Project Dreamer / Dialogue 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 Process View & Detailed 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:

- Easy implementation by level designers.
- Easily managed dialogue, with multiple results based on the designer's choice.
- Ability to be expanded in future versions.

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 dialogue system is paired with some interactable objects. When interacting with these objects it activates the dialogue system and feeds the line or lines and possible names of the character talking.

Created objects that interact with the dialogue system like NPCs, Signposts, and Chests with more possibilities to come. Each of the objects are customised only in the level with no defaults, After placement into the level the designers can then make each one have unique dialogue.

Once interacted with, the character will pull the dialogue from the object and store it in the player state and will freeze character movement and more interaction will continue dialogue if there are multiple lines until there are no more lines. It will also read names to let us know who is talking in the box which has the names shown.

Future behaviour might include choices, branching dialogue, quests, and dialogue that changes based on events of the game.

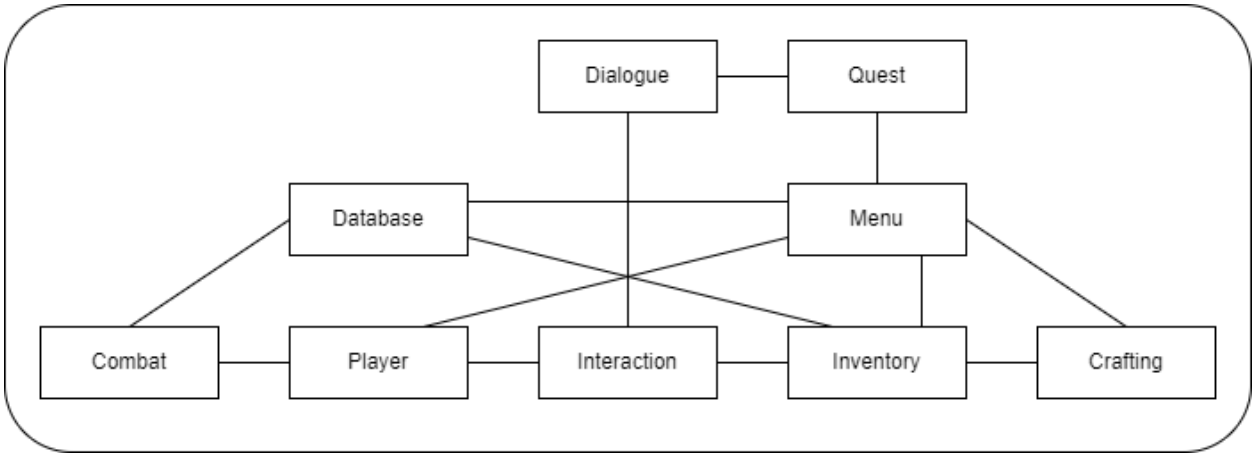
Logical View

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)

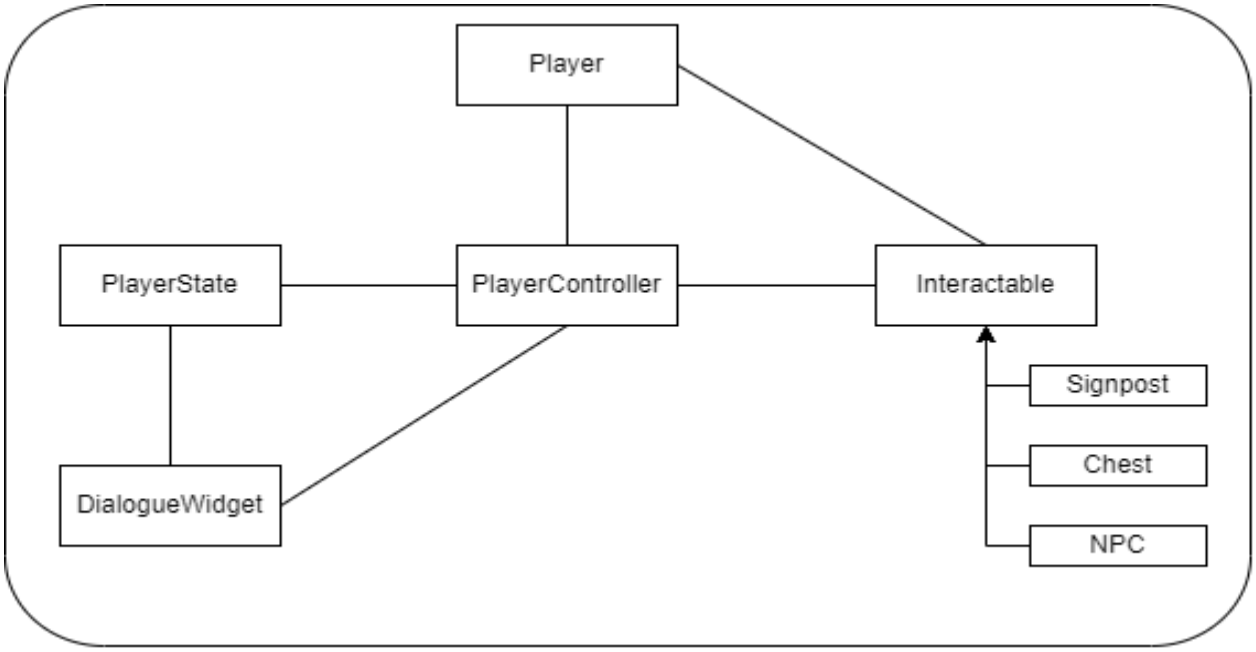
The high-level view or architecture consists of 9 major components:



- **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.

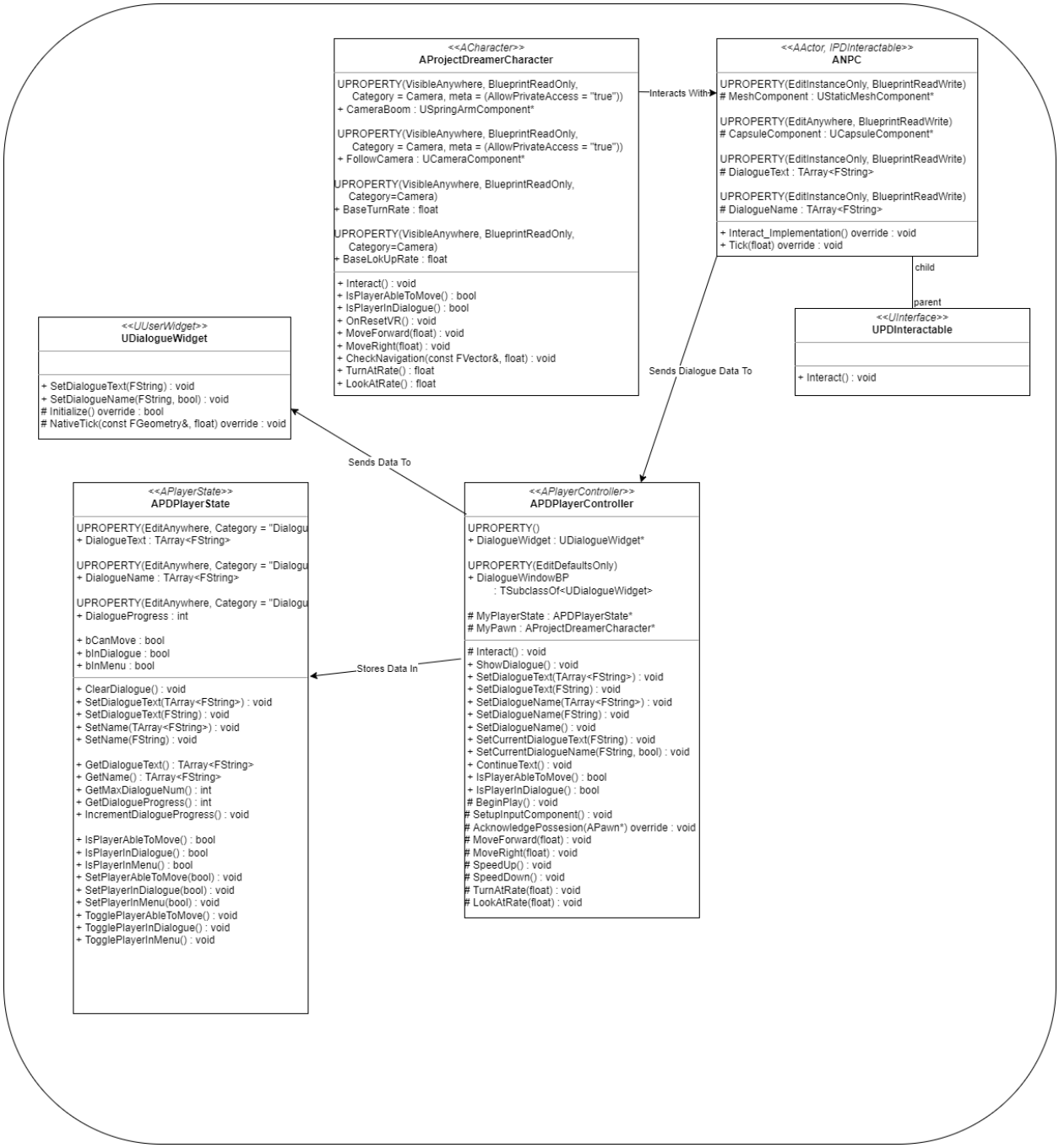
- **Combat:** allows interaction between enemies and player characters as they fight. Controls the flow of battle and which battler may act.
- **Quest:** allows the user to have stored data for quests and be able to receive rewards upon completion.
- **Crafting:** allows the player to combine multiple items together to create new items

Mid-Level Design of Dialogue Module



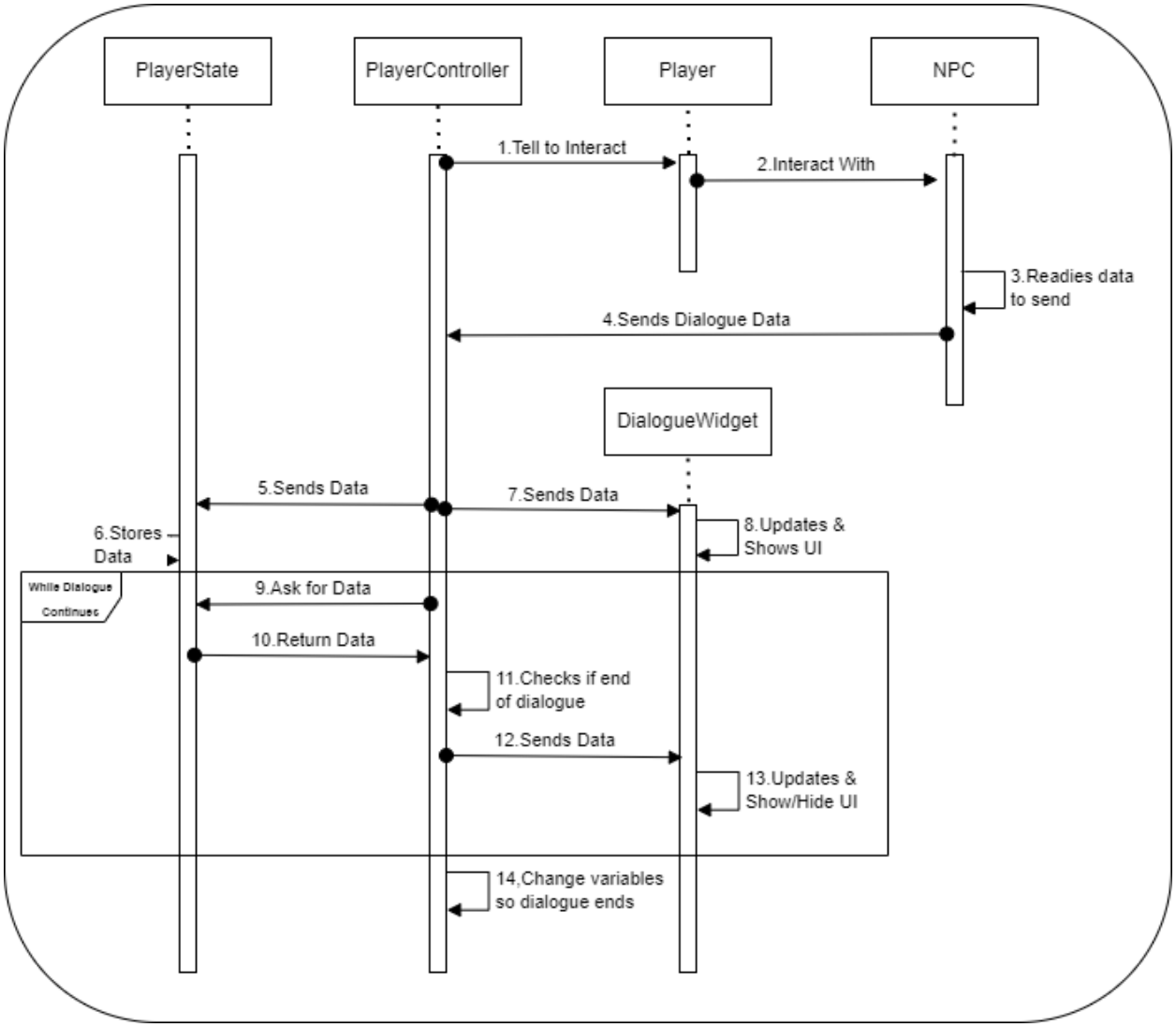
- **Player:** The player object. This object if the interact function detects that the object hit has an interactable and has dialogue and will start the dialogue sequence.
- **Interactable:** This is an Interface that objects can have to determine if the player can interact with it.
- **PlayerController:** The player controller will be the connection between the interface function and the player state which will store the data for the dialogue.
- **PlayerState:** The player state will store data for the current dialogue sequence and will allow the character to continue through the dialogue without the need for more raycasts on the interactable.
- **DialogueWidget:** The dialogue widget is in control of displaying the dialogue and showing or hiding elements of the system.

Detailed Class Design of the Dialogue Module



Process View of the Dialogue Module

This will show the process of interacting with an NPC and starting and continuing dialogue until completion.



When the Player interacts with an NPC the NPC will the dialogue data will then be stored into the Player State and the first line will be sent to the Dialogue Widget to enable, update and show it.

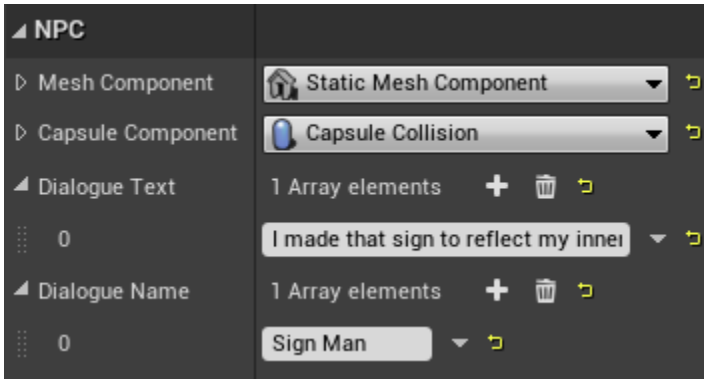
This is when the loop begins as the Controller asks for the data each time we use the Interact Button. Data will then be retrieved and checked in the controller and sent to the Dialogue Widget where multiple cases can happen.

- No more dialogue so the loop ends and the Widget is disabled.
- Dialogue is updated with a new line of dialogue and name.
- Dialogue is updated with a new line of dialogue but no name, so the Name Box is disabled.

Use Case View

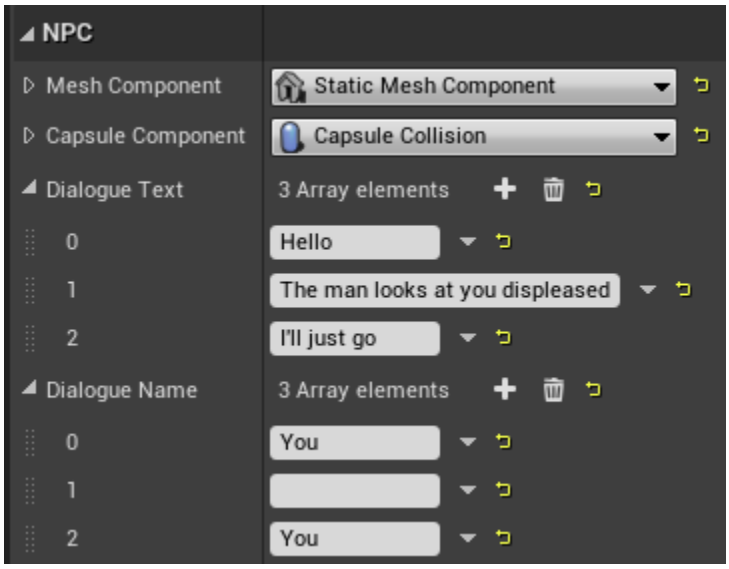
Setting up an NPC

First, move an NPC blueprint into the level. You will be able to adjust the NPC's Mesh, DialogueNames, and DialogueText fields to suit your needs.

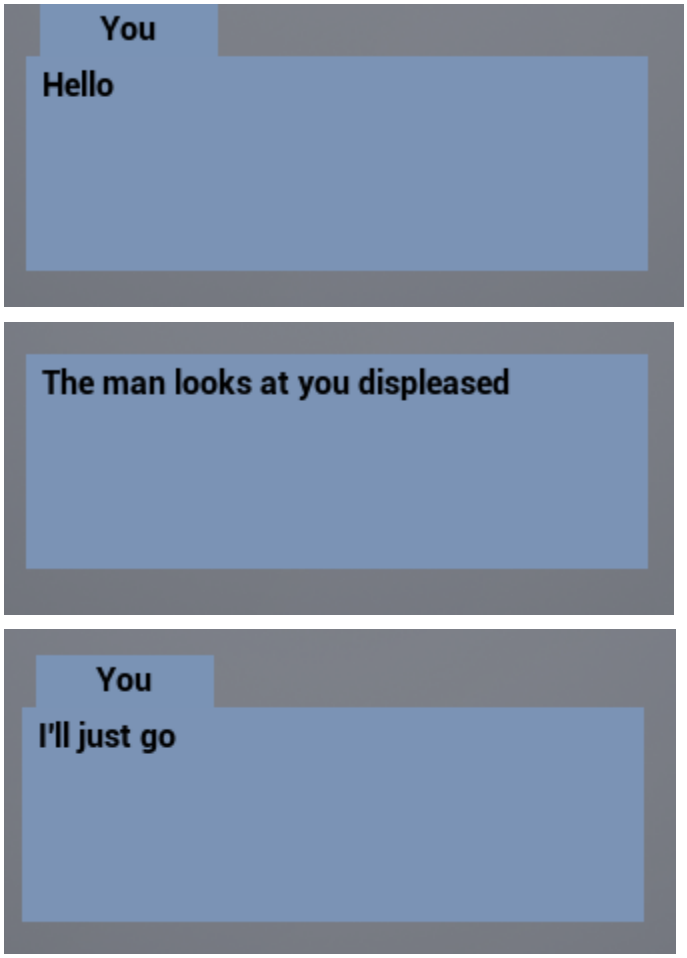


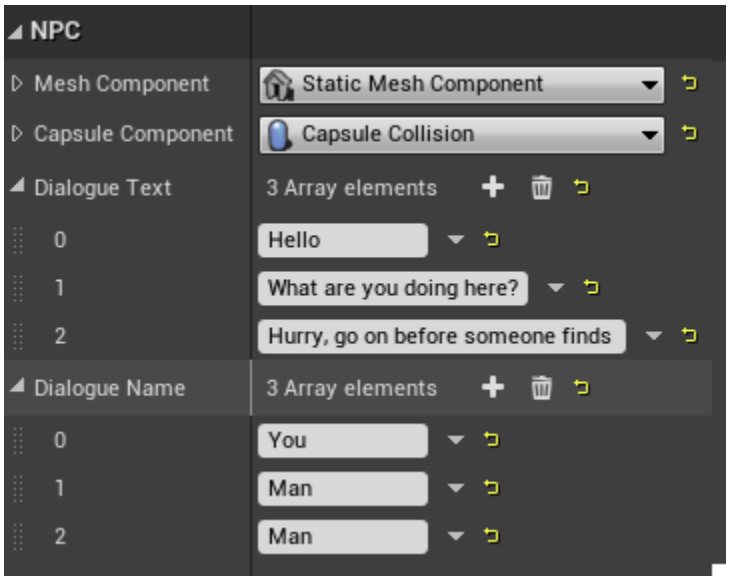
- **Mesh Component:** Appearance of the NPC.
- **Capsule Component:** Collision for NPC.
- **Dialogue Name:** Names to accompany the dialogue to know who is speaking.
- **Dialogue Text:** The text that displays.

You can also set it up in many ways where sometimes the name won't show, like this example below.



This Dialogue Sequence would appear like so:





You can also go ahead and fill in everything like normal

There are also checks to make some things easier like in this image if you instead make the Dialogue Name vector size only be 2 and have the Dialogue Text size be 3, it will use the last name read for the last text to make it a little bit less tedious.