Post Arcana / Door System

Architecture/Design Document

**Table of Contents**

1. Introduction

2. Design Goals

3. System Architecture

4. Mid Level Design

5. Detailed Class Design

6. Process View

7. Use Case View

**Change History**

**Version:** 0.1

**Modifier:** Sebastian Comette

**Date:** 04/09/2022

**Changes:** Design Document Created

**Version:** 0.2

**Modifier:** Sebastian Comette

**Date:** 04/11/2022

**Changes:** Design Document Finished

1. Introduction

This document describes the design and architecture of Post Arcana by the Hex Decimals. Post Arcana is a single player open world RPG, set in a post-apocalyptic small Canadian town after the introduction of magic to the real world.

The purpose of this document is to define the architecture and design of the stat systemin a manner that assists the interests of all major stakeholders. The major stakeholders and their interests are as follows:

* Developers: A design that is easy to implement that minimizes complexity
* Project Manager: A design that can be easily divided among the skills of the programmers
* Maintenance Programmers: A design that can be improved upon easily

1. Design Goals

The design goals for the stat system are as follows:

* The design must minimize complexity
* The design must allow for easy access to abstract and concrete stats
* The design should be easy to amend and balance

1. System Behavior

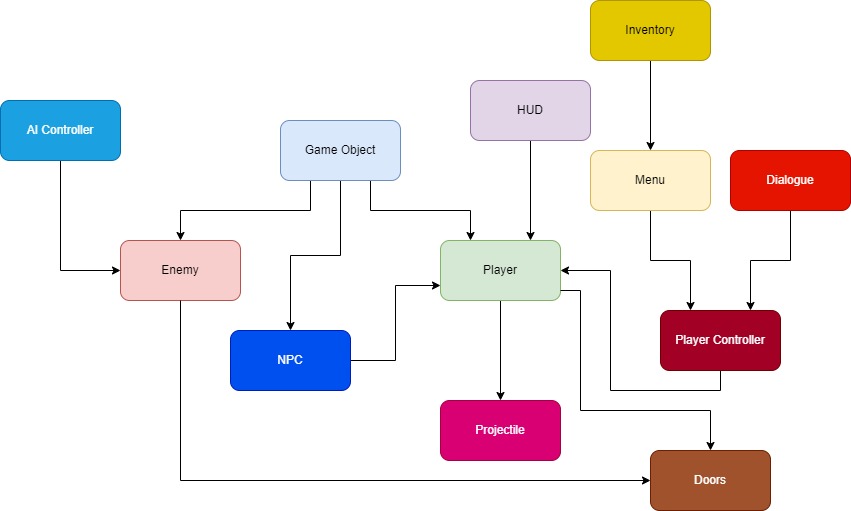
The door system consists of interactions between the player and multiple objects in order to gain access to a certain area. The player may interact with the door itself or other objects that toggle the doors active state. Each type of a door as a specific interaction required to open it. The basic door requires a simple interaction between the player and the door to toggle the doors active state. The button door requires the player to interact with a specific button in the level to toggle the doors active state. The trigger door can be interacted with in 1 of two ways depending on the set up of the level, these interactions consist of either the player entering a trigger volume, or defeating a certain enemy that is linked to the door. Last the locked door requires the player to pick up a specific key that is used to unlock a specific door in the level.

1. 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.

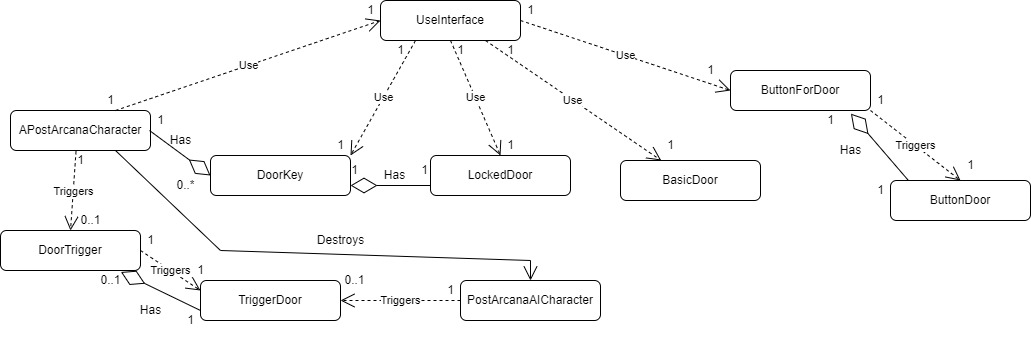
In this section, the modules of the system are first expressed in terms of high-level components (architecture) and progressively refined into more detailed components and eventually classes with specific attributes and operations.

4.1.System Architecture



* GameObject: Base class for the game entities (Players and enemies). Stores abstract and concrete stats.
* Enemy: The class used for enemy-only stats and components
* AI Controller: Moves and controls the enemies
* Player: The class used for player-only stats and components
* Player Controller: Receives player input and controls the player
* Projectile: Is launchd by the player, and deals damage to Game Objects
* Menu: Places a UI element on the screen given a specific input
* HUD: Shows the player important stats\
* Doors: This Module consists of multiple door classes that all provide a different method of access control to an area in the level.

4.2Mid-Level Design



**APostArcanaCharacter(Player):** A game object controlled through the player input that can interact with doors, pick up keys, and buttons, through the Use Interface. This game object can also collide with triggers and destroy AI characters.

**AUseInterface:** A interface class that is the bridge between the player and the object they are interacting with. The player input interacts with this use interface and then the use interface calls the implementation on the object that was interacted with.

**AButtonForDoor:** An actor that the player can interact with to open a specific button door. This button can be used to both open and close button doors.

**AButtonDoors:** An actor that provides access control to an area of the level. This actor has its current state (open or closed) toggled when the player interacts with the a specific button.

**ADoorKey:** An actor that can be picked up by the player. Once picked up this actor stays with the player and can toggle the state of any locked door that requires the this key.

**ALockedDoor:** An actor that provides access control to an area of the level. This actor can only be toggled by the player when the player has acquired the proper key.

**ABasicDoor:** An actor that provides access control to an area of the level. This actor can be toggled when the player interacts with it. No other item is required to toggle the active state of this actor.

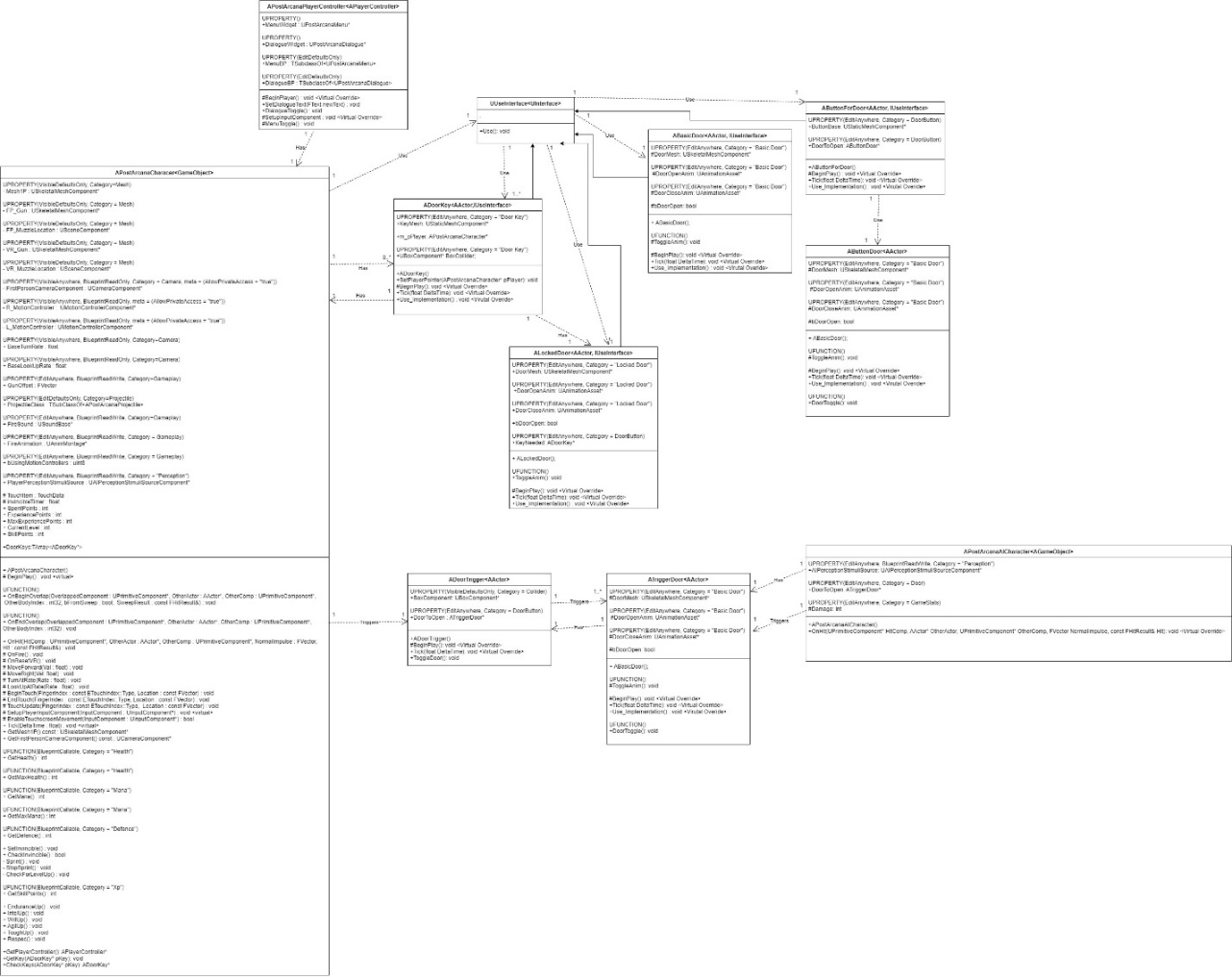
**ADoorTrigger:** An actor that acts as a collision volume. When the player enters this volume, this actor toggles the active state of a specific trigger door

**APostArcanaAICharacter:** A game object that is an enemy to the player. This game object can be destroyed by getting hit with a projectile shoot by the player. This game object can damage the player by hitting them. This game object may or may not have a trigger door actor attached to it. If this game object does have a trigger door attached to it, on death the game object will toggle the active state of the trigger door once.

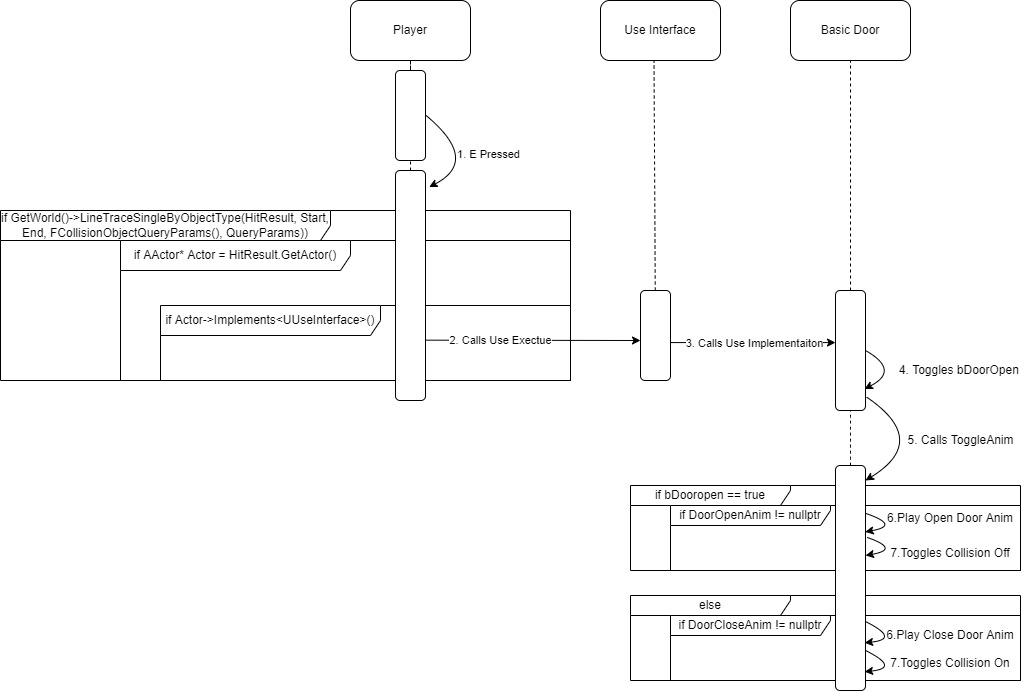
**AtriggerDoor:** An actor that provides access control to part of the level. This actor can have its active state toggled either by a door trigger being walked into by the player or by an Ai Characters death.

.

4.3Detailed Class Design

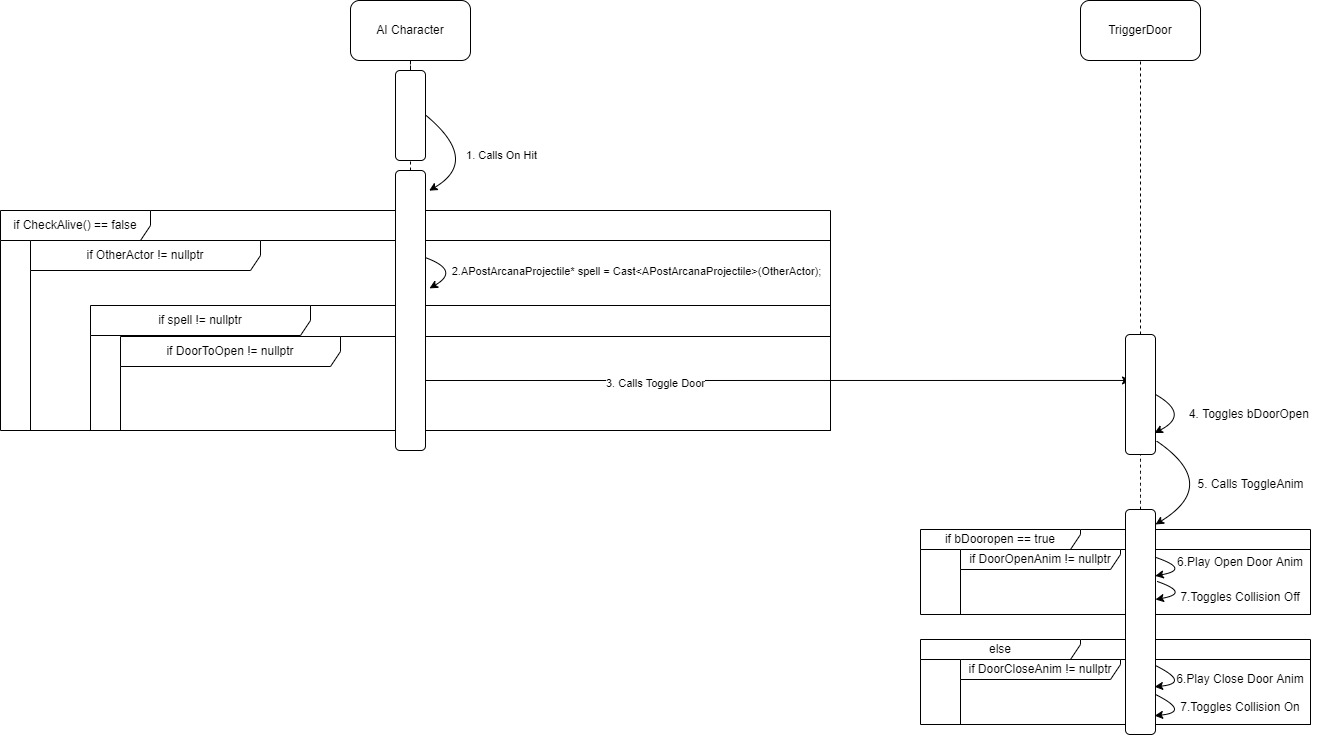


1. Process View
   1. Basic Door

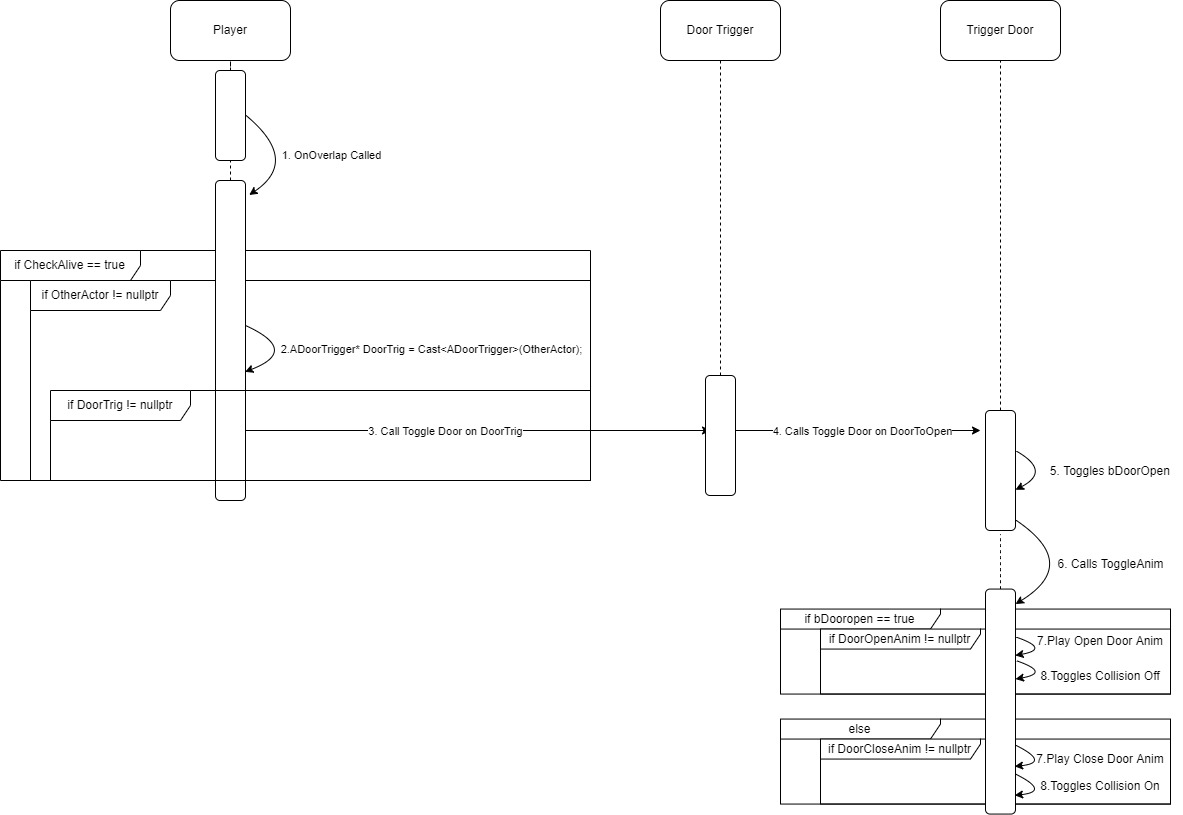


This basic door provides access control to the level. It can be opened by interaction from the player through the use interface. If the player presses E, or the input that has been bound to the interact function, the interact function will be called. When the interact function is called the player will do a ray trace in the direction they are facing. If the ray trace hits an actor, the player will then check if this hit actor has an implementation for the use interface. If the actor does have an implementation of the use interface, then the use interface will call the implementation function on the hit actor, in this case the hit actor is the basic door. The basic door will toggle a Boolean named bDoorOpen and then call the ToggleAnim function on itself. The ToggleAnim function will check if the door is supposed to be open, and if there is a door open animation, if both are true then the function will play the open door animation and remove collision on the door. If bDoorOpen return false then the door will check if there is a close door animation, if this second check returns true the door will then play the close door animation and toggle collision on.

* 1. Target Door

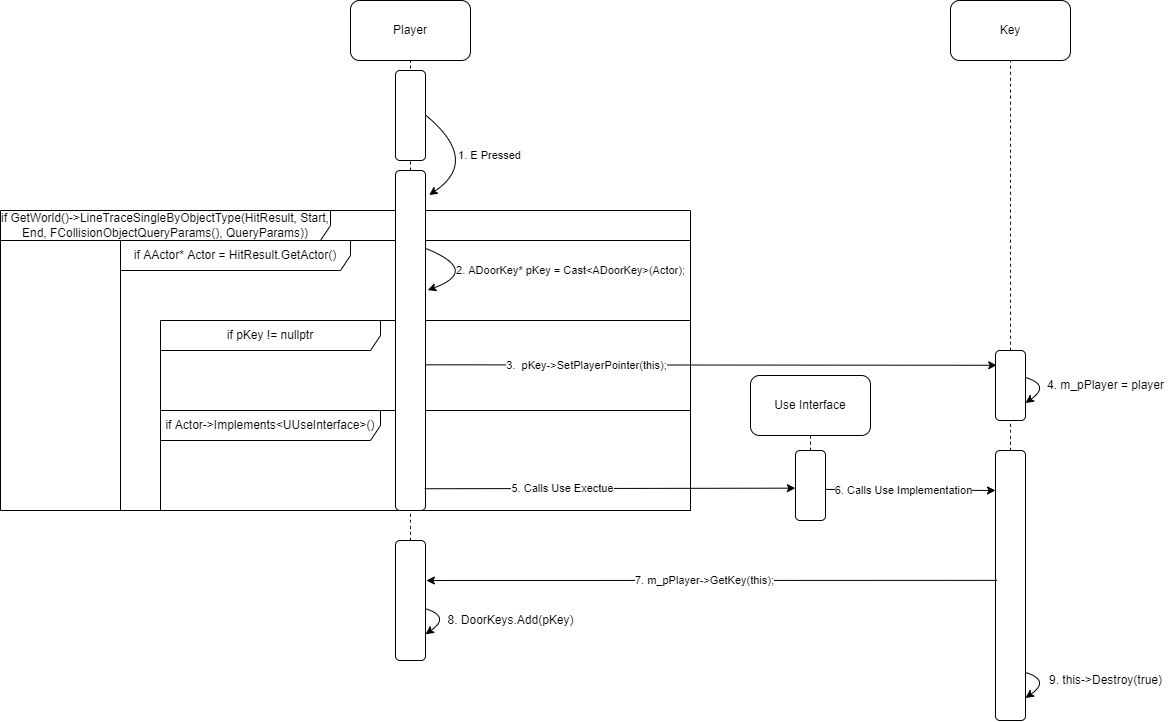


The target door is a trigger door without a trigger but rather an AI character that will toggle the door instead of a trigger. When the AI character is hit, it will check if the AI character is alive, if the AI character is no longer alive it will then check to see what it was hit by. This requires two sub-checks, the first is to ensure that it was hit by an actor, this is done with a null check on the other actor. Then the AI will cast this other actor to projectile and ensure that the AI character was hit by a projectile. If the AI is dead and was hit by a projectile it will make one last check, this check is to see if the AI has a door it is supposed to open on death. If this last check returns true than the AI will call toggle door on the door it is supposed to open, this door must be a trigger door. Then the trigger door that is told to toggle by the AI it will toggle a Boolean named bDoorOpen and then call the ToggleAnim function on itself. The ToggleAnim function will check if the door is supposed to be open and if there is a door open animation, if both are true then the function will play the open door animation and remove collision on the door. If bDoorOpen return false then the door will check if there is a close door animation, if this second check returns true the door will then play the close door animation and toggle collision on. This door will only be triggered once as after the call is made the AI will be destroyed.

* 1. Trigger Door

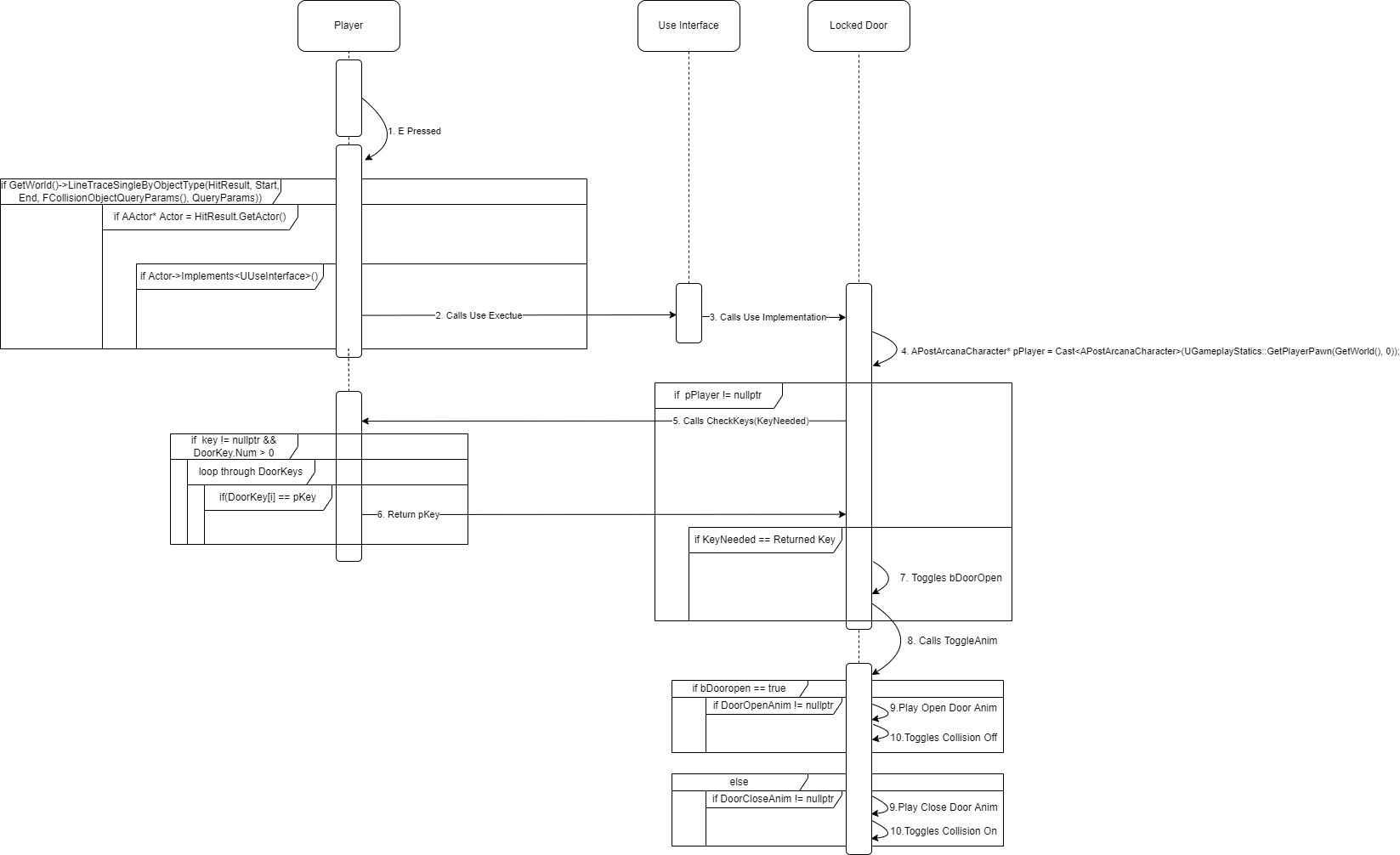
The trigger door is a door that is opened when the player interacts will a door trigger. This door trigger is a volume that is activated when the player overlaps with it. When the player overlaps with a door trigger the player’s OnOverlap function is called. The OnOverlap starts by making two checks, the first is to make sure that the player is alive. If the player is alive, it then checks to make sure that the player overlapped an actor, if this is true the player casts the actor to a door trigger. After casting the actor, the player makes a final check to see make sure the overlapped actor was a door trigger. If it was a door trigger the player calls the toggle door function on the door trigger, this causes the door trigger to call the toggle function of the door attached to the door trigger. This call will toggle a Boolean named bDoorOpen and then call the ToggleAnim function on itself. The ToggleAnim function will check if the door is supposed to be open and if there is a door open animation, if both are true then the function will play the open door animation and remove collision on the door. If bDoorOpen return false then the door will check if there is a close door animation, if this second check returns true the door will then play the close door animation and toggle collision on. This door can be toggled any number of times afterwards by overlapping the door trigger.

* 1. Key Pickup



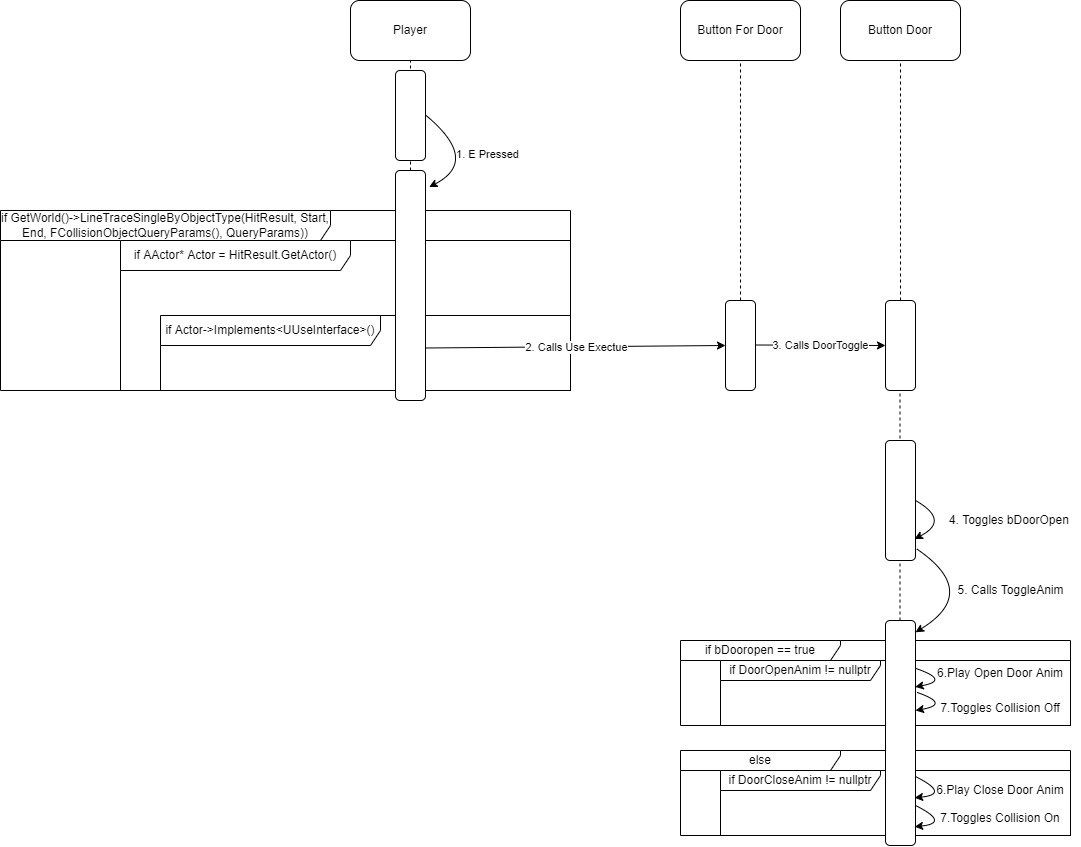
A key is an actor that can be picked up by the player. If the player presses E, or the input that has been bound to the interact function, the interact function will be called. When the interact function is called the player will do a ray trace in the direction they are facing. If the ray trace hits an actor, the player will cast the actor as a key. If the player interacted with a key the player will call the SetPlayerPointer function on they key, passing in itself as the argument. When the SetPlayerPointer function is called on the key the key will set its member variable for m\_pPlayer to the argument that was passed into the function. Once this is done the player will check if the key has an implementation of the use interface, if the key does have an implementation the player will call execute on the use interface which will call the implementation function that the key has. When the implementation function is called on the key the key will call the GetKey function on the member variable that was set earlier to m\_pPlayer, passing itself in as the argument. The player will then add the key to a Tarray which is seen as a virtual key ring. Once the player has added this key to the keyring, the key will destroy itself. A key can only be picked up once but will remain with the player.

* 1. Locked Door



A locked door is a door that can only be opened when the player is in possession of a specific key. If the player presses E, or the input that has been bound to the interact function, the interact function will be called. When the interact function is called the player will do a ray trace in the direction they are facing. If the ray trace hits an actor, the player will then check if this hit actor has an implementation for the use interface. If the actor does have an implementation of the use interface, then the use interface will call the implementation function on the hit actor, in this case the hit actor is the locked door. The locked door will then create a temporary variable and fill it with the player pawn at index 0. If this temporary variable is set properly and the player is not null, the locked door will then check if the player has the correct key. This check is done by calling the CheckKeys function on the player passing in the key needed to open this door as the argument. When the CheckKeys function is called on the player, the player will do three actions. The first action is a check that makes sure that the key passed in is not null, and that the player has at least one key in their possession. Then the player will loop through all of their keys and check if they have the needed key for that door. If the player has the right key it will then return a pointer to that key back to the locked door. The locked door will then make sure that the key returned is the correct key to open itself. If the key returned is the correct key the locked door will toggle a Boolean named bDoorOpen and then call the ToggleAnim function on itself. The ToggleAnim function will check if the door is supposed to be open and if there is a door open animation, if both are true then the function will play the open door animation and remove collision on the door. If bDoorOpen return false then the door will check if there is a close door animation, if this second check returns true the door will then play the close door animation and toggle collision on. This door can be toggled any number of times afterwards as the key stays in the players possession.

* 1. Button Door



The button door is a door that can be opened when the proper button is interacted with. If the player presses E, or the input that has been bound to the interact function, the interact function will be called. When the interact function is called the player will do a ray trace in the direction they are facing. If the ray trace hits an actor, the player will then check if this hit actor has an implementation for the use interface. If the actor does have an implementation of the use interface, then the use interface will call the implementation function on the hit actor, in this case the hit actor is the door button. The door button will then call the toggle function of the button door that is attached to it. The button door will toggle a Boolean named bDoorOpen and then call the ToggleAnim function on itself. The ToggleAnim function will check if the door is supposed to be open and if there is a door open animation, if both are true then the function will play the open door animation and remove collision on the door. If bDoorOpen return false then the door will check if there is a close door animation, if this second check returns true the door will then play the close door animation and toggle collision on. This door can be toggled any number of times afterwards by interacting with the button again.

1. Use Case View

All Creation is done in the *instance* of the blueprint that has been placed in the level.

* 1. Creating a Target Door

To create a target door, you first need to add a trigger door blueprint to the level.

Second you need to drag an AI character blueprint into the level. For this demonstration I will be using a target AI.

Once both blueprints are in the level select the AI character that is in the level and go to the Door category. This category will have a “Door to Open” drop-down menu, in this menu select the trigger door you put into the level.

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

* 1. Creating a Trigger Door and Trigger

To create a target door, you first need to add a trigger door blueprint to the level.

Second drag in a door trigger blueprint into the level. This trigger can be resized in the transformation section of the blueprint under scale

Graphical user interface, application

Description automatically generated

Next, below transformation there is a section to add a “Door to Open”. If click on the Door to Open a drop-down menu will appear, select the target door that you would like to have this trigger open.

A picture containing graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

* 1. Creating Locked Door and Key

To create a locked door, you first need to add a Locked door blueprint to the level.

Second you need to drag a door key blueprint into the level.

Select the locked door that you have placed into the level. In the instance of the blueprint you have dragged into the level look for a section labeled “Key Needed” and click the drop down menu to select the key that is needed to open that door.

Graphical user interface

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

* 1. Creating a Button Door and Button

To create a button door, you first need to add a button door blueprint to the level.

Second you need to drag a button for door blueprint into the level.

Select the button for door that you have placed into the level. In the instance of the blueprint you have dragged into the level look for a section labeled “Door to Open” and click the drop down menu to select the door that the button will open.



A screenshot of a computer

Description automatically generated with medium confidence