****

**COLD NITES**

Alpha-2

**AI System Design Document**

Designed & Implemented by

Yash Chamria

**Change Log**

**Version** – 0.1

**Modifier** – Yash Chamria

**Date** – 1th April, 2021

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

**Version** – 0.4

**Modifier** – Yash Chamria

**Date** – 4th April, 2021

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

**Version** – 0.9

**Modifier** – Yash Chamria

**Date** – 6th April, 2021

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

**Version** –

**Modifier** – Yash Chamria

**Date** – April, 2021

**Description** –

**Table of Contents**

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

**A. Base AI Character**

**B. AI Controller**

**C. Behaviour Tree and Task Nodes**

1. **Logical View 7**

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

**B. Mid-Level Design of AI System**

**C. Detailed Design of AI System**

1. **Process View 11**

**A.**

**B.**

**C.**

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 AI System. AI System is responsible for various enemy types in the game. It will also handle the TileMap Grid-Based and Turn-Based behaviour of the game for the enemy characters.

This document will describe the architecture and design choices that make the AI System 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** – All the AI inherits from one base class and is behaviour tree driven. As a result, new AI behaviours can be easily created by just making new BTTaskNodes.

**Level Designer** - One of the main targets of the AI System is to achieve ease of AI Creation. AI can be created by just creating the blueprint of the BaseAI class. It will provide the Task nodes of different behaviour to extend the existing AI behaviour or creating a new altogether.

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

* The design should minimize the complexity of creating and defining new AI for the game.
* The design will allow easy setup for the programmers to define new task nodes for the AI.
* The design should allow the level designer to experiment with different behaviours for the AI on the fly.

**3. System Overview and Behaviour**

The AI System allows the easy creation of game enemies using the single BaseAI class and Behaviour Tree Task Nodes.

**Base AI Character -**

Note - The Base AI Character is already documented in BaseGrid Classes Module in great detail. This module will include those details and will discuss the new functionality added to it.

AIBase Grid Character inherits from Base Grid Character and will be the base class of all the AI blueprints and the AI behaviours will be extended using the Behaviour Tree.

Since it inherits from the BaseGrid Character, it comes with all the movement and registration to the TileMap System hooked up.

It also comes with additional features such as Player Detection, Player Knockout and Actor in Range check.

**AI Controller -**

The main job of the AI Controller is to set up the behaviour tree. Every AI is supposed to be using its own AI Controller with different Behaviour Trees and Blackboard.

**Behaviour Tree and Task Node -**

The Behaviour Tree is the one, responsible for the AI Behaviour and what makes every AI unique. It uses a set of nodes that determine the tasks AI needs to perform.

In AI System, all the tasks are created in code, based on BTTaskNode Class. Then, these nodes can be attached to the behaviour tree for different enemy types. The nodes to handle player detection, handling the turn system, movement, knocking out the player, patrolling and so on are available to use.

**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 AI System Module, using a UML Class Diagram.

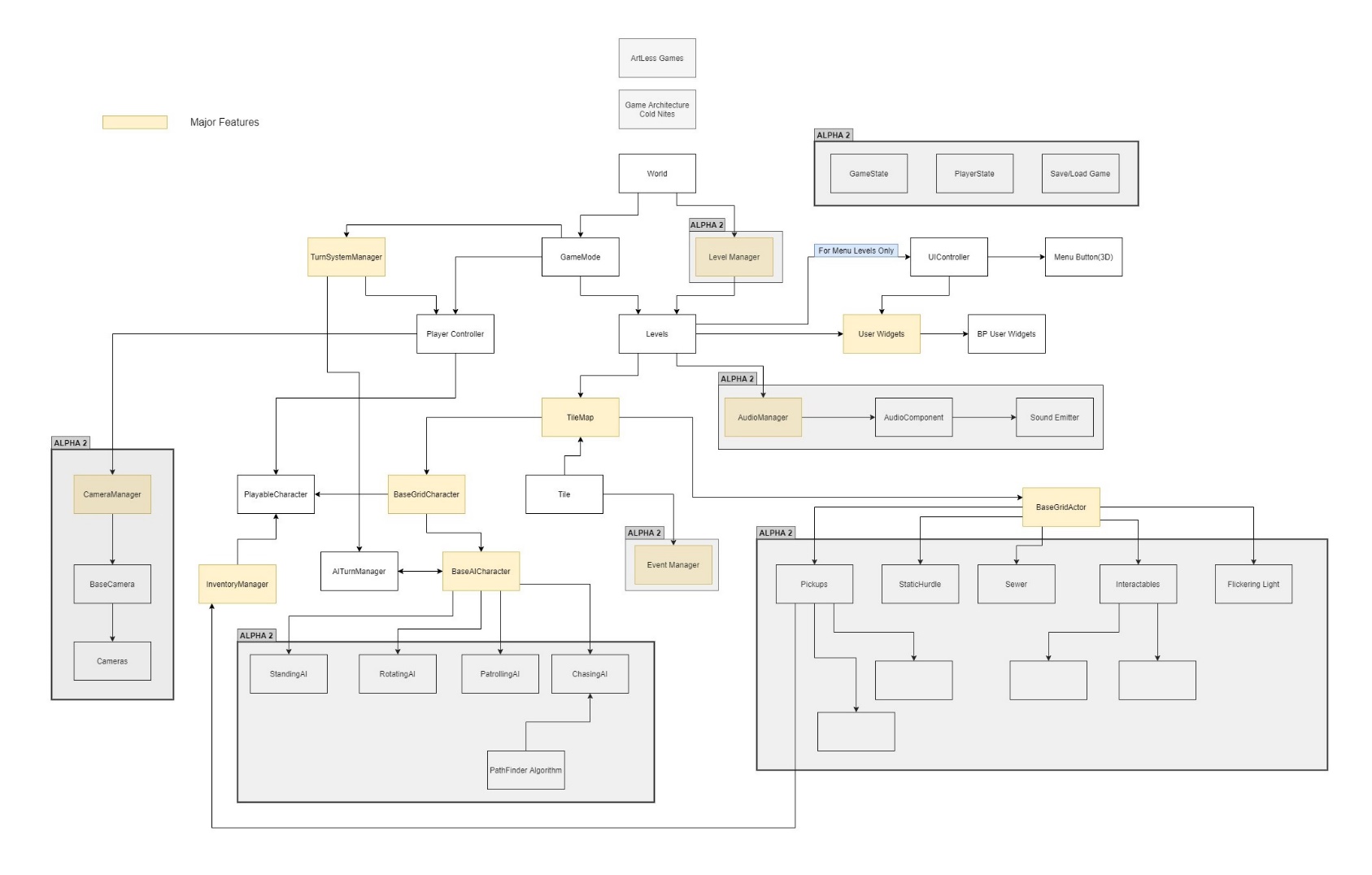
**A. High-Level Design Architecture of the Entire System**

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.



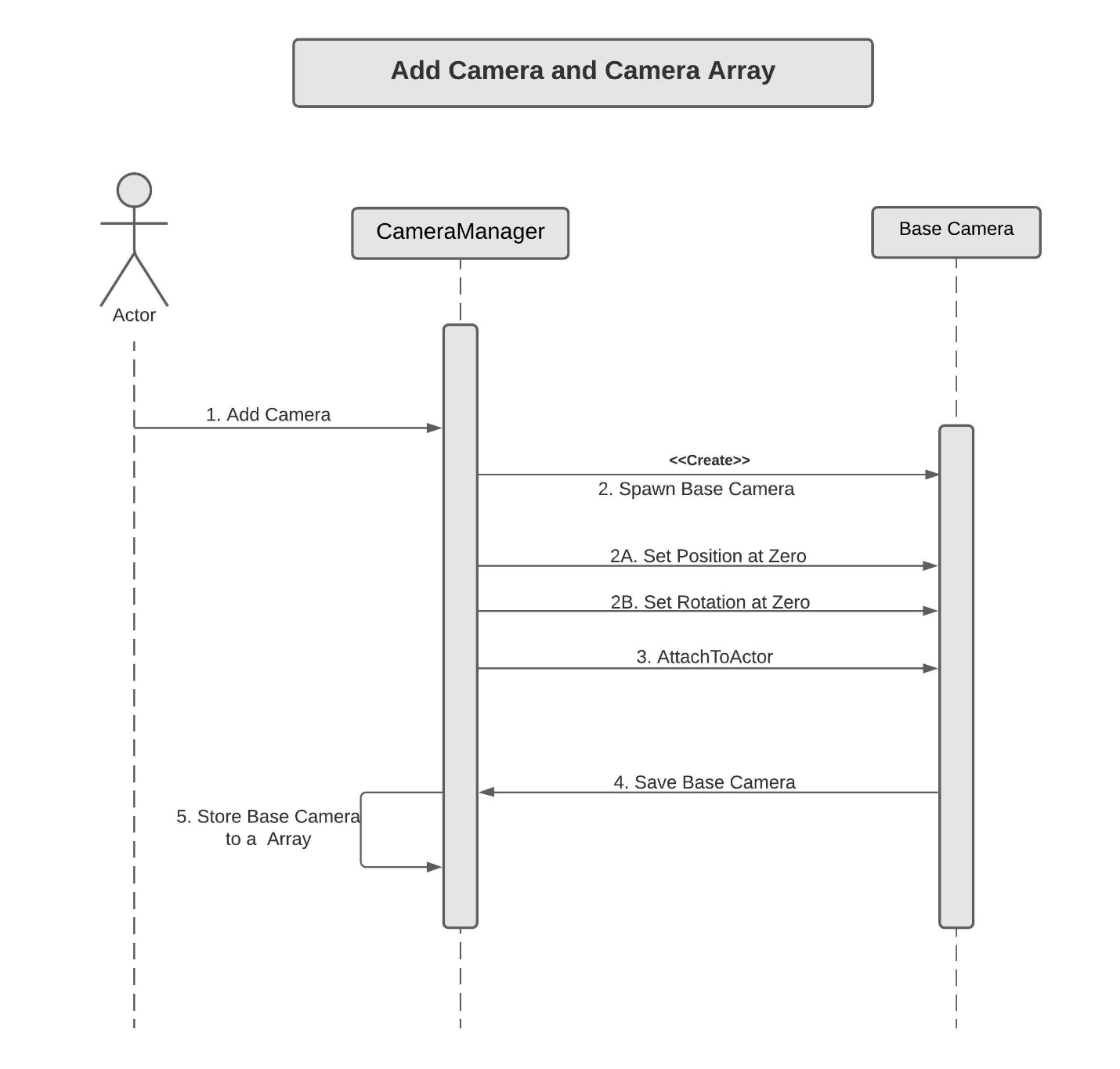
**B. Mid-Level Design of AI System**

**C. Detailed Design of AI System**

**5. Process View**

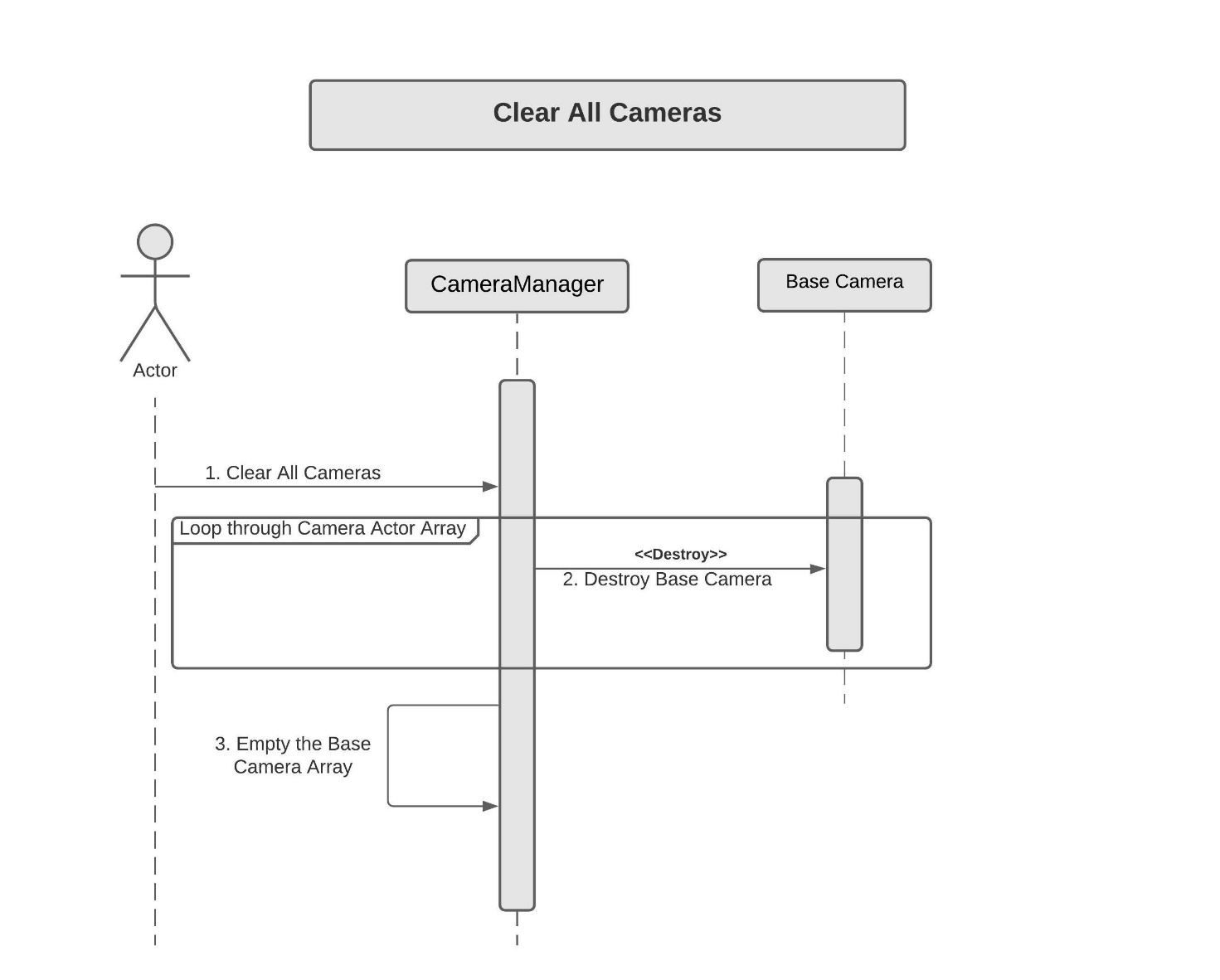
The process view will explain the relation and interaction between various cases using Sequence and Collaboration Diagrams.

**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.



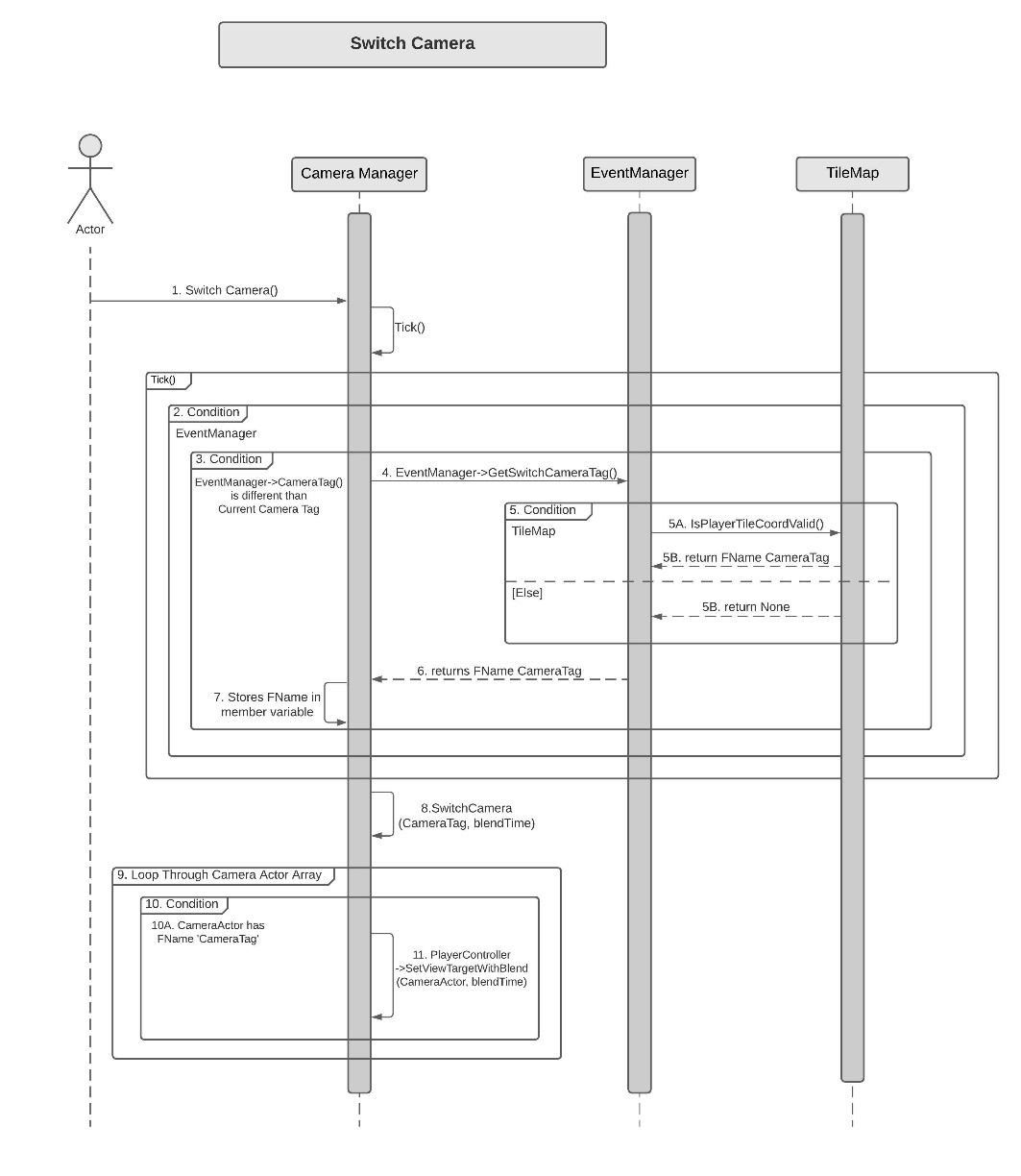
**C. Camera Transition/Switch**

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 AI System, covering how to create a Behaviour tree or create new task nodes in code. Also, it will explain its application so that the AI System module can act as a guide/reference for someone not quite familiar with the AI codebase.

**A. In Code:**

**AI Base Grid Class**

Going over the AIBasegridCharacter class, it comes with movement, player detection and knockout functions already in place. Also, inheriting from BaseGridCharacter allows access to all the standard components and functions of the base class.

Note - If one wants a function that allows the AI to do something, it should go in this class only.

For instance, if I want AI to interact with pickup, I will create a function to detect the pickup and a function to interact with it. These functions will later be called in the task nodes to define a new AI Behaviour.

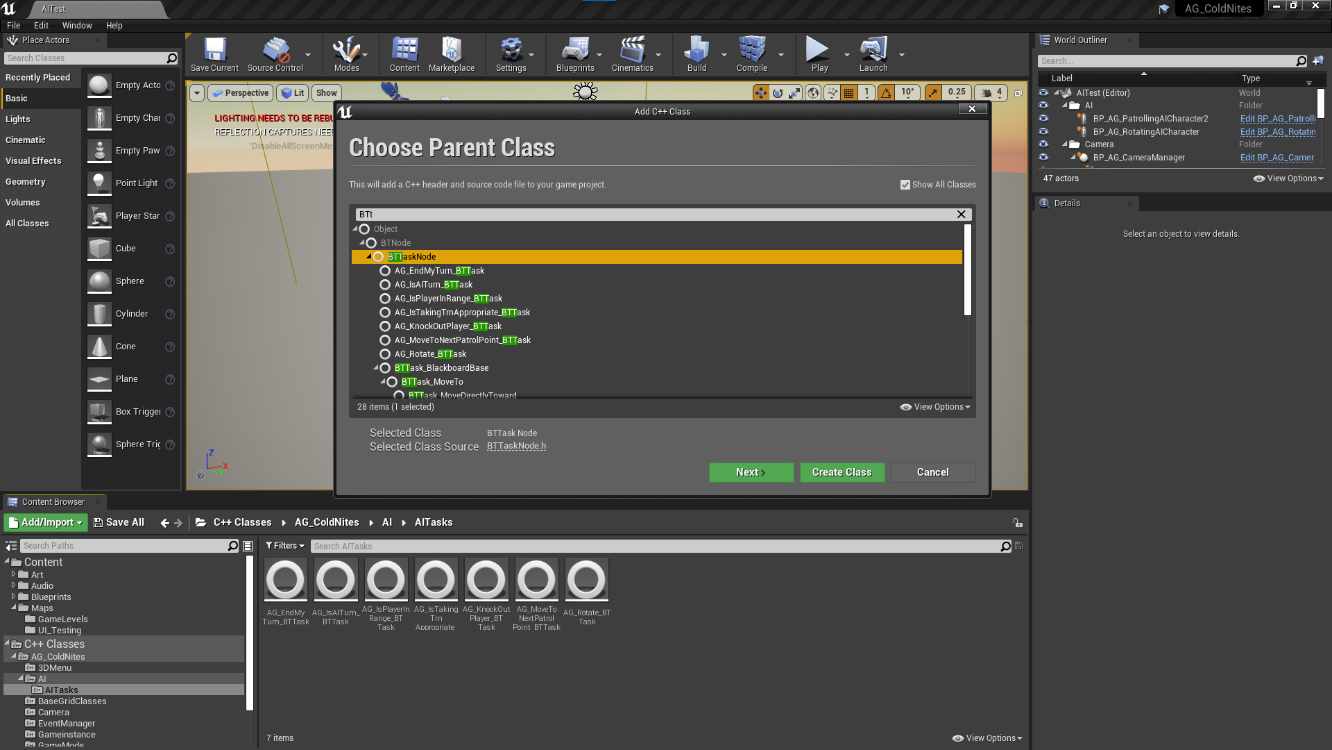
Note - For a detailed description of the BaseGrid classes refer to the previous Base Grid Classes Module.

**AI Controller**

AI Controller is mainly used for the Behaviour tree setup. It has Behaviour Tree, Behaviour Tree Component and Blackboard Component. It also locally stores the AI Character it possesses. Every AI in the game will use a unique controller with desired Behaviour tree attached to it.

**Behaviour Tree Task Nodes**

Behaviour Tree Task Node (BTTaskNode) is used to create the task node which is then used in Behaviour Tree to create various AI Behaviour Pattern. These nodes inherit from BTTaskNode class.



The basic pattern for these nodes is to, first of all, create a constructor and an ExecuteTask override function. The ExcuteTask function return 'Succeded' or 'Failed' based on the logic.

For Instance, if a node is about 'Player Detection', it will check if the player is detected or not, by calling the BaseAI class function, and if true, it will return 'Succeded', else 'Failed'.

Another repetiting standard pattern is to get the owner BaseAI Character, which can be achieved simply by

AIController = Cast<AAG\_BaseGridAIController->(OwnerComp.GetAIOwner());

if(AIController)

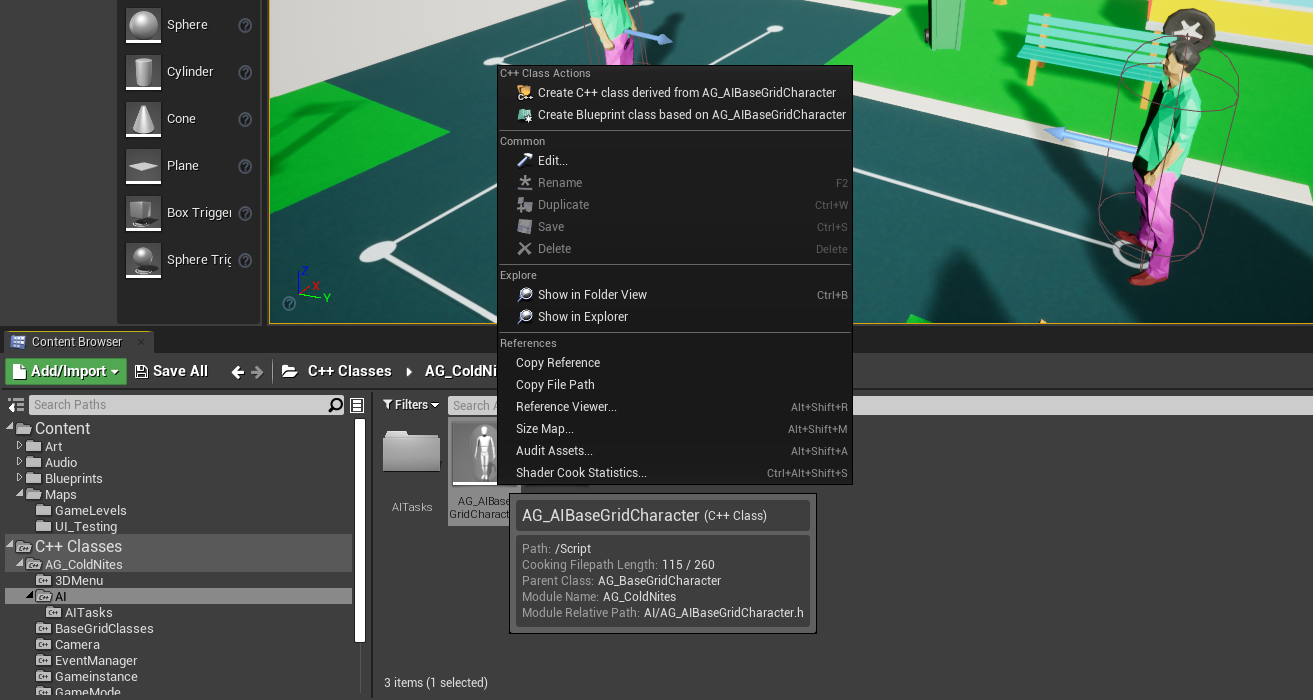
AAG\_AIBaseGridCharacter\* AICharacter = AIController->AICharacter;

**B. In Editor:**

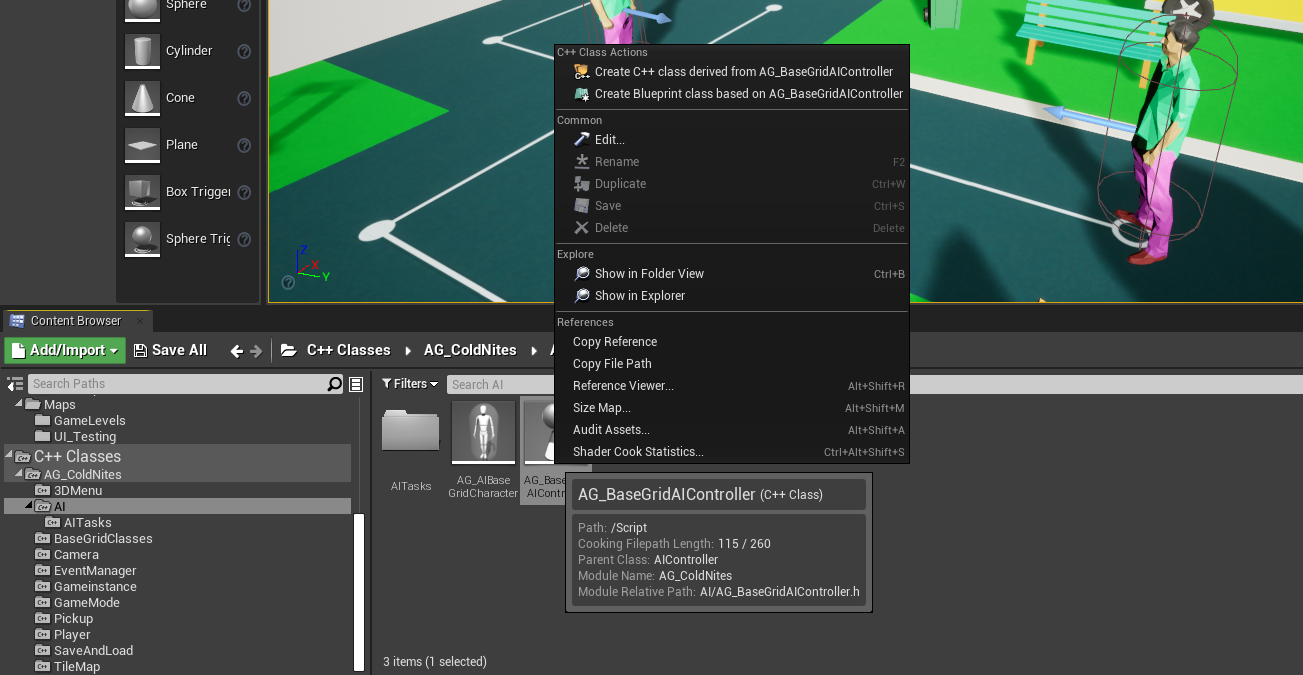
AI System setup for AI creation –

**Creating an AI Character, AI Controller and Behaviour Tree**

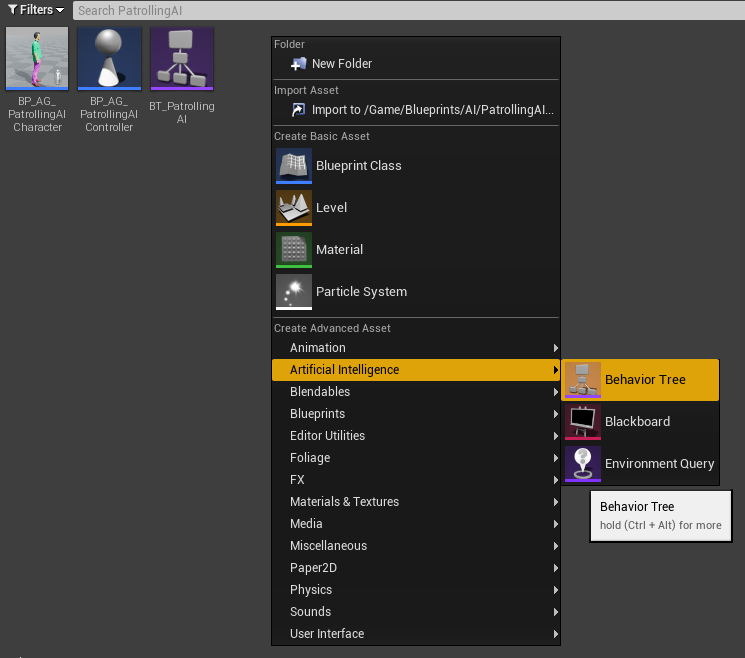
To create the AI Character, simply create a blueprint class based on AG\_AIBaseGridCharacter. Then rename the AI. Set any desired mesh.



Similarly, create a blueprint based on AG\_BaseGridAIController and rename it.

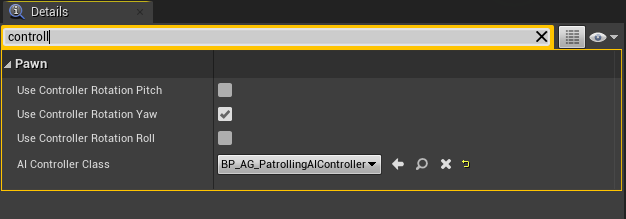


Finally, create a Behaviour tree for the AI and again rename it accordingly.

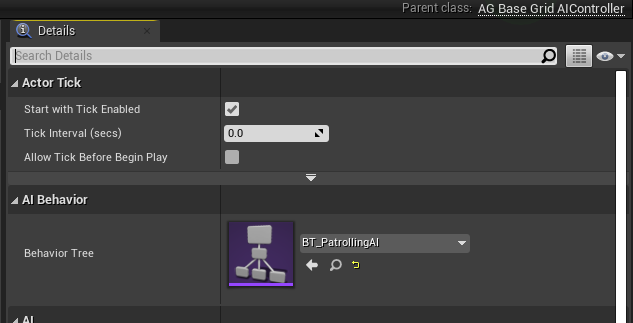


**Hooking up the AI Character, Controller and Behaviour Tree**

Firstly, hook up AI Character with the AI Controller.

****

Then, feed the Behaviour Tree to the Controller.

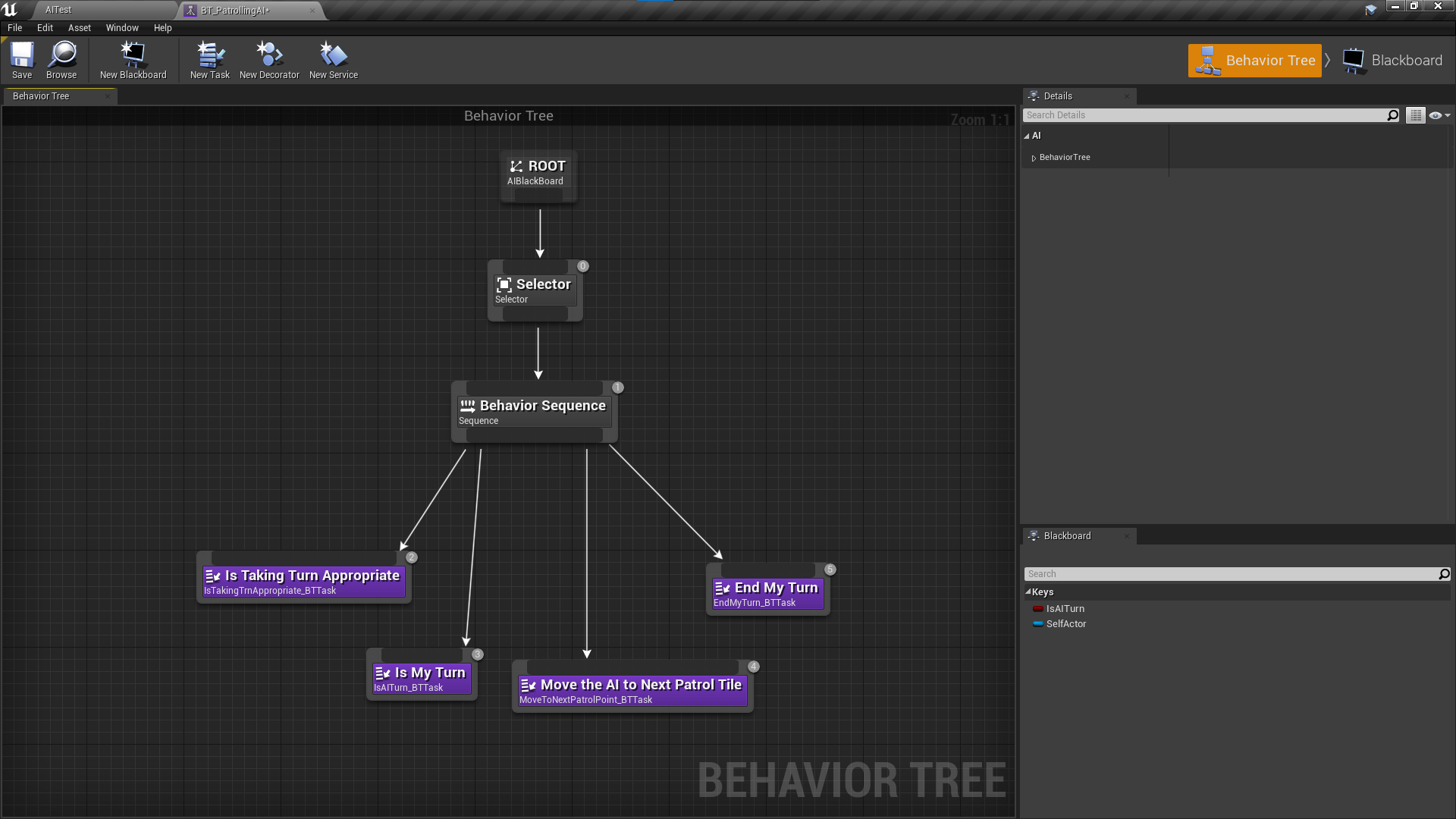
****

**Creating the Behaviour Tree**

Lastly, the fun part is to create the actual behaviour tree.

Since ColdNites is a Turn-based game, the First node for a sequence should be to check 'Is AI Turn Allowed'. This is handled with a couple of nodes, first one checks 'Is Taking Turn Appropriate' - this check if the player is knocked out or won - if true the node will return 'Failed'. The second node is 'Is AI Turn', this node asks the Event Manager for the AI turn (detailed description in the Event Manager Module).

Once these two nodes return success, we can mention the actual behaviour for the AI. For Instance, Move to Next Patrol Point or Rotate or do nothing, anything is possible (if the code is present for the node).

Finally, we can the 'End Turn' node, this lets' Event Manager know that the AI Turn is over.