Project: Abyss / Photography Module

Architecture/Design Document

**Table of Contents**

1 Introduction.. 3

2 Design Goals.. 3

3 System Behavior.. 4

4 Logical View... 4

4.1 High-Level Design (Architecture)… 5-6

4.2 Mid-Level Design… 7

4.3 Detailed Class Design… 8

5 Process Views... 9-18

6 Use Case Views... 19

Change History

**Version:** 0.1

**Modifier:** Austin Morris

**Date:** 10/28/2023

**Description of Change:** Started the module with basic overviews.

**Version:** 0.2

**Modifier:** Austin Morris

**Date:** 11/06/2023

**Description of Change:**  Added mid level, high level, and start of process view section.

**Version:** 0.3

**Modifier:** Austin Morris

**Date:** 11/07/2023

**Description of Change:**  Finished process views and finished use case view.

# **1 Introduction**

This document describes the architecture and design for “Project: Abyss”, a game being developed by Astralwater Interactive. Embark on a groundbreaking cooperative underwater adventure with "Project Abyss," where the uncharted depths of Hollow Earth reveal a hidden world teeming with enigmatic flora, fauna, and ancient mysteries. As intrepid researchers contracted by a major tech conglomerate, you and your partner dive into the abyssal unknown armed with cutting-edge technology. However, beneath the waves, the shadows of corporate indifference threaten your every move.

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

# **2 Design Goals**

The design priorities for the Photography system are:

● Allow players to take pictures underwater.

● Implement a film system with limited capacity until reloaded at computer.

● Provide UI feedback on newly discovered fish.

● Store pictures in a gallery and upload relevant ones to the Collection entry.

● Have unique interactions with the stress & horror element.

# **3 System Behavior**

The photography module is built using 2 key elements. A photography camera actor, and a photography component. This was crucial to make modular to enable the Divebot to also be able to take photographs and have storage of pictures.

The player will be able to aim the camera, then take a picture with another input. When aiming, the camera with highlight any creatures that are currently in focus. Fish that are in focus will be surrounded by a box indicating the capture status of that fish. If it’s been added to the collection already, the player will see the fish name. If it hasn’t been added to the collection, the player will see ??? with a red outline.

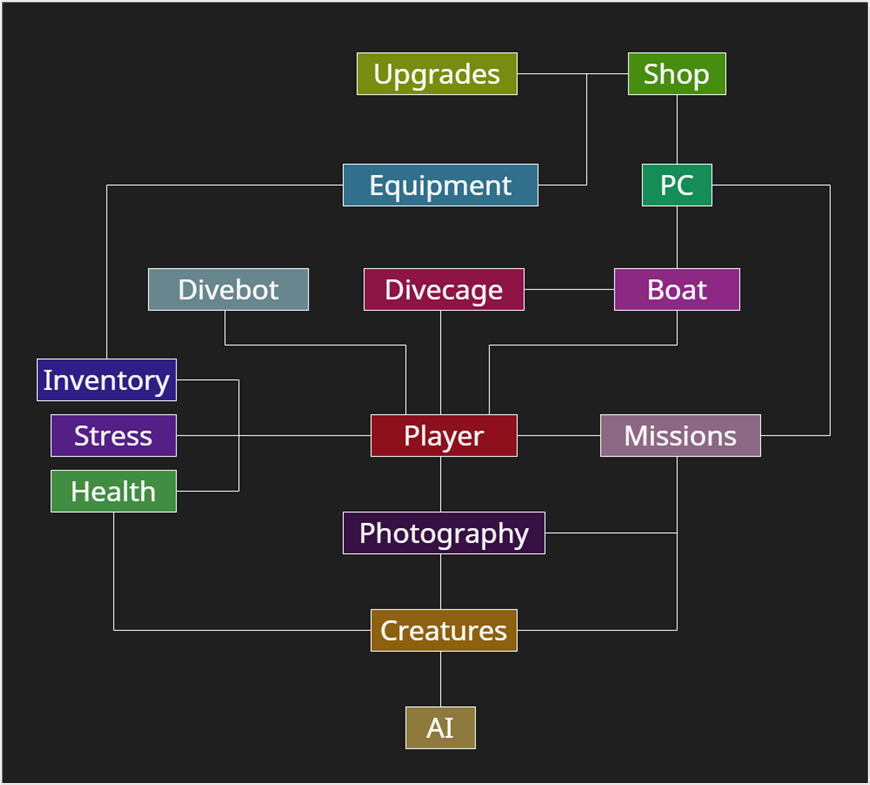
When photographed fish are in focus, the fish “photographed” will be stored in an array separate from the main photographs to ensure proper tracking of money/collection rewards once uploaded. The photography component will handle most of the storage and scene capture elements. This system will interact with creatures, collection system, player, and the PC Terminal.

# **4 Logical View**

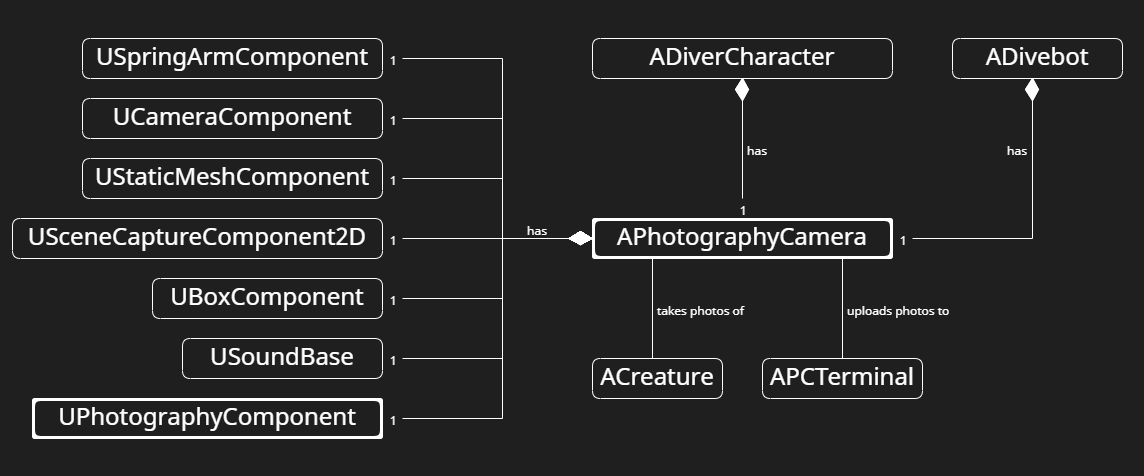
The logical view describes the main functional components of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

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

The high level view consists of…

* **Player System** is the central component that manages the player's actions, interactions, and progression throughout the game. It includes the player character, input handling. Key responsibilities involve controlling the player's movements, handling inputs, and triggering movement states such as swimming.
* **Photography System** is responsible for implementing the mechanics related to underwater photography. It includes the Photography Camera and associated components. This system allows players to take pictures during dives, manage a limited film capacity, store pictures in a gallery, and upload relevant ones to the Collection journal, contributing to the player's knowledge about underwater creatures.
* **Creature System** manages the behavior, interactions, and characteristics of underwater creatures. This system includes various types of fish, their AI, and the logic governing their movement, appearance, and responses to the player's actions. The system ensures a dynamic and realistic underwater ecosystem, providing challenges and opportunities for the player.
* **AI System** governs the artificial intelligence of both hostile and non-hostile entities in the game. It includes the behaviors and decision-making processes of underwater creatures, as well as any AI-driven challenges or enemies the player may encounter during dives.
* **Missions System** oversees the organization of missions into distinct categories, monitors the progression of objectives, and governs the allocation of rewards and unlocks. Accessible through the PC Terminal, players can seamlessly navigate through available, active, and completed missions. The PC Terminal serves as the central hub for mission management in the form of an email system, allowing players to accept new missions and submit completed ones, providing a comprehensive interface for tracking and advancing in the game's narrative.
* **Health Component System** manages the player's health. It includes components that handle damage, healing, and any other health-related mechanics. This system ensures that the player's survival is a key consideration during dives and encounters with underwater creatures.
* **Stress Component System** adds a psychological horror element to the game. It manages the player's stress levels based on in-game events, environmental factors, and encounters. High stress may affect performance or trigger adverse effects, adding an immersive layer to the overall experience.
* **Boat System** encompasses the functionality related to the player's boat. This includes navigation between dive locations, boat inventory, and serving as a safe zone between dives. The boat system also facilitates equipment upgrades, PC Terminal interaction, and overall progression.
* **DiveCage System** is used as a later game upgrade that can be used to speed up travel time to get to deeper locations.
* **Equipment System** is the main item system in the game. Players will be able to buy and use equipment from the shop, including a camera, deployable light sources, and more. Some equipment is consumable, and some is permanent.
* **Divebot System** encompasses the functionality to the controllable Divebot available in the game. The Divebot will be possessed on death as a spectator mechanic, and can be possessed on command to allow remote control for scouting purposes. This will include a custom movement component to assist in executing some of these tasks.
* **Upgrades System** enables the enhancement and customization of the player's equipment, boat, and dive-related tools. Players can earn or purchase upgrades to improve their camera, dive cage, and other components, providing a sense of progression and empowerment.
* **Shop System** allows players to buy in-game equipment and upgrades. It provides a marketplace where players can spend earned resources to acquire new equipment, upgrade existing gear, or replenish consumables.
* **PC System** refers to the personal computer within the game. It serves as the hub for managing various aspects, such as reviewing your research journal, viewing photographs, accessing the in-game shop, and planning future dives. The PC system contributes to the player's overall agency and strategic decision-making.
* **Inventory System** refers to the inventory component that is attached to the player and storage chest. The player will be able to swap equipment to and from the storage chest.

**4.2 Mid-Level Design of the Player Module**

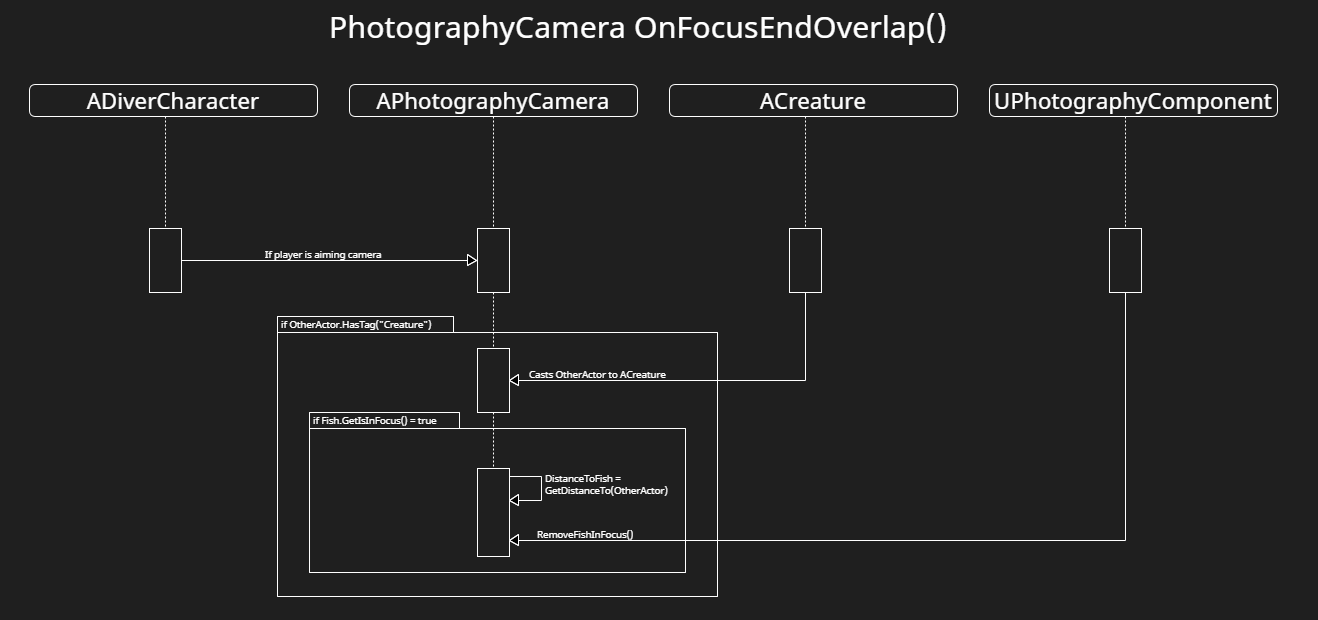
The photography camera consists of a few major elements and key interactions, here’s a description as to what each accomplishes.

* **SpringArm** – This component is attached to the player’s look at location to ensure that the camera will always be where the player is looking.
* **Camera** – The camera is attached to the spring arm, and when the player holds up their camera, this component will be the new View Target.
* **Mesh** – This is the mesh for the camera itself.
* **SceneCapture2D** – This component is key to the photography system, as this is what will be rendering our post processing and the CaptureScene() function itself. This is expanded on later.
* **BoxComponent** – This hitbox is used for activating the Focus widget that is placed on each creature to show when it’s in focus.
* **SoundBase** – The picture taken sound effect & zoom sound effect
* **PhotographyComponent** – This is a custom component which enables most of the photography system, here’s a few functions which will be expanded on soon.
* TakePicture() handles the logic of taking a photograph.
* ResetCamera() this resets the camera to the default state.
* UpdateFishInFocusDistances() this is called by the camera on tick to display the correct fish in focus

## **4.3 Detailed Class Design of the Photography Module**

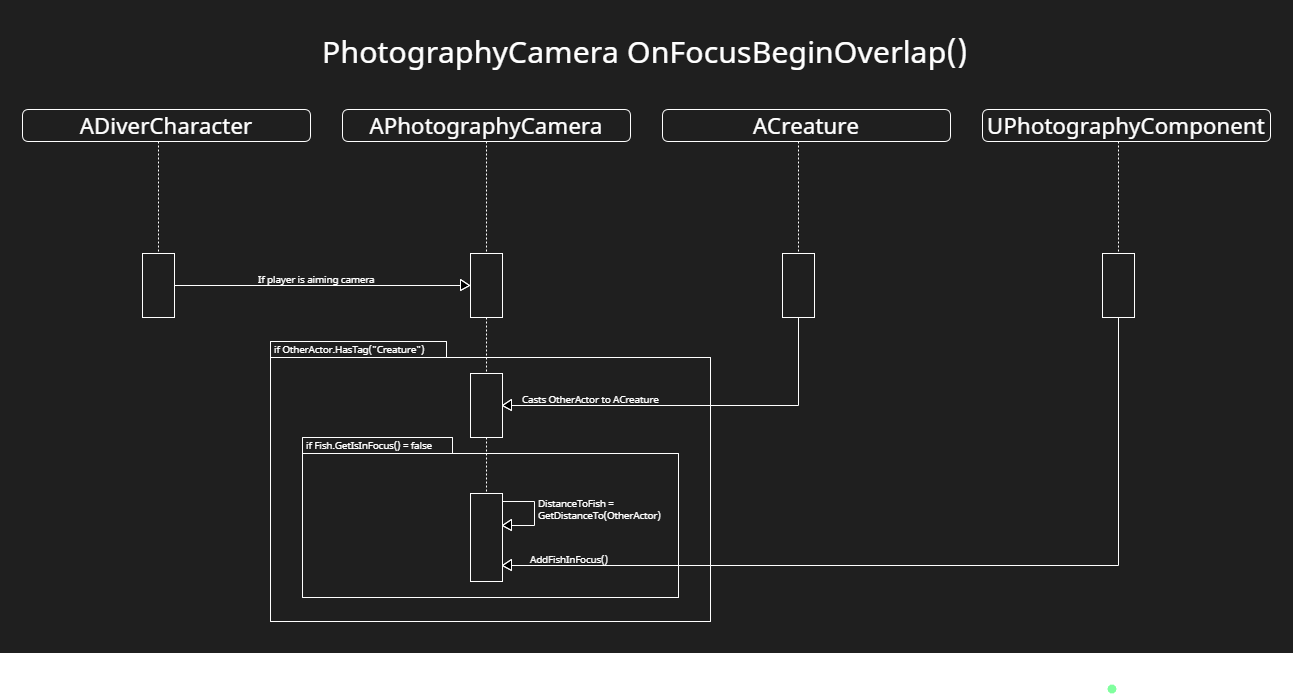
# **5 Process View of the Photography Module**

**PhotographyCamera Actor Functions**

 **OnFocusEndOverlap(etc)**

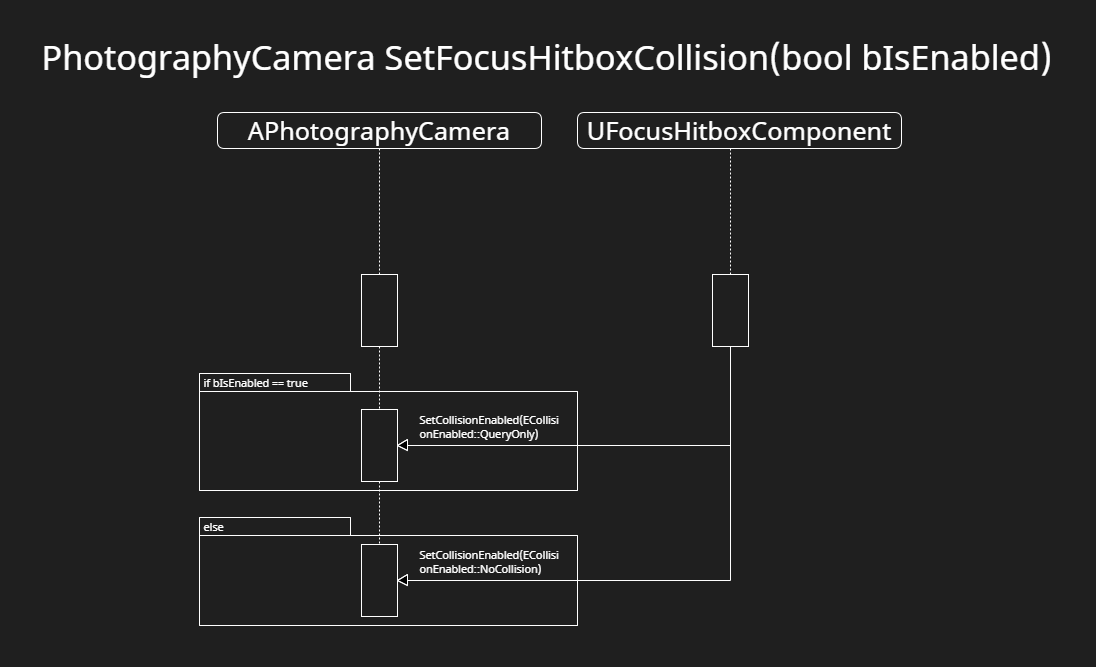
This function is called when an actor exits the overlap of the hitbox. It’s used primarily for disabling the focus widget from a fish that isn’t in focus.

1. If the player is aiming the camera, go into an if statement.
2. If OtherActor that’s overlapping has the tag “Creature”, proceed.
3. Casts OtherActor to the creature that’s overlapping.
4. If that fish is currently in focus, GetsDistanceTo the fish.
5. Removes the fish in focus from the array in photography component.

**OnFocusBeginOverlap(etc)**

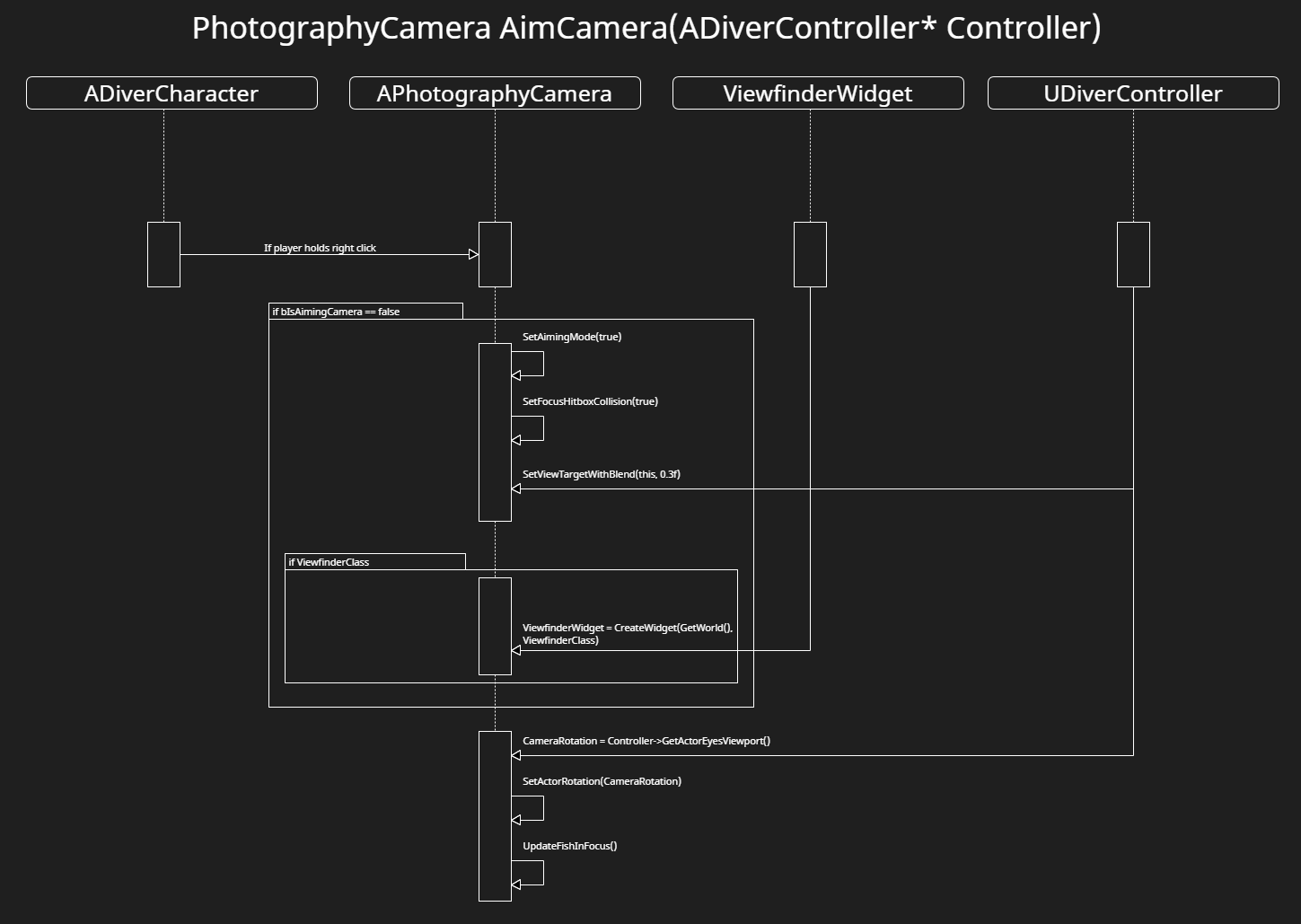
This function is called when an actor enters the overlap of the hitbox. It’s used primarily for enabling the focus widget from a fish that’s newly in focus.

1. If the player is aiming the camera, go into an if statement.
2. If OtherActor that’s overlapping has the tag “Creature”, proceed.
3. Casts OtherActor to the creature that’s overlapping.
4. If that fish is currently NOT in focus, GetsDistanceTo the fish.
5. Adds the fish to the focus array in photography component

**SetFocusHitboxCollision(bool bIsEnabled)**

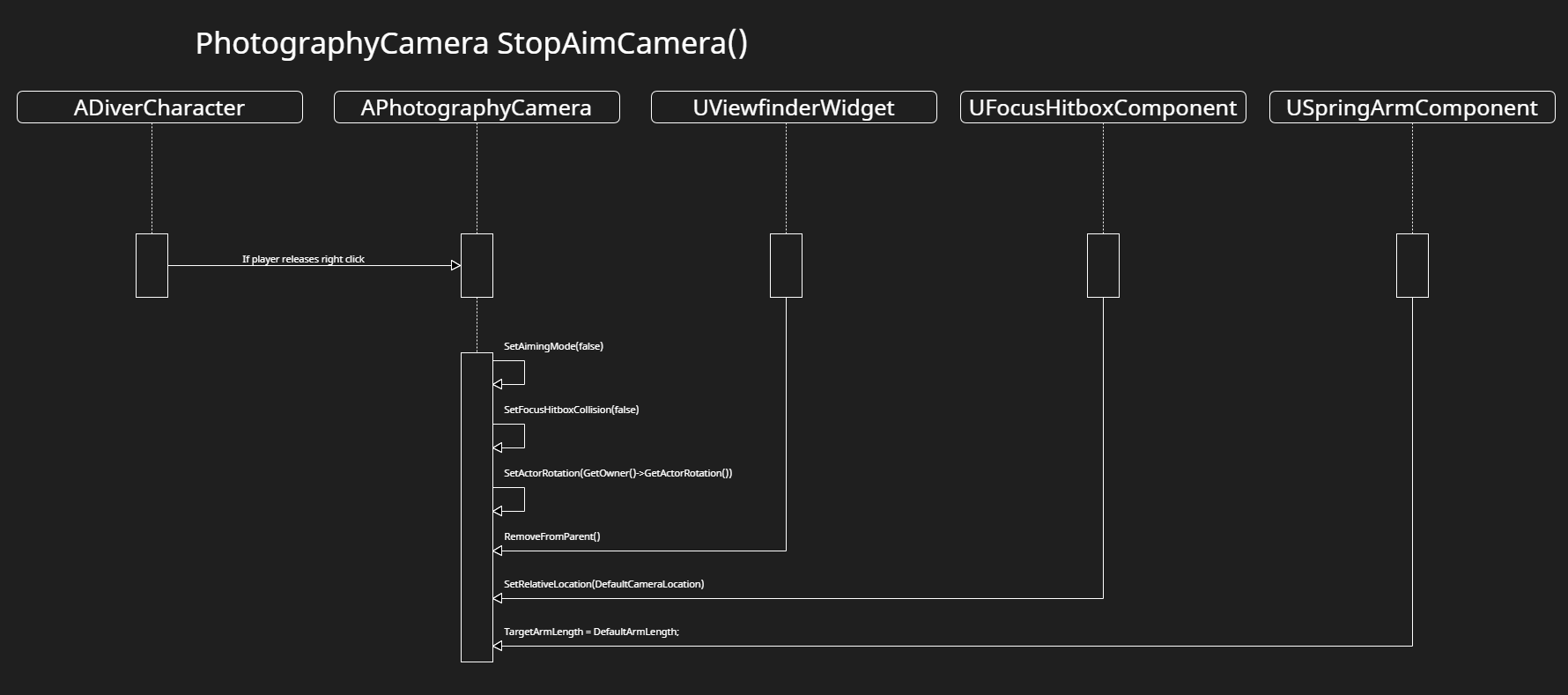
This function simply toggles collision on/off for the FocusHitbox component.

1. If it’s enabled, then disable it.
2. If it’s disabled, then enable it.

**AimCamera(ADiverController\* Controller)**

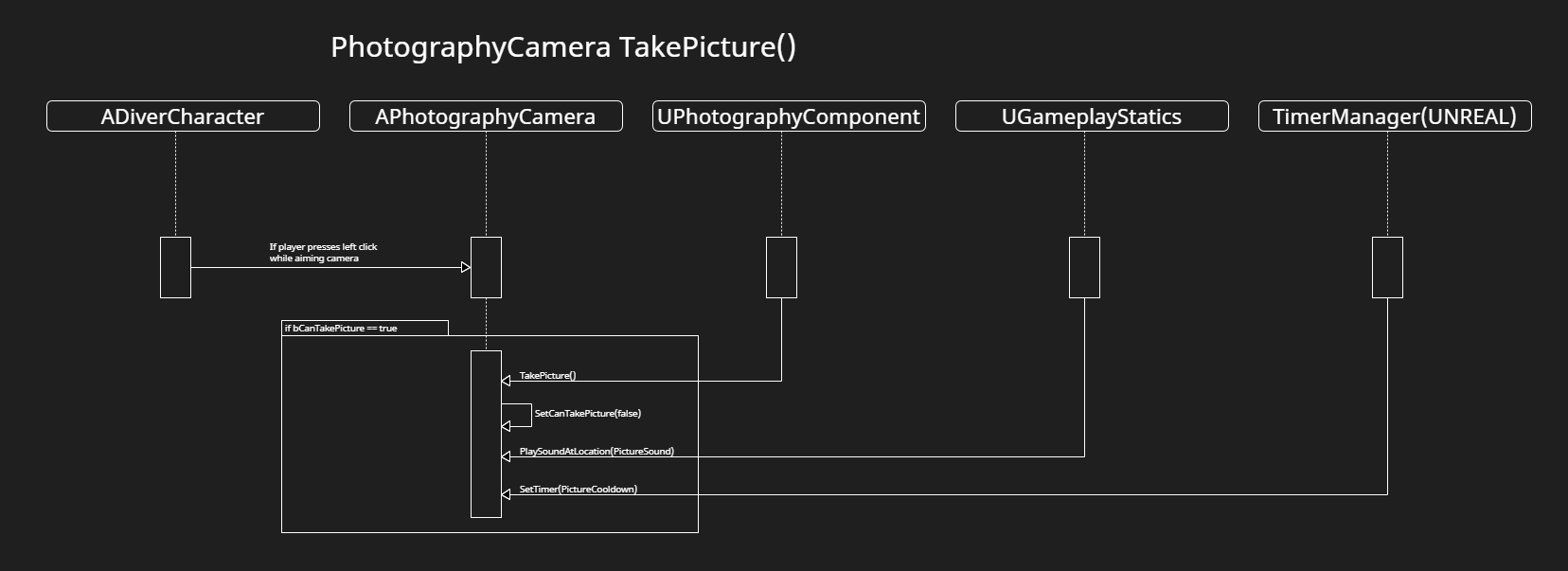
AimCamera is the main function in PhotographyCamera, as it handles logic for setting view targets, enabling collisions, bools, creating the viewfinder widget, and updating fish in focus.

1. If player is holding right click, run the code.
2. If the player currently isn’t aiming the camera, run this code. This is using a bool just to run specific code once for optimization, since AimCamera is updated per frame while being held down.
3. SetsAimingMode to true, SetsFocusCollision enabled, and sets the view target camera to be the camera on this actor.
4. If the viewfinder class is valid, create and add it to the viewport.
5. Outside of these loops, these calls are updating every frame. The camera rotation, and finally,
6. Updates every fish in focus to enable proper focus widget display.

**StopAimCamera()**

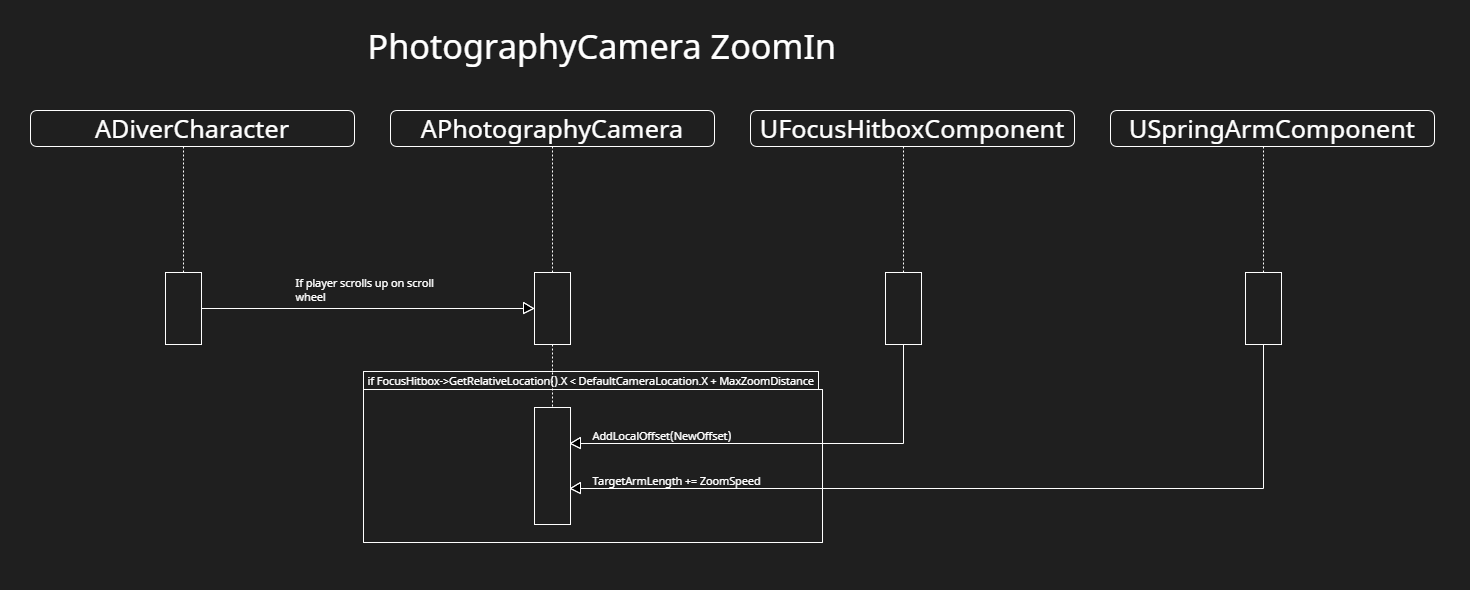
This function is called when player releases right click, and is used to disable a lot of the aiming mode variables/functions.

1. SetsAimingMode and FocusHitboxCollision to false.
2. SetsActorRotation of the camera to the default player’s rotation, so it’s stuck in place on the players body.
3. Removes the viewfinder widget from the player’s screen.
4. Sets the location of the hitbox component to the default location (This is done if the player zooms in the camera, it gets reset.)
5. Also sets the target arm length of the camera to default (Same reason as above.)

**TakePicture()**

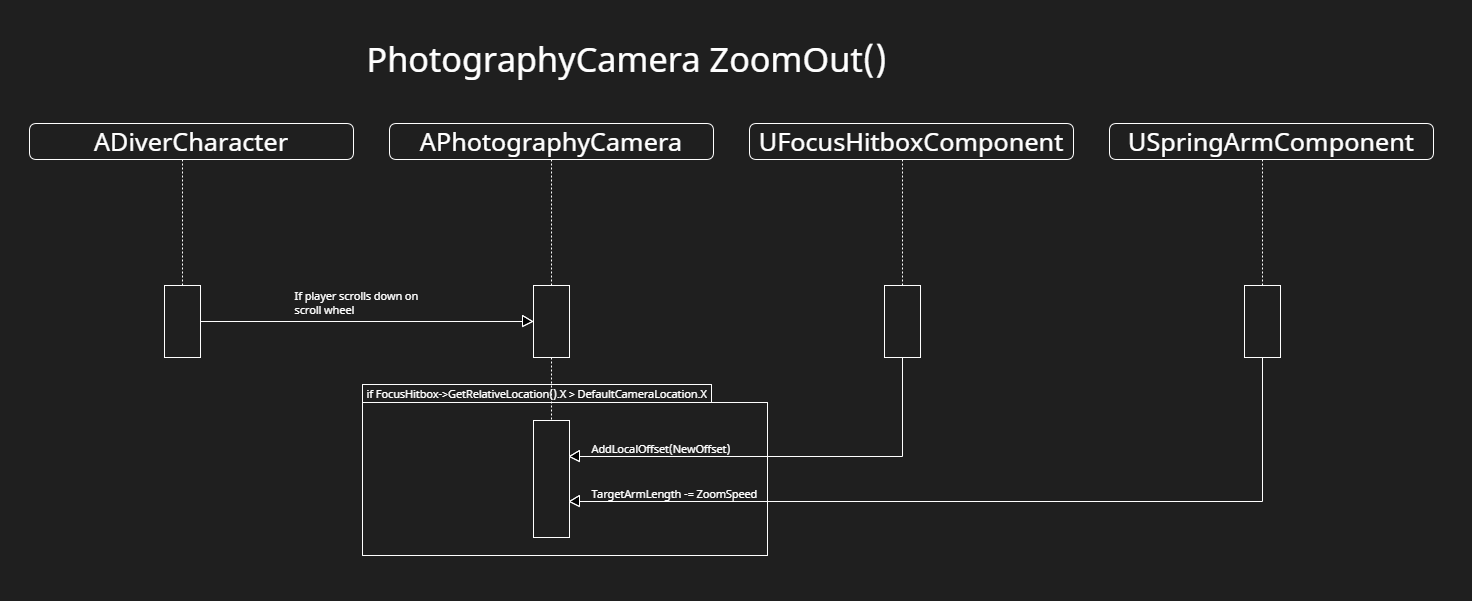
This function is an intermediary between the player input and the PhotographyComponent TakePicture() function. The cooldown is managed here, along with a sound effect.

1. If player presses left click while aiming camera, this is called.
2. If the player can take a picture, proceed
3. Calls TakePicture() on PhotographyComponent
4. Sets CanTakePictue to false which disables the picture taking until the timer re-enables it.
5. Plays a photo sound effect at the players location.
6. Sets a timer to reset the picture cooldown bool.

**ZoomIn()**

This function zooms the camera in.

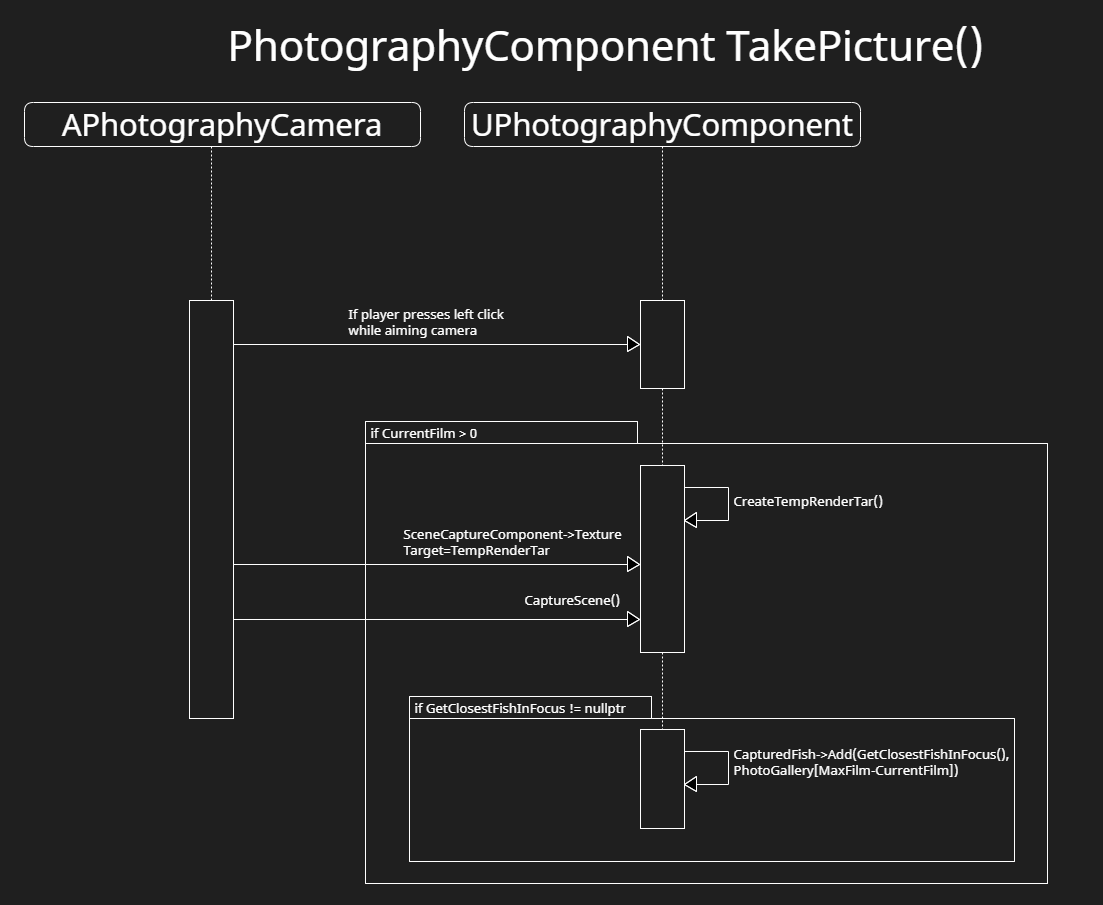
1. If player is scrolling up on the mouse wheel, call this function
2. If the hitbox location is LESS than the default + maxzoomdistance, the camera can zoom in, proceed.
3. AddsLocalOffset to the HitboxComponent to ensure focus isn’t super close to the player.
4. Sets target arm length to be increased by the zoom speed. The spring arm has the camera and scene component attached to the end of the arm.

**ZoomOut()**

This function zooms the camera out.

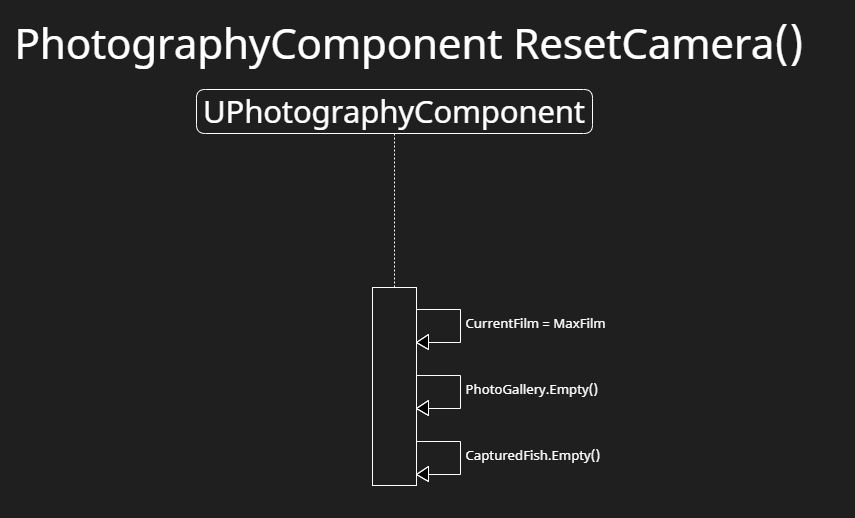
1. If player is scrolling down on the mouse wheel, call this function
2. If the hitbox location is GREATER than the defaultcameralocation, the camera can zoom out.
3. AddsLocalOffset to the HitboxComponent to ensure focus isn’t super close to the player
4. Sets target arm length to be slowly decreased by zoom speed.

**PhotographyComponent Functions**

**TakePicture()**

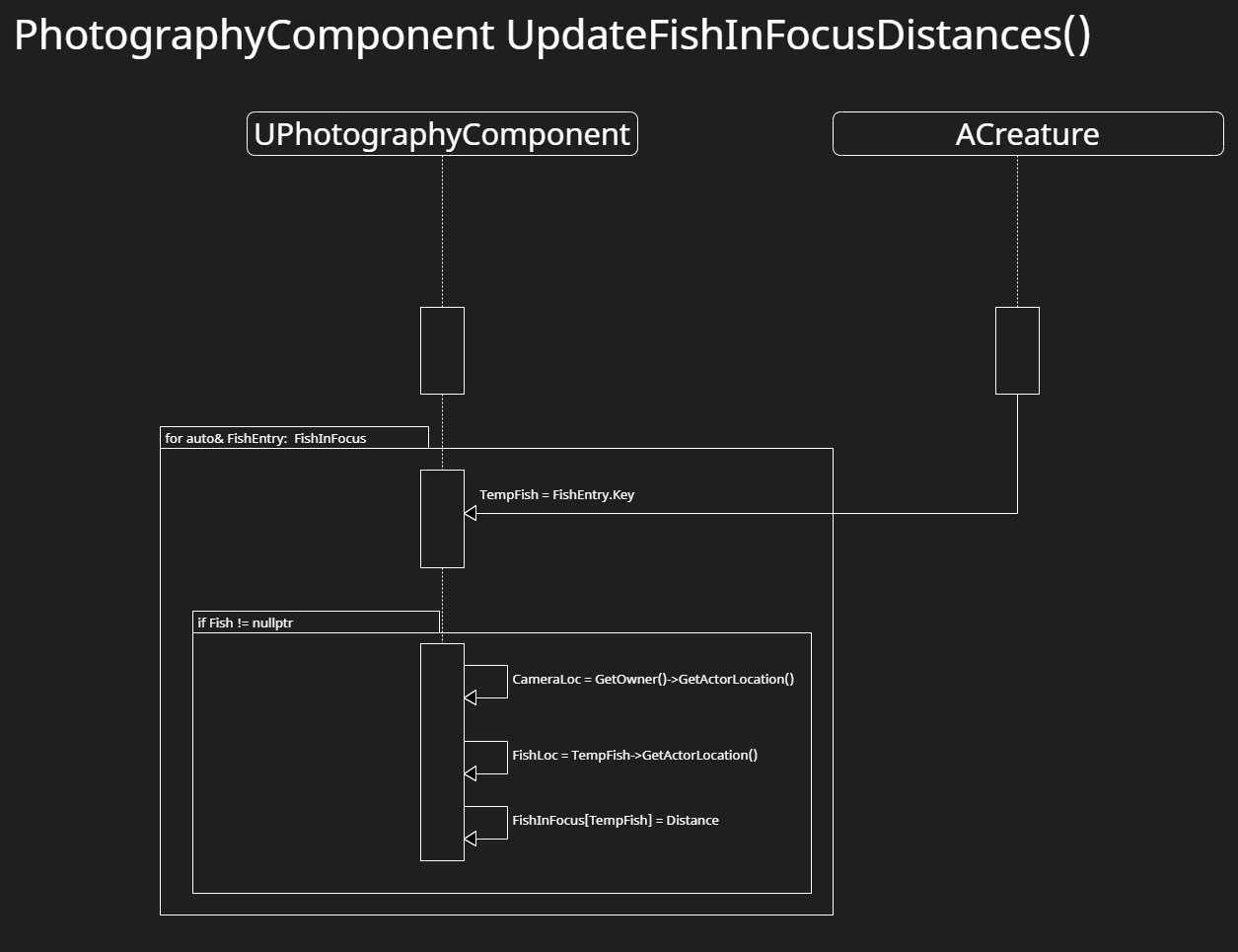
This function takes a picture of the current scene and stores it in a render target on the scene capture. That texture target 2D is then stored onto the PhotoGallery, which once uploaded to the PC, will be used to display images onto the screen.

1. If player takes picture with the camera, run this function.
2. If the current film on the camera is > 0, proceed.
3. Create a temp render target with specific settings, then Capture the scene with the SceneCaptureComponent on the Camera.
4. If the ClosestFishInFocus is valid, add it to the CaptureFish array. This is used to reward the players with money based on what fish are uploaded to the PC.

**ResetCamera()**

This function resets the camera to the default state by resetting film count, and emptying all important maps and arrays. This will be called when the player dies, and when photos are uploaded.

1. Resets film to max film.
2. Empties the photo gallery map.
3. Empties the captured fish map.

**UpdateFishInFocusDistances()**

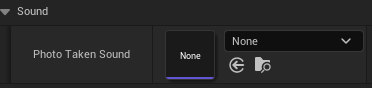
This function updates all the distances of every fish that are currently in focus. This is called in the photography camera tick to ensure proper distances at all times while the player is aiming.

1. For ranged based loop on the FishInFocus map which has distances assigned to references to fish.
2. Assigns a tempfish to the current entry.
3. If the temp fish is valid, set the distance of the fish to the camera.

# **6 Use Case View**

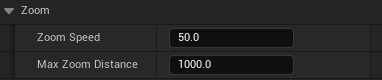
Below are the designer variables which can be found within the PhotographyCamera actor.

PhotoTakenSound



The photo taken sound is triggered when the player takes a photo. This can be changed by designers to the desired sound effect.

Zoom



There are two variables the artists can manipulate to affect the zoom of the cameras.

1. The zoom speed, which affects how fast the camera zooms in/out.
2. The max zoom distance. This is how far the camera can zoom in.

PictureCooldown



When you take a picture, there is a slight delay to make it so the player cannot spam take photographs. The designers can adjust this cooldown which is clamped between 0 and any max.

MaxFilm



This variable can be adjusted to change the max amount of pictures a camera can currently hold before the player dies, or uploads their photos.