Department of Computer Science & Engineering
The University of Texas at Arlington

Architectural Design Specification
CSE 4317: Senior Design II
Fall 2022

Euporie Studios
Project Calamity

Matthew Green
Robert Kembel
Jacob Lanham
Bradley Miettinen

REVISION HISTORY

Revision	Date	Author(s)	Description
0.1	08.25.2022	RK	document creation
0.2	08.26.2022	JL, RK, MG, BM	document section initialization
0.3	09.03.2022	JL, RK, MG, BM	first draft of diagrams
1.0	09.04.2022	JL, RK, MG, BM	complete document draft

Contents

List of Figures

List of Tables

1 Introduction

This section provides the reader with an overview of Project Calamity. The basic components which make up the game are explained here. Project Calamity is a PC video game which takes input from the player via mouse and keyboard which in turn interacts with the game software to control the Player Character. Next there are a series of game components the player will interact with which make up the bulk of the functionality of the game.

The Environment includes all non-character assets which make up the world. Within the environment, the player will find various non-player character (NPC) inhabitants. These include enemies, allies, and bystanders such as shopkeepers which are used to add life to the environment. As the player explores the world, they will require a means of gathering resources - such as equipment - to face the challenges ahead. This will be done via a Quest System - at fixed points in the game, the player can choose to complete quests which will provide a reward of money, gear, or some other kind of item. The Quest System also tracks which quests are unfinished, in progress, or complete. The Inventory System will be implemented to allow both players and NPC's to carry and access a fixed number of items. The player can use this to equip and remove clothes, weapons, money, etc. as well as trade items. The Combat System controls how players and enemies fight within the game as well as how assets respond to damage. The User Interface is what the Player sees on screen as the Player Character interacts which each of the various game components. This includes everything from menus and dialogue boxes to health bars and navigation aides. Finally, Player Management Stations (PMS) are where many of the various layers and subsystems come together to be managed. The PMS are a set of terminals placed throughout the game which allow the player access to menus for managing quests, buying items, upgrading suits and weapons, saving the game, and much more.

2 System Overview

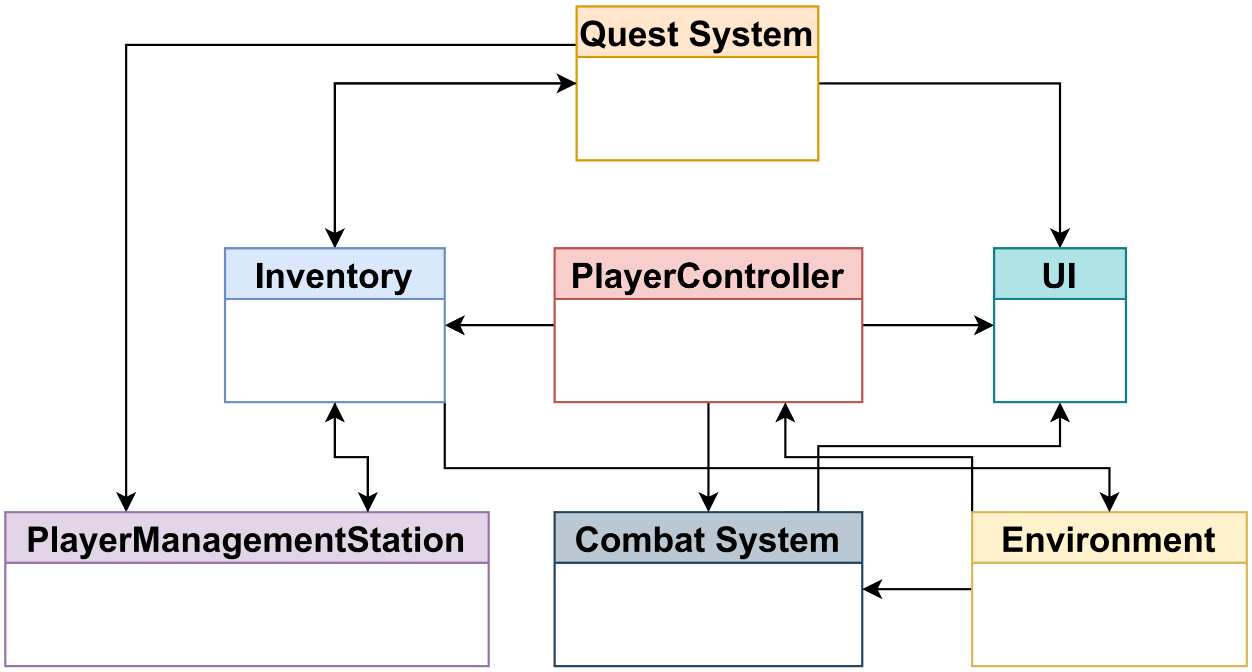

Figure 1: Project Calamity architectural layer diagram

2.1 Player Controller Description

The Player Controller handles all the input from the user and converts it to in-game actions such as moving, looking, and toggling menus. It also handles gravity and inertia acting on the player.

2.2 Inventory Description

Inventory holds all the data on what the player has equipped and what is contained within said equipment. It also manages moving around items between and inside of equipment slots and containers.

2.3 Player Management System Description

There are two different types of PMS, an alpha and beta version. Alpha PMS have access to all modules and functionality while Beta PMS have less modules and reduced functionality. These stations allow the player to upgrade their suits and weapons. It also allows them to purchase recipes for crafting and stash items.

2.4 Combat System Description

The Combat System is in charge of managing all the combat-specific systems and assets of the game. For Project Calamity, this includes weapons, enemies, and the health management system. The Player Controller interacts with the Combat System via the Weapon Controller to indicate when a weapon is being activated. From there, the Weapon Controller regulates the weapons and there characteristics to complete attacks on targets. Enemy behavior is controlled via the Enemy Controller which also takes input from the Player Controller in the form of player location data to tell enemies where to attack. Both weapon data, enemy data, and gravity data from the Environment Layer are passed to the Health System to manage the health status of all relevant assets in the game. Health information is displayed to the player through the UI.

2.5 UI Description

The User Interface processes everything the player sees in the game besides the environment and the assets within it. Thus any menus, status bars, maps and other visual aids needed to guide the player and provide him or her with information is displayed via the UI. As many different layers of the game require displaying information to the player, the UI connects directly or indirectly to almost every system layer through one of its subsystems.

2.6 Quest System Description

The Quest System handles the giving, receiving, tracking and removing of quests. In addition, it is responsible for the distribution of rewards for completed quests. It interfaces with the inventory and UI systems in order to inform the player of quest progression and track completion of quests that require having some amount of items in the inventory, such as a fetch quest.

2.7 Environment Description

The environment involves anything that relates to the environment, such as lighting, gravity, and flora/fauna. It provides the health system with fall damage parameters and updates the movement system with gravity parameters.

3 Subsystem Definitions & Data Flow

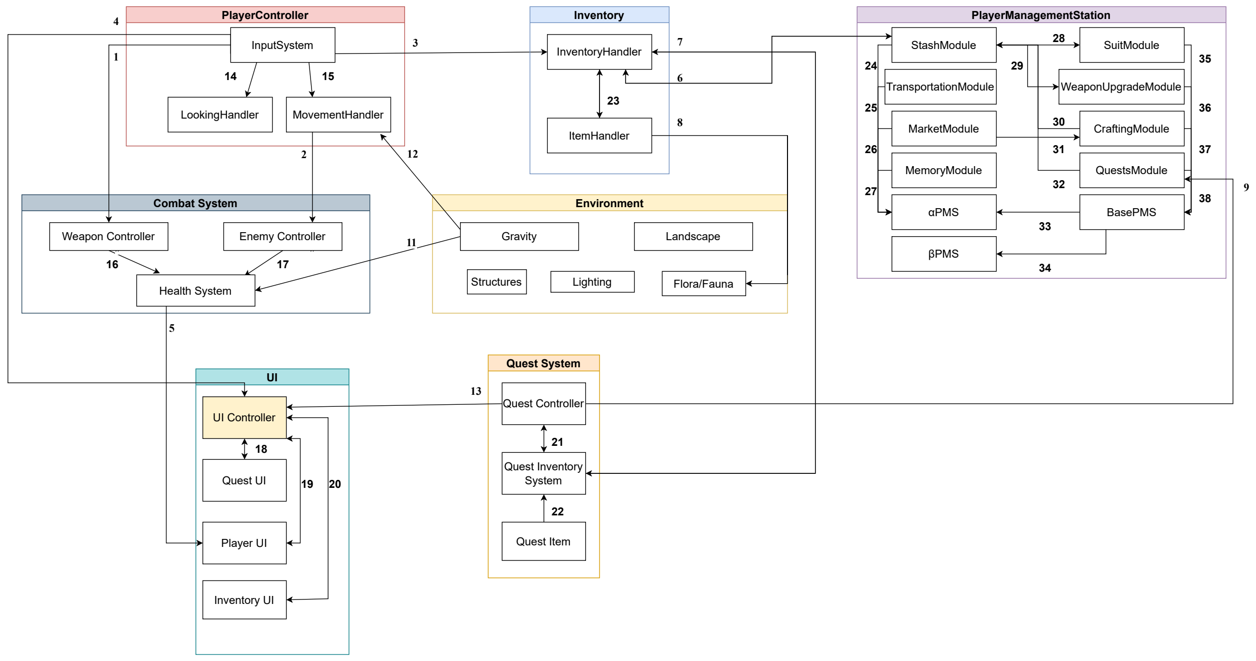

Figure 2: Subsystem Data Flow Diagram

4 Player Controller

The Player Controller handles all the inputs from the user and converts them to in-game actions such as moving, looking, and toggling menus. It also handles gravity and inertia acting on the player.

4.1 Input System

This subsystem gets the input from the user's keyboard and mouse and passes the data to various other systems depending on what the user is attempting to control.

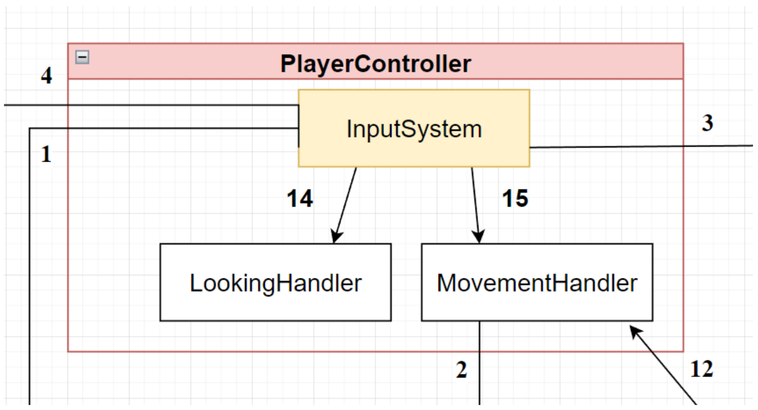

Figure 3: Player Controller: Input System Subsystem

4.1.1 Assumptions

Unity - the game engine of Project Calamity - handles all user inputs. The user input is ONLY from a keyboard and mouse. It does not support a controller, joysticks, etc.

4.1.2 Responsibilities

It gets the user's keyboard and mouse inputs every frame.

4.1.3 Subsystem Interfaces

Table 2: Input System Interfaces

ID	Description	Inputs	Outputs
#1	Weapon Controller Interface	N/A	Keyboard Input Mouse Input
#3	Inventory Controller Interface	N/A	Keyboard Input
#4	UI Controller Interface	N/A	Keyboard Input Mouse Input
#14	Looking Handler Interface	N/A	Mouse Input
#15	Movement Handler Interface	N/A	Keyboard Input Mouse Input

4.2 Looking Handler

Manages the player looking around his environment.

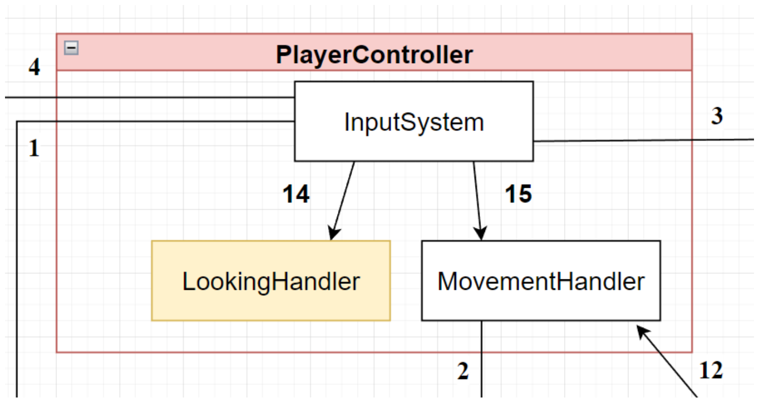

Figure 4: Player Controller: Looking Handler Subsystem

4.2.1 Assumptions

N/A

4.2.2 Responsibilities

Using the inputs of the user it rotates the player on the y-axis and rotates the camera attached to the player on the x-axis.

4.2.3 Subsystem Interfaces

Table 3: Looking Handler Interfaces

ID	Description	Inputs	Outputs
#14	Input System Interface	Mouse Input	N/A

4.3 Movement Handler

Manages the player moving around his environment as well as handling gravitational acceleration and inertia.

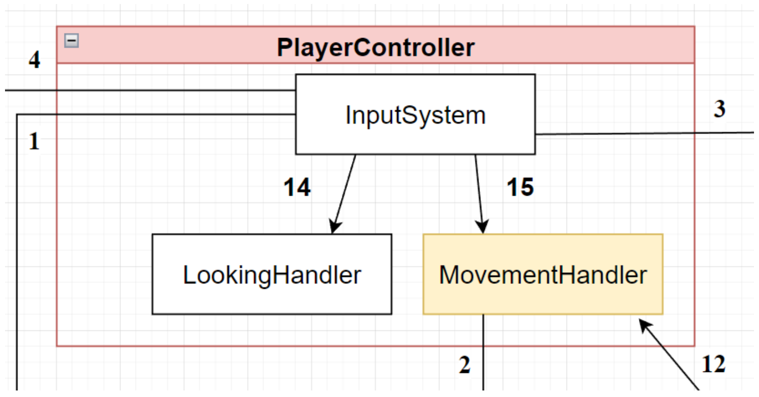

Figure 5: Player Controller: Movement Handler Subsystem

4.3.1 Assumptions

Gravitational acceleration on the planet is the same as the Earth's at 9.81 m/s. Any mesh the player can stand/move on top of is part of the Ground Layer.

4.3.2 Responsibilities

Apply a constant gravitational force upon the player. Check if the player is touching the ground. If the player is grounded apply a force upward on the y-axis when the player uses the jump button. If the player is grounded, the WASD keys are used to move the player in a one of 8 directions at a constant velocity. If the Left Shift key is down the players starts sprinting. All Actions and corresponding animations for the player's limbs and equipped items.

4.3.3 Subsystem Interfaces

Table 4: Movement Handler Interfaces

ID	Description	Inputs	Outputs
#2	Enemy Controller Interface	N/A	Player Postion Player Rotation
#12	Gravity Interface	Gravity	N/A
#15	Movement Handler Interface	Keyboard Input Mouse Input	N/A

5 Inventory

Holds all the data on what the player has equipped and what is contained within said equipment. Also manages moving around items between and inside of equipment slots and containers.

5.1 Inventory Handler

Holds all the data on what the player has equipped and what is contained within said equipment.

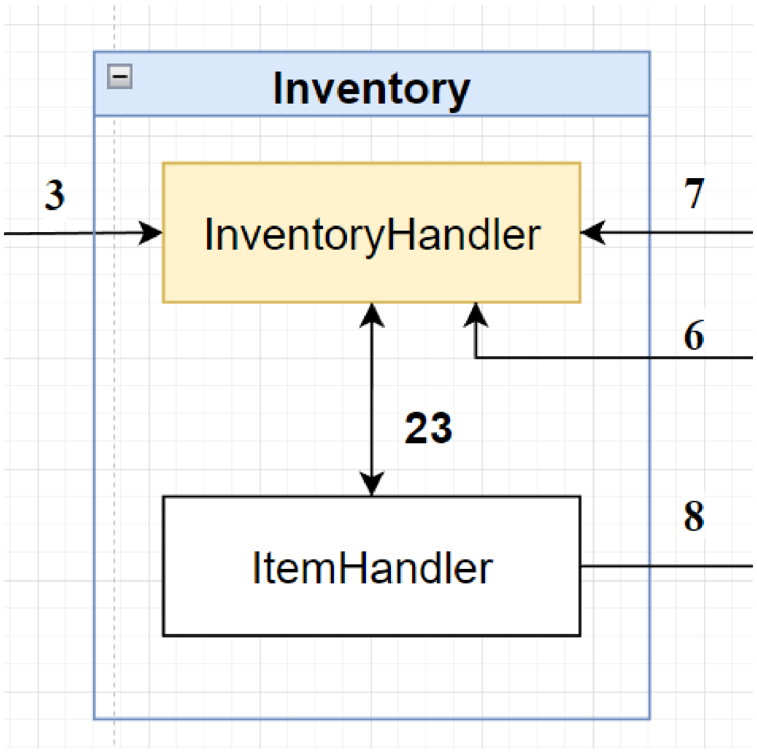

Figure 6: Inventory: Inventory Handler Subsystem

5.1.1 Assumptions

N/A

5.1.2 Responsibilities

Stores the data that the player has equipped or stored.

5.1.3 Subsystem Interfaces

Table 5: Inventory Handler Interfaces

ID	Description	Inputs	Outputs
#3	Input System Interface	Keyboard Input Mouse Input	N/A
#6	Stash Module Interface	Items	Items
#7	Quest Inventory System Interface	Items	Items
#23	Item Handler Interface	Items	Items

5.2 Item Handler

Manages moving around items between and inside of equipment slots and containers.

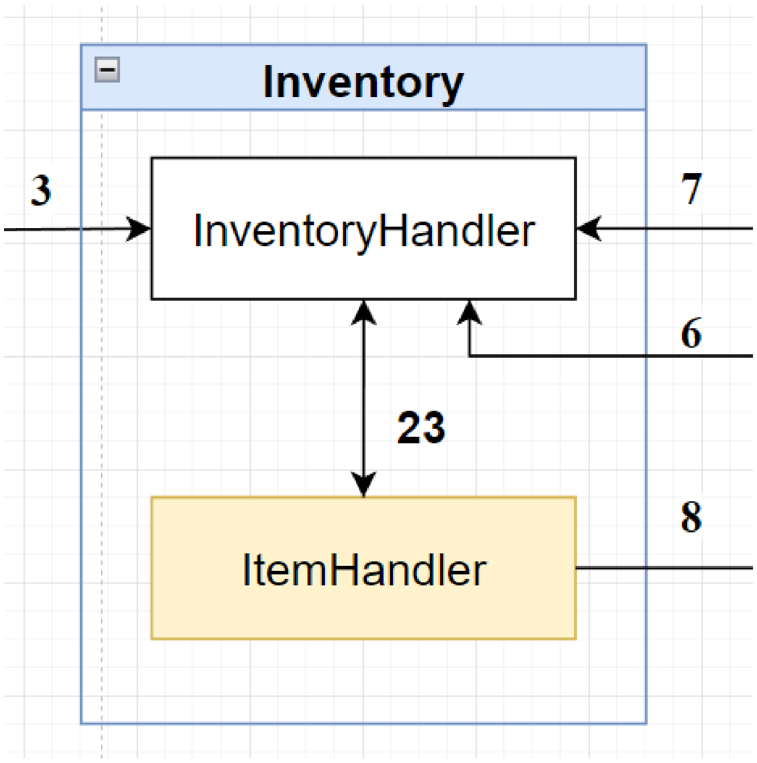

Figure 7: Inventory: Item Handler Subsystem

5.2.1 Assumptions

N/A

5.2.2 Responsibilities

Allows the user to move and rotate items between equipment slots and containers.

5.2.3 Subsystem Interfaces

Table 6: Item Handler Interfaces

ID	Description	Inputs	Outputs
#8	Flora/Fauna Interface	N/A	Items
#23	Inventory Handler Interface	Items	Items

6 Combat System

The combat system is in charge of managing all of the combat-specific systems and assets of the game. For Project Calamity, this includes the weapons, enemies, and health management system. Weapons are managed by the Weapon Controller. This is a set of scripts which define the behavior of each weapon such as fire rate, range, muzzle flash, damage to targets etc. The Enemy Controller specifies the behavior of enemies such as movement, attacks, and reactions to the actions of the player. Finally, the Health System controls the health status of all characters in the game as well as other assets capable of being damaged.

By structuring things this way, all of the combat specific functionality in the game is decoupled from the player controller which then has to interact with the combat system through specific interfaces. This allows us to add additional enemies and weapons and have the Player Controller interact with them all in the same way. The Health System was specified as its own subsystem so that any asset we wish to sustain damage can easily instance its own health data.

6.1 Weapon Controller

The weapon controller manages the characteristics and behaviors of weapons in the game. The most basic behavior of a weapon is whether or not it is active (shooting for a gun or swinging for a sword) at any present moment. The activity of the weapon depends on if the player is currently using it. Therefore this subsystem must have an input connection from the Player Controller's input system. Finally, the Weapon Controller is responsible to transferring damage information to the targets of the weapon. This is done by sending damage data to the health system which will then dictate how the damage is dealt to a target.

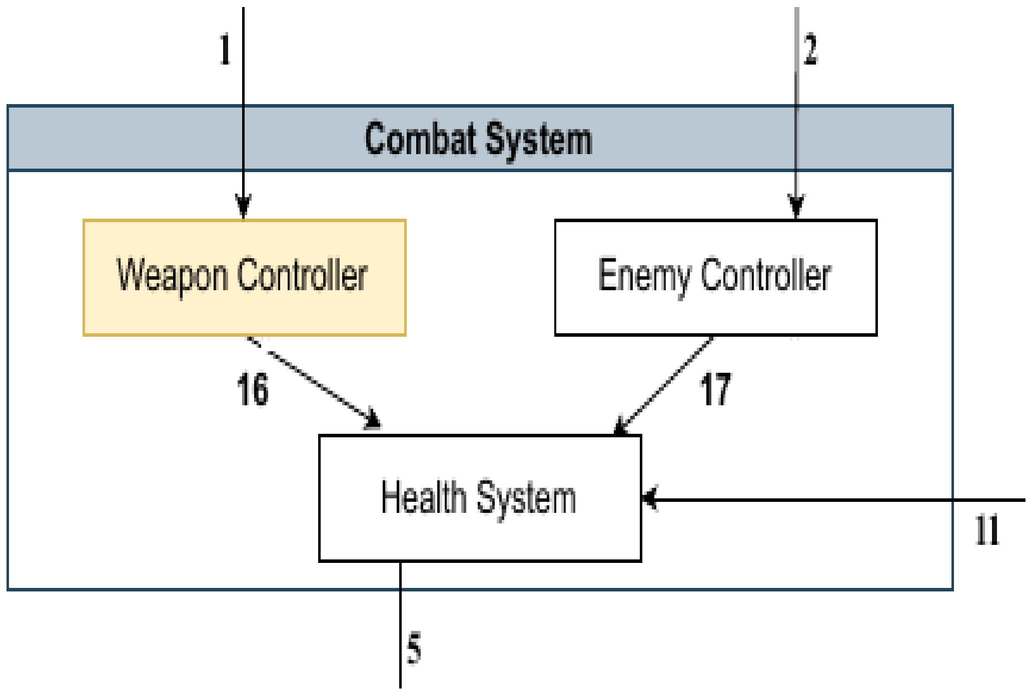

Figure 8: Weapon Controller Description Diagram

6.1.1 Assumptions

Weapon activation inputs from the player come via the Player Controller's input system. Targets of the weapon's attack update their health data via the Health System.

6.1.2 Responsibilities

The Weapon Controller manages the characteristics and behaviors of weapons in the game. For firearms, the Weapon Controller will set characteristics including - but not limited to - the fire rate, range, muzzle flash, damage to targets, audio, shooting animations, weapon health, ammunition, and reloading. It will also control the shooting behavior such as input handling and recoil.

6.1.3 Subsystem Interfaces

Table 7: Weapon Controller Interfaces

ID	Description	Inputs	Outputs
#1	Player activation of weapon system	Activity	N/A
#16	Damage dealt to weapon target or weapon itself	N/A	Target Damage Weapon Damage

6.2 Enemy Controller

The Enemy Controller subsystem controls the behavior and status of enemy characters the player will encounter during combat. Enemy behavior will be dependent on the location of the player in the game (i.e. the enemy might only actively attack when the player is nearby), so there is an interface providing player location data from the Movement Handler of Player Controller as input to the Enemy Controller. If an enemy's health status changes via means other than being attacked by a player's weapon (i.e. environment damage/healing), such data will be sent to Health System where the enemy character's health state will be updated.

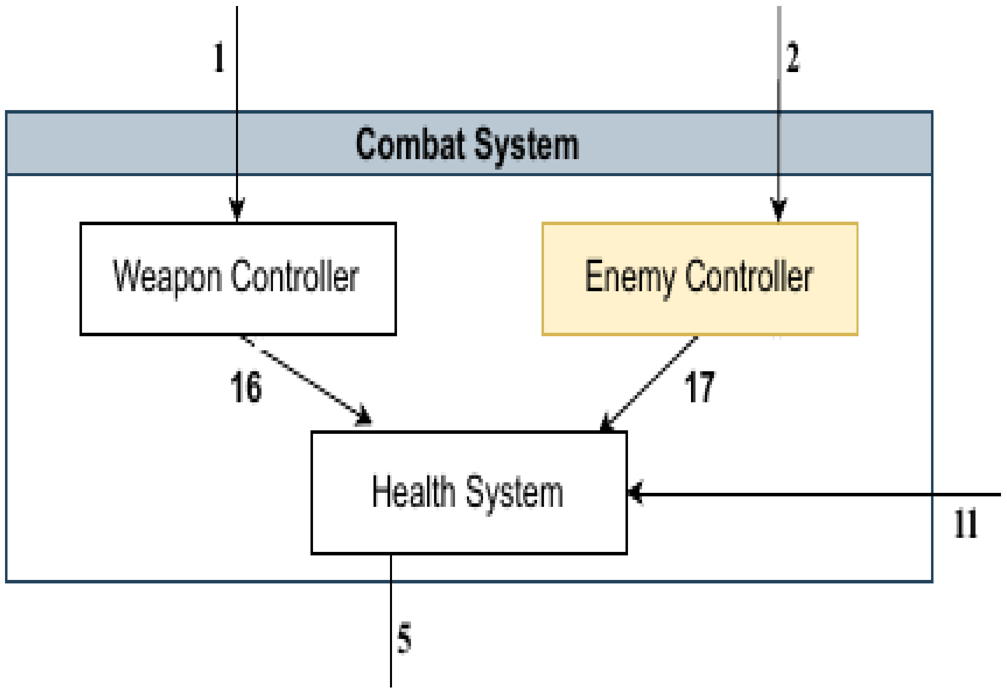

Figure 9: Enemy Controller Description Diagram

6.2.1 Assumptions

The Player Controller's Movement Handler is providing the Enemy Controller with player location data, and the Enemy Controller is providing the Health System with health status updates for the enemy which are directly caused by the enemies actions or location in the world.

6.2.2 Responsibilities

The Enemy Controller subsystem controls the behavior and status of enemy characters in the world. This includes dictating the enemy's movement and actions made in response to the player location such as attacks. If an enemy needs to spawn at a specific time, that functionality will also be controlled here.

6.2.3 Subsystem Interfaces

Table 8: Enemy Controller Interfaces

ID	Description	Inputs	Outputs
#2	Player location information	Location	N/A
#17	Enemy health status updates	N/A	Health

6.3 Health System

The Health System is in charge of managing the health data for any assets in the game which can sustain damage. This can be the player's health, enemy health, or the health of weapons meant to break after their health bar reads zero. There are three inputs and one output for the Health System. The inputs come from the Weapon Controller, Enemy Controller, and the Gravity subsystem of the Environment Layer. These inputs convey health data updates for weapons or targets of weapons, enemy characters, and fall damage dealt by the environment to the characters respectively. After the health status of an asset is updated in the Health System, this data is sent to the Player UI subsystem of the User Interface Layer to be displayed to the player as needed.

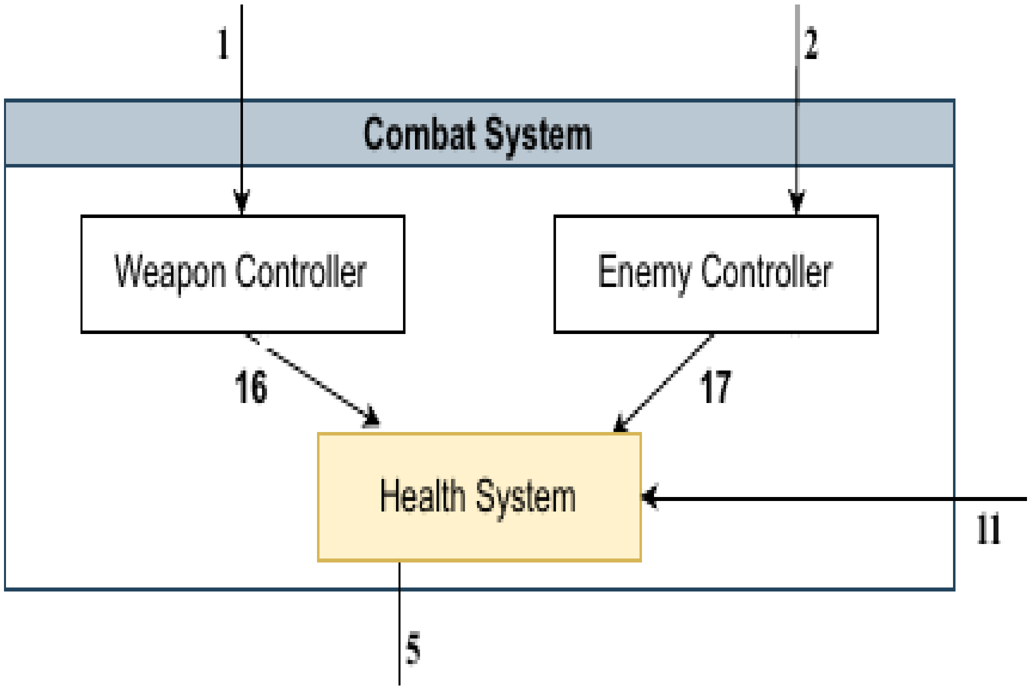

Figure 10: Health System Description Diagram

6.3.1 Assumptions

Health updates to weapons and damage dealt by weapons will be provided via the Weapon Controller. Enemy character health updates not caused by weapon damage or fall damage will come from the Enemy controller. Fall damage will be updated via input from the Gravity subsystem. All health updates will be send to the Player UI subsystem as needed for player viewing.

6.3.2 Responsibilities

The Health System is in charge of managing the health data for any assets in the game which can sustain damage. It is in charge of receiving health data updates from relevant subsystems, updating health data objects for each asset being tracked, and then sending that health data information to the UI to be viewed.

6.3.3 Subsystem Interfaces

Table 9: Health System Interfaces

ID	Description	Inputs	Outputs
#16	Health updates caused by weapons or targets of weapons	Target Damage Weapon Damage	N/A
#17	Miscellaneous enemy health updates	Enemy Damage	N/A
#11	Fall damage caused by environment gravity	Fall Damage	N/A
#5	User interface health updates	N/A	Health Updates

7 Quest System

Handles the giving, receiving, tracking, removing of quests and distribution of rewards of completed quests and any progress-tracked activity.

7.1 Quest Controller

Manages individual quest items, distribution of the quest rewards, handing out full quests, deleting quests from the quest inventory, adding quests to the quest inventory system, and forwarding current quests to the UI system.

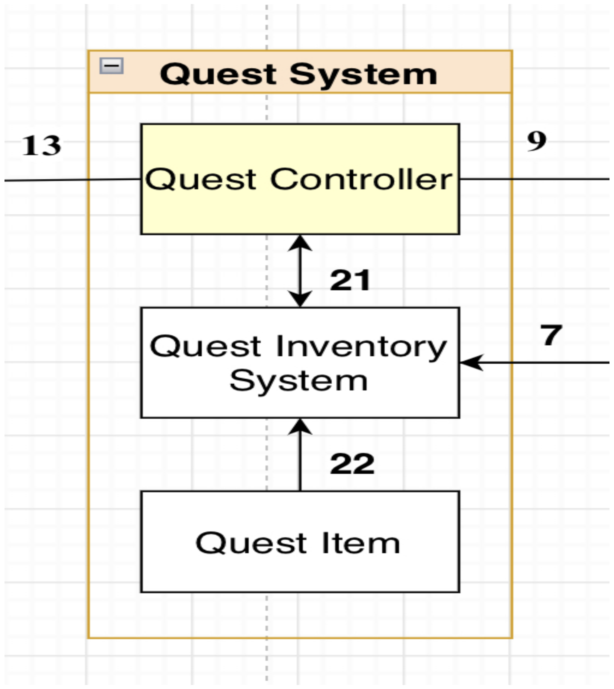

Figure 11: Quest System: Quest Controller Subsystem

7.1.1 Assumptions

The UI layer will take information given and display that to the player in a meaningful way, and the inventory system will report changes in completion status and currently held quests to the controller.

7.1.2 Responsibilities

Adding quests to the quest inventory system will group quest items into one quest and add the full quest to the quest inventory system.

Deleting quests will make sure any earned rewards are given to the player and then remove the quest from the quest inventory system. Failed quests will also be deleted after informing the UI system about the failed quest.

While keeping track of quests, it will see how much of a quest is complete to report that to the UI system. If the quest is complete it will inform the player on how to collect the rewards so the quest can be deleted from the inventory system once it has distributed rewards.

Rewards will be distributed when the quest steps are marked as complete in an appropriate way to be accessible to the player.

7.1.3 Subsystem Interfaces

The quest controller outputs to the UI System the status of current quests. The quest controller outputs to the quest inventory system any updates to the inventory system, and receives the current status of individual items in said inventory system.

Table 10: Quest Controller Interfaces

ID	Description	Inputs	Outputs
#9	QuestModule in the PMS	N/A	QuestModule
#13	UI	N/A	UI Controller
#21	Inventory System	Quest controller	Quest Controller

7.2 Quest Inventory System

Holds quest items in groups to constitute a quest and forwards status of quests to the Controller.

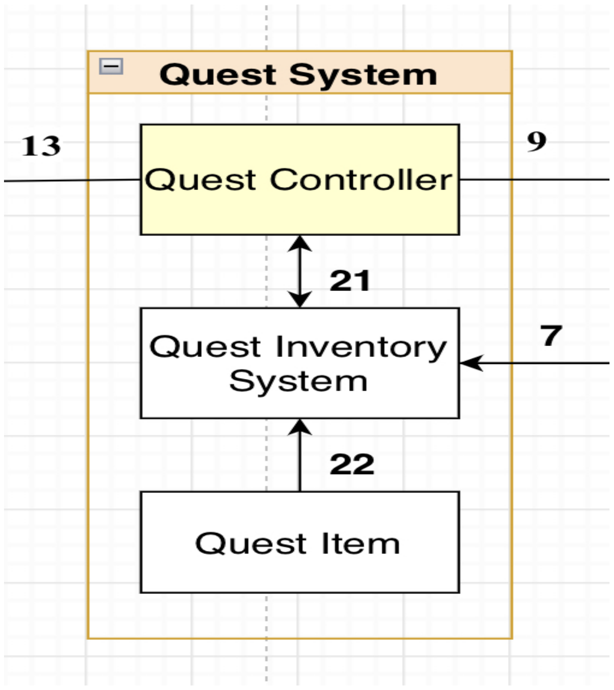

Figure 12: Quest System: Quest Inventory Subsystem

7.2.1 Assumptions

This will report information to the inventory system to constitute a subsystem to the inventory system

7.2.2 Responsibilities

Combine quest items to constitute a full quest, update any inventory items needed for quests that need to see what is in the inventory system itself, and report any relevant information to the quest controller.

7.2.3 Subsystem Interfaces

Table 11: Quest Inventory Interfaces

ID	Description	Inputs	Outputs
#7	Inventory Management	Inventory Handler	Inventory Handler
#21	Inventory Management	Quest Controller	Quest Controller

7.3 Quest Item

Individual items that constitute quest steps.

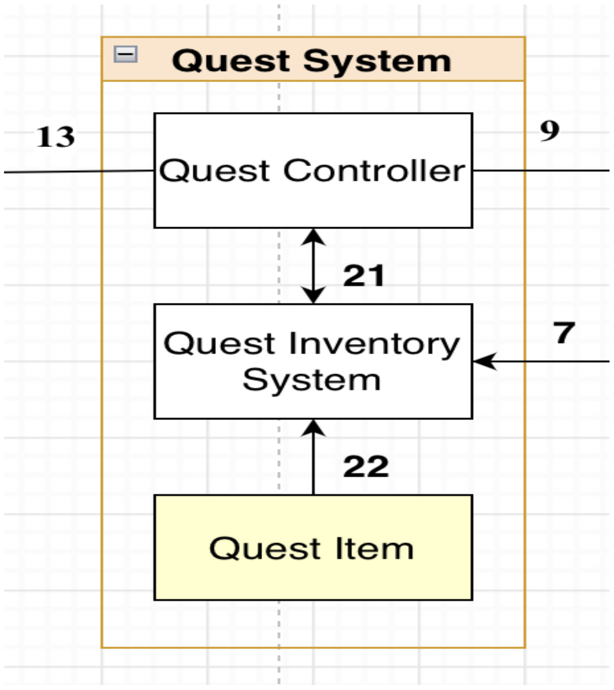

Figure 13: Quest System: Quest Items Subsystem

7.3.1 Assumptions

These will be collected into a larger item inside the Quest Inventory System to make a complete quest.

7.3.2 Responsibilities

Keep track of individual quest steps on a fundamental level.

7.3.3 Subsystem Interfaces

Informs the inventory system of every aspect that constitutes an individual quest step.

Table 12: Quest Item Interfaces

ID	Description	Inputs	Outputs
#21	Inventory Management	Quest Controller	Quest Controller

8 Environment

Handles the lighting, gravity, wildlife, and any other items involved the environment.

8.1 Gravity

Applies gravity to objects interacting with the environment.

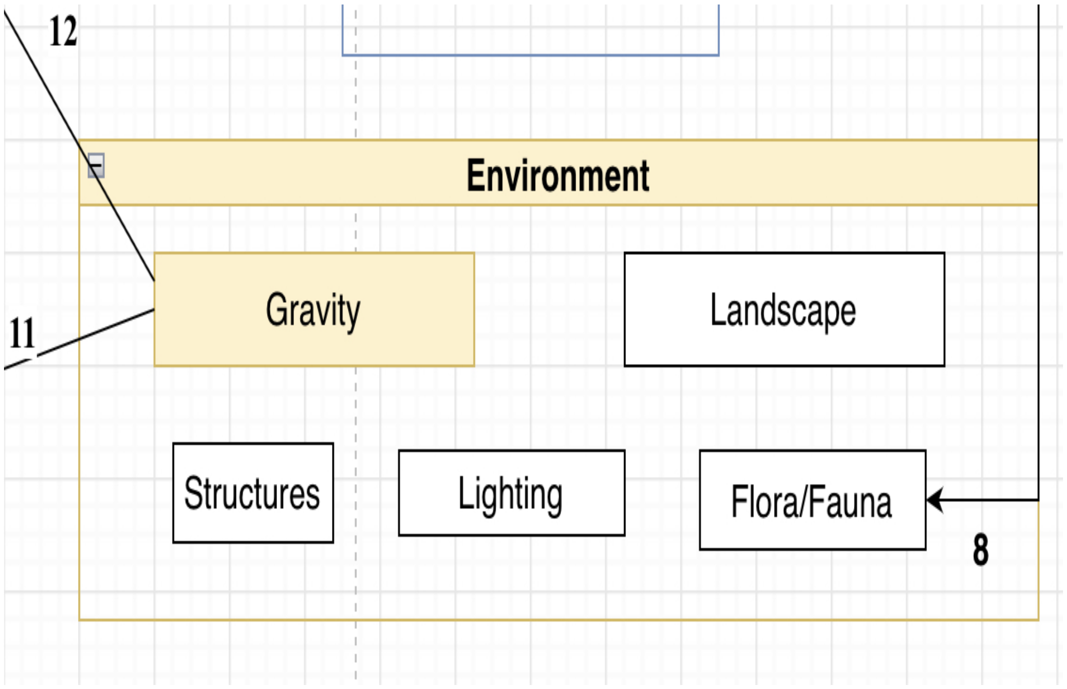

Figure 14: Environment: Gravity Subsystem

8.1.1 Assumptions

The Health System will take an input to apply damage. The Movement Handler will take an input to modify movement speed.

8.1.2 Responsibilities

Apply damage if the player takes a large fall and current gravity parameters cause a large change in velocity.

8.1.3 Subsystem Interfaces

Table 13: Gravity Interfaces

ID	Description	Inputs	Outputs
#11	Fall damage output to health system	N/A	Damage
#12	Gravity parameter output to movement handler	N/A	Gravity

8.2 Landscape

Covers the natural or modified landscape of the environment.

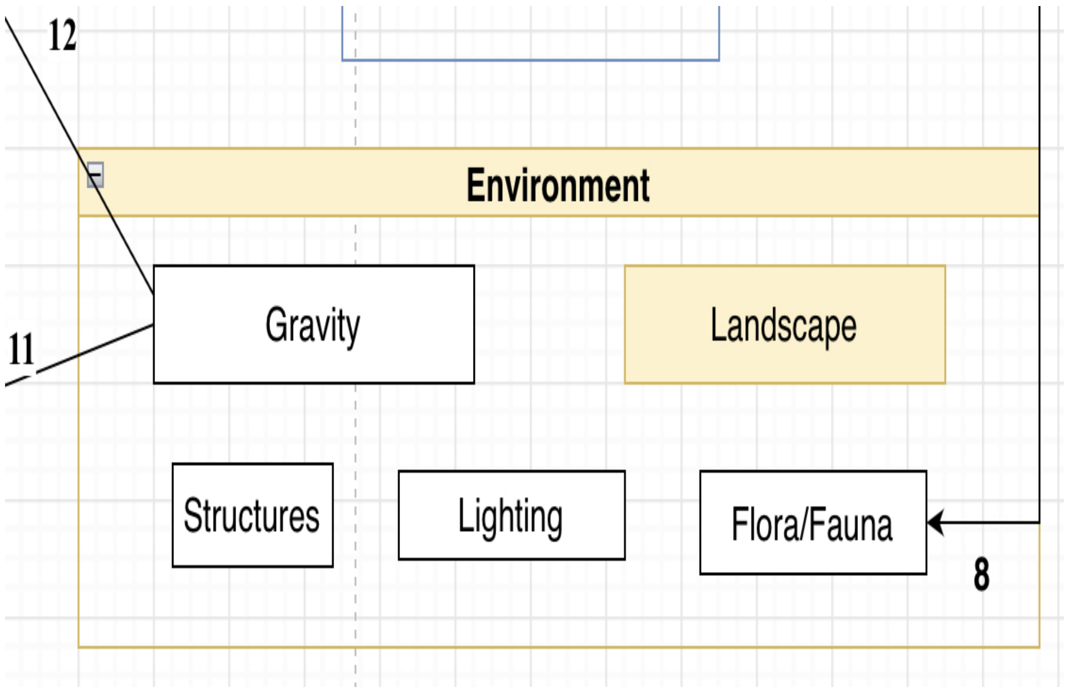

Figure 15: Environment: Landscape Subsystem

8.2.1 Assumptions

None

8.2.2 Responsibilities

Cover placement, animations, and interaction with the landscape.

8.2.3 Subsystem Interfaces

N/A

8.3 Structures

Any buildings, equipment, and things that are not natural or covered by another category are considered structures.

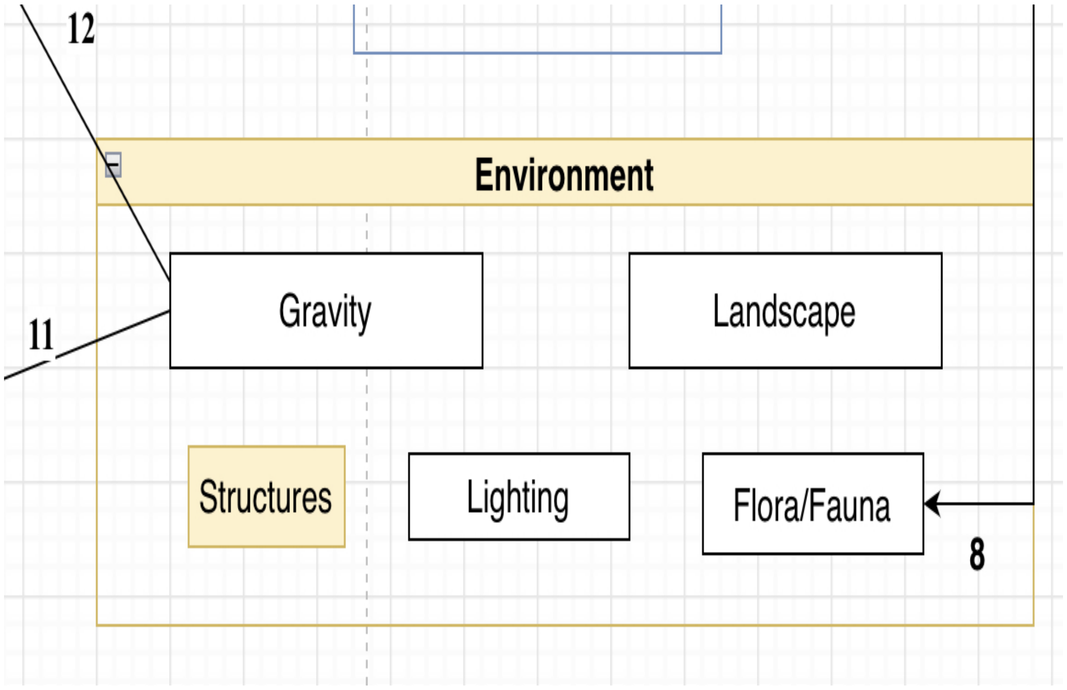

Figure 16: Environment: Structures Subsystem

8.3.1 Assumptions

None

8.3.2 Responsibilities

Control all aspects of structure interaction and placement.

8.3.3 Subsystem Interfaces

N/A

8.4 Lighting

General lighting, shadows, and reflections

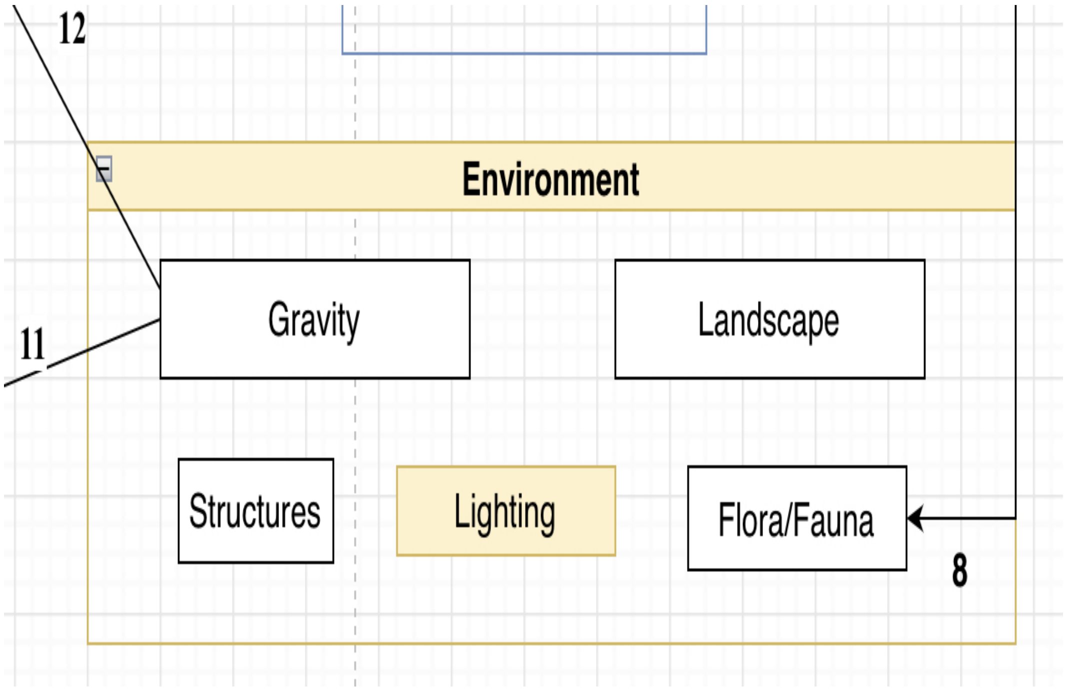

Figure 17: Environment: Lighting Subsystem

8.4.1 Assumptions

Items will be in place that can effect lighting

8.4.2 Responsibilities

Produce lighting and shadows from all environmental factors that can effect lighting

8.4.3 Subsystem Interfaces

N/A

8.5 Flora/Fauna

Flowers, animals, and the like that can be collected for various purposes

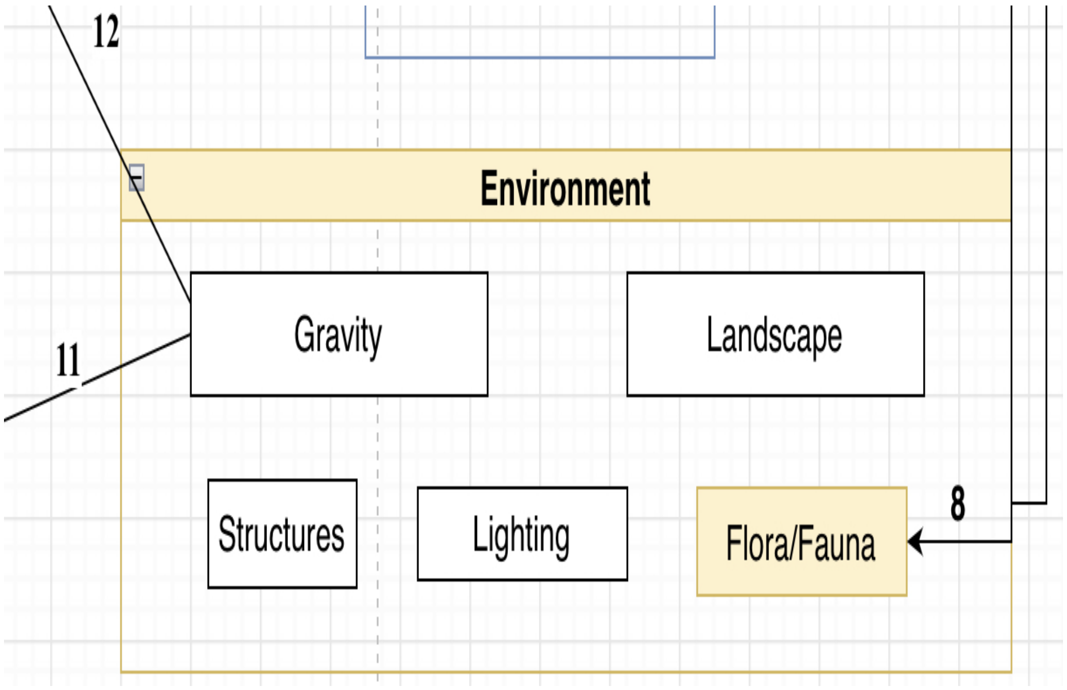

Figure 18: Environment: Flora/Fauna Subsystem

8.5.1 Assumptions

Item Handler allows for adding items to the players inventory

8.5.2 Responsibilities

Flora/Fauna growth, placement, and movement giving the player the ability to collect items from the environment.

8.5.3 Subsystem Interfaces

Table 14: Flora/Fauna Interfaces

ID	Description	Inputs	Outputs
#8	Collection of items	N/A	Item Handler

9 Player Management Station

There are two different types of PMS, an alpha and beta version. Alpha PMS have access to all modules and functionality while the beta has less modules and reduced functionality. These stations allow the player to upgrade their suits and weapons. It also allows them to purchase recipes for crafting. Stash items.

9.1 BasePMS

Defines the basic functionality of all Player Management Systems.

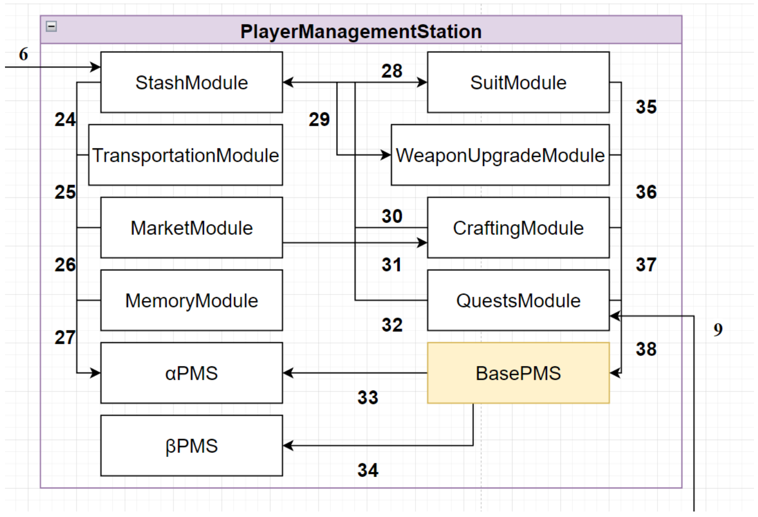

Figure 19: Player Management Station: BasePMS Subsystem

9.1.1 Assumptions

N/A

9.1.2 Responsibilities

Define subsystems which the other Player Management Stations will have access to.

9.1.3 Subsystem Interfaces

Table 15: BasePMS Interfaces

ID	Description	Inputs	Outputs
#35	Suit Module Interface	Functions	N/A
#36	Weapon Upgrade Module Interface	Functions	N/A
#37	Crafting Module Interface	Functions	N/A
#38	Quests Module Interface	Functions	N/A

9.2 AlphaPMS

The major type of PMS that allows you to access all modules.

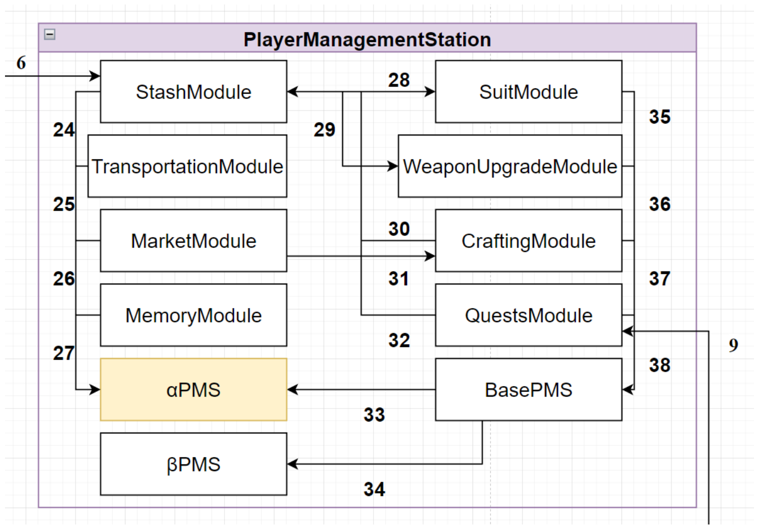

Figure 20: Player Management Station: AlphaPMS Subsystem

9.2.1 Assumptions

N/A

9.2.2 Responsibilities

Heals and respawns all enemies on the map upon opening. Allows access to all modules and their functionalities. Auto-saves the game upon accessing.

9.2.3 Subsystem Interfaces

Table 16: AlphaPMS Interfaces

ID	Description	Inputs	Outputs
#33	BasePMS Interface	Modules	N/A
#24	Stash Module Interface	Functions	N/A
#25	Transportation Module Interface	Functions	N/A
#26	Market Module Interface	Functions	N/A
#27	Memory Module Interface	Functions	N/A

9.3 BetaPMS

Has only access to half the modules.

9.3.1 Assumptions

N/A

9.3.2 Responsibilities

Allows access to half the modules and their functionalities.

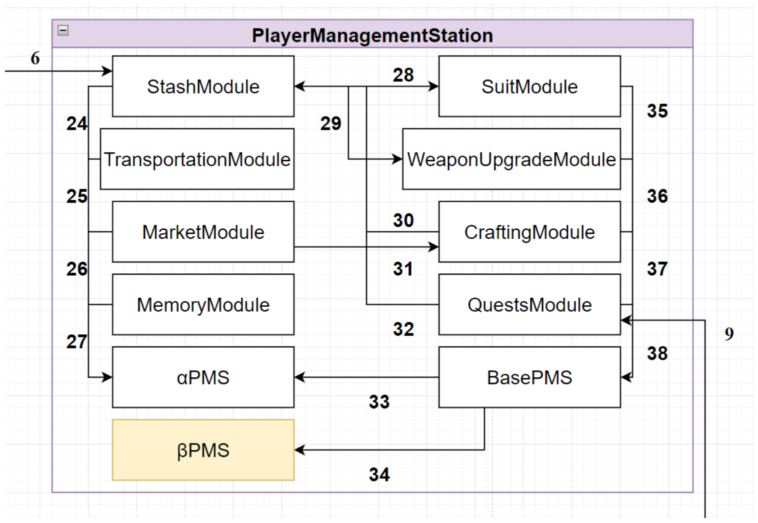

Figure 21: Player Management Station: BetaPMS Subsystem

9.3.3 Subsystem Interfaces

Table 17: BetaPMS Interfaces

ID	Description	Inputs	Outputs
#34	BasePMS Interface	Modules	N/A

9.4 Suit Module

Allows the player to upgrade their suit.

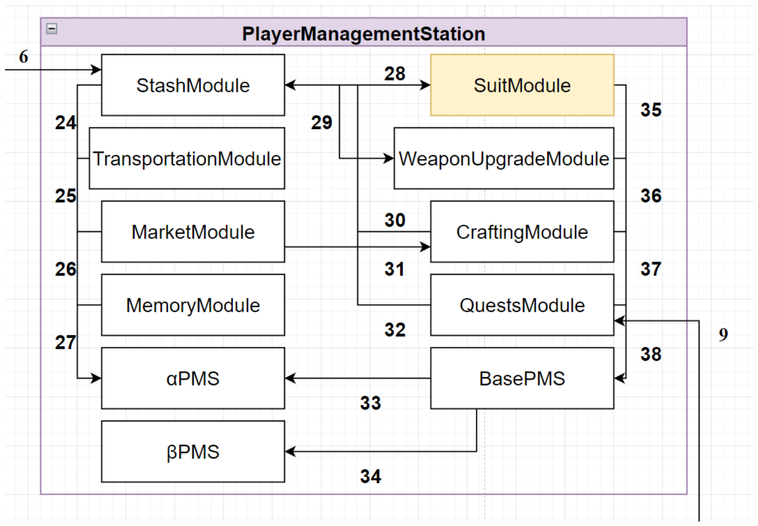

Figure 22: Player Management Station: Suit Module Subsystem

9.4.1 Assumptions

N/A

9.4.2 Responsibilities

Allow the player to upgrade their max health and movement speed.

9.4.3 Subsystem Interfaces

Table 18: Suit Module Interfaces

ID	Description	Inputs	Outputs
#28	Stash Module Interface	Items	N/A
#35	BasePMS Interface	Functions	N/A

9.5 Weapon Upgrade Module

Allows the player to upgrade their weapons.

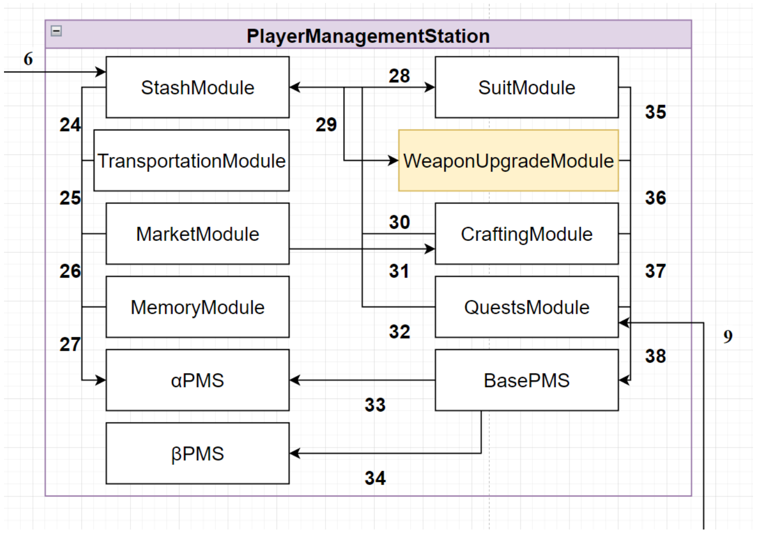

Figure 23: Player Management Station: Weapon Upgrade Module Subsystem

9.5.1 Assumptions

N/A

9.5.2 Responsibilities

Allow the player to upgrade their weapon's fire rate, damage, reload time, and range.

9.5.3 Subsystem Interfaces

Table 19: Weapon Upgrade Module Interfaces

ID	Description	Inputs	Outputs
#29	Stash Module Interface	Items	N/A
#36	BasePMS Interface	Functions	N/A

9.6 Crafting Module

Allows the player to craft items.

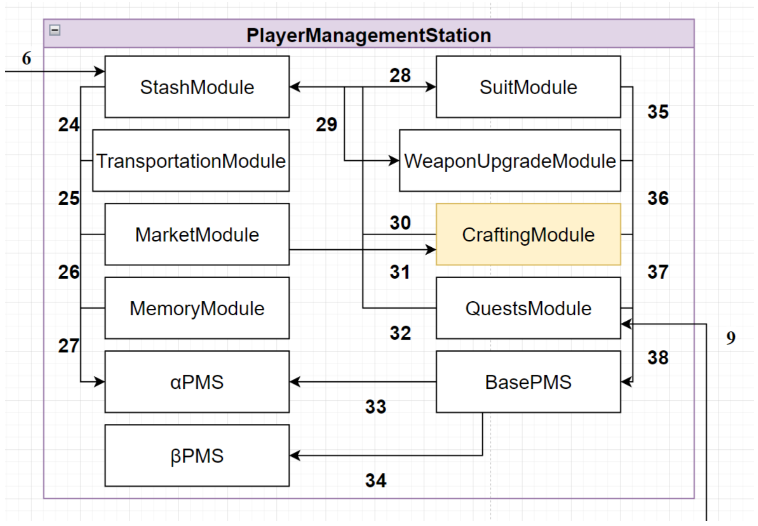

Figure 24: Player Management Station: Crafting Module Subsystem

9.6.1 Assumptions

N/A

9.6.2 Responsibilities

Allows the player to craft items using the recipes purchased from the market module. It can use resources from the player's inventory or stash (if an alpha PMS) with the recipe to produce the product item(s).

9.6.3 Subsystem Interfaces

Table 20: Crafting Module Interfaces

ID	Description	Inputs	Outputs
#30	Stash Module Interface	Items	Items
#31	Market Module Interface	Recipes	N/A
#37	BasePMS Interface	N/A	Functions

9.7 Quests Module

Allows access to quests.

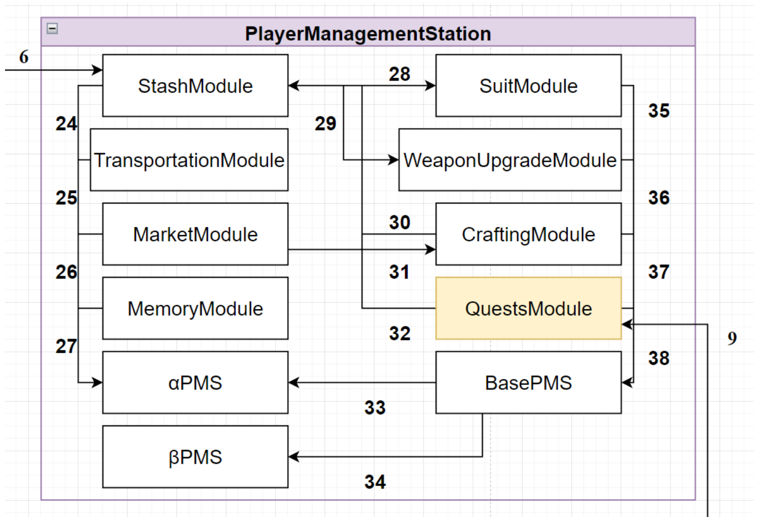

Figure 25: Player Management Station: Quests Module Subsystem

9.7.1 Assumptions

N/A

9.7.2 Responsibilities

Allows the user to accept quests. Allows user to turn in items for a quest. Allows the user to collect quest rewards upon finishing them.

9.7.3 Subsystem Interfaces

Table 21: Quests Module Interfaces

ID	Description	Inputs	Outputs
#32	Stash Module Interface	N/A	Items
#38	BasePMS Interface	N/A	Functions
#9	Quest Controller Interface	Quests	Quests

9.8 Stash Module

Allows the user to stash items inside.

9.8.1 Assumptions

N/A

9.8.2 Responsibilities

Allows the user to move items to and from the stash. Allows the upgrading of the stash size. Allows the player to add a recycler upgrade. This recycler upgrade allows the player to get 50% of the resources that make up the recipe of an item, so it can only recycle non-atomic items.

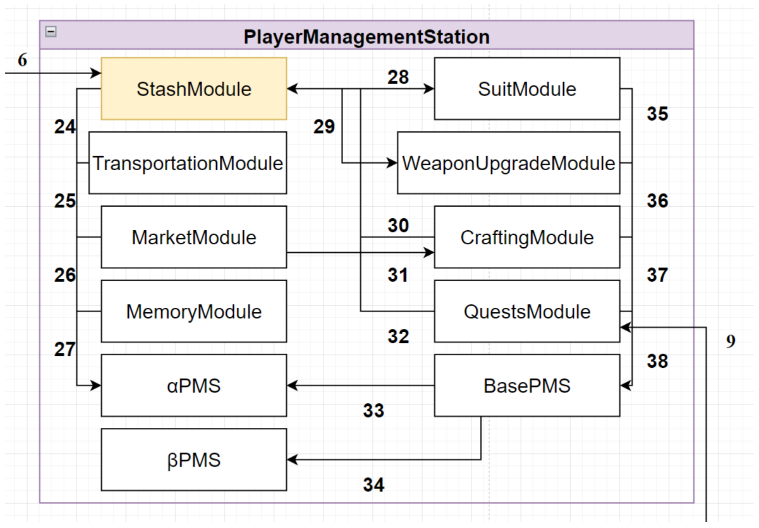

Figure 26: Player Management Station: Stash Module Subsystem

9.8.3 Subsystem Interfaces

Table 22: Stash Module Interfaces

ID	Description	Inputs	Outputs
#6	Inventory Handler Interface	Items	Items
#24	AlphaPMS Interface	N/A	Functions
#28	Suit Module Interface	Items	Items
#29	Weapon Upgrade Module Interface	Items	Items
#30	Crafting Module Interface	Items	Items
#31	Quests Module Interface	Items	N/A

9.9 Transportation Module

Allow the player to transport to other PMS that have a Transportation Module.

9.9.1 Assumptions

N/A

9.9.2 Responsibilities

Allows the player to teleport between discovered PMS that contain Transportation Modules.

9.9.3 Subsystem Interfaces

Table 23: Transportation Module Interfaces

ID	Description	Inputs	Outputs
#25	AlphaPMS Interface	N/A	Functions

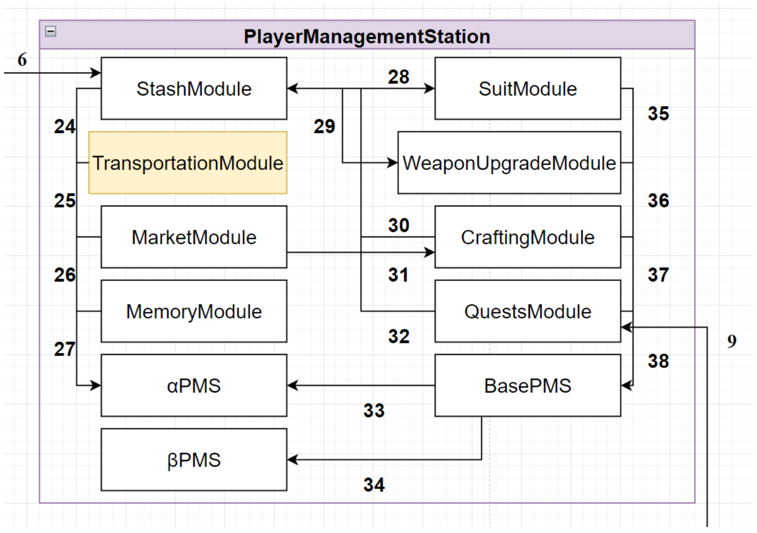

Figure 27: Player Management Station: Transportation Module Subsystem

9.10 Market Module

Allows the player to purchase recipes for crafting.

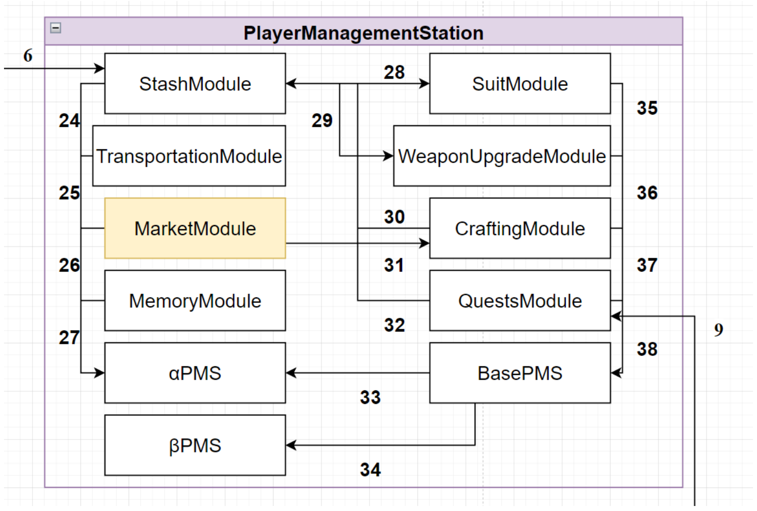

Figure 28: Player Management Station: Market Module Subsystem

9.10.1 Assumptions

N/A

9.10.2 Responsibilities

Allows the player to sell items. Allows the player to purchase single and multi-use recipes using the credits they find or obtain through selling items.

9.10.3 Subsystem Interfaces

Table 24: Market Module Interfaces

ID	Description	Inputs	Outputs
#26	AlphaPMS Interface	N/A	Functions
#31	Crafting Module Interface	N/A	Recipes

9.11 Memory Module

Allows the user to save, load, and exit the game.

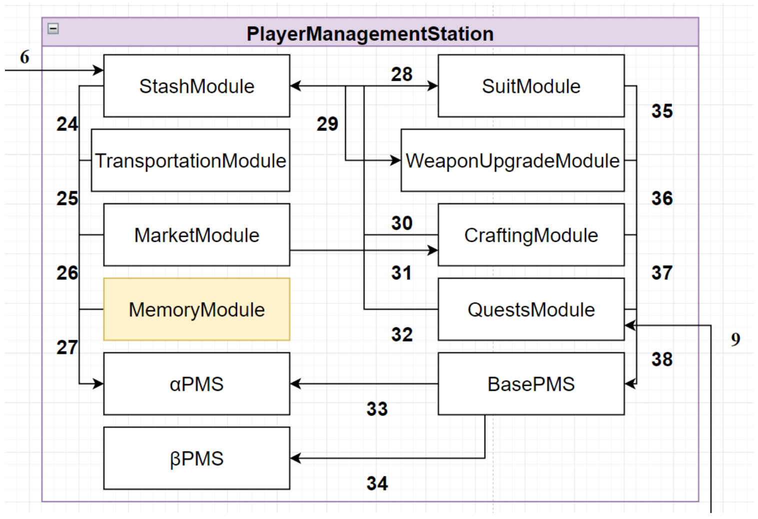

Figure 29: Player Management Station: Memory Module Subsystem

9.11.1 Assumptions

N/A

9.11.2 Responsibilities

Allows the user to save the current state of the game. Allows the user to load an older state of the game. Allows the user to gracefully exit the game.

9.11.3 Subsystem Interfaces

Table 25: Memory Module Interfaces

ID	Description	Inputs	Outputs
#27	AlphaPMS Interface	N/A	Functions

10 UI

The User Interface allows for the interaction of the player and the game's components through a visual medium. The UI primarily informs the player of in-game states such as player status, quest status as well as allowing for inventory interaction.

10.1 UI Controller

The UI controller manages the other UI components: Quest, Player and Inventory.

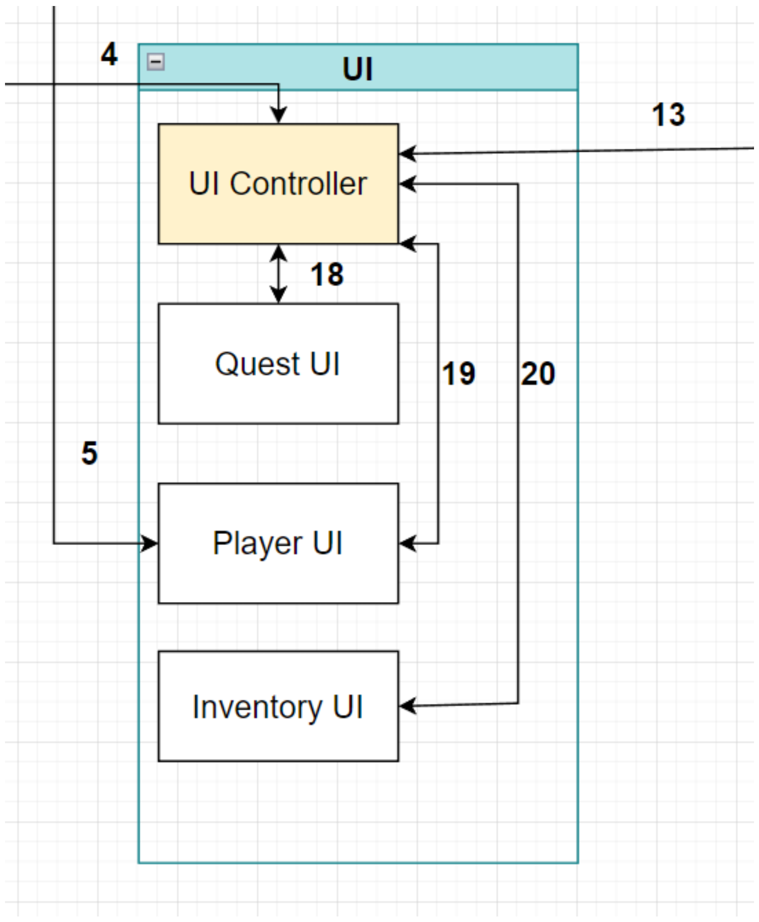

Figure 30: UI Controller

10.1.1 Assumptions

Proper data being received from the different controllers within system. This includes the Player and Quest Controllers in addition to input from the UI subsystem.

10.1.2 Responsibilities

The UI controller will receive data from the Player Controller and Quest Controller systems. Proper interaction for later display within the subsystems will be the primary responsibility of the UI controller. In addition to interaction with the Player and Quest Controllers, the UI controller will receive data from the UI subsystems to follow, including the Quest UI, Player UI and Inventory UI.

10.1.3 Subsystem Interfaces

Table 26: UI Controller Interfaces

ID	Description	Inputs	Outputs
#4	Player Controller data via Input System	InputSystem	N/A
#13	Quest data via Quest Controller	Quest Controller	N/A
#18	Quest data via Quest UI	Quest UI	Quest UI
#19	Player UI Management	Player UI	Player UI
#20	Inventory Management data	Inventory UI	Inventory UI

10.2 Quest UI

The Quest UI is designed to allow the player to interact with and keep track of components related to the current task/quest.

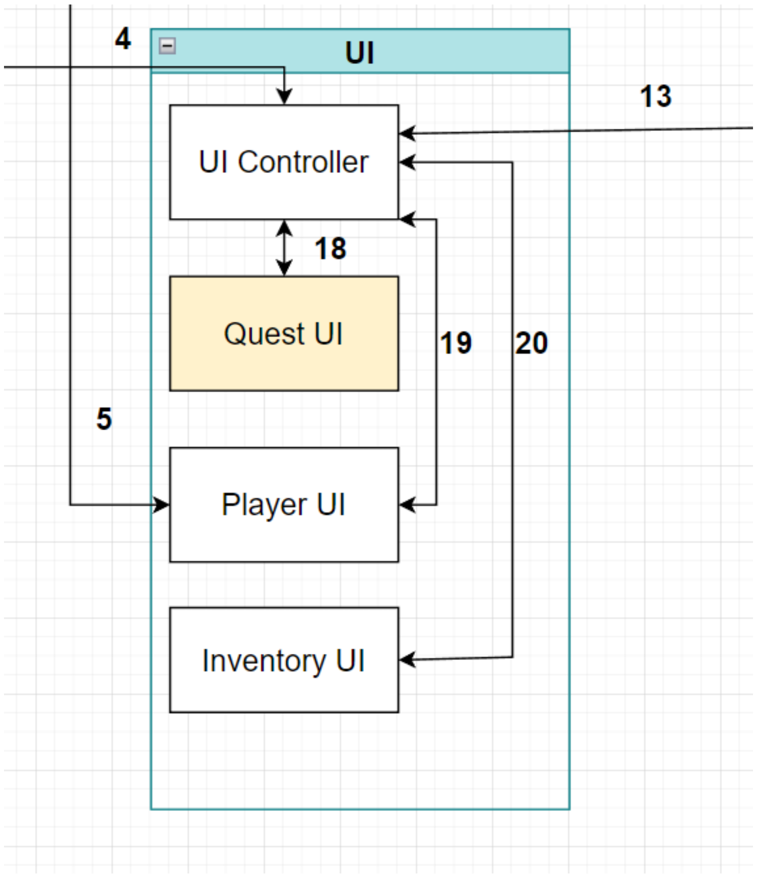

Figure 31: Quest UI

10.2.1 Assumptions

Proper synergy and input from the game's Quest and Inventory systems.

10.2.2 Responsibilities

Maintaining and editing of the quest's primary components. This includes allowing the player to keep track of quest statuses such as milestones and rewards in addition to interacting with the status of the overall quest itself, such as deleting and accepting quests. In addition to viewing current objectives, the Quest UI system needs to inform the player of the completion of each goal of the quest, as well as signaling the completion of any completed quests.

10.2.3 Subsystem Interfaces

Table 27: Quest UI Interfaces

ID	Description	Inputs	Outputs
#18	Quest Data via UI Controller	UI Controller	UI Controller

10.3 Player UI

The Player UI is designed with the purpose of displaying information related to the character's status as it relates to game mechanics. Primarily, the status of the characters health.

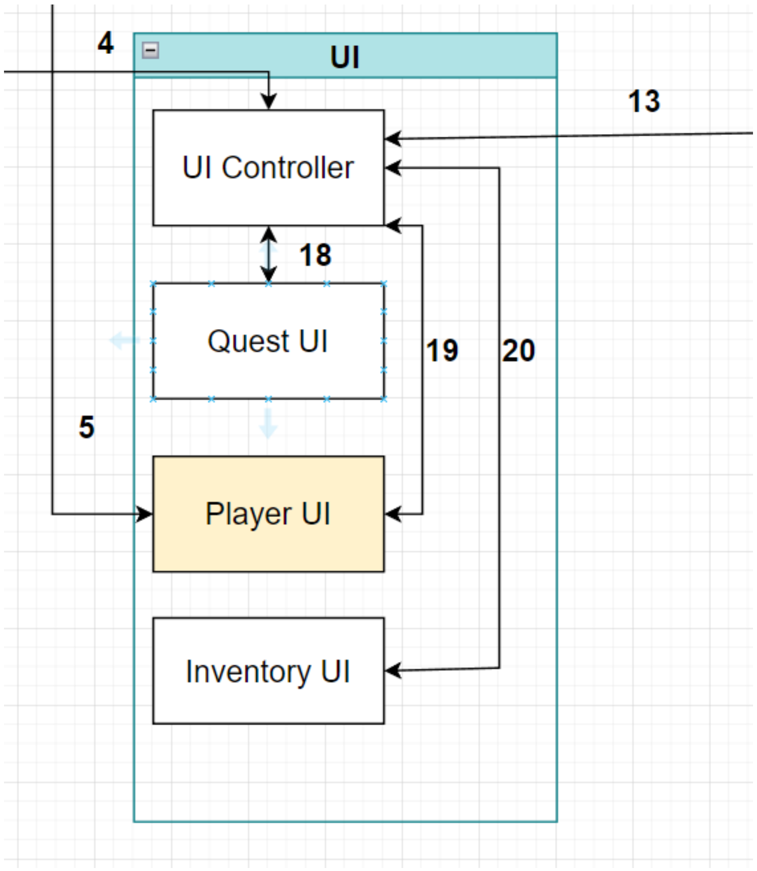

Figure 32: Player UI

10.3.1 Assumptions

Proper functionality of the Health System as it exists within the Combat System.

10.3.2 Responsibilities

Proper interaction with the combat system will be necessary for the ability to accurately display the player's health. This includes real-time updates to the health status as the player takes damage in combat (or falling) as well as recuperating health during healing.

10.3.3 Subsystem Interfaces

Table 28: Player UI Interfaces

ID	Description	Inputs	Outputs
#5	Player Health data	Health System	N/A
#19	UI Controller Management	UI Controller	UI Controller

10.4 Inventory UI

The Inventory UI purpose is to allow for the player to visually interact with items collected within the inventory. Expected items include, but are not limited to, weapons, quest items and consumables. This system includes the display of items, inventory slot management, and inventory item interactions.

10.4.1 Assumptions

The ability to receive inventory data from the Inventory System.

10.4.2 Responsibilities

Primarily, the responsibility of the Inventory System is to display input received from the Inventory Handler within the Inventory System.

10.4.3 Subsystem Interfaces

Table 29: Inventory UI Interfaces

ID	Description	Inputs	Outputs
#20	Inventory data via UI Controller	UI Controller	UI Controller

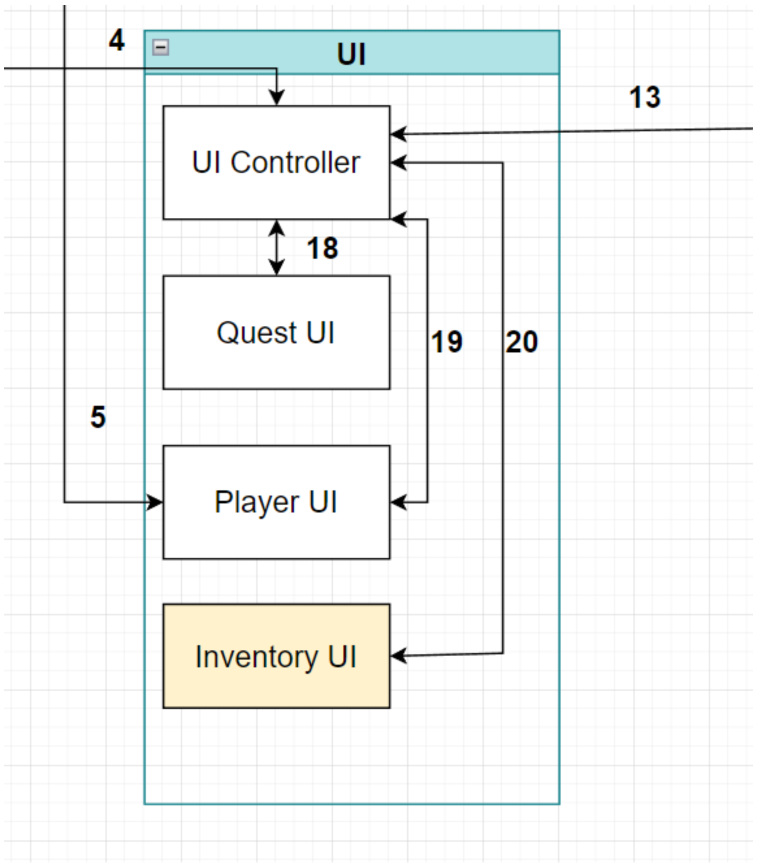

Figure 33: Inventory UI

REFERENCES