Project Abyss / Divebot Module

Architecture/Design Document

**Table of Contents**

1 Introduction.. 3

2 Design Goals.. 3

3 System Behavior.. 3

4 Logical View... 4

4.1 High-Level Design (Architecture) 5

4.2 Mid-Level Design. 7

4.3 Detailed Class Design. 8

5 Process View... 9

6 Use Case View... 16

Change History

**Version:** 0.1

**Modifier:** Alexander McTernan

**Date:** October 11th, 2023

**Description of Change:** Initial creation. Added a rough UML and Mid-level design, along with filling out the Design goals and system behaviour.

**Version:** 0.2

**Modifier:** Alexander McTernan

**Date:** October 12th, 2023

**Description of Change:** Started Sequence Diagrams.

**Version:** 0.3

**Modifier:** Alexander McTernan

**Date:** October 19th, 2023

**Description of Change:** Fixed UML diagram after implementation of base Divebot and functionality. Added Sequence diagrams for the Possess Divebot, UnPossess Divebot and Setup Diver Reference. Will do Movement when component is setup.

**Version:** 0.4

**Modifier:** Alexander McTernan

**Date:** November 7th, 2023

**Description of Change:** Completed Document. Added a High-level view image and description. Updated the UML and Possess Sequence diagram for backwards documentation. Completed the rest of the sequence diagrams and their descriptions. Also completed the 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 Divebot 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 Divebot Module system are:

* This design should allow designers to have easy access to all necessary properties to create desired effects.
* This design should keep the Divebot and its separate components easy to understand and should keep everything modular and simple.

# **3 System Behavior**

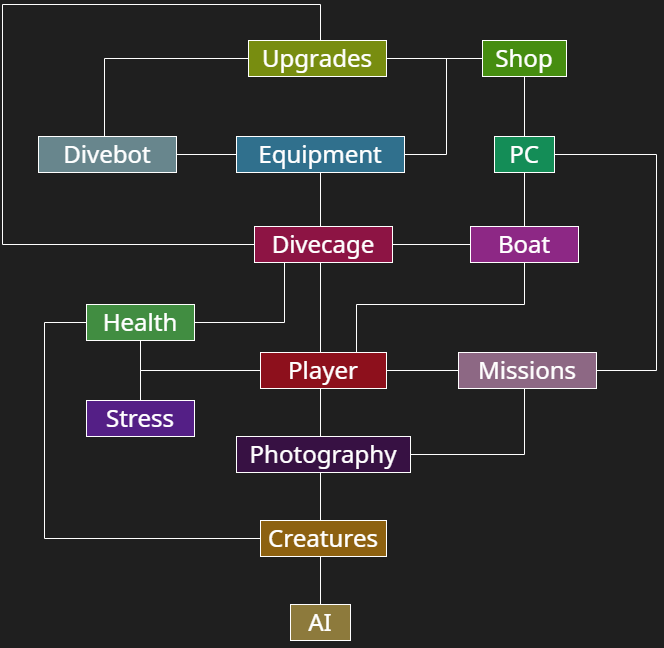
The Divebot module is derived from the APawn class in the Unreal Engine. This is done so a user is able to possess the object. This class will build an object that will follow the player around while in cave systems or dock on the Dive Cage at all other times. When not docked, the Divebot will follow the player around and shine a light ahead to assist them. When the other player possesses it, they are given the ability to move on their own and take their own pictures and use equipment. The player will also be able to pet the Divebot later.

# **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.

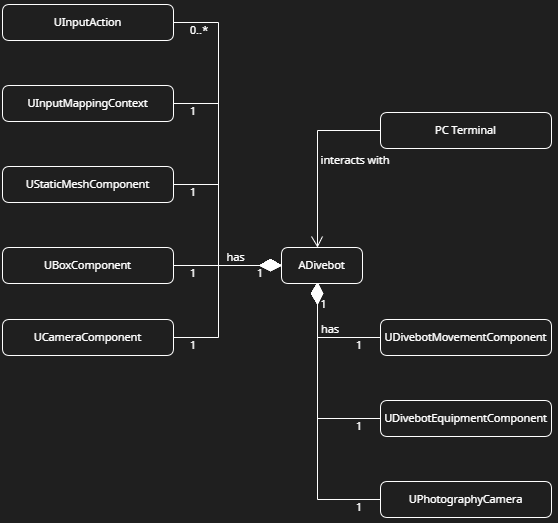
## **4.1 High-Level Design (Architecture of the Entire system)**



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**
* **Health Component System** manages the player's & divecage’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 the main source for underwater exploration, providing a somewhat “safe” way for the researchers to take photographs of the flora and fauna of the world. It will have health which unless managed properly while being sent down, can lead to player death and loss of progress.
* **Equipment System** will be completed later.
* **Divebot System** encompasses the functionality to the controllable Divebot available in the game. This includes being controlled through the PC Terminal, the Boat operator will be able to control it and use the Photography Camera to take pictures. When it is not being controlled, it will follow the Diver and shine a light ahead of them. 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, divebot, and other components, providing a sense of progression and empowerment.
* **Shop System** allows players to buy in-game items 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.

## **Mid-Level Design of the Divebot Module**



* **ADivebot** inherits from the APawn class. This class holds the Mesh and the Collider for the system, as well as taking all inputs from the user when it is possessed. This class is the base character and it calls all the attached components to do most of the functionality.
* **PC Terminal** will be explained in a separate module.
* **UDivebotMovementComponent** inherits from UActorComponent. This deals with the movement and rotation of the object. The input from the Divebot will call functions in this component.
* **UPhotographyCamera** is explained in a separate module. This module only explains its interaction with it.
* **UDivebotEquipmentComponent (Not Implemented)** inherits from a custom Inventory component. This is explained in separate module.

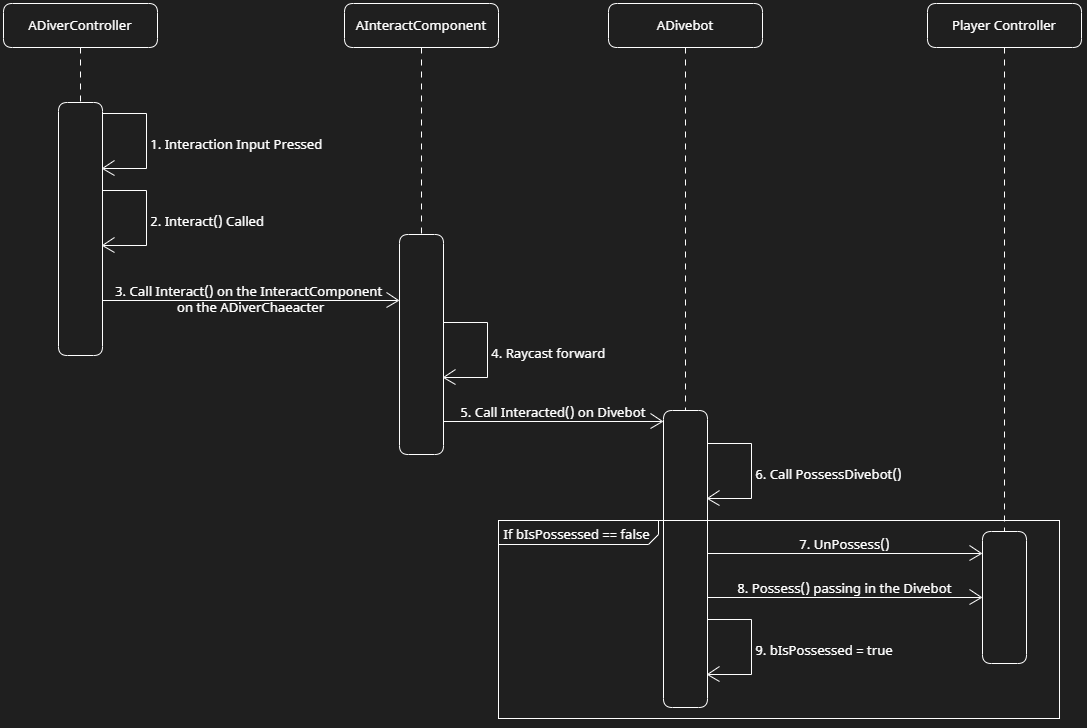
## **4.3 Detailed Class Design of the Divebot Module**

A screenshot of a computer program

Description automatically generated

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

Possess Divebot



The above diagram shows the process of how the ADiverCharacter will possess the Divebot.

1. If the user is possessing the ADiverCharacter, when they press the InteractAction, The Interact() method will be called, which will call the Interact() on the InteractComponent that is on the ADiverCharacter, passing in the ADiverCharacter in as a parameter.
2. In this method, it will then Raycast forward and if it hits a ADivebot, it will call the Interacted() method on the ADivebot, continuing to pass the aDiverCharacter in as a parameter.
3. In this method, it will set the BoatOperatorReference to be the ADiverCharacter parameter, then it will call the PossessDivebot() method.
4. If the Divebot is not possessed, it will then call UnPossess() on the ADiverController that was controlling the ADiverCharacter.
5. If the server made this call, it will call the OnPossess() method on the ADiverController, passing the ADivebot as a parameter.
6. If a client made the call, it will call NMC\_Possess(), which does the same as the previous step and then replicates it down to the client.
7. Finally, it sets bIsPossessed to true.

UnPossess Divebot

A diagram of a diver controller

Description automatically generated with medium confidence

The above diagram shows how the ADivebot is UnPossessed and it repossesses the ADiverCharacter.

1. UnPossess input is received. This calls the UnPossessDivebot() method on the ADivebot.
2. In the method, it checks if the ADivebot is currently Possessed. If it is, then it calls UnPossess() on the on the ADiverController.
3. It then calls OnPossess() on the ADiverController, passing in the BoatOperatorReference as a parameter.
4. Finally, bIsPossessed is set to false.

Setup Diver Reference

A screenshot of a computer

Description automatically generated

The above diagram shows how the ADivebot assigns a value to the DiverReference variable. This is used for the movement, which is described in its own module.

1. On BeginPlay(), the SetupDiverReference() method is called.
2. In this method, it loops through the all the ADiverCharacter actors and assigns the closest one to be the DiverReference

Setup Photography Camera

A screen shot of a computer

Description automatically generated

The above diagram shows how the ADivebot creates a APhotographyCamera for itself, to be used for all the Photography interactions.

1. On BeginPlay(), the SetupPhotographyCamera() method is called.
2. In this method, if the CameraActor is null and the CameraTemplateClass is not null, it checks if the server is the user calling this method, if so, it spawns and actor using the CameraTemplateClass and assigns it to the CameraActor variable. It then attaches it to the ADivebot and sets it to be on the front of the ADivebot.
3. It then sets the Owner of the CameraActor to be the ADivebot.

Aim Photography Camera

A screen shot of a computer screen

Description automatically generated

The above diagram shows how the user aims the APhotographyCamera so they can later take a picture.

1. When the Aim Camera input is received, the AimCamera() is called.
2. In this method, if the CameraActor and the ADiverController are not null, it calls the AimCamera() method on the CameraActor, passing in the ADiverController.

Stop Aiming Camera

A screen shot of a computer

Description automatically generated

The above diagram shows how the user stops aiming the APhotographyCamera.

1. When the Aim Camera input is released, the StopAimCamera() method is called.
2. In this method, if the CameraActor and the ADiverController are not null, it calls the StopAimingCamera() method on the CameraActor, passing in the ADiverController.
3. Finally, call SetViewTargetWithBlend() on the ADiverController, passing in a reference to this ADivebot.

Zoom In/Out Photography Camera

A screenshot of a computer program

Description automatically generated

The above diagram shows how the user controls the zoom level on the APhotographyCamera.

1. When the Zoom In or Out input is pressed, the ZoomIn() (or ZoomOut()) method is called.
2. In these methods, if the CameraActor and the ADiverController are not null, it then checks if the user is aiming.
3. If true, it calls the ZoomIn() (or ZoomOut()) method on the CameraActor.

Take Picture

A screenshot of a computer

Description automatically generated

The above diagram shows how the user takes a picture with the APhotographyCamera.

1. When the TakePhoto input is pressed, the TakePhoto() method is called.
2. In these methods, if the CameraActor and the ADiverController are not null, it then checks if the user is aiming.
3. If true, it calls the TakePicture() method on the CameraActor.

All Movement will be shown in the Divebot Movement Component Module

# **6 Use Case View**

Setting up Photography Camera Template

A screenshot of a computer

Description automatically generated

This the Divebot Blueprint, this will allow you to swap out the Photography Camera template as needed. The Blueprint needs to be derived from APhotographyCamera.