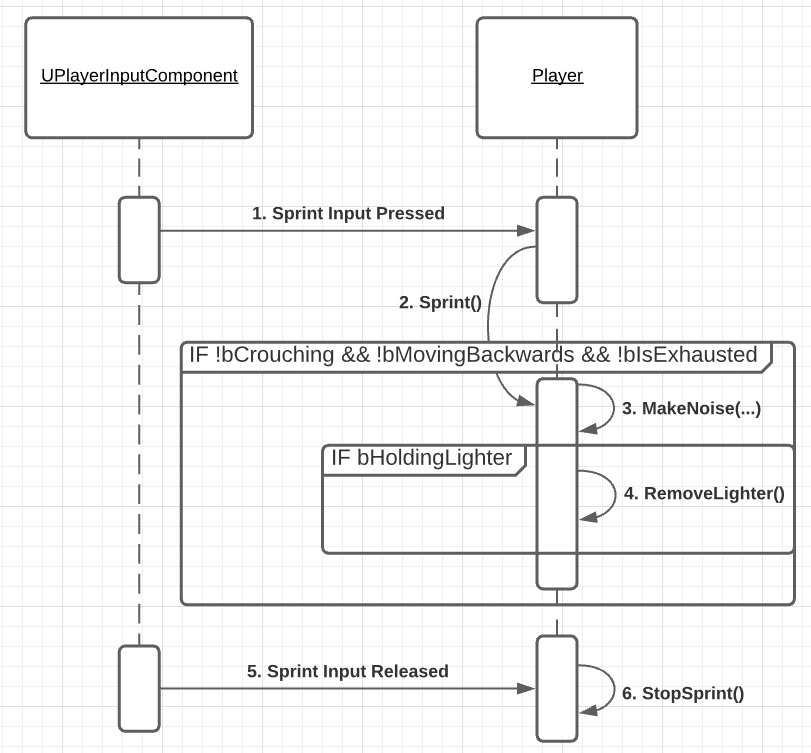


# **5 Process View of the Player Module**

There’s a number of actions that the Player can perform, such as sprint, drain stamina, pull out a light, teleport via Portals and update their spawn location with Checkpoints.

For Player interactions with the HealthComponent, Trigger Boxes, and Interactables, see their respect Module Design Documents.

Sprinting and Stamina



The sprinting sequence starts when the Player presses the sprint key (Left Shift on Keyboard, LB/L1 on Controller) which calls the Player’s Sprint function.

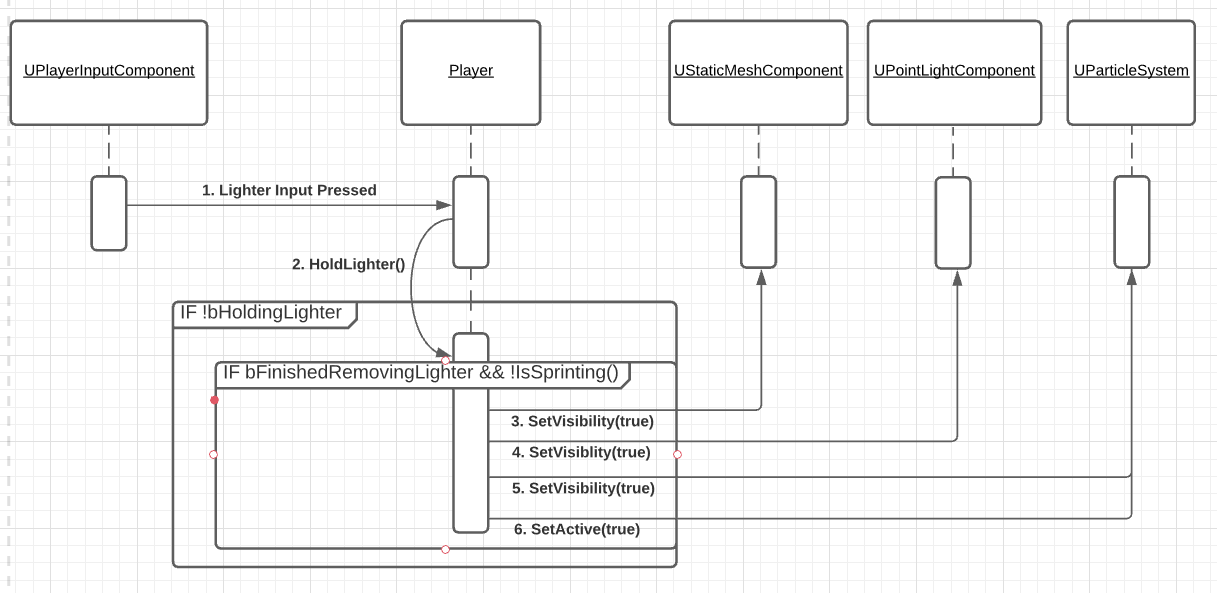
The Player must pass several checks before they can successfully start sprinting. If they are not crouching, not moving backwards, and not exhausted (not suffering from a Stamina penalty: this is detailed in the Use Case View), then they can start sprinting.

When the Player starts to sprint, their MaxWalkSpeed is increased (200.0f to 350.0f), their Stamina starts to drain, and MakeNoise is called. In The Stroke of Midnight, sprinting is loud and can attract the attention of enemies, so you should use it sparingly. (Note: Sprinting to make noise is currently bugged and as a result is not included in this release. Instead, for the sake of testing, MakeNoise can be called by pressing 2/RB/R1)

If the Player was currently holding their Lighter when they started sprinting, they will put it away.

When the Sprint input is released, StopSprint is called, which sets the player’s MaxWalkSpeed is set back to normal and their Stamina starts to regenerate.

Lighter - On

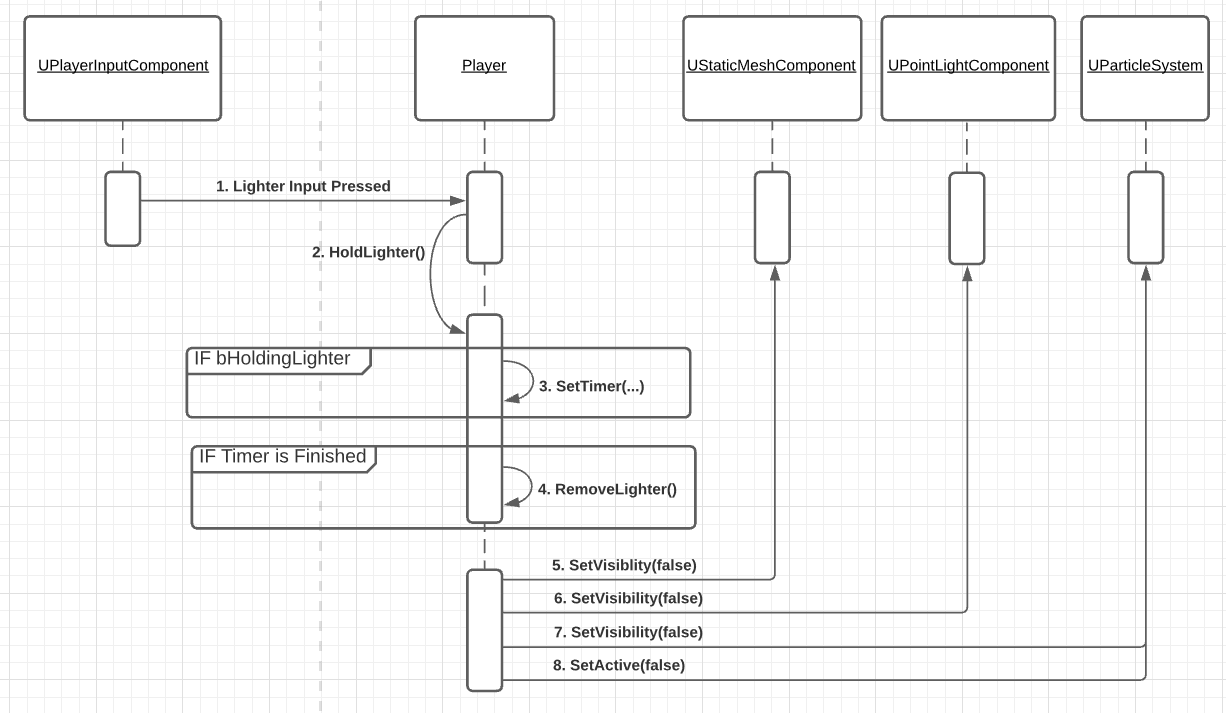


Like sprinting, the Lighter toggling sequence is tied to an input. (C on Keyboard, Y/Triangle on Controller) Once the input is pressed, HoldLighter is called. From here, there’s two possible paths, depending on whether bHoldingLighter is true (below) or false (above).

If the Lighter is not being held, the Player must pass two more checks before it can be turned on. If the Player is not sprinting, and if the Player is in the middle of the Lighter animation, they will pull out their Lighter.

To do so, The Lighter’s StaticMesh, PointLight, and ParticleSystem have their visibility turned on. The ParticleSystem is also set to active.

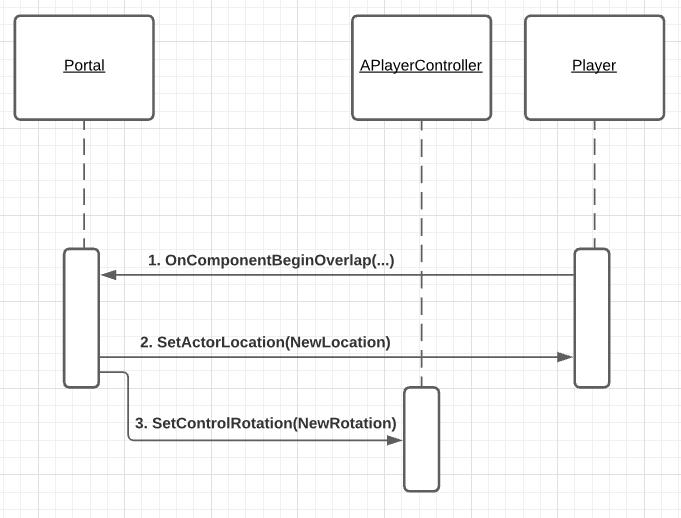
Lighter - Off



To turn off the Lighter, a very similar sequence is used. If bHoldingLighter is true this time, a Timer is set. Once this Timer is finished, RemoveLighter is called, which calls the same functions on the Lighter’s components, but passes in false this time to turn them off.

The turning off of the Lighter is tied to a Timer so the Player can properly finish the animation of them setting their arm down before the light is disabled.

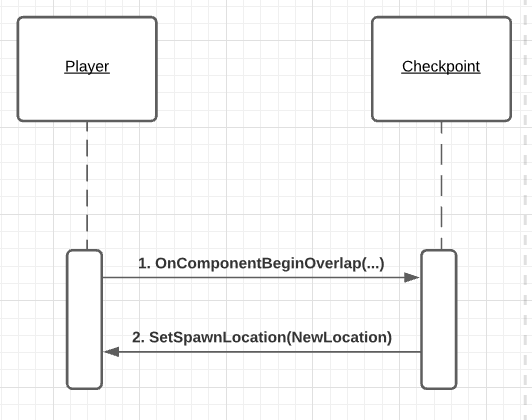
Portals



In its current state for this release, Portals - which will be used to travel between different parts of the level - are extremely simple. Once the Player overlaps with the Portal’s trigger box, it calls the Portal’s OnComponentBeginOverlap event.

In this event, SetActorLocation is called on the player, passing in the Portal’s location. SetControlRotation is also called on the Player’s controller, so the Player is facing the same direction of the Portal when they exit. (With their back facing the location they were teleported to)

Checkpoints



The Checkpoint sequence is even simpler: when the Player triggers the Checkpoint’s OnComponentBeginOverlap event by overlapping with it’s trigger box: the Checkpoint updates the Player’s spawn location.

The Player will be sent to this new location the next time they die and reload.

# **6 Use Case View**

Modifying Stamina and Sprinting

Note: For this release, both Player Health and Stamina are displayed via debug messages. In normal gameplay, the exact values will be hidden to the Player, and they will have to rely on visual and audio cues to determine how low each stat is.

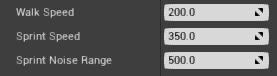
The Player has several variables related to Stamina - all editable either within code or inside the Unreal Editor.

* **MaxStamina**: The maximum amount of Stamina the Player can have. Defaults to 100.0f
* **StaminaIncrementRate:** How quickly the Player regenerates their Stamina. Defaults to 10.0f. It is not required for this to be the same value as the DecrementRate
* **StaminaDecrementRate:** How quickly the Player drains their Stamina. Defaults to 10.0f. It is not required for this to be the same value as the IncrementRate
* **ExhaustionThreshold:** When the Player is Exhausted, how much Stamina do they need to recover before they can Sprint again. Defaults to 25%.



Additionally, there are a few variables related to Sprinting that can be edited in the Player as well

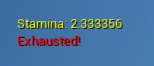
* **WalkSpeed**: How fast the player moves while walking. Defaults to 200.0f
* **SprintSpeed**: How fast the player moves while sprinting. Defaults to 350.0f
* **SprintNoiseRange**: How far the noise made by sprinting can be heard. Defaults to 500.0f. Note: This feature is currently disabled in this release.



**Additional Note:** Attempting to sprint while holding out the Lighter will cause the Player to automatically put the Lighter away before sprinting.

Exhaustion

Every frame that the Player is sprinting, their Stamina decreases by a certain amount. (StaminaDecrementRate \* DeltaTime). If their CurrentStamina reaches 0, StopSprint is called to force them to stop sprinting, and bIsExhausted gets flipped to true. For this release, a debug message will show when the Player gets Exhausted.



When exhausted, the Player will be unable to sprint (due to the !bIsExhausted condition in Sprint()) until their Stamina regenerates to a certain threshold. By default, this is 25%, meaning one quarter of the Player’s Stamina must be regenerated before they can sprint again.

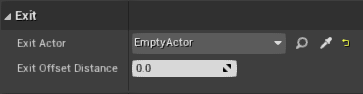
In order to avoid this, The Player must use their Sprint sparingly, and their character a chance to catch their breath.

One-Way Portals

One-Way Portals are portals that simply teleport the Player to a desire location that they cannot teleport back to.

To set up a One-Way Portal

1. Drag the Portal blueprint into the scene. Depending on your usage it would have a mesh (like a doorway) or would just be a simple trigger box without a mesh chosen.
2. Drag an Empty Actor into the scene, and place it where you want the Player to teleport to.
3. Under “Exit” in the Portal settings, set the “Exit Actor” to the empty Actor. Keep Exit Offset to 0.0



Two Way Portals

Note: These are being used in our demo level, but likely will not be used in actual gameplay

Two-Way Portals allow the Player to teleport back and forth between two Portals.

1. Drag the Portal blueprint into the scene. Depending on your usage it would have a mesh (like a doorway) or would just be a simple trigger box without a mesh chosen.
2. Instead of using an Empty Actor, bring a second Portal into the scene.
3. Assign the “Exit Actor” for each Portal to the opposite Portal.
4. **Important:** For Two-Way Portals to work, the Player needs to be offset from the exit Portal’s location. Failure to do so will result in an infinite loop and eventual stack overflow, as the Player will be constantly teleporting between the two Portals (due to constantly overlapping with each Portal’s trigger box).

