Project Dreamer / Interaction Module

Architecture/Design Document

Change History

Version: 0.1
Modifier: Joshua Griffis
Date: 12/03/2022
Description of Change: Design Document started

Version: 0.2
Modifier: Joshua Griffis
Date: 15/03/2022
Description of Change: Use Case & Detailed View updated

Introduction

This document describes the architecture and design for the Project Dreamer application being developed by Radical Dreamers. Project Dreamer is a Third Person Role Playing Game where you explore the dream world and interact with its inhabitants and fight bad dreams.

The purpose of this document is to describe the architecture and design of the Project Dreamer 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 minimise 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 specialise 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 into the future.

Design Goals

The goals we decided on for the design were:

- Easily implemented into objects we wanted to be interactable
- Player will raycast to check if there is an interactable in front of them.

The Interaction Module uses an Interface class as a base parent. An object that inherits this Interactable interface can then be interacted with by implementing what it does in the Interact function in the object. Interactables will include anything the user will be able to interact with like chests, NPCs, signs, pickups, doors, etc.

With this system, when the user presses the “interact” button, it will allow the user to check what they are looking at and check if that object has the interactable interface. If it does have the interface, the object will then run the features it is designed to do as an interactable.

System Behaviour

The use case view is used to both drive the design phase and validate the output of the design phase. The architecture description presented here starts with a review of the expected system behaviour in order to set the stage for the architecture description that follows. For a more detailed account of software requirements, see the requirements document (GDD).

The interaction in the game will be using an Interface class called Interactable with only one function called Interact. When the player raycasts to check for an interactable it will check to see if that object has that interface, and if it does, it will use the objects Interact function.

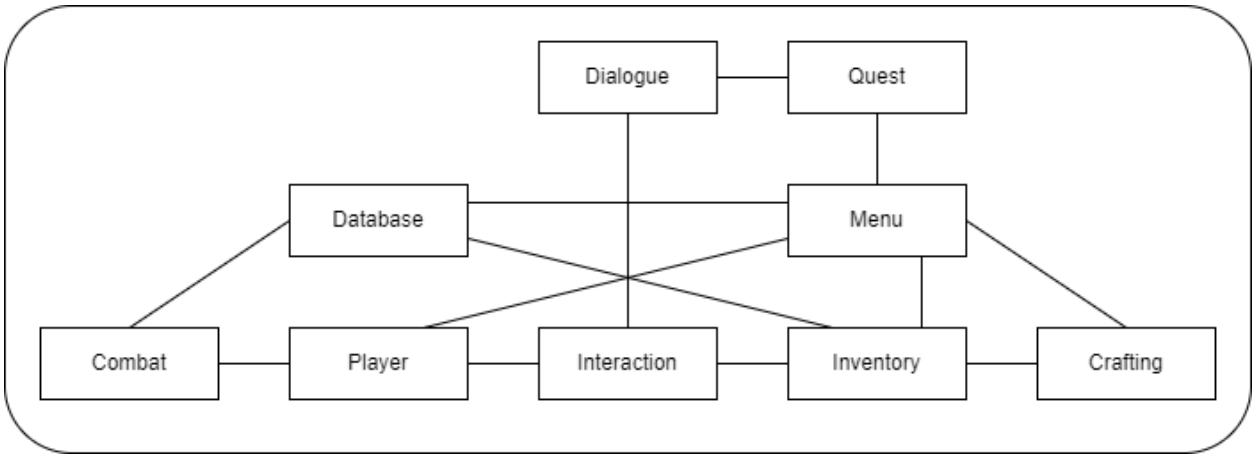
Logical View

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

High-Level Design (Architecture of the Entire system)

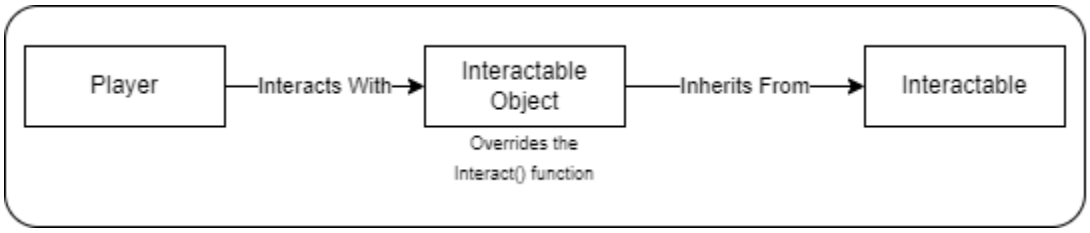
The high-level view or architecture consists of 9 major components:



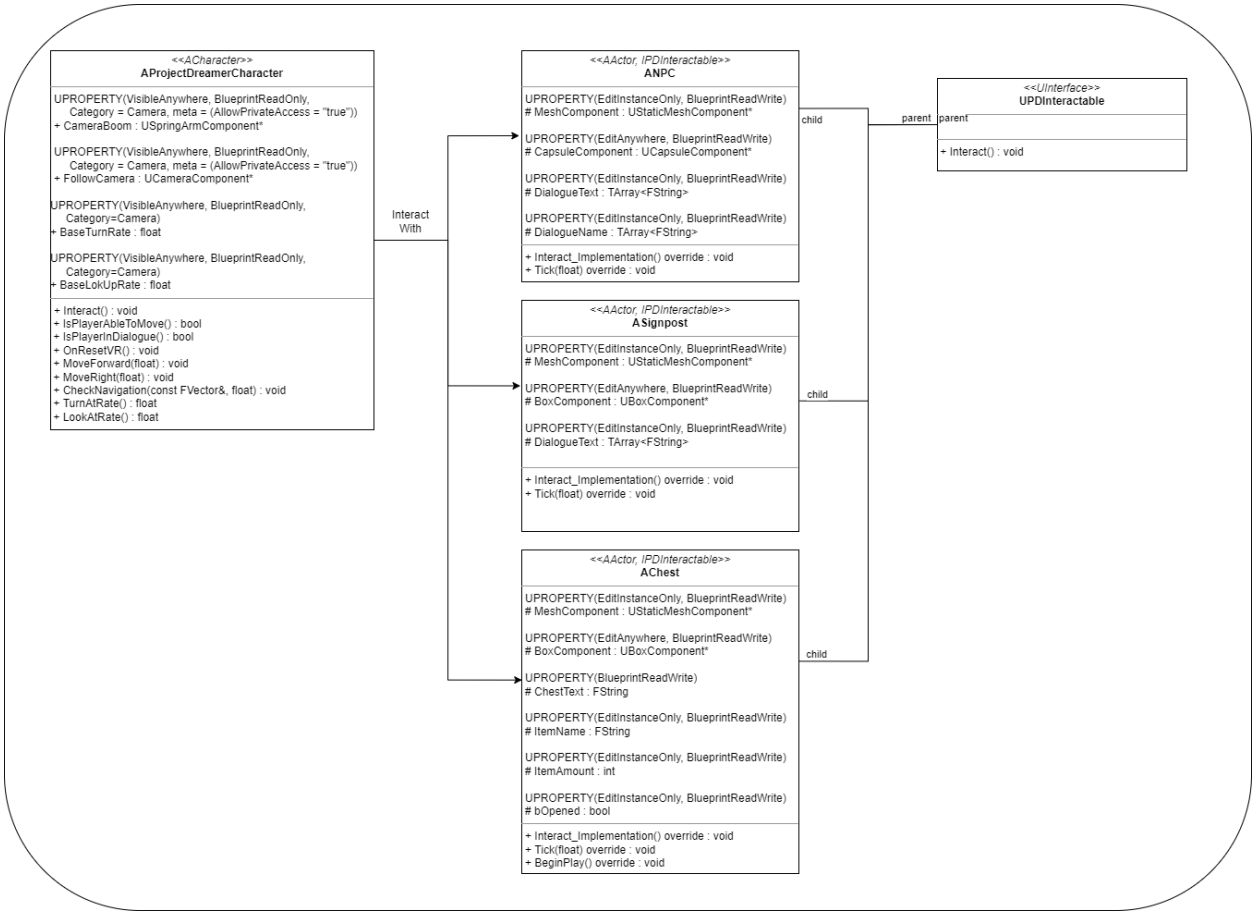
- **Player:** is the main control over the character in the world and allows the user to interact with the world.
- **Database:** stores the data needed for many features to work.
- **Interaction:** handles the objects the player can interact with like talking to characters, opening chests, opening doors, etc.
- **Dialogue:** is responsible for handling the flow of conversations and displaying dialogue to the user.
- **Inventory:** manages the items picked up by the player and money stored. Allowing the player to use them at a later point.
- **Menu:** allows interaction with some modules and features.
- **Combat:** allows interaction between enemies and player characters as they fight. Controls the flow of battle and which battler may act.
- **Quest:** allows the user to have stored data for quests and be able to receive rewards upon completion.

- **Crafting:** allows the player to combine multiple items together to create new items

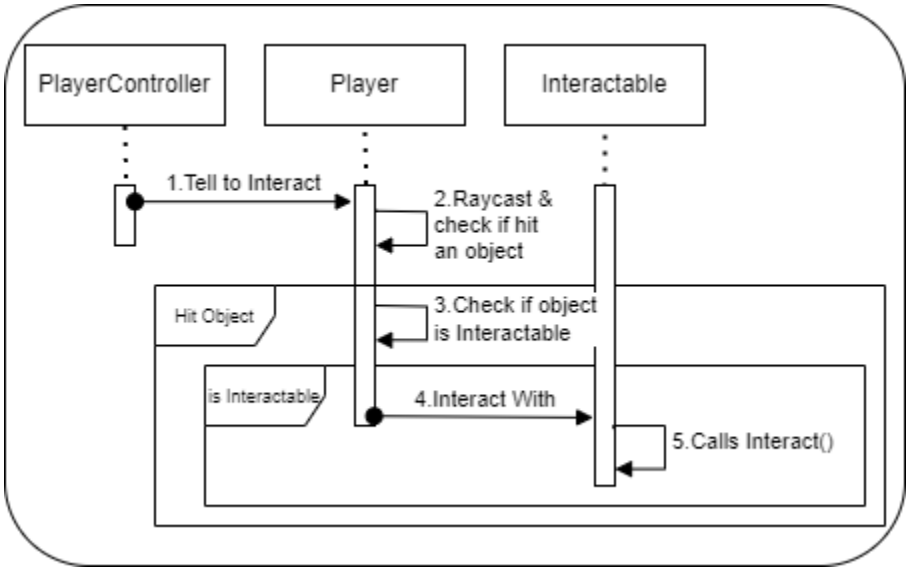
Mid-Level Design of Interaction Module



Detailed Class Design of the Interaction Module



Process View of the Interaction Module



When the Interaction button is pressed, the Player Controller will tell the Player to check if the object is an interactable and if it is, it will use the interactable's Interact function.

Use Case View

Setting up Interaction

To set up an Interactable object, you must have the object inherit from the IPDInteractable Interface class. Then you must override the Interact function and implement all your logic in that function.

```
AChest : public AActor, public IPDInteractable
```

```
  virtual void Interact_Implementation() override;
```

After the class is finished you can create a blueprint out of it and be able to place it in the level.