**ShadowStalk Entity Module**

Architecture/Design Document

**Table of Contents**

[1 Introduction 2](#_Toc66789270)

[2 Design Goals 3](#_Toc66789271)

[3 System Behavior 3](#_Toc66789272)

[4 Logical View 4](#_Toc66789273)

[4.1 High-Level Design (Architecture of the Entire system) 4](#_Toc66789274)

[4.2 Mid-Level Design of Entity Module 5](#_Toc66789275)

[4.3 Detailed Class Design of Entity Module 7](#_Toc66789276)

[5 Process View of Entity Module 8](#_Toc66789277)

[6 Use Case View 8](#_Toc66789278)

Change History

**Version:** 0.1

**Modifier:** Arianne Fennell, Christian Young, Hamidreza Ghasemi

**Date:** 03/15/2021

**Description of Change:** Creation of base Entity classes and functionality.

3/16/2021 – Hamidreza - Updated use case view.

# Introduction

This document describes the architecture and design for the ShadowStalk application being developed for Particle Interactive. ShadowStalk is a heavily stylized asymmetrical co-op horror game where the players must work together to gather keys with the goal of helping lost souls escape the confines of Limbo.

The purpose of this document is to describe the architecture and design of the Entity 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.

# Design Goals

The design priorities for the Entity Module system are:

* The design should be open to and encourage future iterations without imposing unnecessary complications on external modules.
* The design should be network ready and should facilitate network implementation in its child classes.
* The design should be compatible with the classes inheriting from it, without breaking any extra logic.

# System Behavior

The Entity Module is built from a single Entity parent that inherits from Unreal Engine’s APawn. The Entity Module is responsible for passing off movement functionality from the custom movement component to the Entity child classes. The Entity class is an abstract class, which means it will never appear alone and is simply meant as a template class for all of the game’s entities.

# Logical View

This module has fairly dynamic interactions with the game world and other modules of its own type.

It receives the player input from their controller, then informs the movement component over the network, which moves the root collider. It also handles the rotation of the camera and the collider using input from the mouse.

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.

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

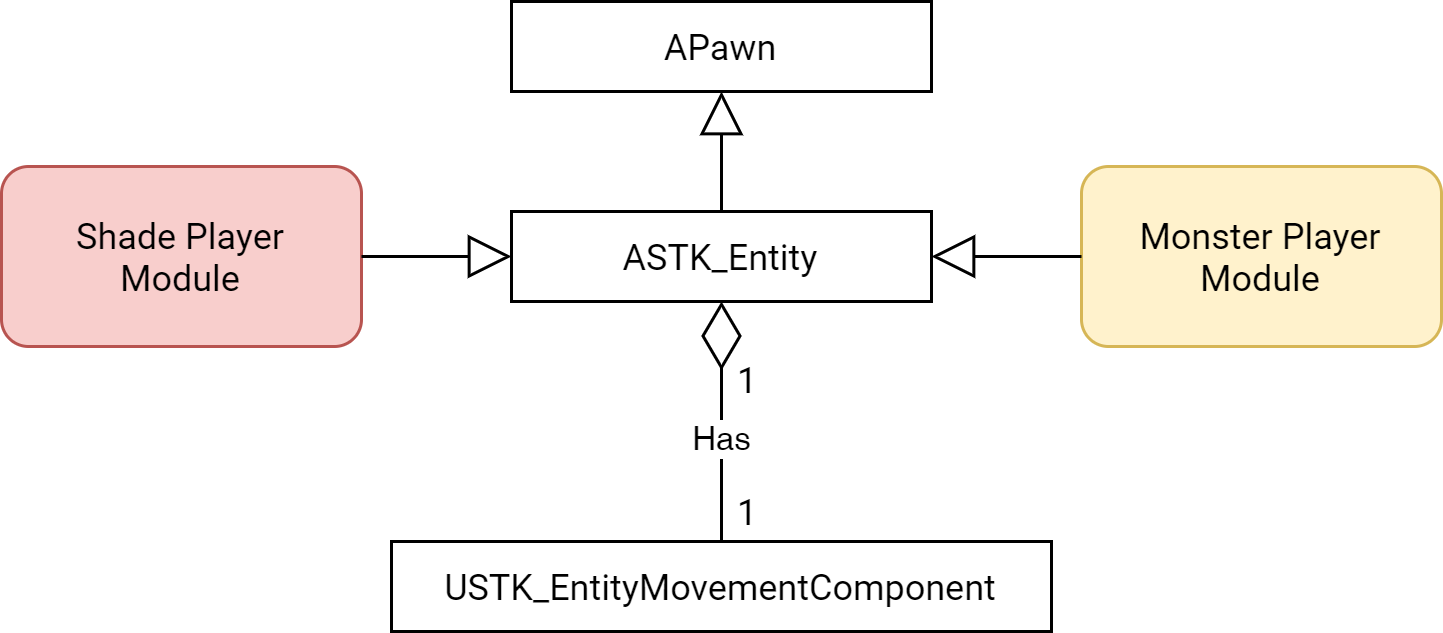
Graphical user interface

Description automatically generated

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

* The **Entity Module** is a wrapper around Unreal Engine’s standard Pawn class. It holds the game’s custom movement component.
* The **Shade Player Module** is the main driver of the Shade players’ character. It reacts to user input and Monster attacks, interacts with the Inventory Module, and controls the Shade’s eyes.
* The **Inventory Module** contains information related to what a Shade player is currently carrying (keys, items, etc.)
* The **Match GameMode Module** dictates the rules of the game, manages the game state, spawners, pickups, and doors.
* The **User Interface Module** is responsible for the creation of a series of interfaces and screens that allow players to interact with the game world.

## Mid-Level Design of Entity Module

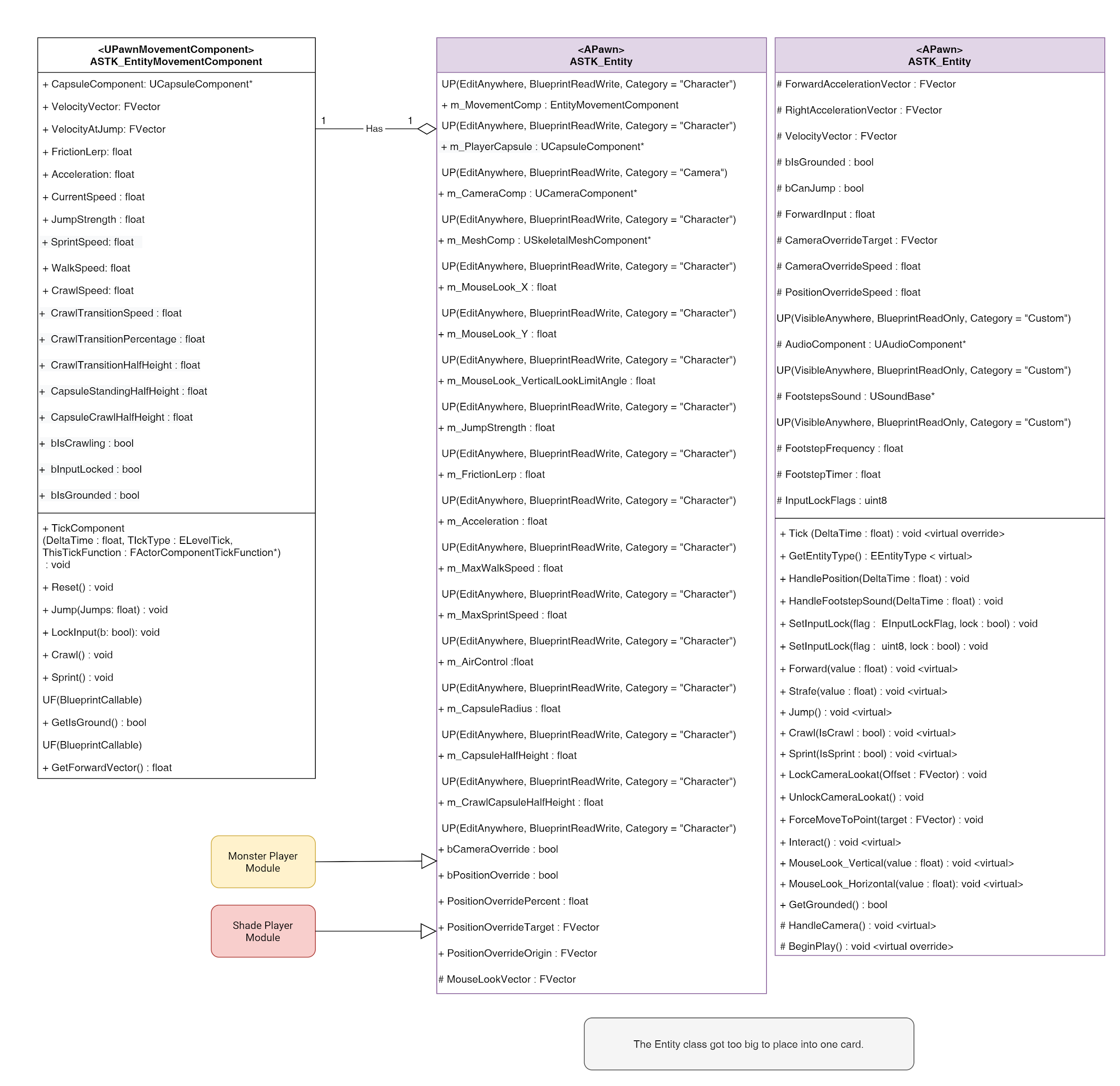


**ASTK\_Entity:** The base class for all Entities. This class is responsible for handling any shared functionality between the Shade and Monster classes. It provides a mesh and a collider, and links them to the game’s custom Movement Component to allow for movement. It also handles MouseLook, Sprinting, Crouching and Interacting. It also provides helper functions for overriding the camera and position. A large chunk of networking happens in this class as well so far as it pertains to movement.

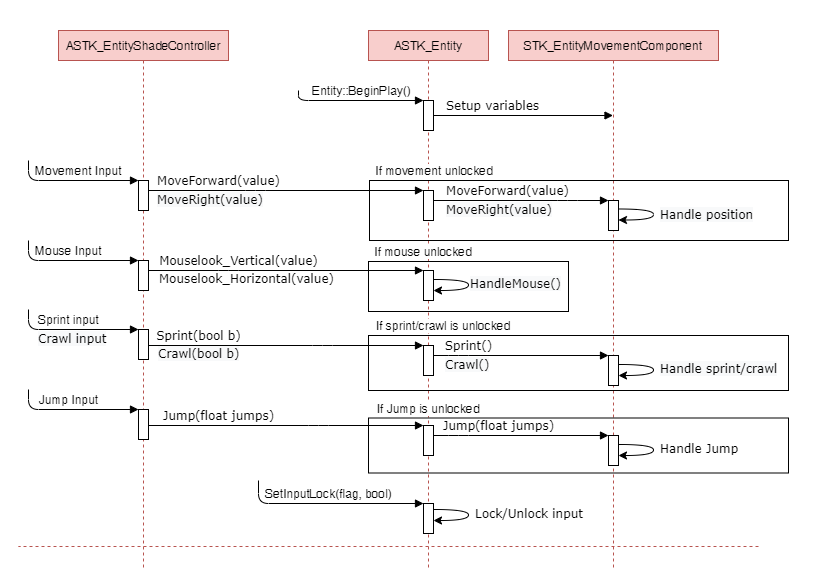
**USTK\_EntityMovementComponent:** A custom movement component with ground checking, movement handling, jumping and crouching. It reads the input acceleration vector and calculates a desired movement per frame. It then applies that movement value after resolving all collisions. This component partially handles movement replication.

## Detailed Class Design of Entity Module

Please view **UML/EntityUML.png** for the high-resolution image.



# Process View of Entity Module



# Use Case View

Since this module is abstract, there’s no particular use case for it beyond inheriting from it. We’ll look at the Shade Module document for an example of this class in action.

Simply inherit from the class, ensure you have a constructor, and override Interact() and GetEntityType():



In the .cpp of the inherited class, we use the constructor to set up the variables in Entity:



