**ShadowStalk Inventory Module**

Architecture/Design Document

**Table of Contents**

[1 Introduction 3](#_Toc131876418)

[2 Design Goals 3](#_Toc131876419)

[3 System Behavior 4](#_Toc131876420)

[4 Logical View 4](#_Toc131876421)

[4.1 High-Level Design (Architecture) 5](#_Toc131876422)

[4.2 Mid-Level Design 6](#_Toc131876424)

[4.3 Detailed Class Design 7](#_Toc131876425)

[5 Process View 7](#_Toc131876426)

[8 Use Case View 8](#_Toc131876429)

Change History

**Version:** 0.1

**Modifier:** Jeffrey Armstrong

**Date:** 03/09/2021

**Description of Change:** Creation of base Inventory Component for game pickups.

# 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 Interface 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 Inventory Component system are:

* The design should be open to future development and iterations if needed by the developers.
* The design should have as little complexity as possible in order for it to be more accessible and organized.
* The design should be optimized and openly compatible so items that are added to it at any point during development have as little conflicts in the process as possible.

# System Behavior

The Inventory Component’s purpose is to store and manage collected items with the player being able to toggle between each collected item in the order of when each item was collected.

# Logical View

The Inventory Component in its current state is completely isolated from all other modules in the system. Future development of the Inventory Component will have it working with the Shade Player Module so when the Entity Shade picks up an item, it will be sent to and managed in the Inventory Component.

The STK\_PickupBase works closely with the Inventory Component because it manages what a pickup is and the different types of pickups there are while the Inventory Component manages how much of each pickup the Player has.

The Inventory Component will also work with the STK\_EntityShadeController class so the Shade can toggle and use the different items that are contained and managed in the Inventory Component. Once used, there will be a function to remove the used item from the Inventory Component.

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

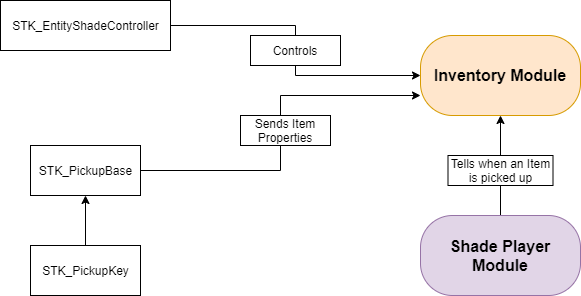
Graphical user interface

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, 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** is responsible for managing the game state, item 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 Inventory Module



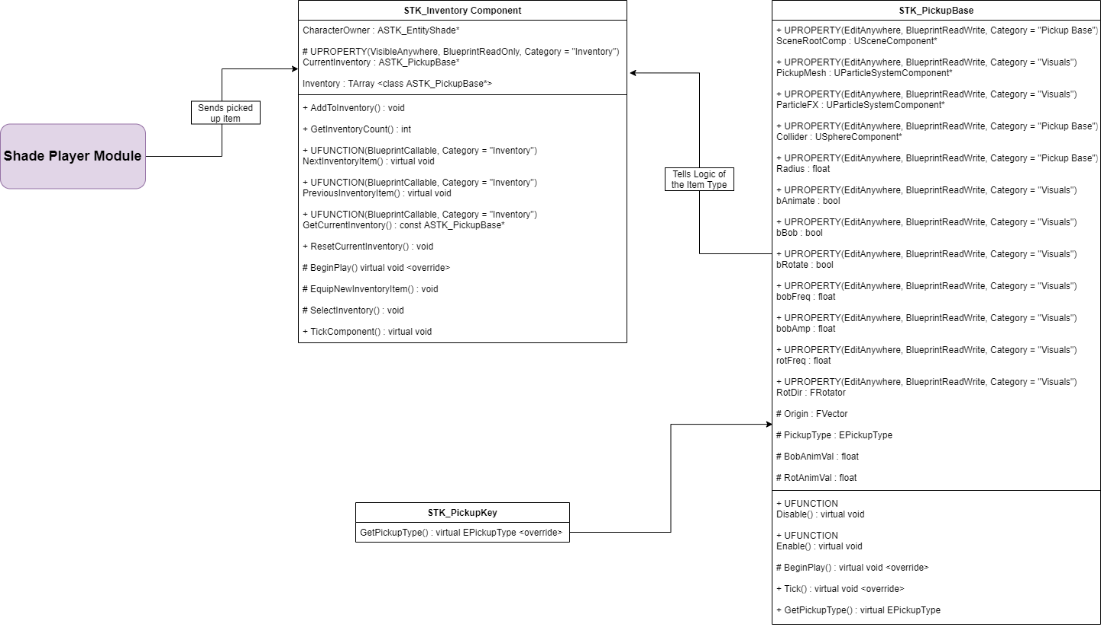
**STK\_PickupBase:** The base class for all pickups in the game. It handles the Collision, Mesh, an enum list for all the different types of Pickups, a Particle System, and variable for animation of the Pickups rotating, and bobbling.

**STK\_PickupKey:** Tells the STK\_PickupBase of it’s identity as a Key type item in the enum list.

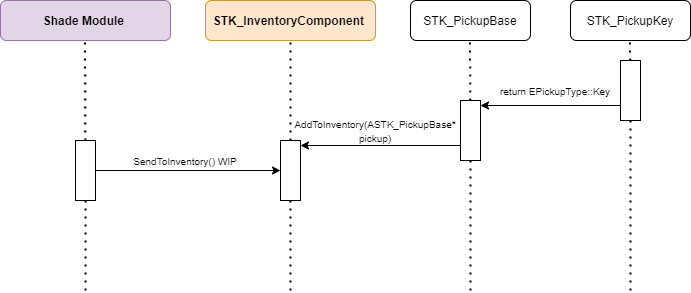
**STK\_ShadeController:** The controller of the Shade Player and controls the use of the Inventory Component as well as tells the Component that an item has been picked up by the Player.

## Detailed Class Design of Inventory Module

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



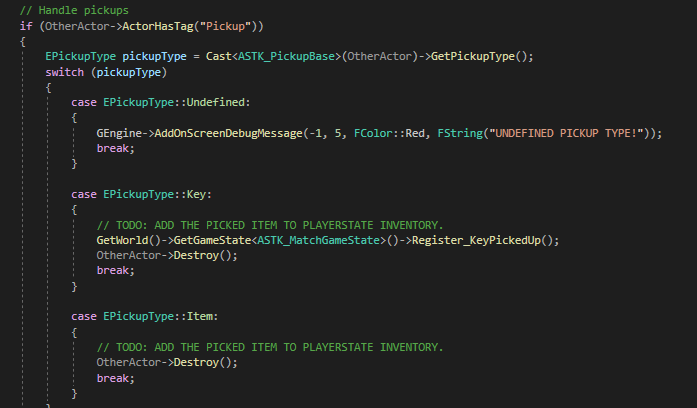
# Process View of Inventory Module



NOTE: The following diagram contains some concept that are not implemented and may change in future builds.

# Use Case View

When the Player picks up an item, STK\_EntityShade class will send it to STK\_InventoryComponent to be managed. (WORK IN PROGRESS)



STK\_InventoryComponent will then add the item to the Inventory to be saved and managed for whenever the player wants to use it. (WORK IN PROGRESS)

When the item is being added to the Inventory Component, STK\_PickupBase will handle the specific information of said item that was picked up and call from the class that the specific item. (WORK IN PROGRESS)

