Project Dreamer / Inventory 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: Updated Mid and Detail View

Version: 0.3
Modifier: Joshua Griffis
Date: 13/04/2022
Description of Change: Updated Detail View

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:

- Store data of what items the user has acquired
- Store the money of the user
- Store data in way to conserve data
- Easy to put Item into the inventory
- Can check what items are in our inventory and how much money user has

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 inventory system will store a struct that holds a name and amount for easy management. When the inventory needs data on the item it will use the name of the item to find its data in the Database Manager. Each item name is unique as to not have any conflicts with other possible items.

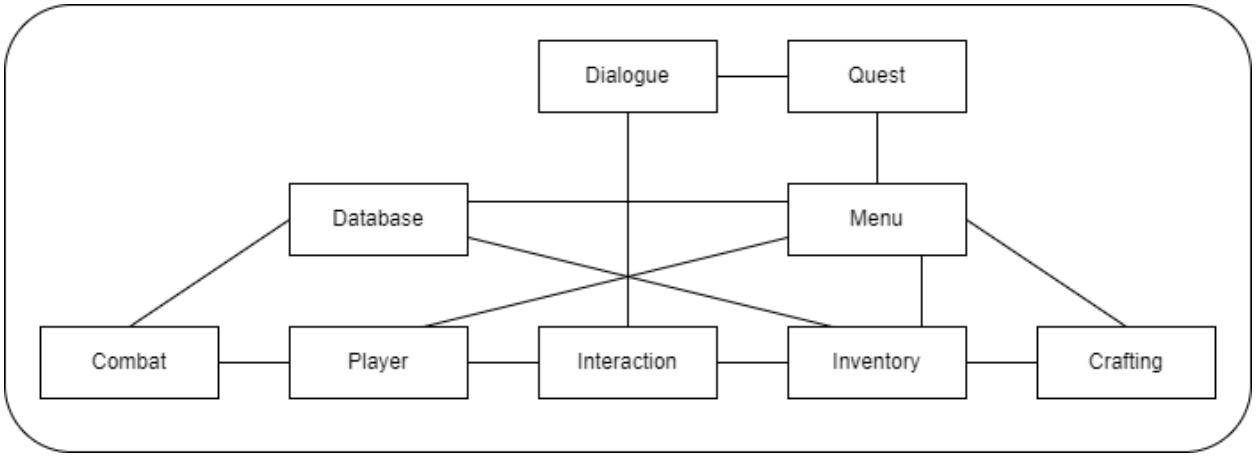
Logical View

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

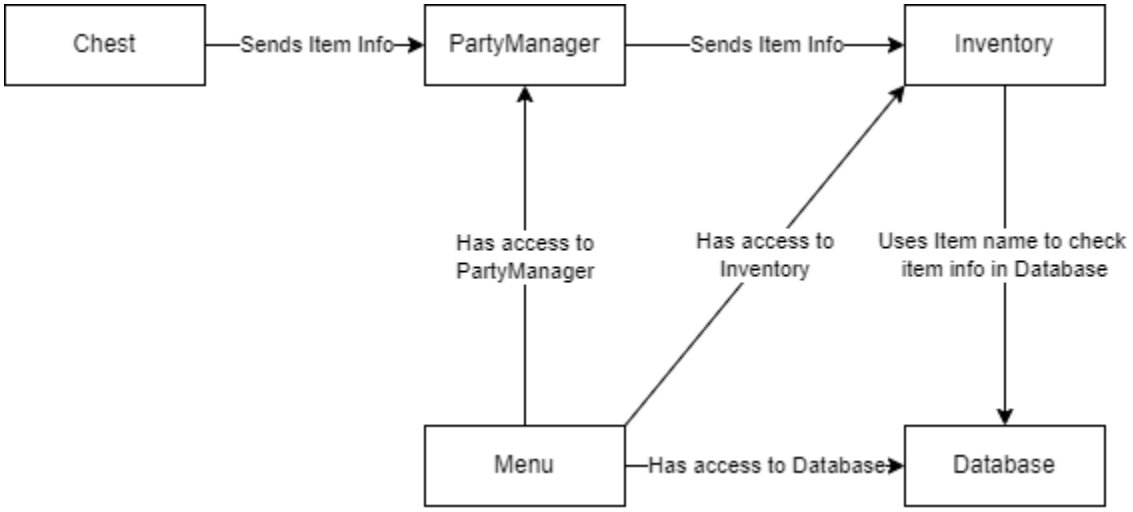
High-Level Design (Architecture of the Entire system)

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 Inventory Module

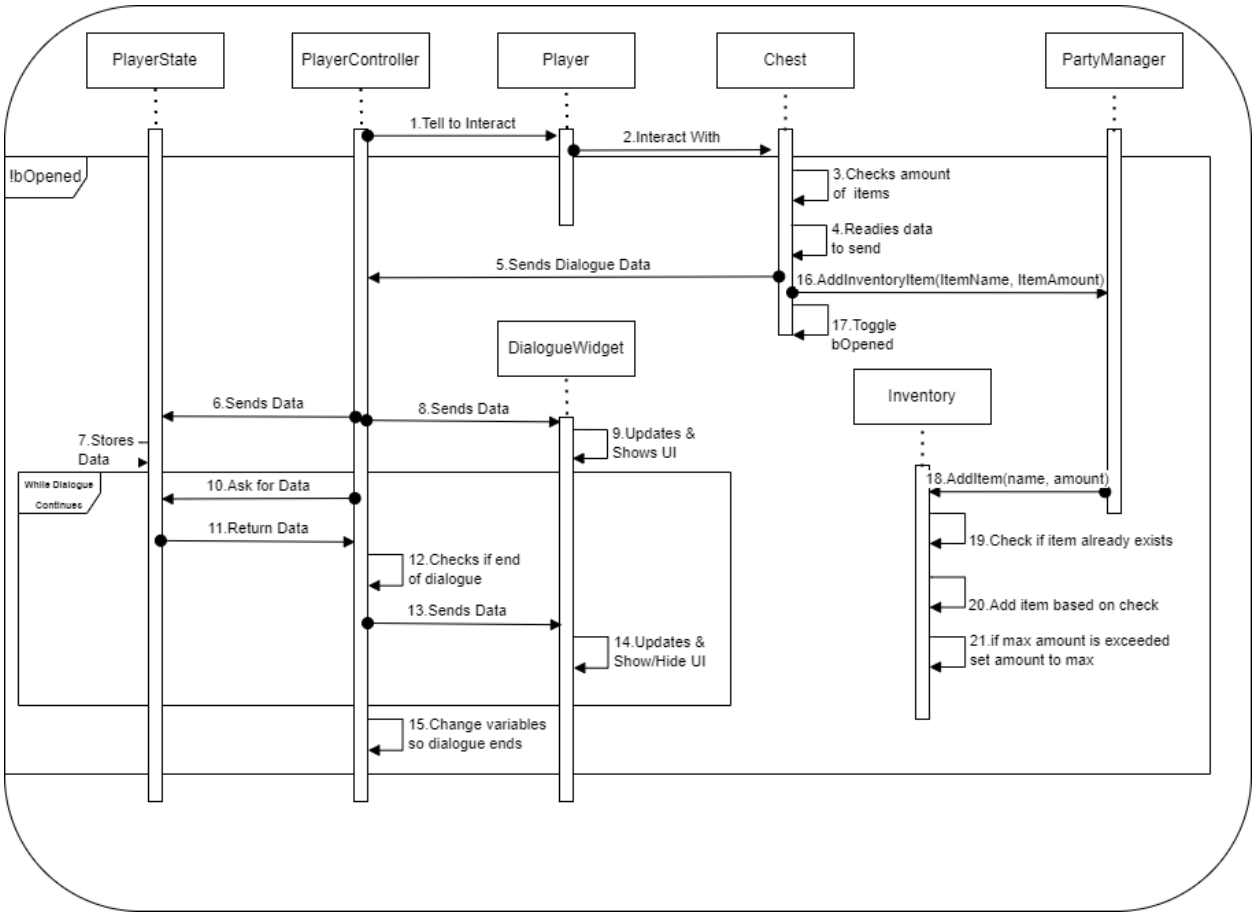


Detailed Class Design of the Inventory Module

See UMenuWidgetRelationsUML.png

Process View of the Inventory Module

Below we have the process of opening a Chest and how it works with multiple systems:



When the chest is unopened, the Chest will set the dialogue text for the user to read and then sends the name and amount of the item to the Party Manager and then toggling that it is now opened. The Party Manager then sends the item info to the Inventory and then does checks to see if the item already exists in the inventory and one of two outcomes occurs:

- The item exists and the amount is added onto the existing item
- The item is not in the inventory and it is then added with its amount

If the max amount is exceeded, then the amount of the item is set to the max, in this case it being 99 as our max for item amounts.

Use Case View

Setting up a Chest

When setting up a chest there are two major variables that interact with the inventory and that is:

- The **Item Name**
- The **Amount** of the **Item**

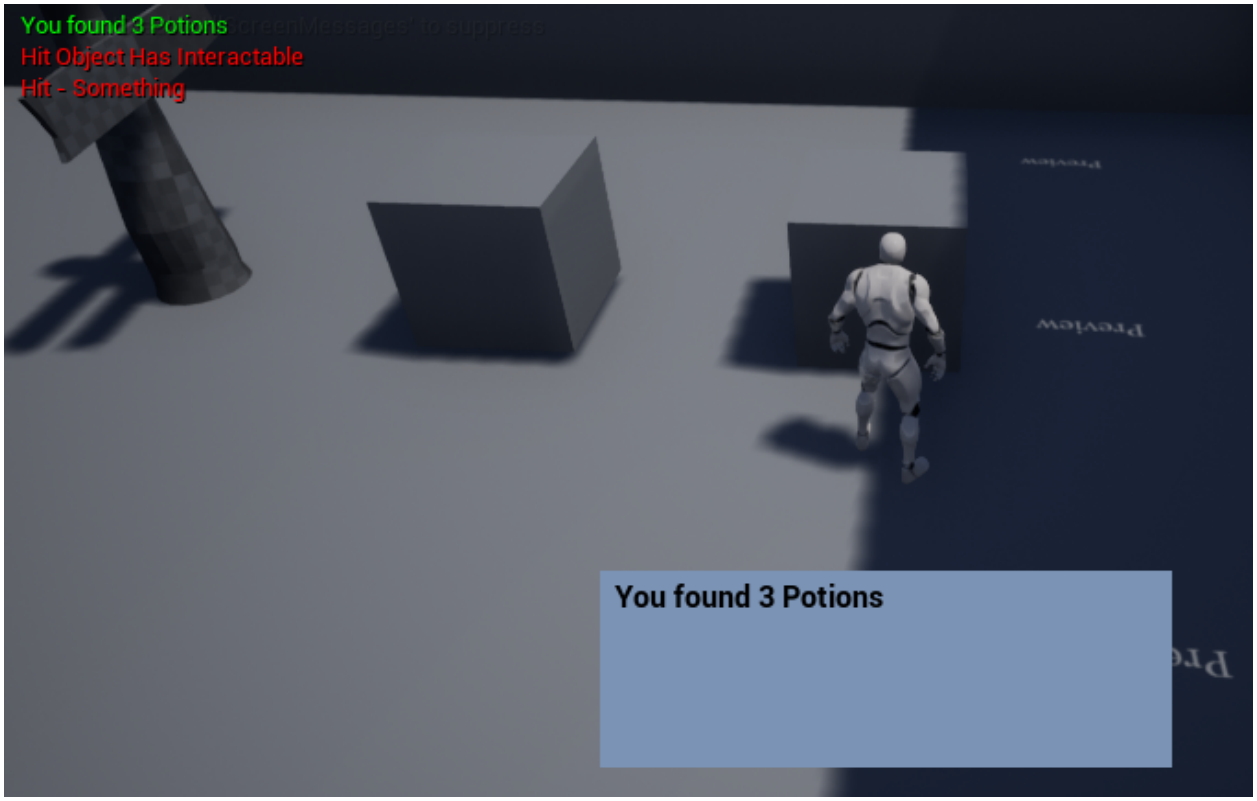
I have two test cases for a Chest.

1.

2.

When the chest is opened we have these 2 things happen.

1.



In these cases the text has shown us we have found one Potion or three Potions to let the player know they have obtained the item. In the top left we have a debug message that is shown when it is successfully added into the inventory so the designers know it is successfully in the inventory since the menu system is still being worked on to allow the player to see and use items from a menu we are using the debug message to know for sure that it is working.