**ShadowStalk User Interface 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 9](#_Toc131876425)

[5 Process View 10](#_Toc131876426)

[6 Use Case View 13](#_Toc131876429)

Change History

**Version:** 0.1

**Modifier:** Arianne Fennell

**Date:** 03/01/2021

**Description of Change:** Creation of base User Widgets for Main Menu screen.

**Version:** 0.2

**Modifier:** Arianne Fennell

**Date:** 04/16/2021

**Description of Change:** Implemented Pause, Invite and Confirmation widgets.

# 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 User 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 User Interface Module system are:

* The design should be highly open to future iterations and changes with possibility of rearranging widget hierarchies.
* The design should minimize complexity and development effort by being organized and accessible.
* The design should be optimized to facilitate the creation of new widgets without impeding on the construction of new game features.

# System Behavior

The User Interface Module is created by implementing game-specific user widgets in the game’s Instance. Each custom user widget is derived from a custom UUserWidget subclass. By default, this allows for new user widgets to be built with the same base standard for the ShadowStalk application. Organizing the creation of widgets in this manner ensures that the system remains clean and functional throughout all future iterations.

# Logical View

The inner workings of the User Interface Module are completely isolated from all other modules in the system with an exception being made when it comes to adding the widgets to the viewport. The only aspects of the game’s user interface that are meant to be accessed directly from the player or the level are the user widgets’ Setup and Teardown functions.

These functions are what adds and removes the specified widget from the viewport while also modifying the player’s input controls based on whether they are currently meant to be navigating an interface or the game itself.

Each user widget is designed to handle one job which helps to keep the module organized. For example, the Pause Menu user widget would need to be accessed by the player in-game. When called on by the player controller, the Pause Menu user widget will be added to the viewport on top of the player’s HUD as a separate visible interface.

## 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 User Interface Module



**USTK\_GameInstance:** The game’s main manager class. When it comes to the User Interface Module, the GameInstance is responsible for the implementation and switching of the main UI screens and levels of the game, as well as implementing functions that handle the game’s integration with Steam.

**ASTK\_MainMenuLevel:** The class is a child of Unreal Engine’s LevelScriptActor and allows for the Level’s Blueprint to be directly modified in C++. The class is mainly responsible for attaching the UWMainMenu class to the Main Menu Map.

**ASTK\_MainMenuGameMode:** The MainMenuGameMode class inherits from Unreal Engine’s GameModeBase and is only responsible for setting the MainMenuLevel’s default pawn.

**USTK\_UserWidget:** The base UserWidget class that is a child of Unreal Engine’s UUserWidget. All the game’s unique user widgets will inherit from this class. This parent class holds two important functions that are responsible for the Setup and Teardown of each widget in relation to the game. The class will also never be used alone and is simply meant as a template class.

**USTK\_UWMainMenu:** Once called for setup by the MainMenuLevel class through the GameInstance class, this user widget will be added to the viewport and make up the title screen of the game. The UWMainMenu class is responsible for enabling the player to load into the game map, giving them access to the options and credits menu, as well as enabling them to quit the game.

**USTK\_UWOptionsPanel [WORK IN PROGRESS]:** Once called for setup by the UWMainMenu class through the GameInstance class, this user widget will be added to the viewport and make up the options panel of the main menu. The UWOptionsPanel class is responsible for enabling the player to change specific user settings like keybindings, sound and resolution.

**USTK\_UWCreditsPanel:** Once called for setup by the UWMainMenu class through the GameInstance class, this user widget will be added to the viewport and make up the credits panel of the main menu. The UWCreditsPanel class is responsible for displaying the project’s developers and their role.

**USTK\_UWPauseMenu:** Once called for setup by the Entity class through the GameInstance class, this user widget will be added to the viewport and make up the pause screen of the game. The UWPauseMenu class holds instances of UWOptionsPanel, UWInviteMenu and UWConfirmQuit and is responsible for enabling the player to return to the main menu, access the game’s options and invite players to the current session.

**USTK\_UWInviteMenu:** The UWInviteMenu is called on by the player’s interaction with the UWPauseMenu and is setup by this same class. Although it is not seen in code, the UWInviteMenu holds an instance of the UWFriendsList class. This displays the player’s list of Steam friends in a way that is readable and accessible.

**USTK\_UWFriendsList:** The UWFriendsList is responsible for sending friend data from the player’s Steam friends list to a newly created instance of the UWFriendRow. If the data exists, the player’s friends’ display name, online presence and unique net ID will be displayed in the widget.

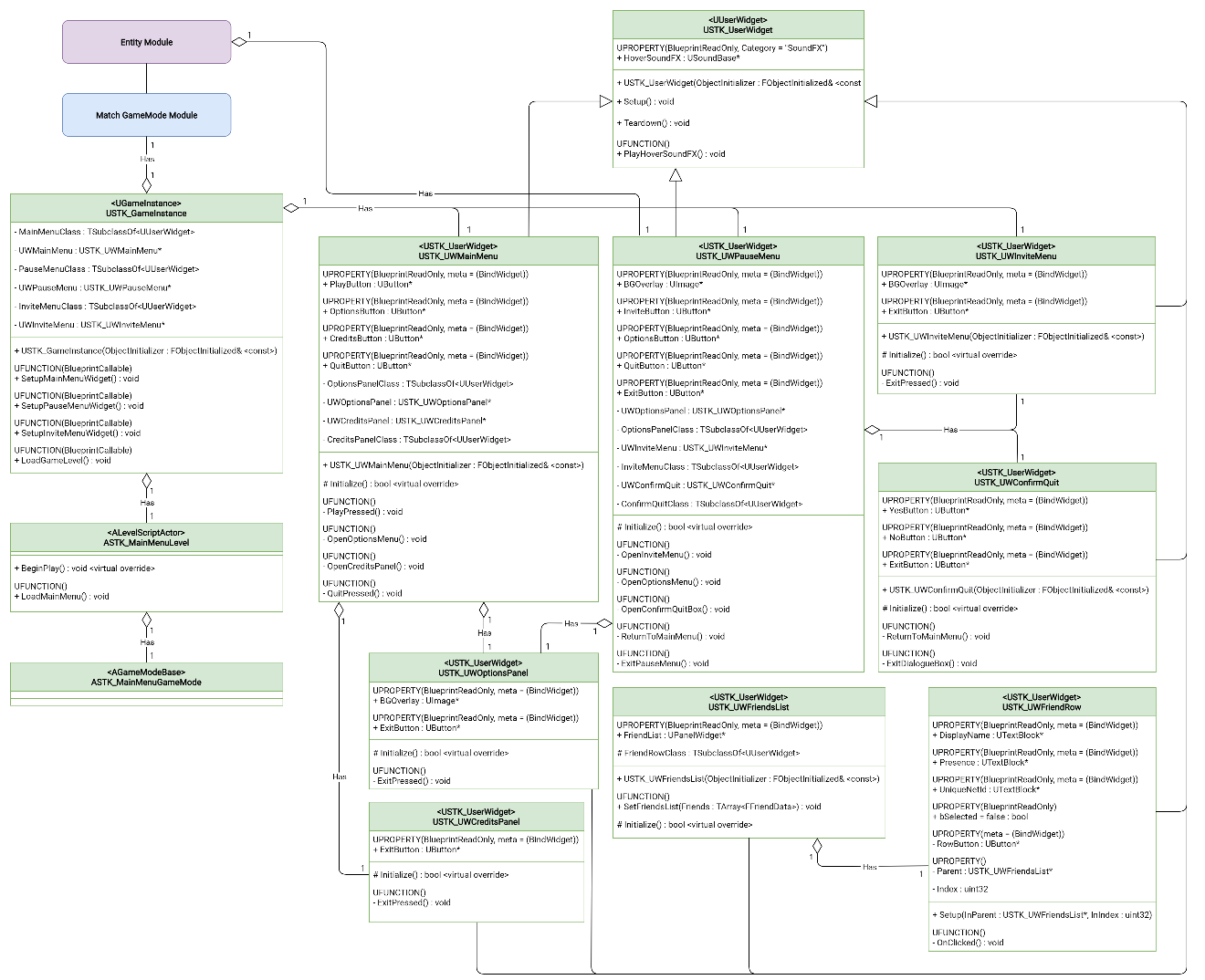
**USTK\_UWFriendsRow:** The UWFriendsRow is primarily designed as a custom button widget that allows players to send an invite to the selected Unique Net ID once interacted with. This widget is created and pushed into a list for every instance of a friend that is found in the player’s Steam friends list.

**USTK\_UWConfirmQuit:** The primary purpose of the UWConfirmQuit widget is to act as a confirmation dialogue box that appears when the player interacts with the “Quit” button in the UWPauseMenu widget. It forces the player to confirm their decision before leaving the current session.

## Detailed Class Design of User Interface Module

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

**Note:** Functions, variables and structs necessary for Steam integration are not included to save on space. Tilan said this was okay.



# Process View of User Interface Module

**Main Menu User Widget:**

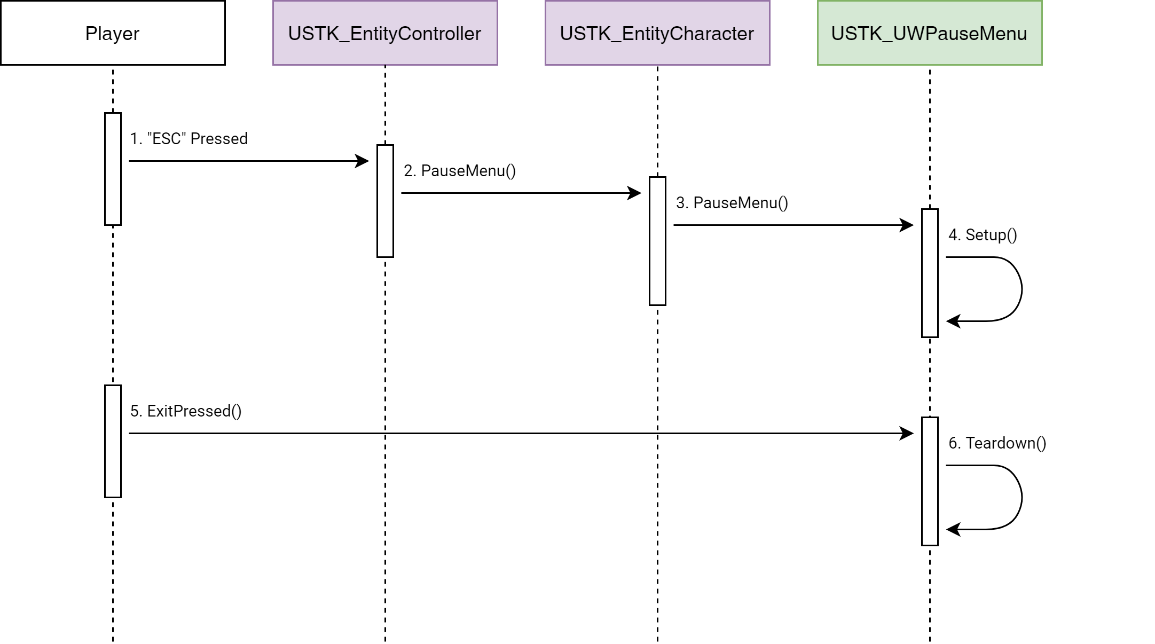
The Main Menu User Widget in the game world follows this sequence when being added to and removed from to the game’s viewport.

Diagram

Description automatically generated

**Pause Menu User Widget:**

The Pause Menu User Widget in the game world follows this sequence when being added to and removed from to the game’s viewport.

****

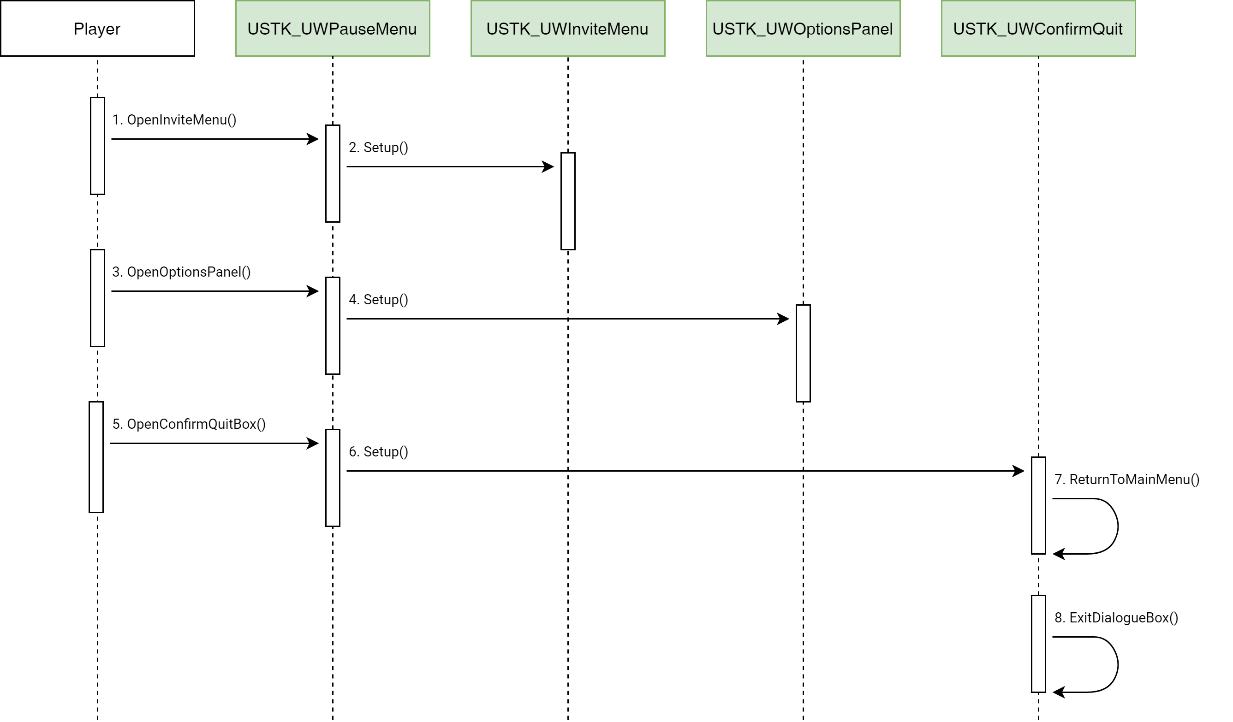
**Main Menu Interactions:**

The Main Menu interface follows this sequence when being interacted with.



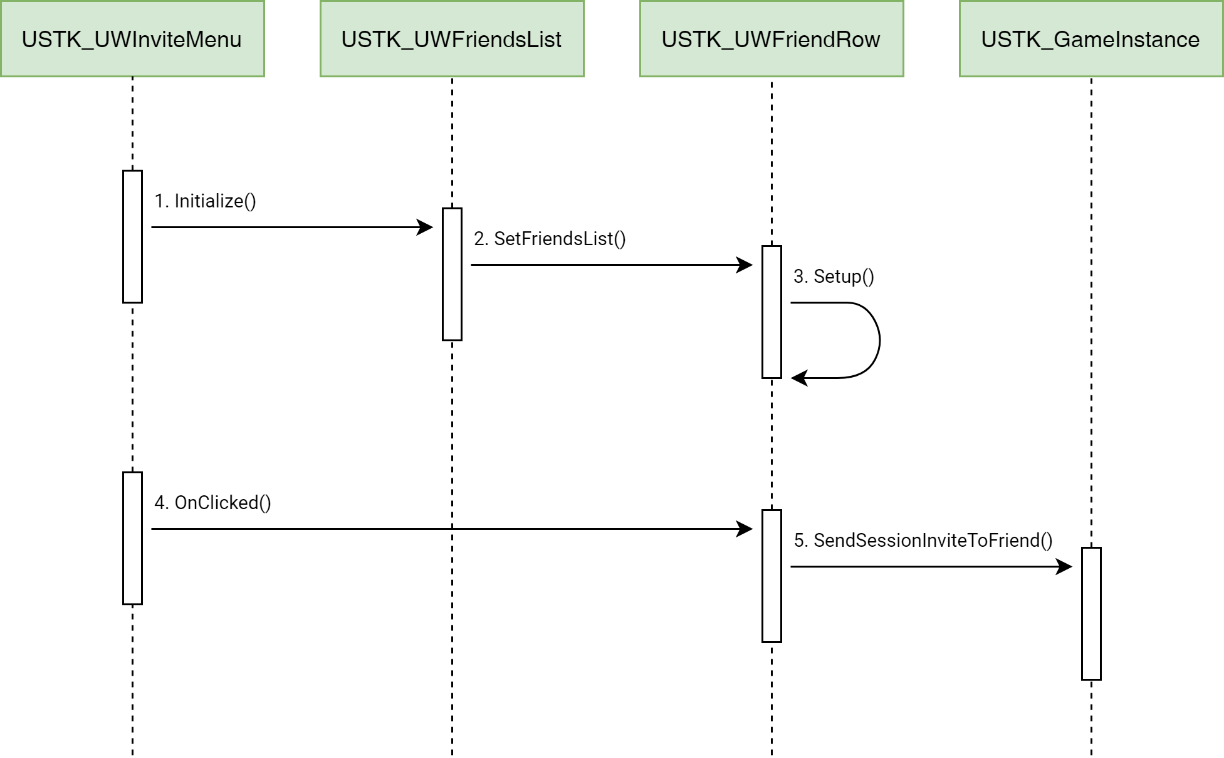
**Pause Menu Interactions:**

The Pause Menu interface follows this sequence when being interacted with.



**Invite Menu Interactions:**

The Invite Menu interface follows this sequence when being interacted with.

****

# Use Case View

The following diagram represents the UI flow. This flow diagram is meant to visually represent the interface screens in the ShadowStalk application and how the user is going to get from one screen to another.

**Note:** This is a work in progress, as the Options panel is currently in development.

