The Stroke of Midnight / Health Component

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... 9

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:** Process and Use Case View completed

**Version:** 0.4

**Modifier:** Zoe Purcell

**Date:** 03 / 15 / 2021

**Description of Change:** Document completed.

**Version:** 0.4

**Modifier:** Zoe Purcell

**Date:** 03 / 16 / 2021

**Description of Change:** Mid Level and UML updated after feedback

# **1 Introduction**

This document describes the architecture and design for The Stroke of Midnight application being developed by Memento Game Studios. The Stroke of Midnight is a single-player 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 AI 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 into the future.

# **2 Design Goals**

The design priorities for the Health system are:

* The design should minimize complexity and development effort.
* The design should allow easy access to values surrounding Health and Health Regeneration both in C++ and in the Unreal Editor

# **3 System Behavior**

The HealthComponent is a custom component that can be attached to any Character. It stores values for health and health regeneration which can be edited both in C++ and in the Unreal Editor. The system should handle any damage taken by the owning Character. If the owning Character inherits from the custom CharacterBase function, the component will initiate the owner’s death behaviour when its health reaches zero.

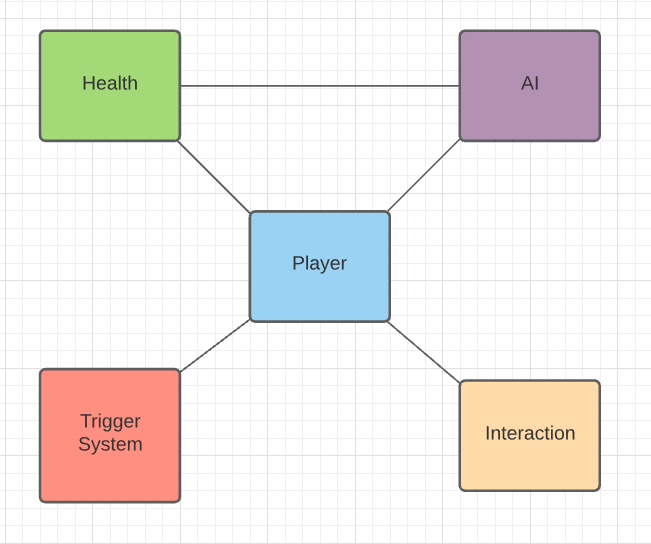
# **4 Logical View**

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.

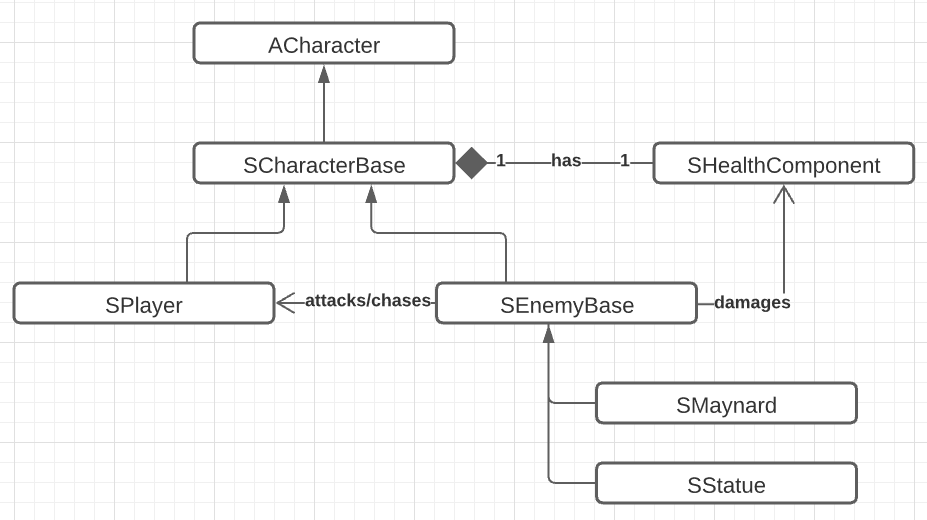
## **4.1 High-Level Design (Architecture of the Entire system)**

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

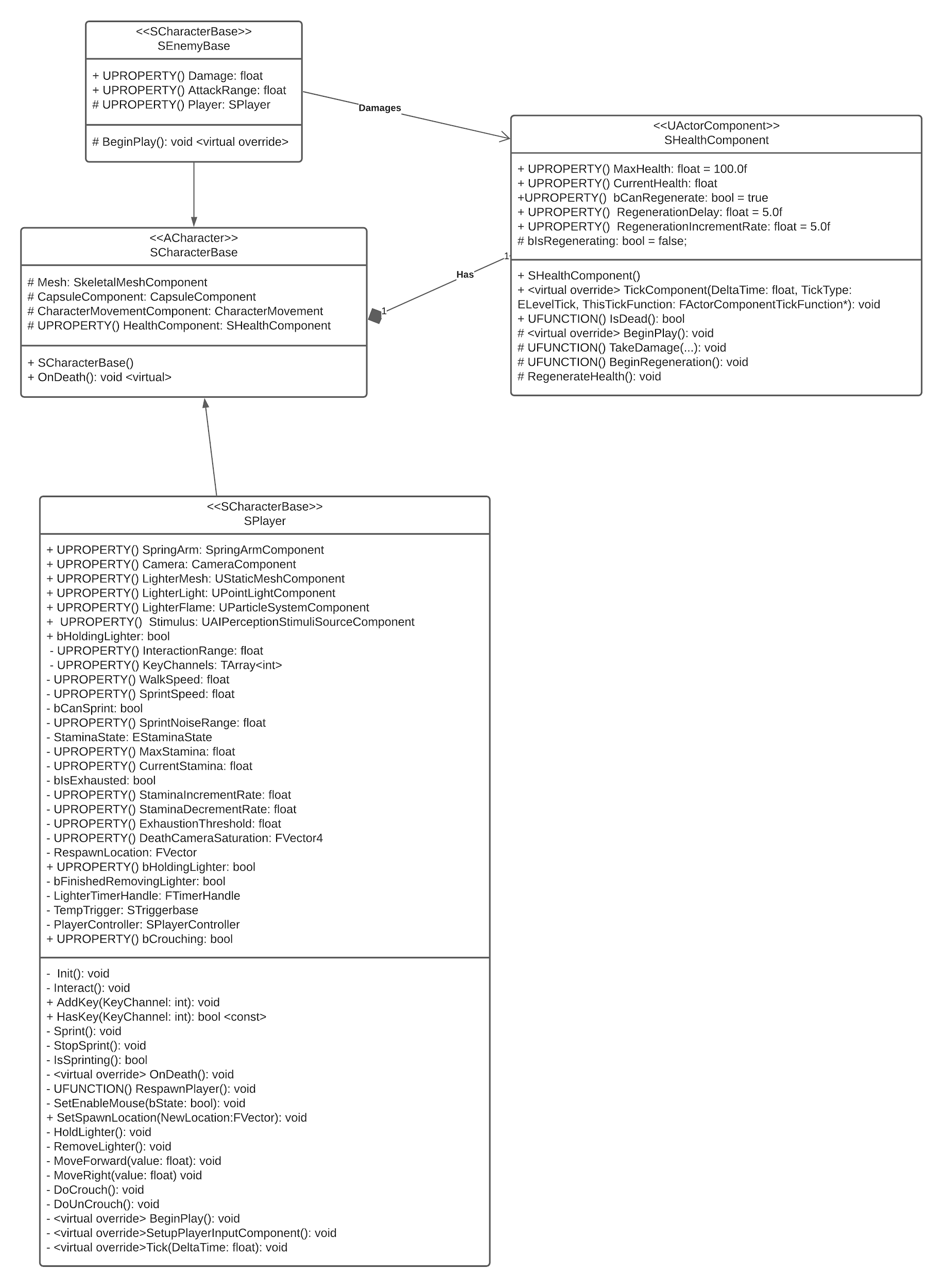


* Health System handles how Characters take damage/receive health.
* AI System is used for all enemy behaviour.
* Event Trigger Module holds all the unique event triggers and their effects on the Player character.
* Player System is the main system, which consists of a controlled character that takes in user input.
* 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 Health Component Module**



## **4.3 Detailed Class Design of the Health Component Module**

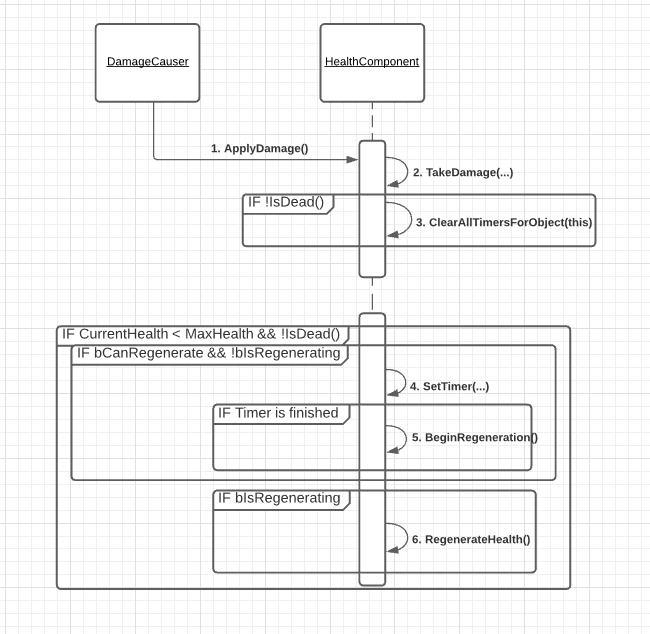


# **5 Process View of the Health Component Module**

Every operation related to the HealthComponent can be separated into 2 sequences: If the player takes damage and survives, or if the player takes damage and dies.

Taking Damage and Living

In our current release, only the Enemies can deal damage to the Player - the Player currently cannot deal any damage to the Enemies.



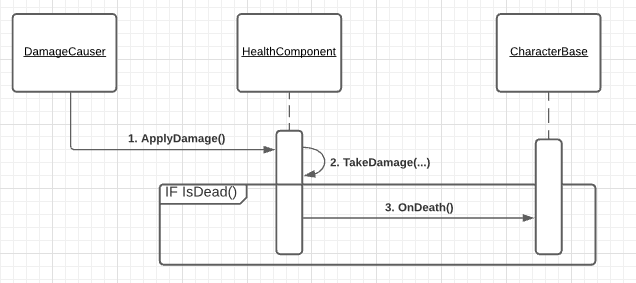
To begin, The Damage Causer calls ApplyDamage to trigger the HealthComponent’s TakeDamage event.

Inside this event, after we check to see if the Owner is still alive, we start by clearing all timers on the HealthComponent. Starting Health Regeneration is tied to a Timer, so the Player doesn’t immediately start regenerating Health as soon as they take damage. However, taking damage again before the delay is finished will start this sequence again, which begins by clearing the timer. This makes the Owner wait for the entire duration before they can heal themselves back up.

Then, inside TickComponent, we check to see if the Owner is missing health and are not already dead. If so, we start the regeneration Timer that was cleared earlier. When the timer is finished without interruption, BeginRegeneration is called, which simply flips the bool bIsRegenerating to true, starting the regenerating process.

Also in TickComponent, we check if the Owner is allowed to regenerate Health (bCanRegenerate) and is currently in the regenerating state. (bIsRegenerating, which we just switched to true) When both of these conditions are true, RegenerateHealth is called every frame, which increments the Health by a certain, editable variable (RegenerationIncrementRate).

Taking Damage and Dying



This sequence starts the same as the previous - by the Damage Causer triggering the Components TakeDamage event. In this case, however, IsDead has returned true.

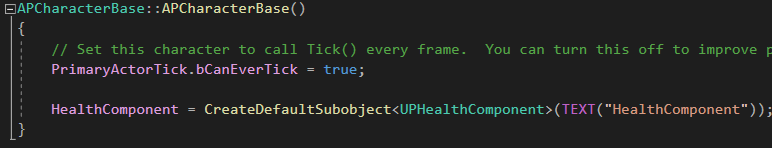
If the Owner inherits from the CharacterBase class - which is currently the Player and all Enemies - then the HealthComponent will call CharacterBase’s virtual OnDeath function, which will result in different behaviour depending on what it was overwritten with.

# **6 Use Case View**

Adding and Modifying a HealthComponent

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.

Like any component, the HealthComponent can be added to a Character by simply declaring it in the .h file and calling CreateDefaultSubObject in the .cpp. However, the CharacterBase class already has its own HealthComponent, so it is unnecessary to add one if the Character is already inheriting from this class.



The HealthComponent comes with some properties that can be accessed and modified in any class that owns it

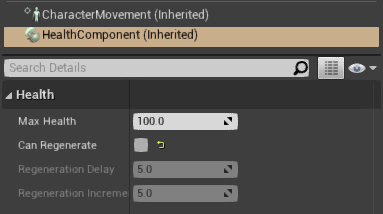
* **MaxHealth:** The maximum Health that the Owner has. Defaults to **100.0f**
* **bCanRegenerate:** Whether or not the Owner has the ability to regenerate health. Defaults to **true**.
* **RegenerationDelay:** How many seconds after taking damage can the Owner start regenerating health. Defaults to **5.0f.**
* **RegenerationIncrementRate:** How quickly the Owner regenerates their health. Defaults to **5.0f**

To edit a value, simply write “HealthComponent->PropertyName = NewValue”



Inside the Unreal Editor, if each instance of a Character should have different values for each HealthComponent property, simply select the instance in the scene, select the HealthComponent, and all values will be editable, under the “Health” category.

“Regeneration Delay” and “Regeneration Increment Rate” can only be edited if “Can Regenerate” is checked to true.



Damage and Death

The HealthComponent is subscribed to the OnTakeAnyDamage event, so all damaging and death logic is handled in its TakeDamage function. To use it, simply call UGameplayStatics::ApplyDamage inside the class of your Damage Instigator, passing in the Owner of the component. The value passed in for Damage will be removed from the HealthComponents MaxHealth, and properly clamped so it doesn’t go below zero.

