**ShadowStalk Match GameMode Module**

Architecture/Design Document

**Table of Contents**

[**1 Introduction 3**](#_Toc66794963)

[**2 Design Goals 3**](#_Toc66794964)

[**3 System Behavior 4**](#_Toc66794965)

[**4 Logical View 4**](#_Toc66794966)

[**4.1 High-Level Design (Architecture of the Entire system) 5**](#_Toc66794967)

[**4.2 Mid-Level Design of Match GameMode Module 6**](#_Toc66794968)

[**4.3 Detailed Class Design of Match GameMode Module 7**](#_Toc66794969)

[**5 Process View of Match GameMode Module 7**](#_Toc66794970)

[**6 Use Case View 8**](#_Toc66794971)

Change History

**Version:** 0.1

**Modifier:** Hamidreza Ghasemi

**Date:** 03/11/2021

**Description of Change:** Initial creation of the system.

# 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 Match GameMode 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 Match GameMode Module system are:

* The module is expected to be network-ready.
* This module should hold game-sensitive information in an organized, accessible manner.
* The module is expected to be flexible to any future changes in gameplay elements.

# System Behavior

The Match GameMode Module is in charge of spawning players properly, and setting up the level by selecting a preset number of spawn points to spawn keys from, and selecting a random exit door from a list of available doors. It then Informs the GameState of the results of the randomization.

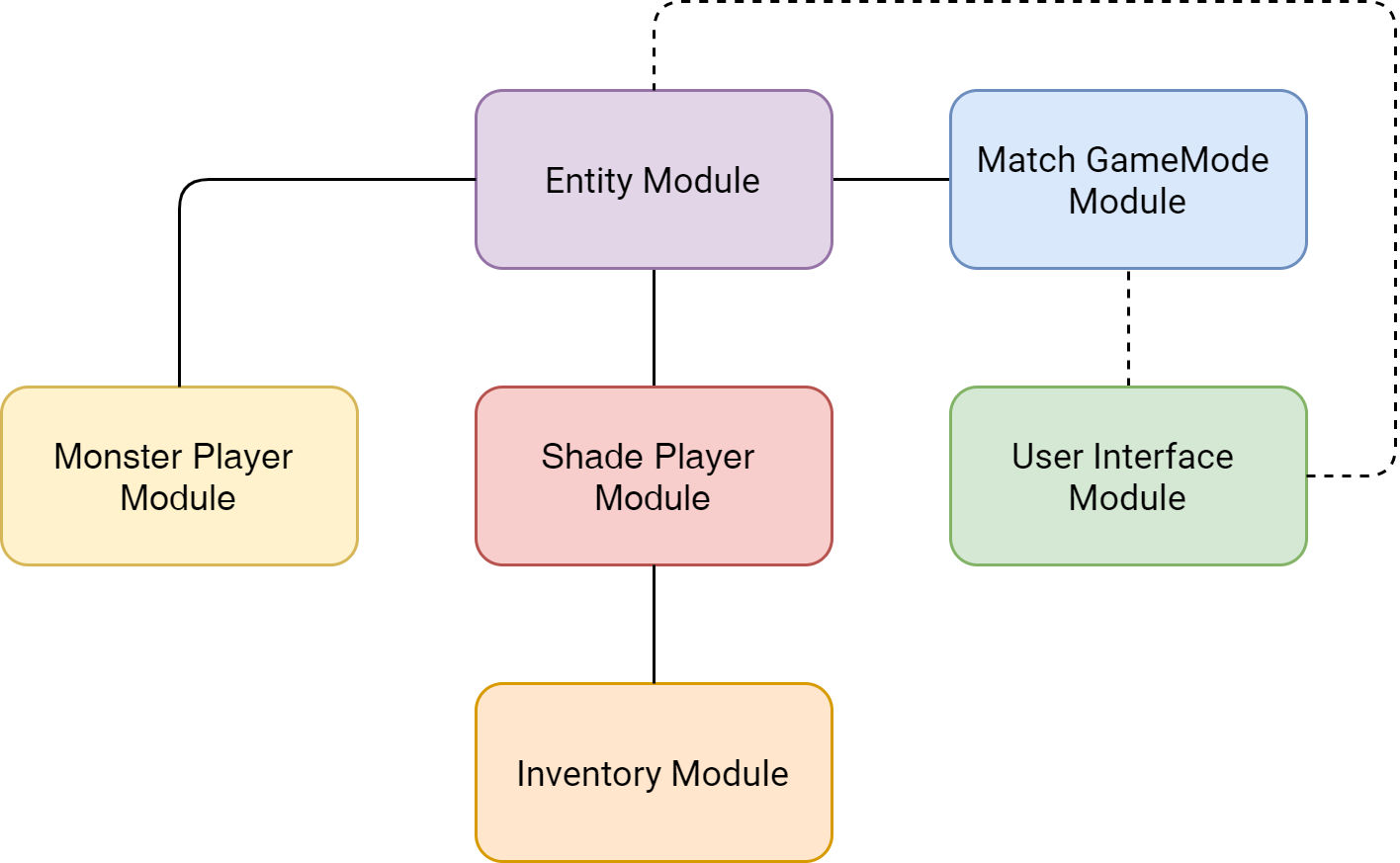
# Logical View

As this module is written nearly exclusively for the sake of the server’s use, it tends to have largely static interactions with other elements in the project.

The majority of its “dynamic” interactions are relegated to scene elements such as spawn points and exit doors registering themselves with the class so that it can manage them.

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.

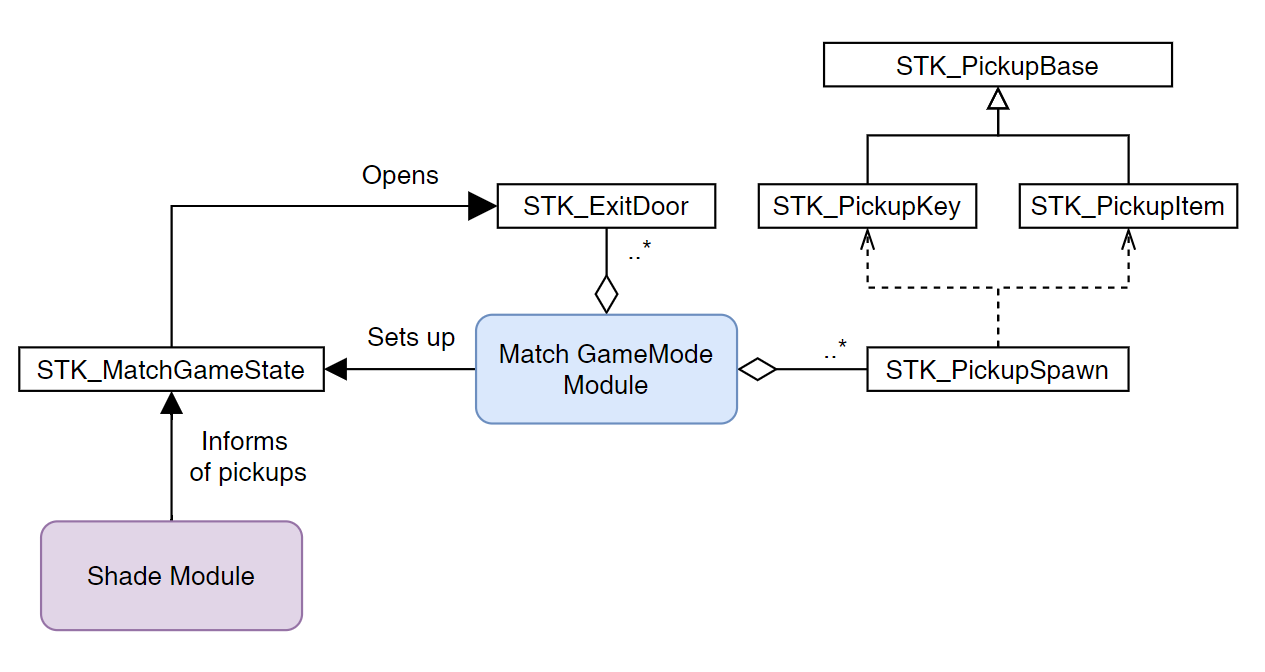
## High-Level Design (Architecture of the Entire system)



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

1. The **Entity Module** is a wrapper around Unreal Engine’s standard Pawn class. It holds the game’s custom movement component.
2. 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.
3. The **Inventory Module** contains information related to what a Shade player is currently carrying (keys, items, etc.)
4. The **Match GameMode Module** is responsible for managing the game state, item spawners, pickups, and doors.
5. 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.

## 4.2 Mid-Level Design of Match GameMode Module

****

**STK\_PickupBase:** Base class for pickups in the game. It has an enum to identify the pickup type, a Collider, a Mesh, a Particle System, and variables for basic bobbing and rotation animation.

**STK\_PickupKey:** The primary derivative of the STK\_PickupBase class, overriding its type to identify the pickup as a key type item.

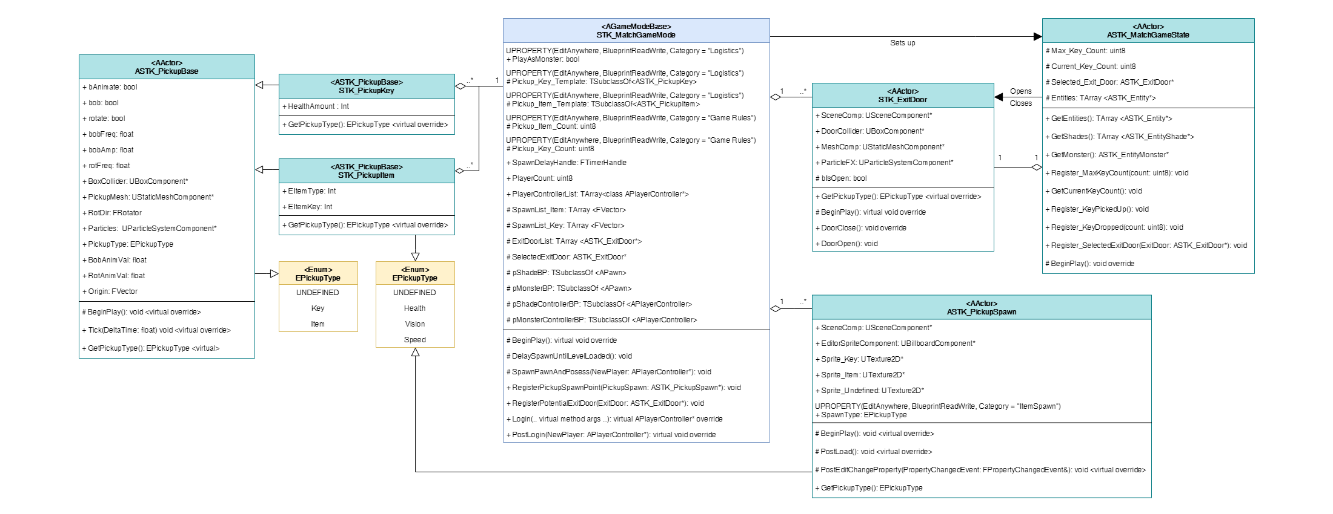
**STK\_PickupItem [WORK IN PROGRESS]:** Intended to be the base class for items later to be added to the game, each with its own unique effects.

**STK\_PickupSpawn:** This class is an interface for the level designers to place item and key spawn points around the map with ease. It has an enum identifier for what pickup object it’s intended to spawn. On the server, this class registers itself with the Match GameMode Module, which decides which spawn points to randomly spawn keys on. A billboard sprite updated through code will change its icon in the editor for more readability.

**STK\_ExitDoor:** The exit doors of the map. On the server, these will register themselves with the Match GameMode Module, which decides which one is to be opened when enough keys are picked up. It has a Mesh, a Collider and a Particle System. When opened, the door disables its collider and turns off its Particle System to visually signal the path now being open.

## Detailed Class Design of Match GameMode Module

View **UML/MatchGameModeUML.png** for the high-resolution image.

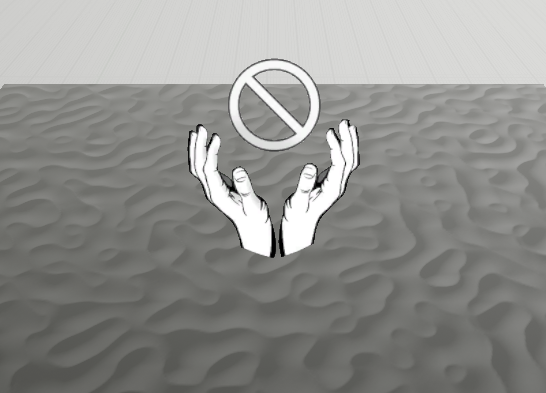


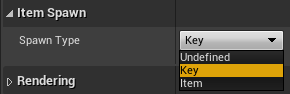
# Process View of Match GameMode Module

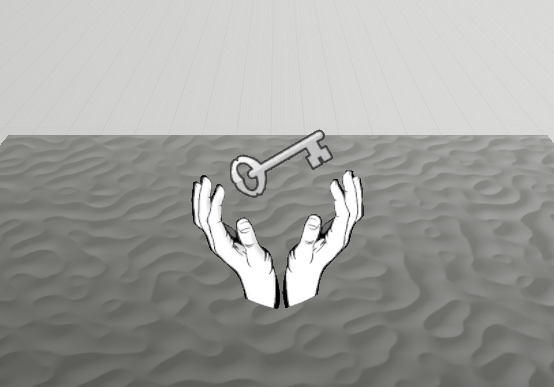
# Use Case View

**Spawning Keys:**

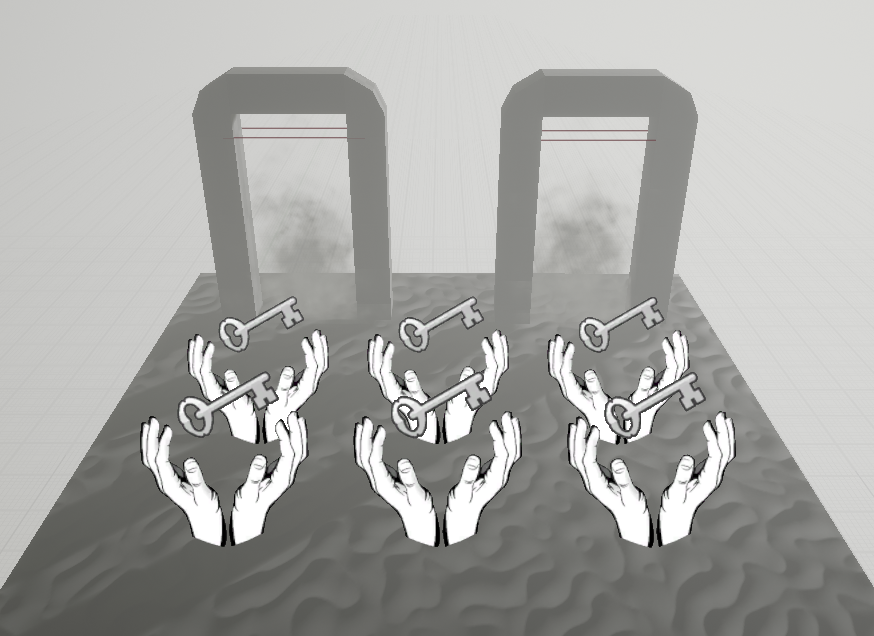
**Step 1:** Place a few pickup spawners. In the details tab in the engine, set their type to Key. Their icons will update.

****

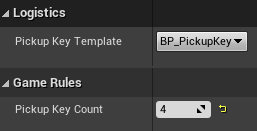
****

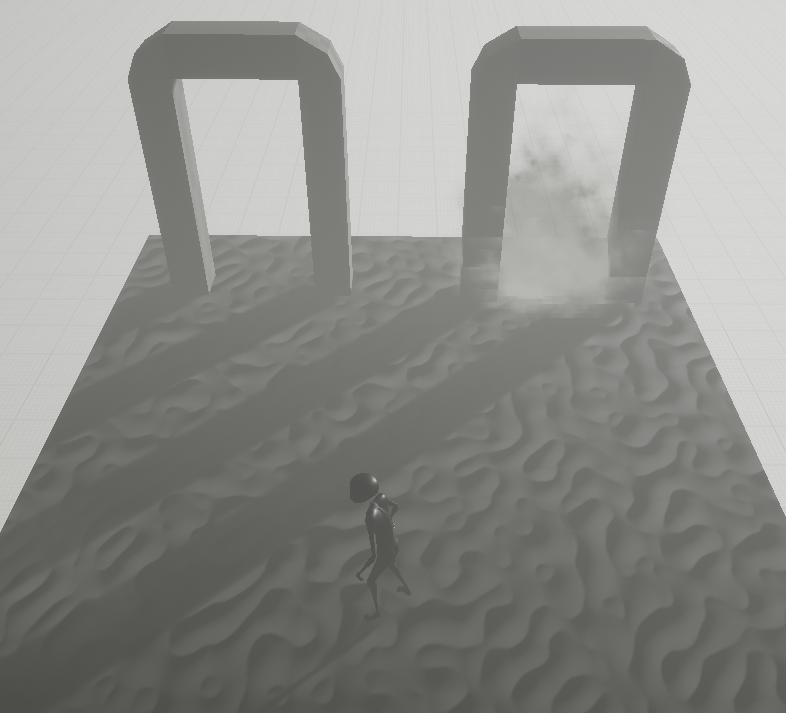
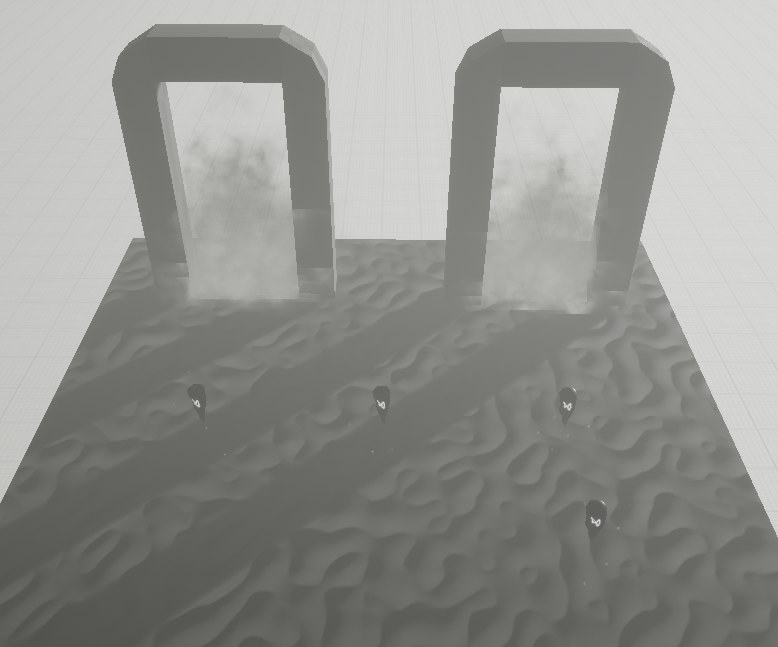
****

**Step 2:** Place some doors.



**Step 3:** In the MatchGameMode blueprint, set the key template blueprint you want to spawn, as well as the number of keys you want to spawn on the provided spawn points.



**Step 4:** Run the game. The module and its components automatically take care of spawning keys and opening/closing a random door when they’re all picked up.

**Spawning Items:**

**[WORK IN PROGRESS] these will follow a pattern similar to spawning keys. Updates to follow.**