Project: Abyss / Equipment 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… 7 - 8

5 Process Views... 9-19

6 Use Case Views... 20 - 24

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:** Alexander McTernan

**Date:** 12/05/2023

**Description of Change:** Added Design Goals and a System Behaviour.

**Version:** 0.3

**Modifier:** Alexander McTernan

**Date:** 12/05/2023

**Description of Change:** Updated High level view.

**Version:** 0.4

**Modifier:** Alexander McTernan

**Date:** 12/07/2023

**Description of Change:** Mid-level and Detailed class design completed. All Process views are created. Case Views completed.

# **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 Equipment 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 Equipment system are:

● Able to create a dynamic system that can be used to create new equipment with ease.

● Create the ability to be purchased at the in-game shop.

● To be able to interact with the inventory system efficiently.

● To be able to be interacted with the player as each one is intended with efficiency.

# **3 System Behavior**

This system gives us the ability to create various types of equipment that can be equipped by the player with the use of the players hot bar. The equipment can be purchased from the in-game shop, and it will be available in the storage chest for the player to pick up. Any repeating behaviours are created and implemented in the Equipment Base and can be called via methods in individual equipment to add the functionality, including the pickup and through mechanic, but still allowing new functionality to be added easily. The different equipment will be described below as well.

# **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 Equipment Module**

A diagram of a computer

Description automatically generated with medium confidence

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

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

AInteractableActor is discussed in its own module

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

Below describes the available features in the Equipment Base.

Is Interacted With

A screenshot of a computer

Description automatically generated

This diagram does not show how the Player interacts as that is described in the Player module. This diagram describes what happens when it reaches AEquipmentBase.

1. Calls Interacted() on the AEquipmentBase, passing in the Player reference.
2. Turns off the collision for the MeshComponent.
3. If it successfully adds the equipment to the HotbarInventory, call the Equip() method.
4. Set bIsThrown to false.
5. Set the desired position you would like the equipment to sit in reference to the player.

Use Primary

A screenshot of a computer

Description automatically generated

The above diagram is described below:

1. First it will check if the Primary use input action is pressed on the ADiverController.
2. This will call the UsePrimary() method.
3. If the MyPlayer variable in the DiverController is not null, set bIsUsingAnything to true.
4. If the Player has an InventoryComponent and has an equipment available in the inventory slot, call UsePrimary() on the selected Equipment.
5. If the bIsCharging variable is set to true, set the bIsThrown bool to true.
6. Turn off the trajectory.

Release Primary

A screenshot of a computer program

Description automatically generated

The above diagram is described below:

1. First it will check if the UsePrimary input action is released on the ADiverController.
2. This will call the StopUsePrimary() method.
3. If the MyPlayer variable in the DiverController is not null, set bIsUsingAnything to false.
4. If the Player has an InventoryComponent and has an equipment available in the inventory slot, call ReleasePrimary() on the selected Equipment.
5. This currently does nothing on the AEquipmentBase, but it can be overridden by other equipment.

Use Secondary

A screenshot of a computer screen

Description automatically generated

The above diagram is described below:

1. First it will check if the UseSecondary input action is pressed on the ADiverController.
2. This will call the UseSecondary() method.
3. If the MyPlayer variable in the DiverController is not null, set bIsUsingAnything to true.
4. If the Player has an InventoryComponent and has an equipment available in the inventory slot, call UseSecondary() on the selected Equipment.
5. If the bCanThrow bool is true, set bIsCharging to true.

Release Secondary

A screenshot of a computer

Description automatically generated

The above diagram is described below:

1. First it will check if the UseSecondary input action is released on the ADiverController.
2. This will call the StopUseSecondary() method.
3. If the MyPlayer variable in the DiverController is not null, set bIsUsingAnything to false.
4. If the Player has an InventoryComponent and has an equipment available in the inventory slot, call ReleasedSecondary() on the selected Equipment.
5. If the bCanThrow bool is set to true, set the ChargeValue to be MinChargeValue.
6. Set bIsCharging to be false.

Equip

(No diagram required as it is too simplified)

When this function is called:

1. Sets the owner of the equipment to the actor that was passed into the function.
2. Attaches it to the actor.
3. Disables the equipment’s collision.

Unequip

(No diagram required as it is too simplified)

When this function is called:

1. Detach from the actor.
2. Enable the equipment’s collision.
3. Reset the MeshComponent Collision Profile.

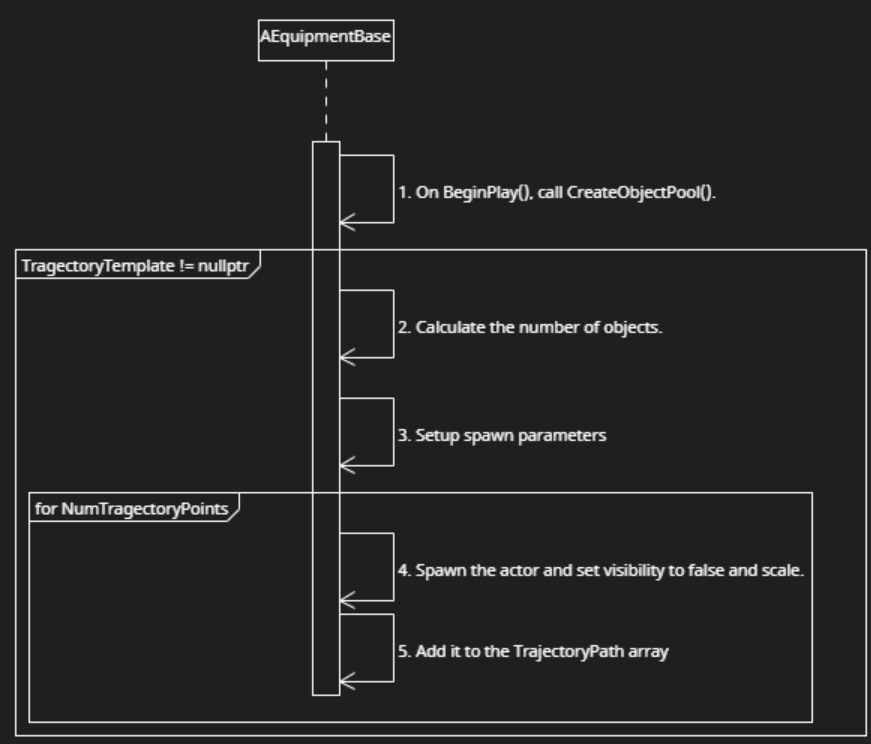
Removing from the Inventory

(No diagram required as it is too simplified)

To execute this task:

1. OnDropped() is called in the AEquipmentBase.
2. In this function, it makes sure the equipment has an owner, casts the owner to be an ADiverCharacter, and checks if the cast was null.
3. If not, it gets the Inventory from that ADiverCharacter and iterates through the inventory to find this equipment.
4. Calls RemoveFromInventory passing in the iterated index.
5. The rest is described in the Inventory module.

Create Object Pool



The above diagram is described below:

1. The CreateObjectPool() method must be called on BeginPlay().
2. If the TragectoryTemplate is not null, Calculate the number of trajectory objects it should create.
3. Setup the spawn parameters for each object.
4. For the NumTragectoryPoints, spawn a Trajectory object and set its visibility to false, and adjust the scale.
5. Finally, add each one to the TrajectoryPath array.

Throw

A screenshot of a computer

Description automatically generated

The above diagram is described below:

1. When it is required, StartThrow() is called.
2. If bIsCharging is true and the equipment has an owner, set the ChargeValue to be the MinChargeValue, bIsCharging to false and reset the WaypointIndex to 0.
3. In Tick(), if the bIsThrown bool is set to true, call ContinueThrow().
4. In this method, if WaypointIndex is greater than or equal to the last index in the TrajectoryPath array, set bIsThrown to false.
5. Else, Set the new position by lerping between the current position and the next waypoints location.
6. If the distance between the current position and the target position is less than x amount, increment the WaypointIndex.

Turn On Trajectory

A screenshot of a computer program

Description automatically generated

The above Diagram is described below:

1. When desired, call ResetTrajectory().
2. In this function, cast the Owner to be an ADiverCharacter and get the forward vector of its FirstPersonCameraComponent.
3. Set up the FPredictProjectilePath parameters and the result variable.
4. Call PredictProjectilePath(), passing in a reference to the equipment, the params and the result variable.
5. Store the path data that is now stored I the result variable, into a pathData Array.
6. For each increment in the pathData array, if it does not exceed the Object pool size, set the TrajectoryPath object at this increment to the corresponding pathData location.
7. Set the Mesh visibility to false.

Turn off Trajectory

(No diagram required as it is too simplified)

When this function is called:

1. Loop through TrajectoryPath array and set the visibility to false.

Below describes the available features in the Deployable Light

The Deployable Light does uses all of the above features, below will describe how they are used. If it is not mentioned below, no modifications were made. Since it uses the throw mechanic, CreateObjectPool() is called in BeginPlay().

Use Primary

A diagram of a call

Description automatically generated with medium confidence

The above diagram shows what happens AFTER all the “Use Primary” sequence from earlier in the document is completed. All of this occurs in the UsePrimary() in ADeployableLight.

1. First, it calls the Unequip() method. This detaches it from the player. A more indepth look at this function is described above.
2. Second, it calls the StartThrow() method. This resets all the needed variables to start the throw. See above for more of an in depth look.
3. Finally, it calls the OnDropped(), which is described in the above section called “Removing from the Inventory”.

Use Secondary

(No diagram required as it is too simplified)

The above diagram shows what happens AFTER all the “Use Secondary” sequence from earlier in the document is completed. All of this occurs in the UseSecondary() in ADeployableLight.

1. Calls ResetTrajectory(), which is described in the section above named “Turn On Trajectory”.

Release Secondary

(No diagram required as it is too simplified)

The above diagram shows what happens AFTER all the “Release Secondary” sequence from earlier in the document is completed. All of this occurs in the ReleaseSecondary() in ADeployableLight.

1. Calls TurnOffTrajectory(), which is described above as well.

On Hit (NEW)

(No diagram required as it is too simplified)

When a rigid body collision is detected, it checks if it is not colliding with the ADiverCharacter. If it is not either of these, it resets the bIsThrown = false.

Set Light Visibility (NEW)

(No diagram required as it is too simplified)

This method is currently not being used, but it will be used to disable the lightsource.

1. Set the LightComponents bAddedToSceneVisiblity to be the value passed into the function

# **6 Use Case View**

Below are the designer variables which can be found within the ADeployableLight and Trajectory Object blueprints.

Finding the example blueprint

A screenshot of a computer

Description automatically generated

Above shows the file path to get to the example blueprint of the ADeployableLight class

Where to find all adjustable variables

A screenshot of a computer

Description automatically generated

You can find all of the customizable variables for the Deployable Light in the Designer section of the Details tab.

Changing the shop related variables

A screenshot of a computer

Description automatically generated

The above variables adjust the shop interaction values.

Changing the Hotbar image

A screenshot of a computer

Description automatically generated

This is the place you would change the image that appears in the hotbar when it is equipped.

Changing the Throw mechanic

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

To enable or disable whether the equipment can be thrown, adjust the Can Throw bool. Enabling this bool will then allow the throwing and Trajectory variables to be visible and adjustable. This is where you can adjust the charging values, the amount of objects in the trajectory and the Trajectory Template blueprint.

Changing the Mesh and Texture for the Trajectory Object

A screenshot of a computer

Description automatically generated

To change the Trajectory Objects mesh and textures, the variables can be accessed through this object.

A screenshot of a computer

Description automatically generated

The you will want to navigate to the Mesh component and the Static Mesh and Material variables are on the right.