****

**COLD NITES**

Alpha-2

**Camera Manager Design Document**

Designed & Implemented by

Yash Chamria

**Change Log**

**Version** – 0.2

**Modifier** – Yash Chamria

**Date** – 28th March, 2021

**Description** – Created the Design Document Body.

**Version** – 0.4

**Modifier** – Yash Chamria

**Date** – 30th March, 2021

**Description** – Completed Introduction and Design Goals.

**Version** – 0.6

**Modifier** – Yash Chamria

**Date** – 5th April, 2021

**Description** – Completed System Overview and Edited High level.

**Version** – 0.9

**Modifier** – Yash Chamria

**Date** – 6th April, 2021

**Description** – Completed Sequence Diagram and Use Cases.

**Version** – 1.6

**Modifier** – Yash Chamria

**Date** – 12th April, 2021

**Description** – Added cleaned up Detailed UML and Mid-Level Diagram.

**Table of Contents**

1. **Introduction 4**
2. **Design Goals 5**
3. **System Overview and Behaviour 6**

**A. Camera Manager**

**B. Base Camera Actor**

1. **Logical View 7**

**A. High-Level Design Architecture of the Entire System**

**B. Mid-Level Design of Camera Manager**

**C. Detailed Design of Camera Manager**

1. **Process View 11**

**A.** **Creating and Storing Camera in the Level**

**B. Removing All Camera Actors**

**C. Detailed Design of Camera Manager**

1. **Use Case View and Practice 14**
2. **In Code**
3. **In Editor**

**1. Introduction**

Cold Nites is a Grid-style turn-based game. The player must strategically navigate through the level to survive the cold night, protecting the boy from all the mischievous elements of the city. And, there are always multiple ways to solve the puzzles along the way.

This design module will focus on the construction(architecture) and implementation of the Camera Manager. Camera Manager allows the designer to spawn any number of cameras and seamlessly blend between them throughout the level.

This document will describe the architecture and design choices that make the Camera Manager implementation easy to use, understand and reusable for all the fellow programmers, level designers, and major stakeholders.

Below are interest points for the mentioned parties:

**Programmers** – Camera Manager along with Event Manager, will handle all the camera events.

**Level Designer** - One of the main targets of the Camera Manager is to achieve ease of level designing. It will provide the nifty functionality of adding any number of cameras in the scene with just a click. And just adding the name of the camera in the tile is enough and the Camera Manager will manage the blend between them.

**Project Manager (and the Team)** - All the tasks during the group meetings were assigned with everyone's and the Project Manager's agreement. This, Camera Manager design module and the code implementation will address all the concerns and will fulfill all the requirements in the game's and team's best interest.

**2. Design Goals**

The design priorities for the Camera Manager are mentioned below:

* The design should minimize the complexity of setting up the cameras in the level.
* The design will allow easy handling of the camera switching and blending throughout the level.
* The design should allow the level designer to experiment on the fly and allow easy level creation.

**3. System Overview and Behaviour**

The Camera Manager is responsible for spawning the 'Base Camera' actors and switching between the desired camera with a blend.

**Base Camera Actor -**

The Base Camera Actor is a basic camera that has a spring arm attached to it. It comes with some of the camera properties and values predefined in the code which allows a consistent look throughout the game and ease of camera creation.

**Camera Manager Actor -**

Camera Manager is the one who spawns and handles all the switching and blending logic for the cameras.

Camera Manager stores an array of all the camera actors spawned in the world. It handles the logic of switching between any of the given cameras using the passed Camera Tag. Event Manager is the one who checks the Tile, the player is on and decides which camera should draw that Tile and let the Camera Manager know.

**4. Logical View**

The logical view describes the high-level architecture for the entire game from all the core classes to high-level relations and interactions between them with a flow chart making it easy to read and understand.

Later, it will dive deep into the high-level and detailed design for the Camera Manager Module, using a UML Class Diagram.

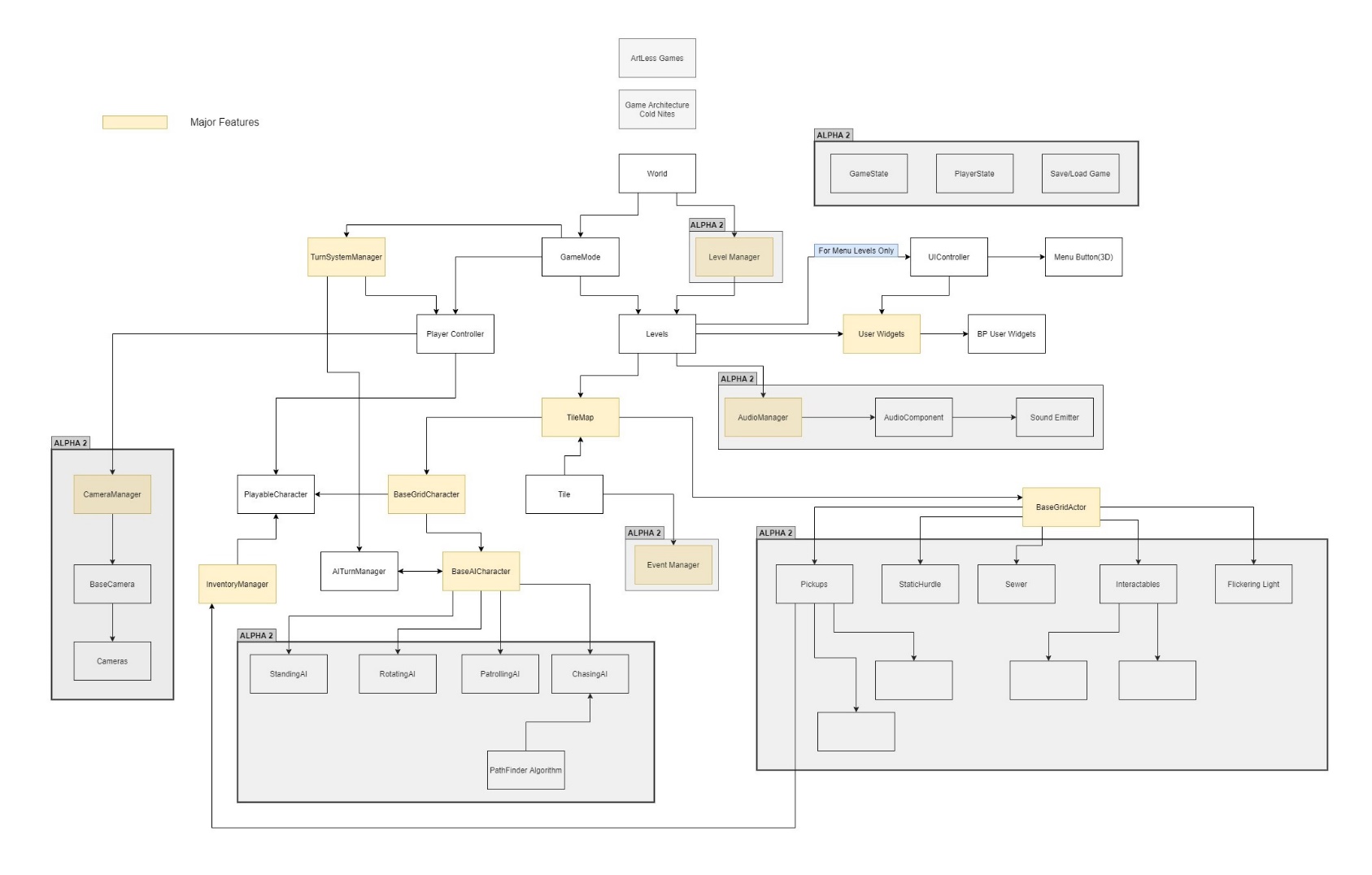
**A. High-Level Design Architecture of the Entire System**

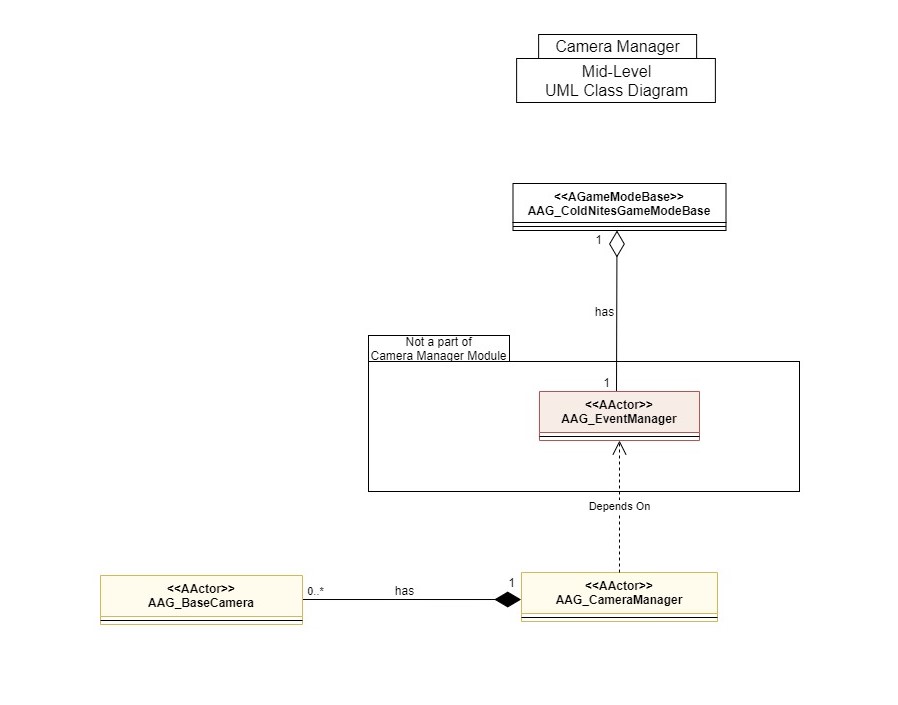
The primary features for the Alpha 1 release:

1. **TileMap** - TileMap provides the grid-based behaviour for the game and will facilitate the event system, based on the actor present on the Tiles.
2. **Turn-Based System** - This provides the turn-based aspect for the game. It is responsible for maintaining the turn order for all the world elements(actors) and the player.
3. **Base Grid Classes** - These classes work as a foundation class for all the actors/characters spawned in the game. These classes are closely integrated with handling the TileMap(Grid-Base) Behaviour of the game.
4. **Player Character** - Player is a controllable character that inherits from BaseGridCharacter, which takes user inputs to perform appropriate moves.
5. **Inventory System** - The pickup function helps the player grab the items on the map. Inventory stores the items for the corresponding actor and will allow the player easy access to any collectible throughout the game, and it also assists in equipping the stored items.
6. **Menu Interface** - The Menu Interface will be responsible for Main Menu and Pause Menu with which the player can interact.

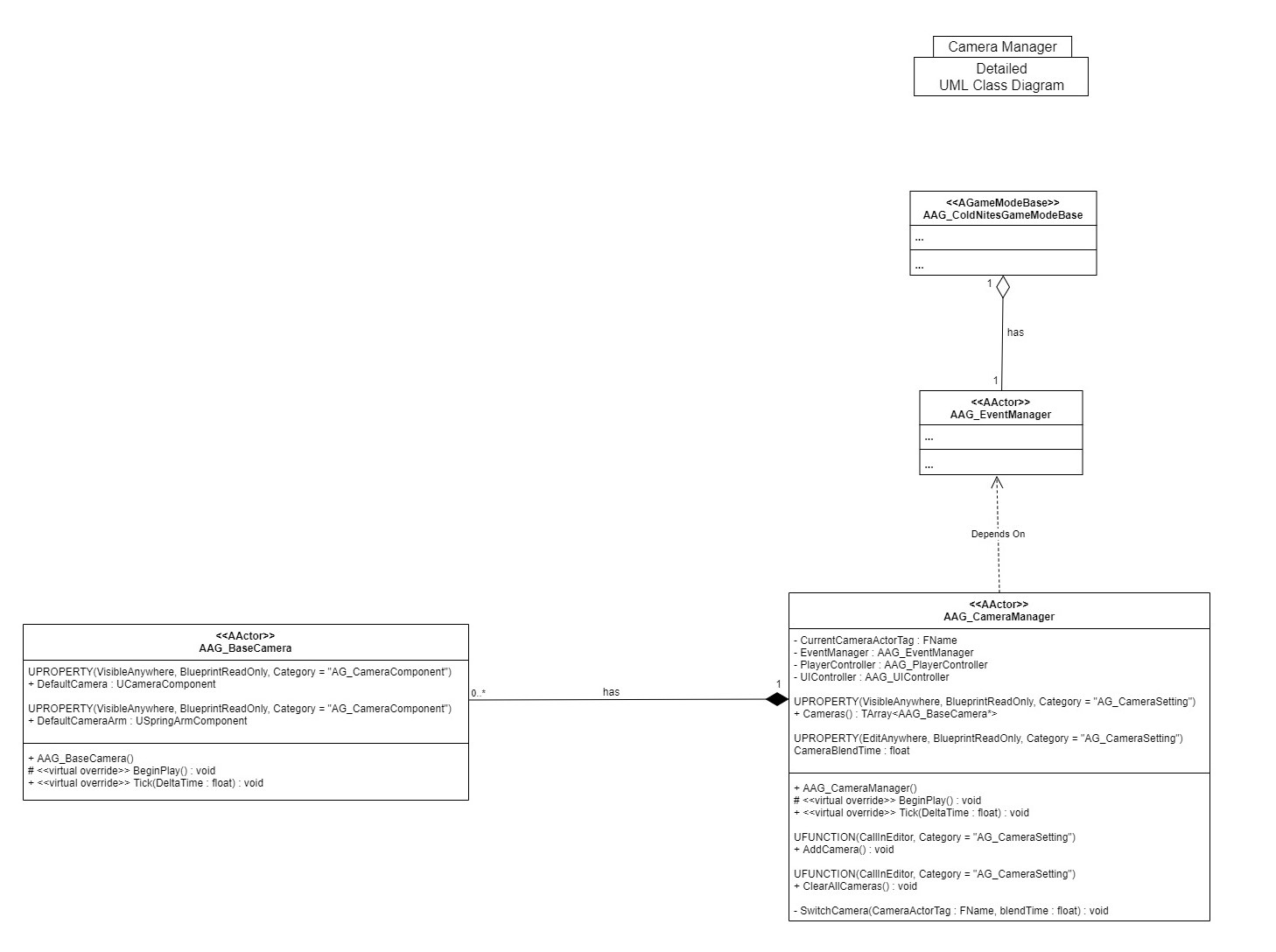
The primary features for the Alpha 2 release:

1. **AI System** - AI System is responsible for various enemy types in the game. The AI System allows the ease of creation using the single BaseAI class and Behaviour Tree Task Nodes.
2. **Event System** –
3. **Camera Manager** - Camera Manager is responsible for handling the game view. It provides the functionality of spawning the camera and handles the switching between the desired cameras.
4. **Level Management System -**



**B. Mid-Level Design of Camera Manager**

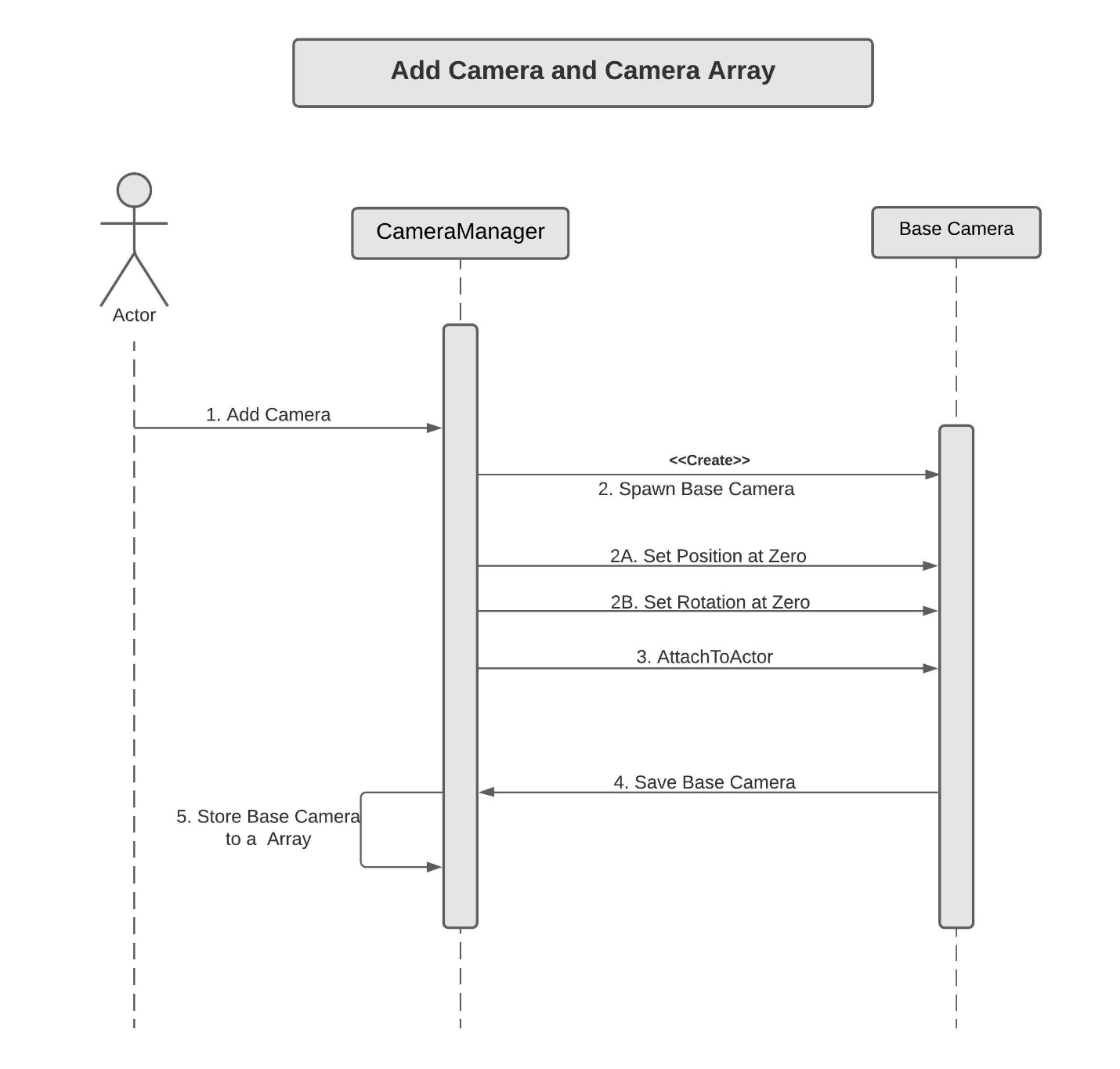
**C. Detailed Design of Camera Manager**

****

**5. Process View**

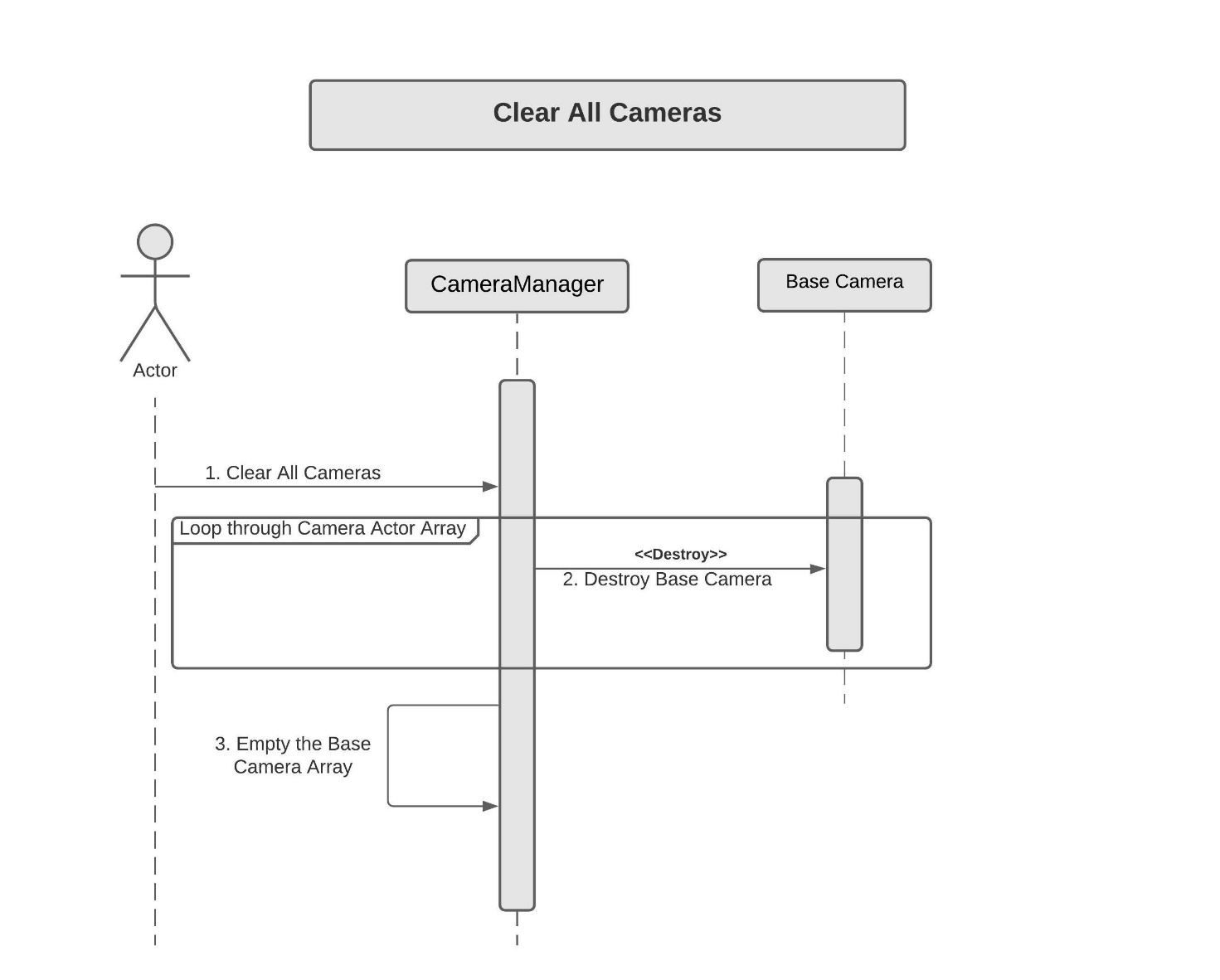
The process view will explain the relation and interaction between various cases using Sequence and Collaboration Diagrams.

**A. Creating and Storing Camera in the Level:**

Pressing 'Add Camera' from the Camera Manager details panel simply calls the AddCamera() function. This function calls the spawn actor function passing BaseCamera actor, ZeroVector for location and ZeroRotator for rotation. It then attaches the spawned camera to the Camera Manager and also stores them locally in an array.

**B. Removing All Camera Actors:**

Similarly, Pressing 'ClearAllCamera' calls ClearAllCamera() Function. This function just loops through the locally store Base Camera Actor and calls destroy on the actor. Once the loop is over, it empties the local array.



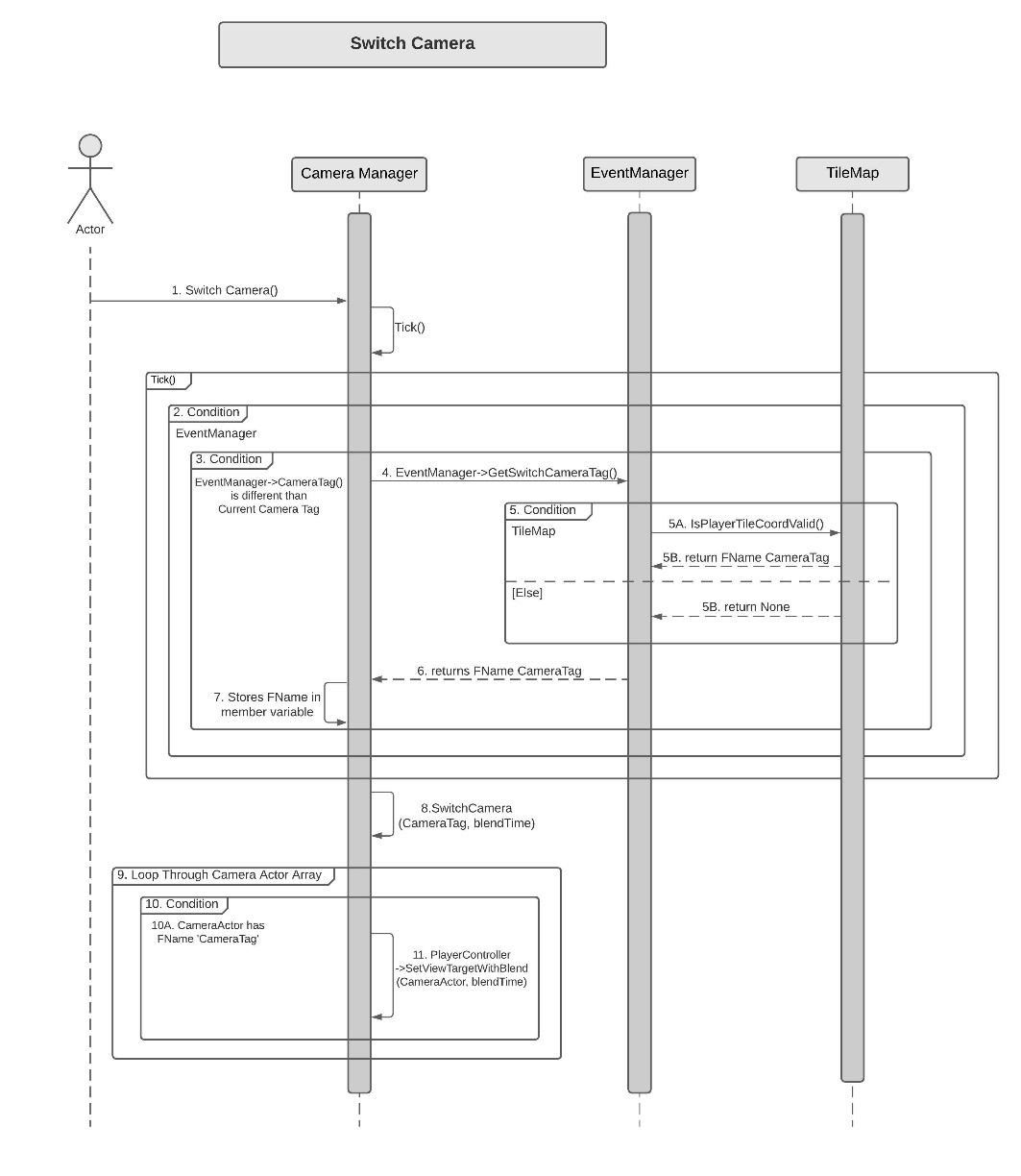
**C. Camera Transition/Switch**

Note - For a better understanding of the setup and camera switching,

refer to \*6B. Use Case View-> In Editor section of the document first. \*

For Camera Switching, the Camera Manager works in sync with the Event Manager. The Event Manager is constantly up to date with the current Player Tile Coordinates. And using the player coordinate, the Event Manager asks the TileMap to get the desired CameraActorTag for that certain Tile. It returns the desired CameraActorTag for that Player Tile.

CameraManager Tick() checks if the CurrentCameraActorTag is different than the DesiredCameraActorTag. If true, it runs a for loop in the SwitchCamera() function and switches the Camera actor with that CameraActorTag.



**6. Use Case View and Practice (Application)**

The use case will focus on showcasing the use of Camera Manager in the level building for the game and will explain its application so that the Camera Manager module can act as a guide/reference for someone not quite familiar with the CameraManager codebase.

**A. In Code:**

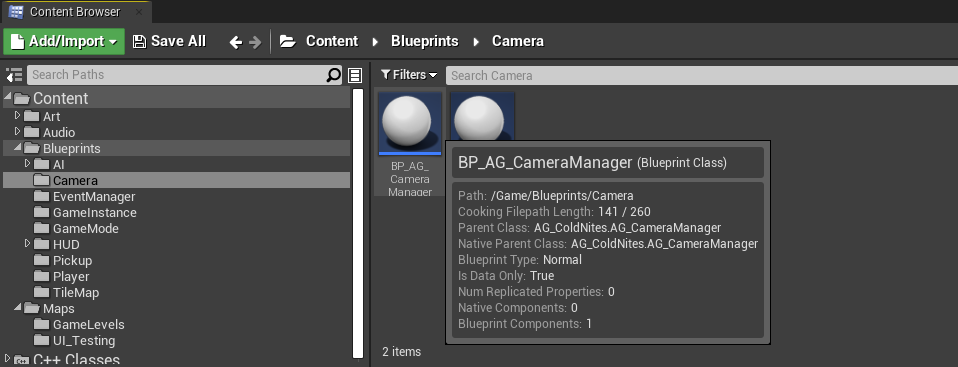
Camera Manager spawns and stores all the Base Camera Actors. Camera Manager works in sync with the Event Manager for the Camera Switching event. And EventManager checks with the TileMap that on which Tile, the player is currently standing. All these things are already hooked up in the code.

**B. In Editor:**

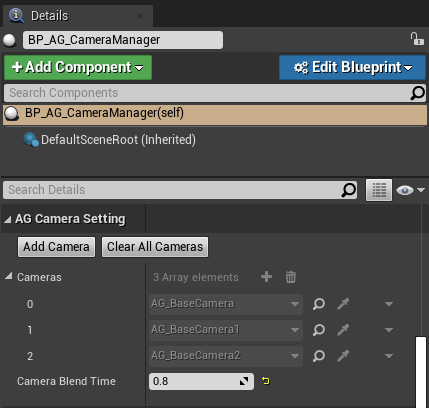
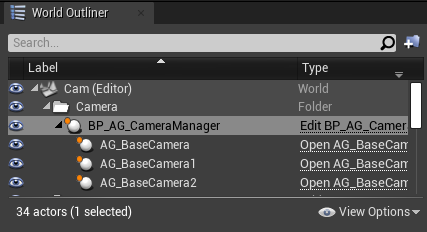
Camera Manager setup for level creation –

**Creating and Adding Camera to the level**

To create the Camera Manager, just drag the Camera Manager blueprint from the content browser.



From the details panel in Camera Manager, you can add any number of cameras to the level.

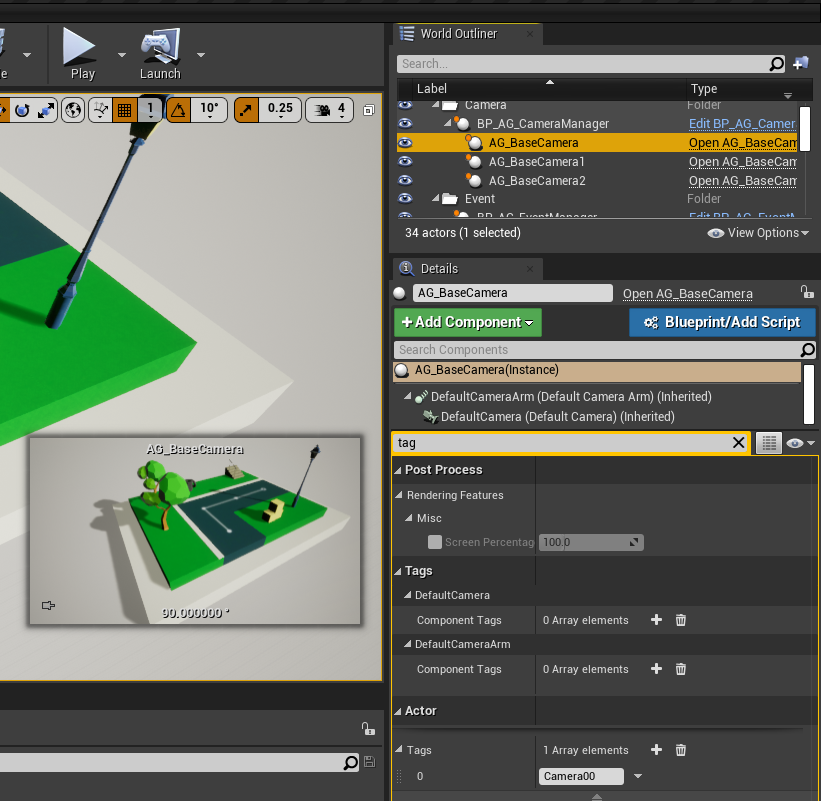


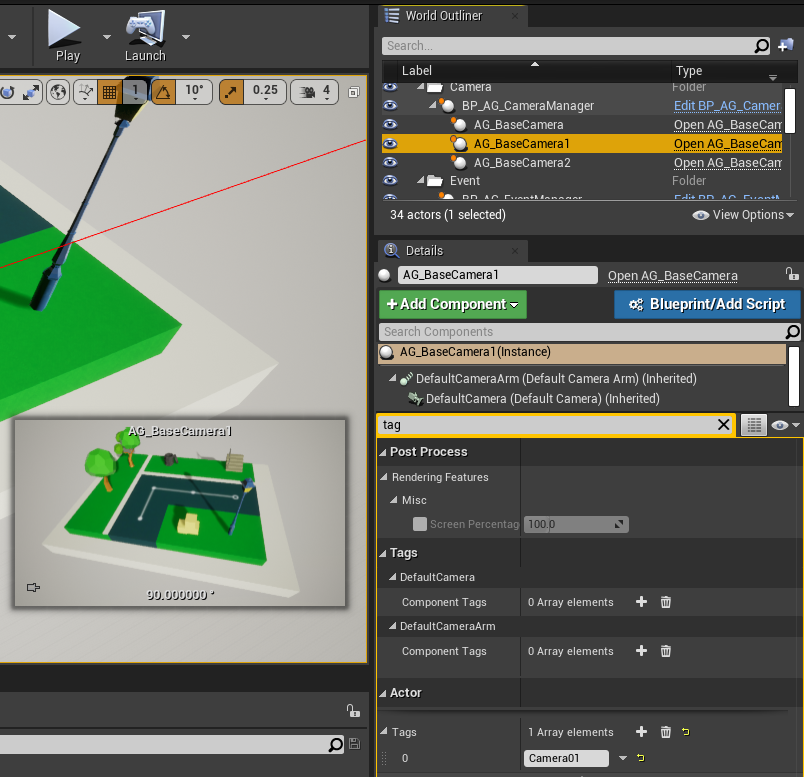
After that, select the individual camera and set its position, rotation or arm length to the desired value.

**Hooking up the cameras for the Transition/Switching**

Now select the camera and add a tag to them.

For the naming convention, I suggest select the first camera and put the tag 'Camera00', for the second 'Camera01', for the third 'Camera02' and go on.





Now to make everything work, select the tiles on which the player can walk. And fill the 'CameraActorTag' field in the Tile. This will result in the use of the camera with that specified tag when the player walks on it.

For Instance, when the player is on the start tile and I want to show that with the second camera. Just put 'Camera01' in the 'CameraActorTag' field for the mentioned tile. And now if I want to switch to Camera five when the player arrive on the next tile, we can set the tag to use 'Camera05' or your given name.

We can also specify the blend time.

Note - Make sure that the desired camera and the tile share the same tag.

Also, you can select multiple tiles at once and add a tag if all the tile shares the same tag.

