****

**COLD NITES**

Alpha-2

**Event Manager Design Document**

Designed & Implemented by

Yash Chamria

**Change Log**

**Version** – 0.1

**Modifier** – Yash Chamria

**Date** – 15th April, 2021

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

**Version** – 0.4

**Modifier** – Yash Chamria

**Date** – 16th April, 2021

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

**Version** – 0.9

**Modifier** – Yash Chamria

**Date** – 16th April, 2021

**Description** – Completed System Overview, and Use Cases.

**Version** – 1.4

**Modifier** – Yash Chamria

**Date** – 17th April, 2021

**Description** – Added Sequence Diagrams and cleaned UML Diagram.

**Table of Contents**

1. **Introduction 4**
2. **Design Goals 5**
3. **System Overview and Behaviour 6**
4. **Logical View 7**

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

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

**C. Detailed Design of Event Manager**

1. **Process View 12**

**A. Set Decor Mesh Property**

**B. Set Player Intial Tile**

**C. Switch Turn State Event**

**D. Check Is AI Turn Over**

**E. Change Turn State**

**F. Set Player Turn Event**

**G. Set AI Turn Event**

**H. Camera Switch Event Update**

**I. Scoring Event On Level Completion**

**J. Level Won Event**

**K. Level Lose Event**

1. **Use Case View and Practice 24**

**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 Event Manager. Event Manager is responsible for handling various events throughout the level. It will handle Turn System Events, Camera Switch Events, Level Win/Lose Events, Scoring Events with other small events.

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

Below are interest points for the mentioned parties:

**Programmers** – Event Manager is a good place to write any ‘Level Specific’ code that will persist throughout the level. Also, it stores all the important actors of the world and centralizes all the code that doesn't belong to any particular class.

**Project Manager (and the Team)** - All the tasks during the group meetings were assigned with everyone's and the Project Manager's agreement. This, Event 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 Event Manager are mentioned below:

* The design should minimize the complexity of the current codebase.
* The design will allow easy creation for the programmers to define new event functions for the level.
* The design should provide a safe place for any level-specific logic and making the code easier to debug.

**3. System Overview and Behaviour**

The Event Manager handles and centralizes level-specific events such as Turn System Events, Camera Switch Events, Level Win/Lose Events, Scoring Events with other small events using the functions and working along with other actors.

Note - The Event Manager in the Cold Nites is a single large file. Due to the later addition of the class and to control the scope of the game in the given time frame, Event Manager only uses internal functions for the events instead of working with the base class of 'Event' and tightly integrating it throughout the code.

The Event Manager will hold all the important actors/character in the level. It will handle various functions throughout along with other classes.

For Instance,

For the Turn system Events - it will check with all the AI Actors and the player and will set the TurnState Enum accordingly in the GameMode.

For Camera Events, it will work with TileMap and player, and it will send the switch event according to the Camera Manager.

For Win/Lose Events - it will handle the enabling and disabling input for the player and AI. It also draws the Win/Lose Widget for the respective situation.

For Scoring System - it keeps the record for all the stars in the level and lets Game Instance know about all the collected stars throughout the level.

It also handles serval small situations and events.

**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 Event Manager Module, using a UML Class Diagram.

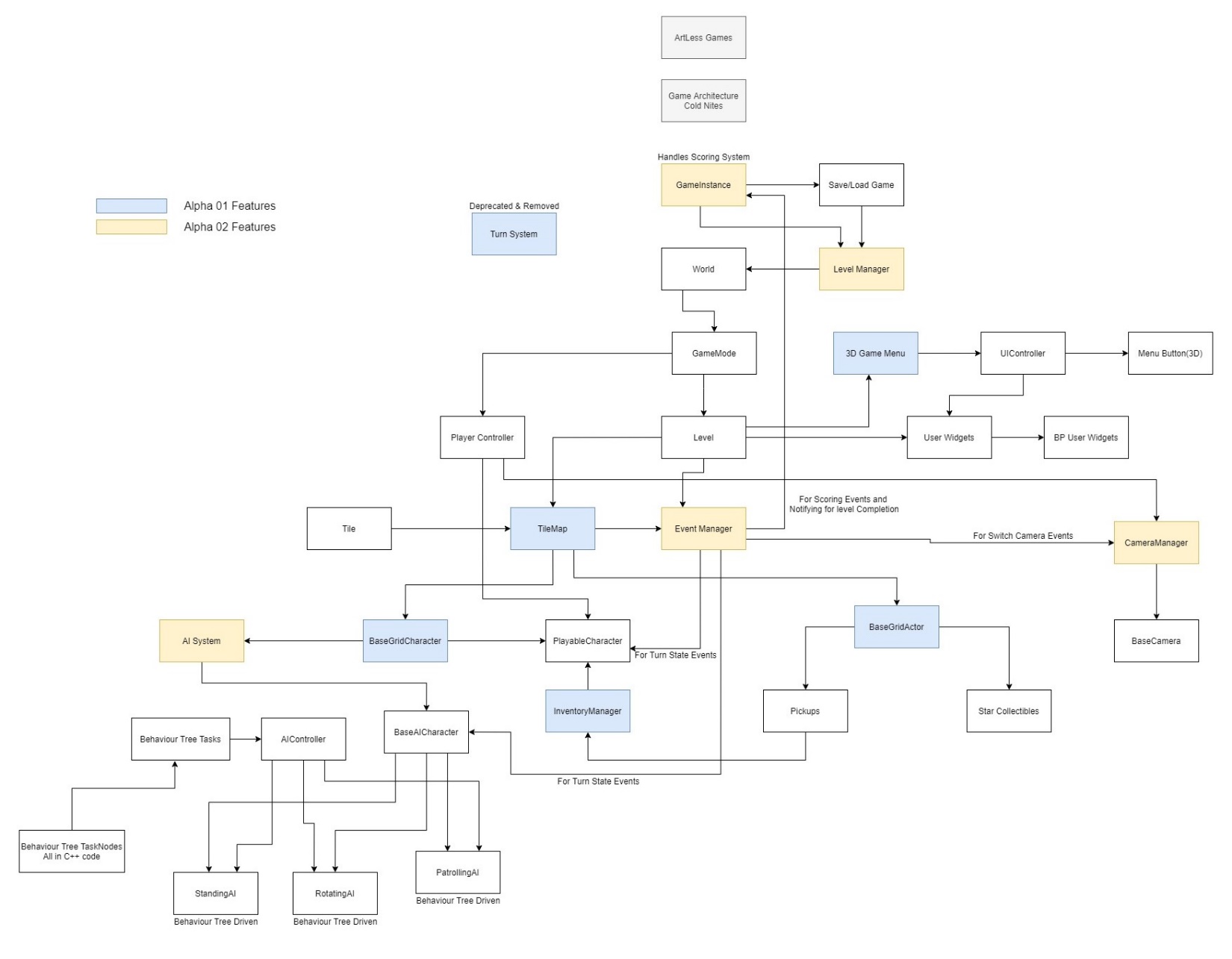
**A. High-Level Design Architecture of the Entire System**

The primary features for the **Alpha 1** release:

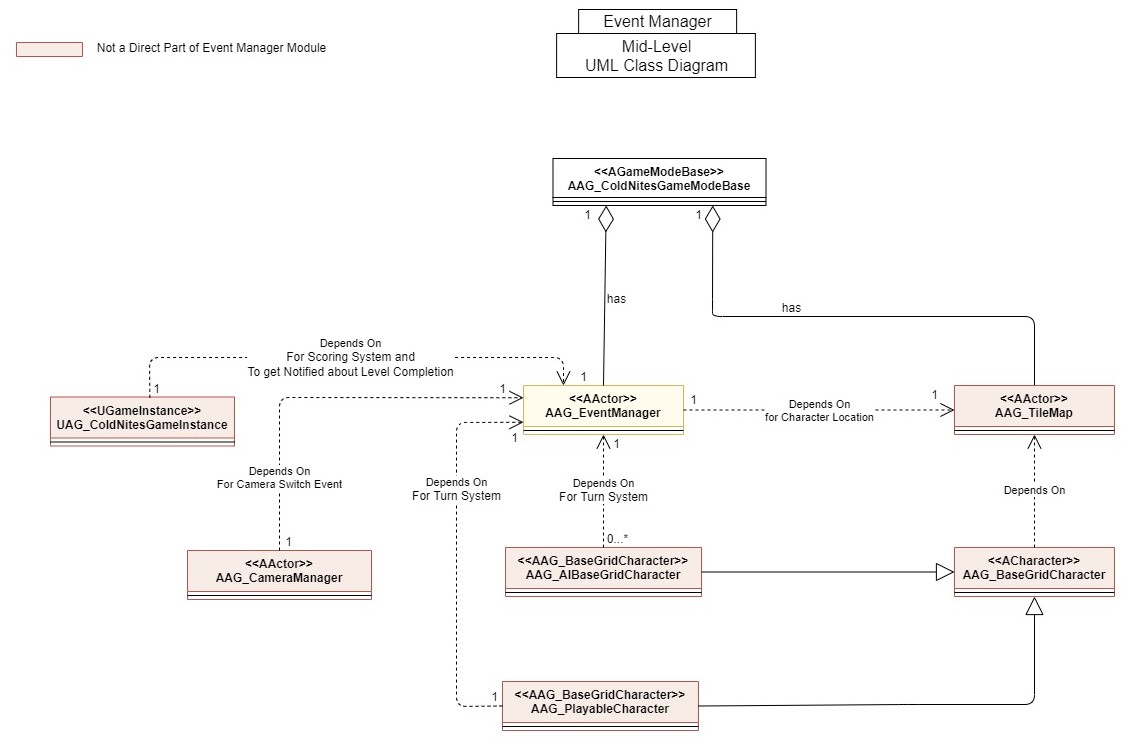
1. **TileMap** - TileMap provides the grid-based behaviour for the game and will facilitate the event system, based on the actor present on the Tiles.
2. **Turn-Based System** - This provides the turn-based aspect for the game. It is responsible for maintaining the turn order for all the world elements(actors) 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:

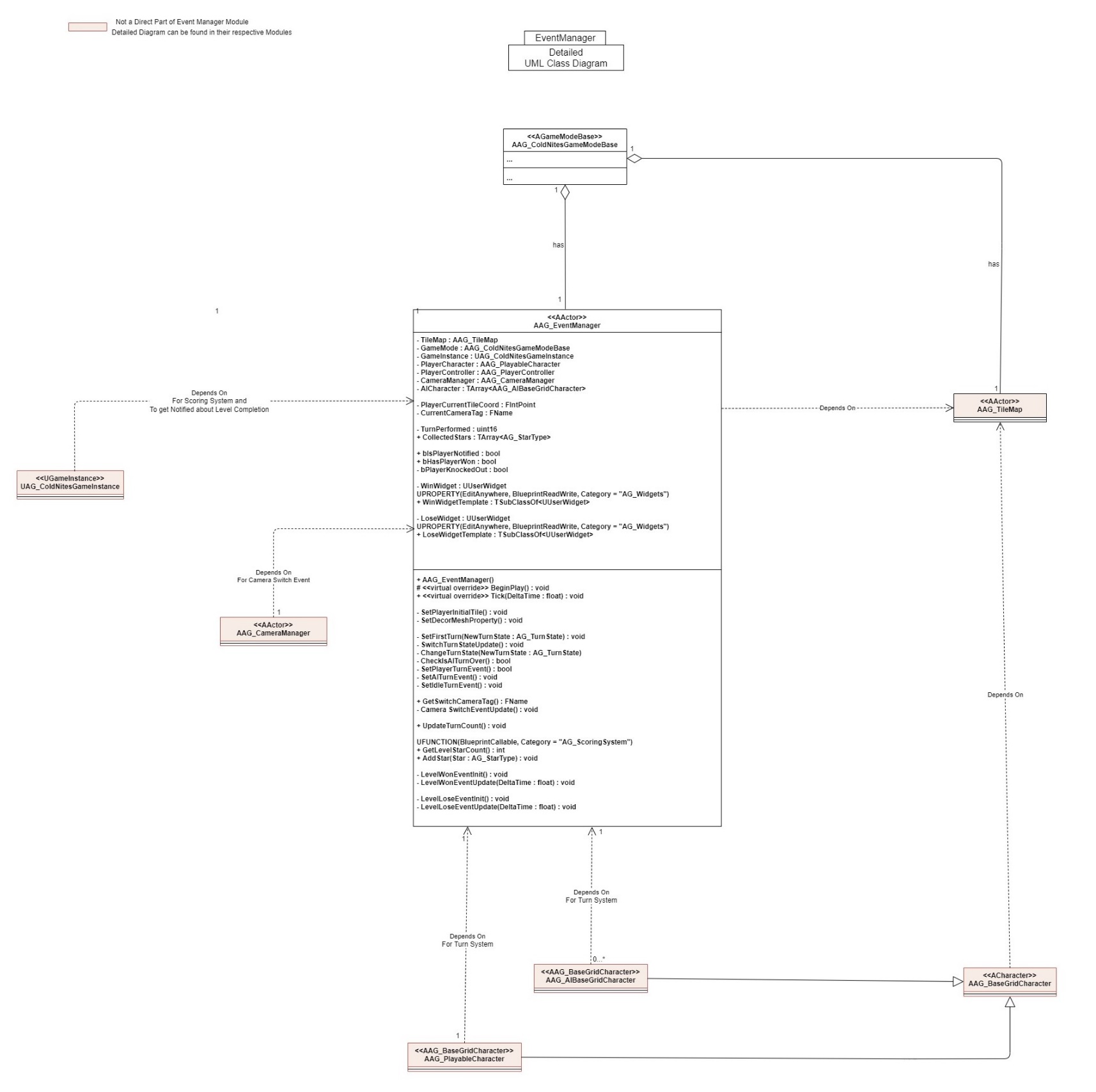
1. **AI System** - AI System is responsible for various enemy types in the game. The AI System allows the ease of creation using the single BaseAI class and Behaviour Tree Task Nodes.
2. **Event Manager** – Event Manager handles and centralizes level-specific events such as Turn System Events, Camera Switch Events, Level Win/Lose Events, Scoring Events with other small events using the functions and working along with other actors.
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 –** The Level Management System will be responsible for switching levels in the game and also saves and loads the level completed progress.
5. **Scoring System -** The Scoring System is responsible for keeping track of number and type of stars earned by the player throughout the game across the levels.



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



**C. Detailed Design of Event Manager**

****

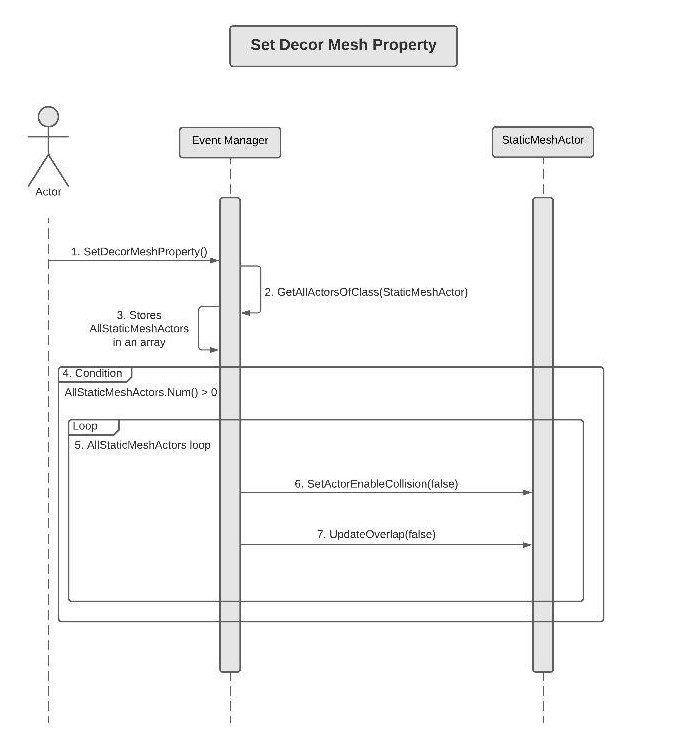
**5. Process View**

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

**A. Set Decor Mesh Property**

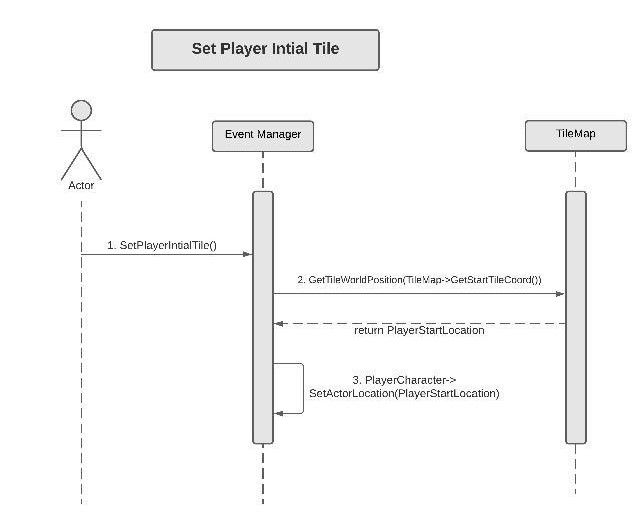
Event Manager disables the collision and overlap updates for all the non-gameplay critical actors such as trees, lamps, poles, and other decorative props. This prevents the player or mouse collision with the props, and also removes unnecessary overhead.

For this to work, in BeginPlay(), the Event Manager calls SetDecorMeshProperty(), which internally gets all the StaticMeshActors from the world. It then checks if the StaticMeshActor array size is valid and if true, then loops over it. In the loop, it disables the collision and UpdateOverlaps for all the array actors by calling SetActorEnableCollision(false) and UpdateOverlap(false) respectively.



**B.** **Set Player Intial Tile**

In BeginPlay(), the Event Manager also set the player location to the Start Tile. For this, EventManager asks the TileMap about the StartTile WorldPosition and sets the player position to this location.



**Turn Switch Events**

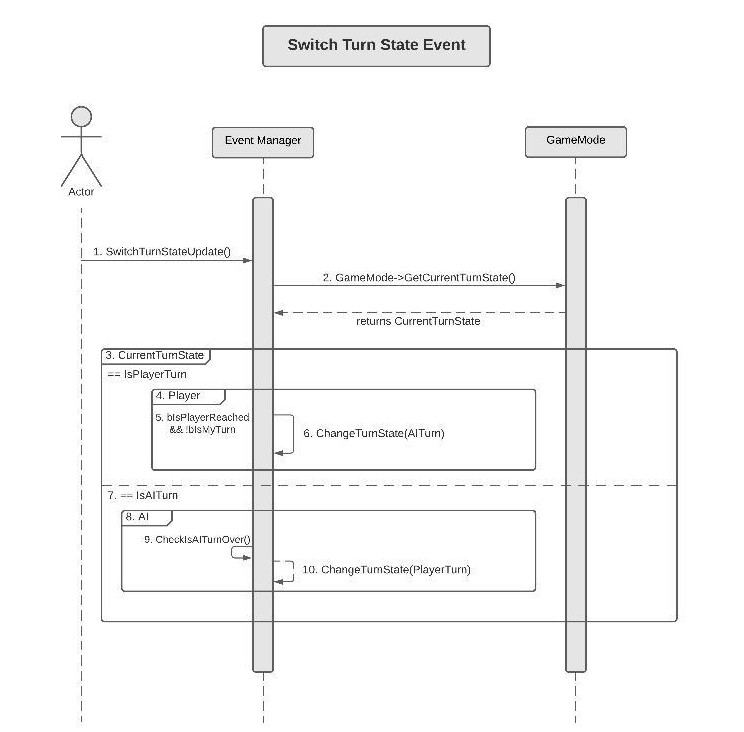
Event Manager is now responsible for handling the turn state throughout the level. For this to work, every character is now only concerns about his turn and uses the bIsMyTurn member variable. Every Character sets the variable to false, once done moving. And Event Manager is the one that sets this variable to true allowing the character to move again.

**C.** **Switch Turn State Event**

Now, for this Turn System to work, the Event Manager calls the SwitchTurnStateUpdate() function in Tick().

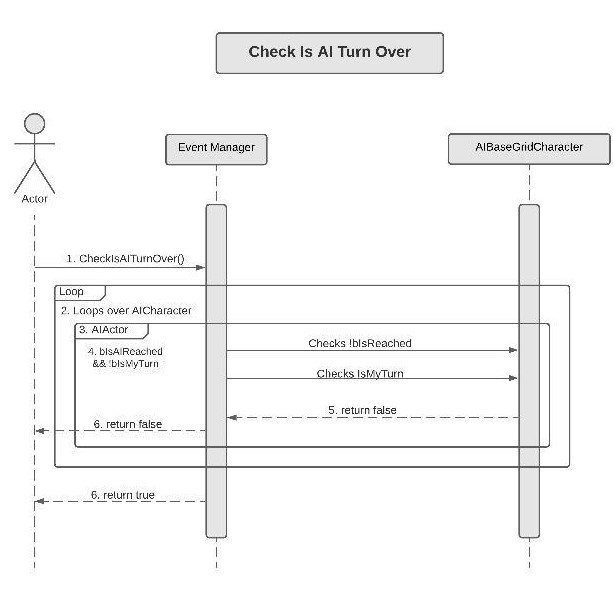
In SwitchTurnStateUpdate, the Event Manager gets the current Turn State from the GameMode. Then if the Turn State is "PlayerTurn", it will check if the player is reached to the tile and player turn is over, if true, it will then call ChangeTurnState() passing in AITurn.

Now, if the Turn State is "AITurn", it will call CheckIsAITurnOver() and if true it will ChangeTurbState to PlayerTurn.



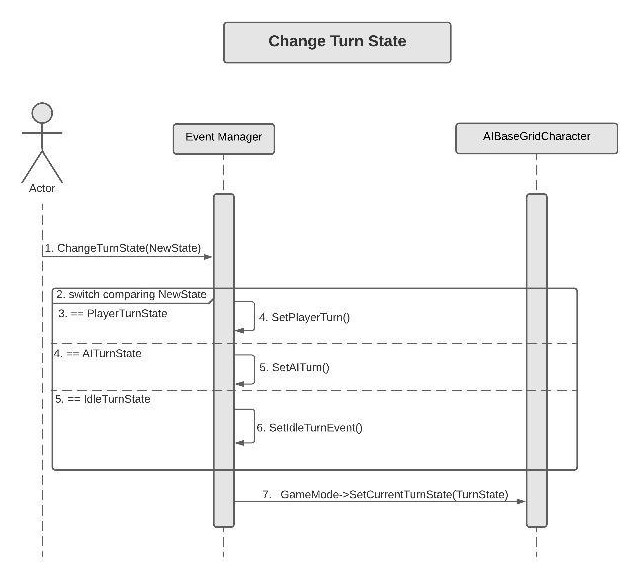
**D. Check Is AI Turn Over**

To check if all the AI Turn is over or not, the Event Manager loops over the array of all the AICharacters and check if their respective turn is over by checking "bIsMyTurn" boolean. If every AI's bIsMyTurn is false, CheckIsAITurnOver will return true, else it will return false.



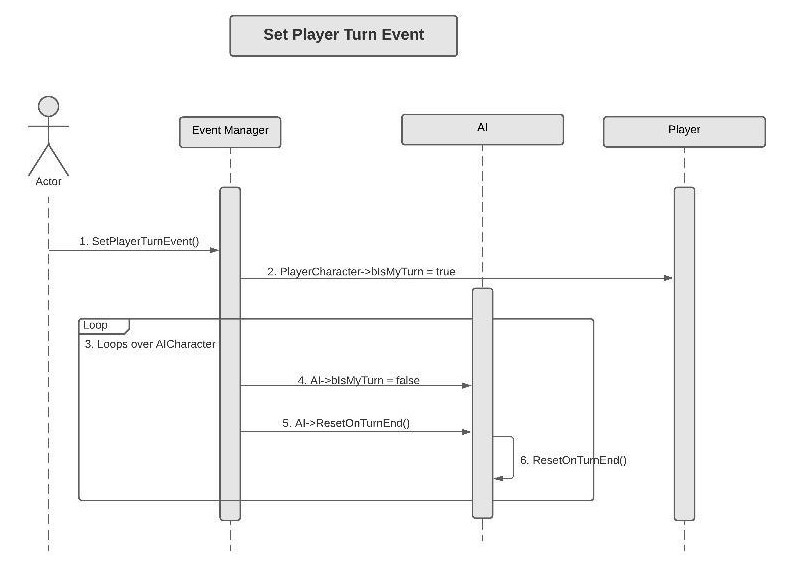
**E.** **Change Turn State**

ChangeTurnState() function is just a switch state, which based on the passing argument calls the appropriate function. For instance, if the passed State is PlayerTurn, it will call SetPlayerTurn() function or if the passed State is AITurn, it will call SetAITurn() function. It also SetCurrentTurnState to the passed one in the GameMode.



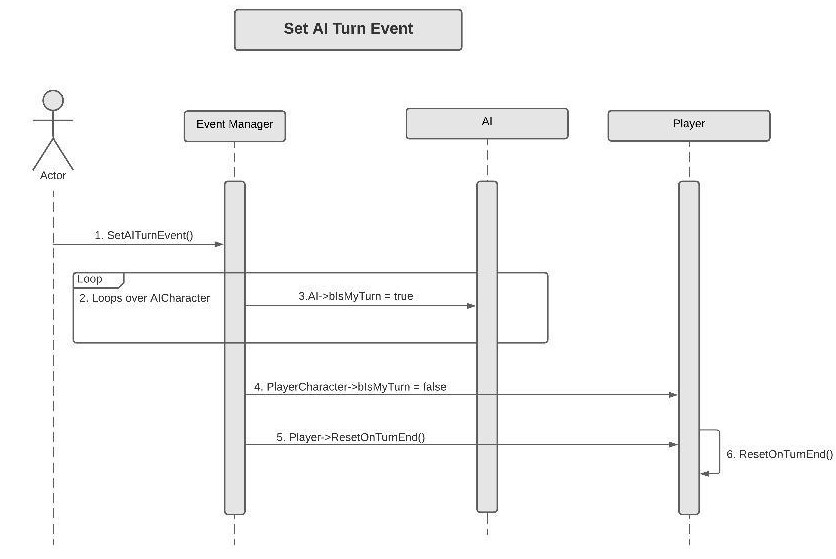
**F.** **Set Player Turn Event**

Now, SetPlayerTurnEvent(), sets the variable bIsMyTurn to true for player, allowing it to move. Then it also loops over all the AI and sets their bIsMyTurn to false, and calls ResetOnTurnEnd() which resets some of the variables in the AI.

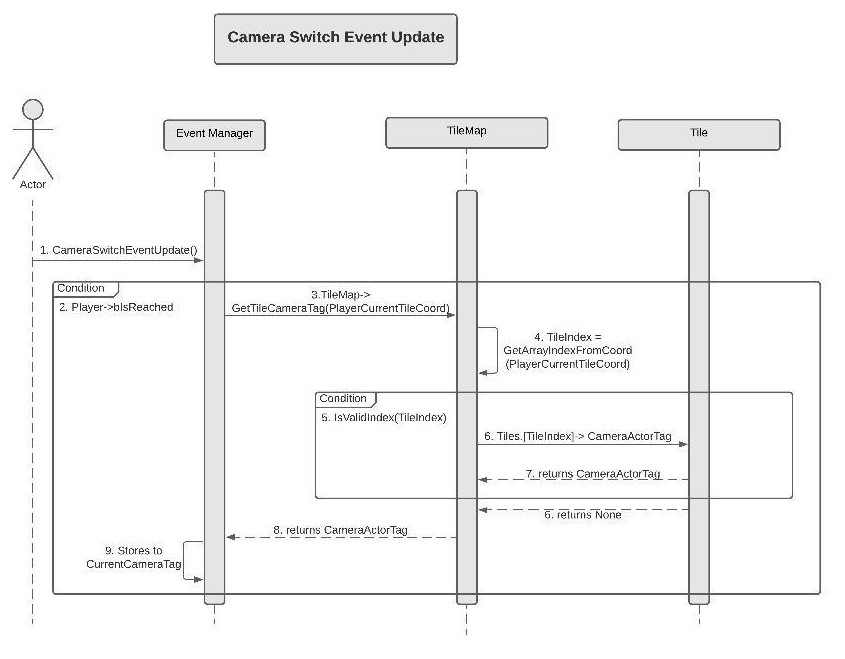


**G.** **Set AI Turn Event**

Lastly, SetAITurnEvent(), loops over all the AICharacter and sets their "bIsMyTurn" to true. It also set the bIsMyTurn variable to false for Player and calls ResetOnTurnEnd() which will reset some of the internal variables.



**H.** **Camera Switch Event Update**

Event Manager also handles when to switch the camera based on the PlayerCurrentTileCoord. In Tick(), it calls CameraSwitchEventUpdate() which firstly check if the player is reached on the Tile by checking bReached. Then it passes the PlayerCurrentTileCoord to the TileMap, calling GetTileCameraTag. In the TileMap class, TileMap gets the index for the tile using the passed PlayerCurrentTileCoord. Then if the TileIndex is valid, it will ask the Tile for its CameraActorTag. Then TileMap will return the CameraActorTag to the Event Manager. Lastly, Event Manager updates its member variable and stores CurrentCameraTag. Event Manager has a getter for the variable used by the CameraManager to set the next Camera Switch.

**Sequence View for Behaviour Trees**

Firstly, there are three possible StarType for a level in the game.

Level Completion Star - Get it by completing the level.

Collectible Star - Get it by collecting in-level Star prop.

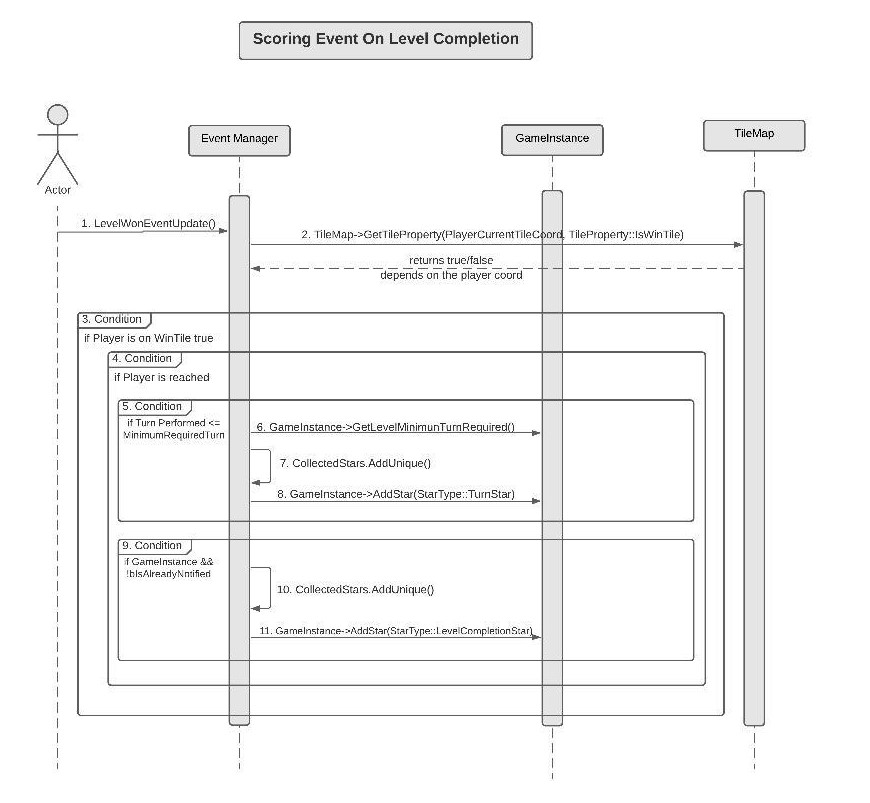
Turn Star - Get it by completing the level in the minimum number of Turns.

**I.** **Scoring Event On Level Completion**

In LevelWonEventUpdate(), TileMap checks if the PlayerCurrentTileCoord's property is IsWinTile. If true, it will check if the player is bReached or not.

If both the player is on the Win Tile and already reached, the Event Manager will check if the Number of Turns performed by the player to get there is less than or equal to the MinimunRequiredTurn for the level. GameInstance holds the MinimumRequiredTurns for each level. If true, then it adds a star to the GameInstance by calling AddStar passing in the StarType::TurnStar.

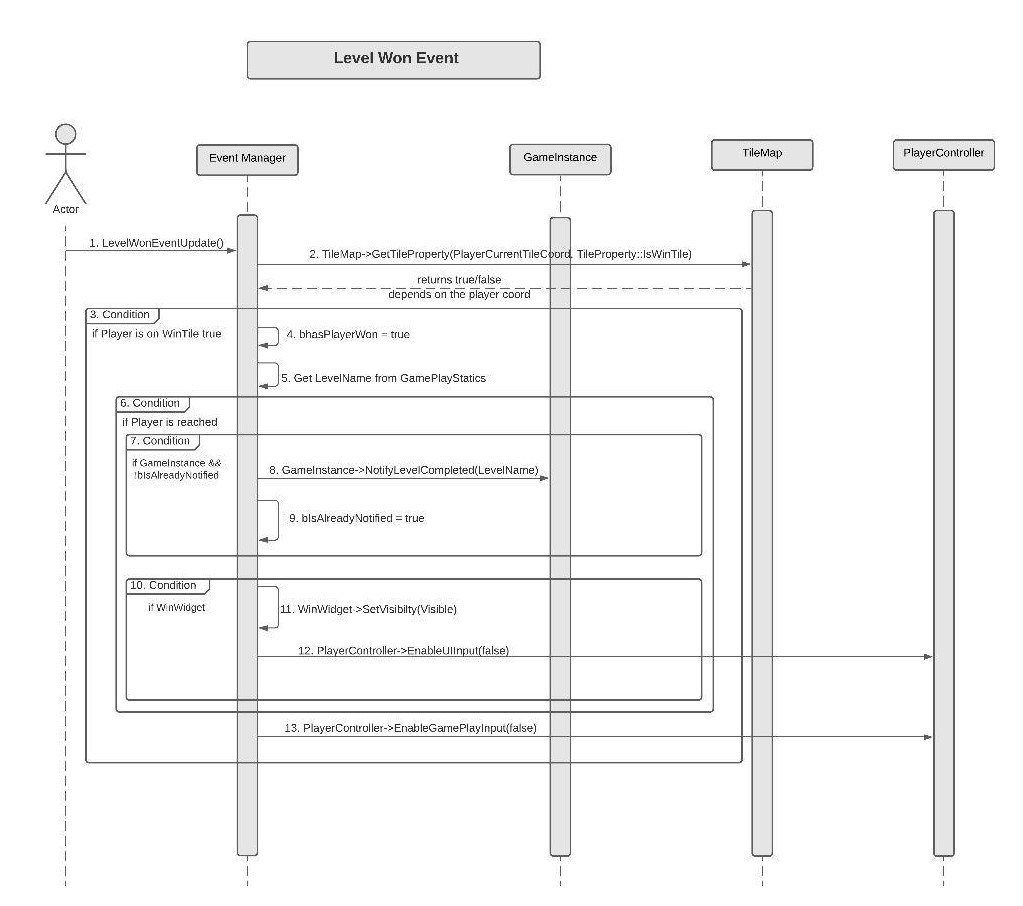
Now, On Level completion, Event Manager also calls AddStar on GameInstance passing in StarType:: LevelCompletionStar.

****

**J.** **Level Won Event**

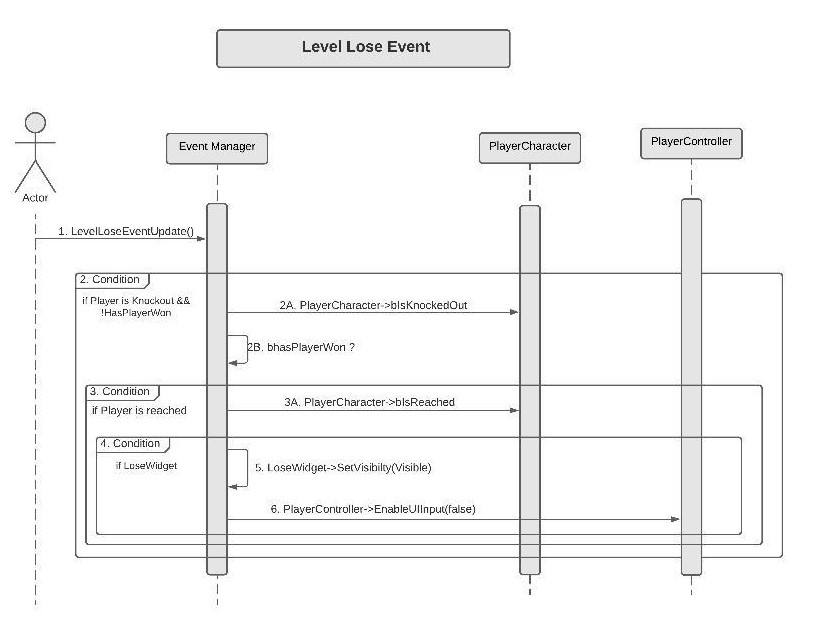
In addition to handling Scoring System, LevelWonEventUpdate() also handles drawing the Win Widget and notifying the GameInstance about Level Completion.

For this, LevelWonEventUpdate(), checks if the PlayerCurrentTileCoord is WinTile. If true, it sets the bHasPlayerWon to true. Now, if the player is reached, it will notify the GameInstance to prepare to load the next level by passing the CurrentLevelName. Then it will set the Win Widget visible and also disables the player GamePlayInput.

****

**K. Level Lose Event**

Similar to LevelWonEvent, LevelLoseEventUpdate is responsible for drawing the Lose widget. For this, LevelLoseEventUpdate() checks if the player is bKnockedOut or not and the player has not already won and, has player bReached. If true it will set Lose Widget visible and disable the controls.

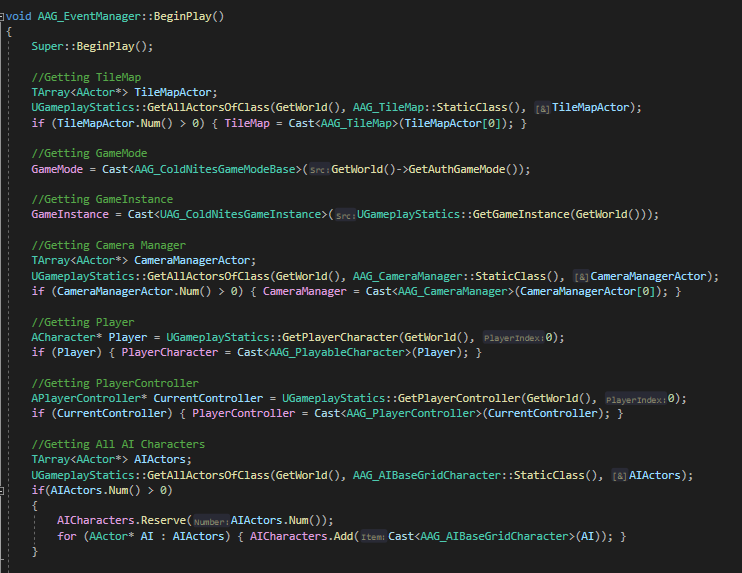
****

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

The use case will focus on showcasing the use of AI System, along with its application so that the Event Manager module can act as a guide/reference for someone not quite familiar with the EventManager codebase.

**Event Manager on Initialization –**

Event Manager stores all the useful actor/character pointer as member variables. It gets almost all the actors from the world and it initializes all these member functions in the BeginPlay() call. And most probably we will need these variables to write any level-specific logic.



In the BeginPlay(), it also does some miscellaneous calls.

For Instance, SetDecorMeshProperty() disables the collision and updates for all the static world actors, mainly props, such as trees, poles, other cosmetic actors so that they don't come in the way of player or mouse clicking.

Similarly, it also sets who is supposed to take the first turn by calling SetFirstTurn(TurnState).

It also snaps the player to the Start Tile by calling SetPlayerIntialTile().

And it also initializes the Win/Lose Widget and sets them Hidden by calling LevelWinEventInit() and LevelLoseEventInit() respectively.

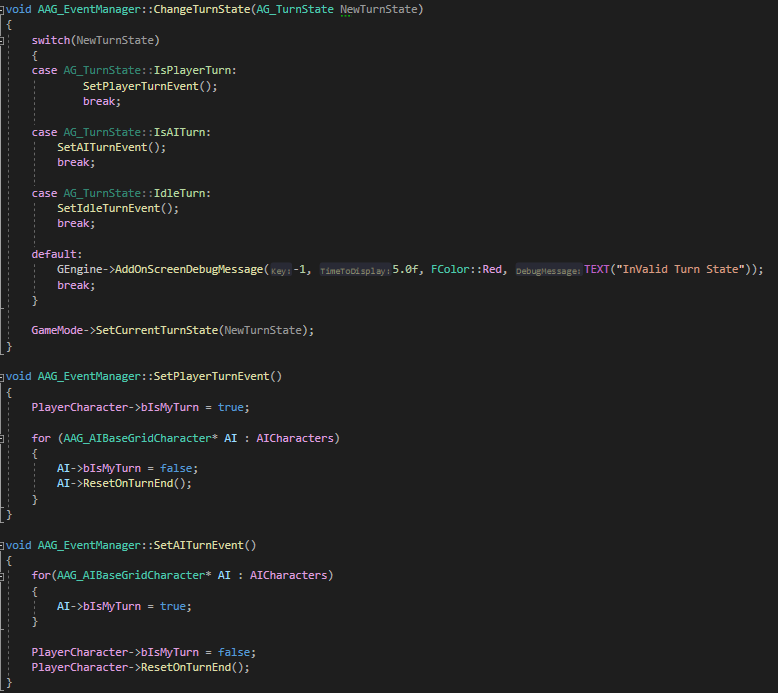
**Turn State Events –**

Event Manages makes the use of the Turn System convenient, consistent and easy to use for all the characters in the game.

Now Event Manager handles all the backend logic to change the TurnState initialized in the GameMode, instead of the character accessing the GameMode directly and changing the Turn State as implemented in the previous Turn-System. This makes debugging extremely easy as well.

Now, to achieve this, every BaseGridCharacter is only concern about their own Turn. BaseGridCharacter holds a boolean "bIsMyTurn" and player or AI is only allowed to move when it's true. Event Manager is the one that sets this variable to true, allowing them to move. And, the Player or AI are supposed to set this variable to false, once they are done moving.

For Instance, the Behaviour tree sets this to false, in the final node of every sequence. And similarly, PlayerControllers StopMove() does the same on KeyReleased.



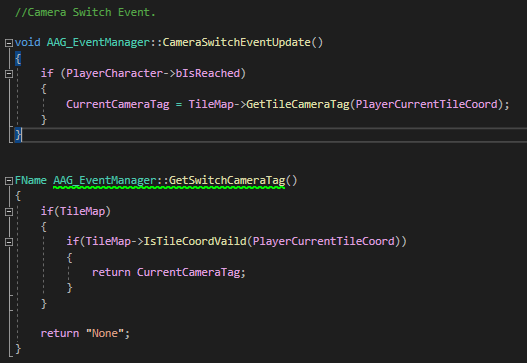
**Camera Switch Events –**

Event Manager always keeps the track of PlayerCurrentTileCoord by calling

PlayerCurrentTileCoord = TileMap->GetTileCoord(PlayerCharacter-> GetActorLocation()) in the Tick().

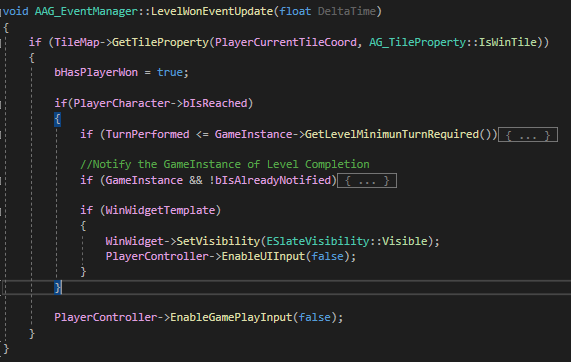
Using this Coord EventManager gets the desired "CameraTag" from the TileMap passing in the playerTileCoord. Lastly, EventManager has a getter of the desired CameraTag, GetSwitchCameraTag(). Using this Camera Manager gets the Current CameraActorTag and switches to that Camera.

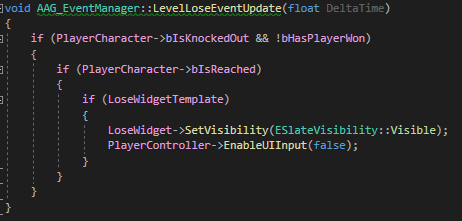
Note - Detailed Explanation for the CameraTag setup is covered in the Camera Manager Module.



**Win/Lose Events –**

Event Manager constantly checks from TileMap about PlayerTileCoord is equal to "IsWinTile" and IsPlayer "bKnockedOut". If true, it draws the Win or Lose Widget accordingly. It also disables the player gameplay Input and notifies the GameInstance, to prepare to load the appropriate level.





**Scoring Events –**

Event Manager keeps track of all the stars collected throughout the level. If the player collects the stars, the Event Manager informs the GameInstance about the collected stars, so that GameInstance can increment the TotalCollectedGameStars, if the star is unique.

Note - Detailed explanation about the TotalCollectedGameStars Increment will be in Scoring System Module.

