****

**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** – 1.4

**Modifier** – Yash Chamria

**Date** – 15th 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**

**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. Check Player On Tile**

**B. Is Taking Turn Appropriate – Task Node**

**C. Is My Turn - Task Node**

**D. End My Turn – Task Node**

**E. Player Detection – Task Node**

**F. Rotate AI by 180 – Task Node**

**G. Move To Next Patrol Point – Task Node**

**H. Knock Out Player – Task Node**

**I. Attack Sequence – Behaviour Tree**

**J. Still AI – Behaviour Tree**

**K. Rotating AI – Behaviour Tree**

**L. Patrolling AI – Behaviour Tree**

1. **Use Case View and Practice 25**
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.

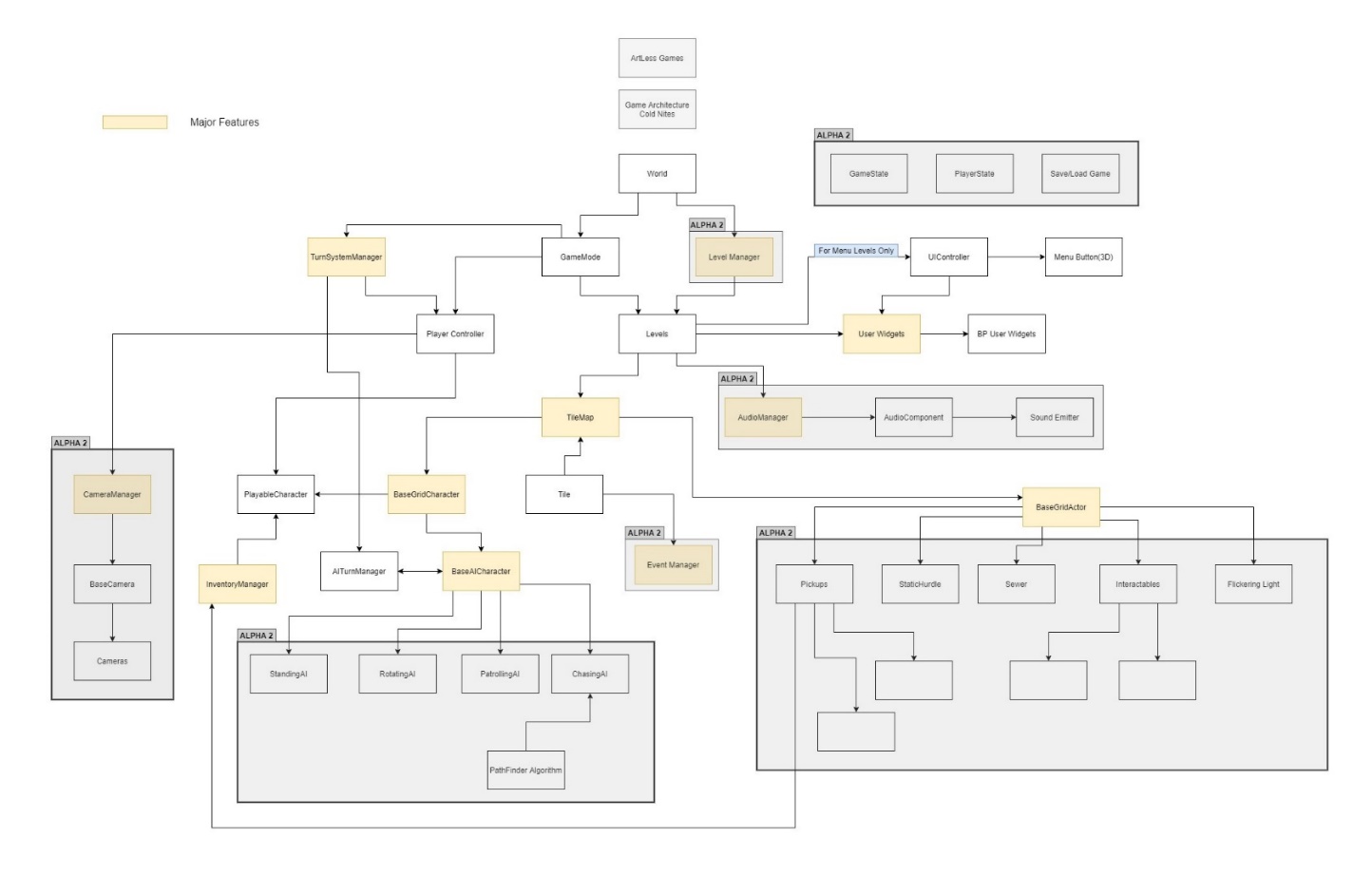
**A. High-Level Design Architecture of the Entire System**

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

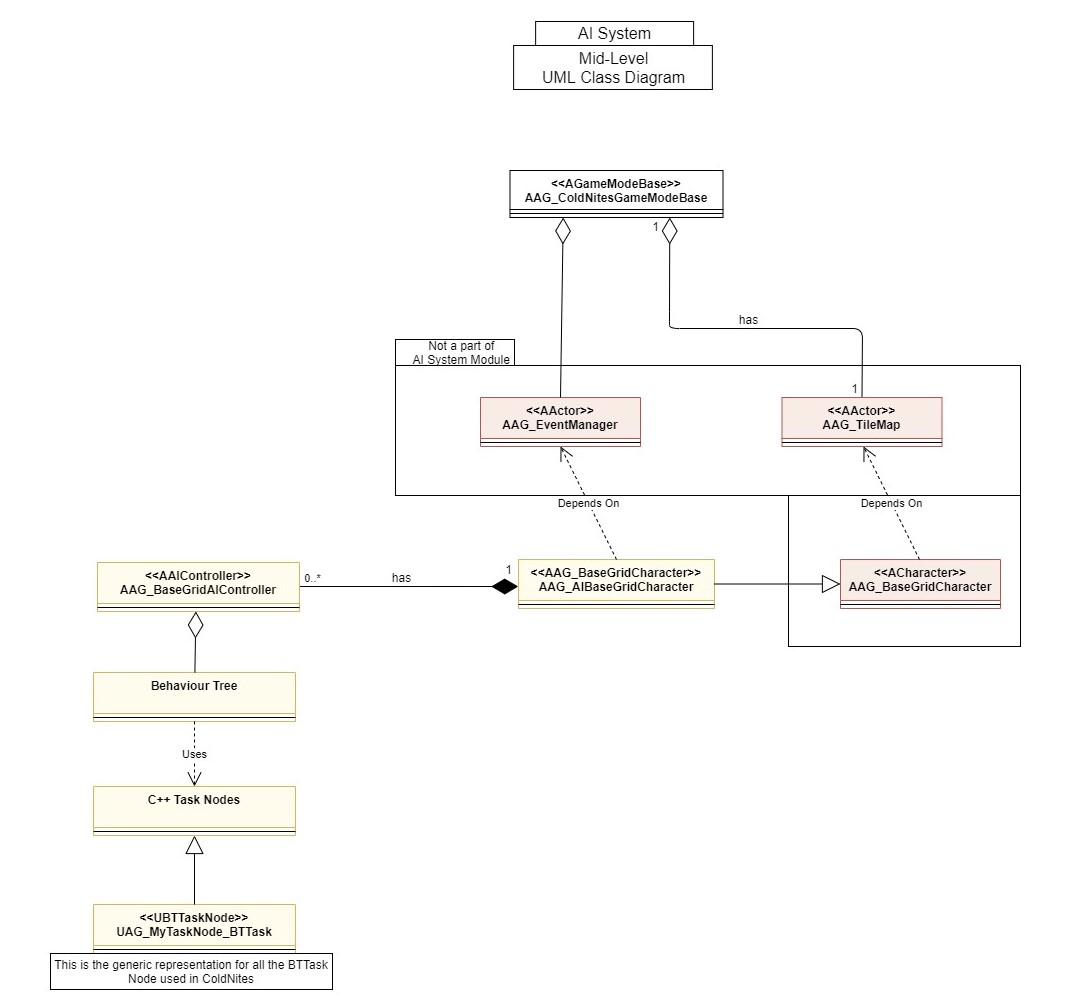
1. **TileMap** - TileMap provides the grid-based behaviour for the game and will facilitate the event system, based on the actor present on the Tiles.
2. **Turn-Based System** - This provides the turn-based aspect for the game. It is responsible for maintaining the turn order for all the world elements(actors)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

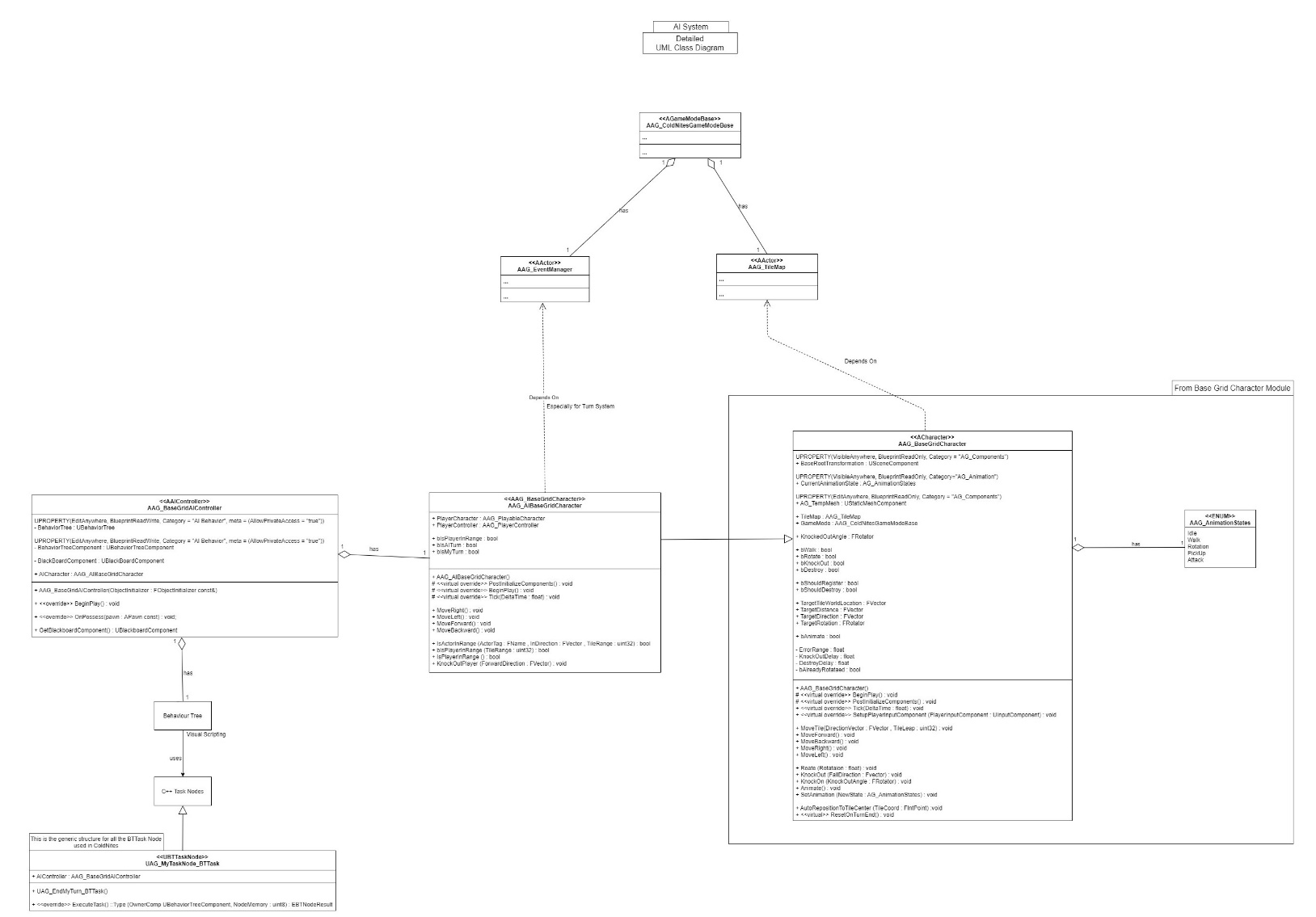
The primary features for the **Alpha 2** release:

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.
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.



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

****

**C. Detailed Design of AI System**

**5. Process View**

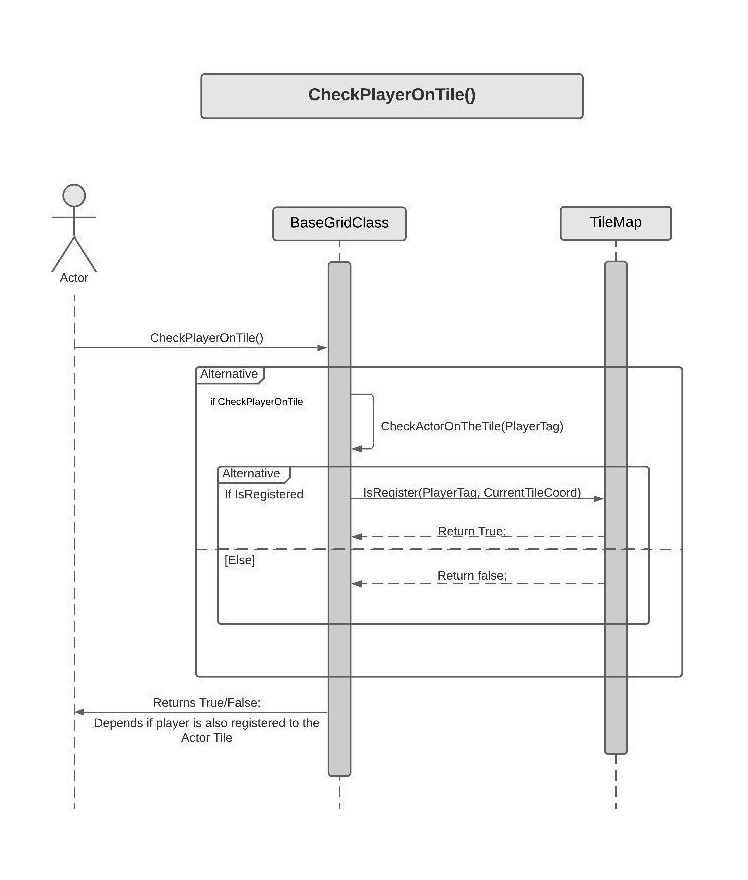
The process view will explain the relation and interaction between various cases using Sequence and Collaboration Diagrams.

**Note** - Most of the Sequence diagrams for AIBaseGridCharacter covered in Base Grid Actors Module still holds up. So, to keep the module self-contained as much as possible, I will reiterate some of the old sequence diagrams.

Also, the Sequence Diagrams for inherited functions from the BaseGridCharacter such as AutoRespositionToTileCentre, CharacterMovement, CharacterRotation and CharacterKnockOut can be found in Base Grid Classes Module.

**A. Check Player On Tile**

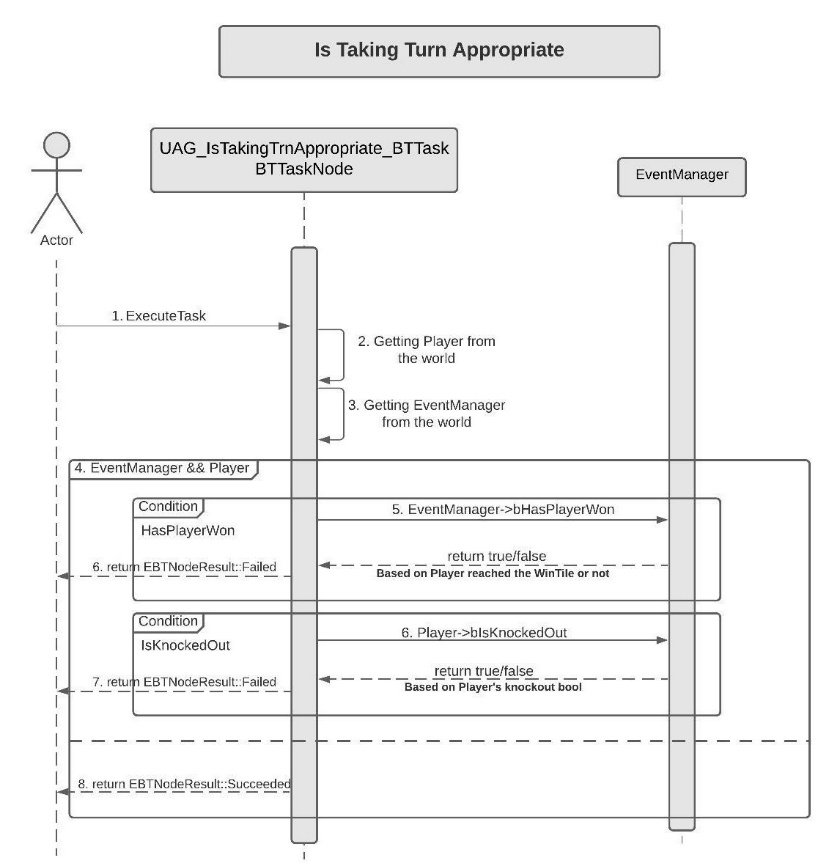
AIBaseGrid Character can check if the player is in range or not, just by calling CheckPlayerOnTile(). Internally, it will call a CheckActorOnTile() passing player tag as a parameter. This functions 'if checks' if the Current Actor Tile, also consist of the player character in the RegisterActor array. If true, the CheckPlayerOnTile returns true or vice versa.

****

**Task Node Sequence Diagrams**

**B. Is Taking Turn Appropriate**

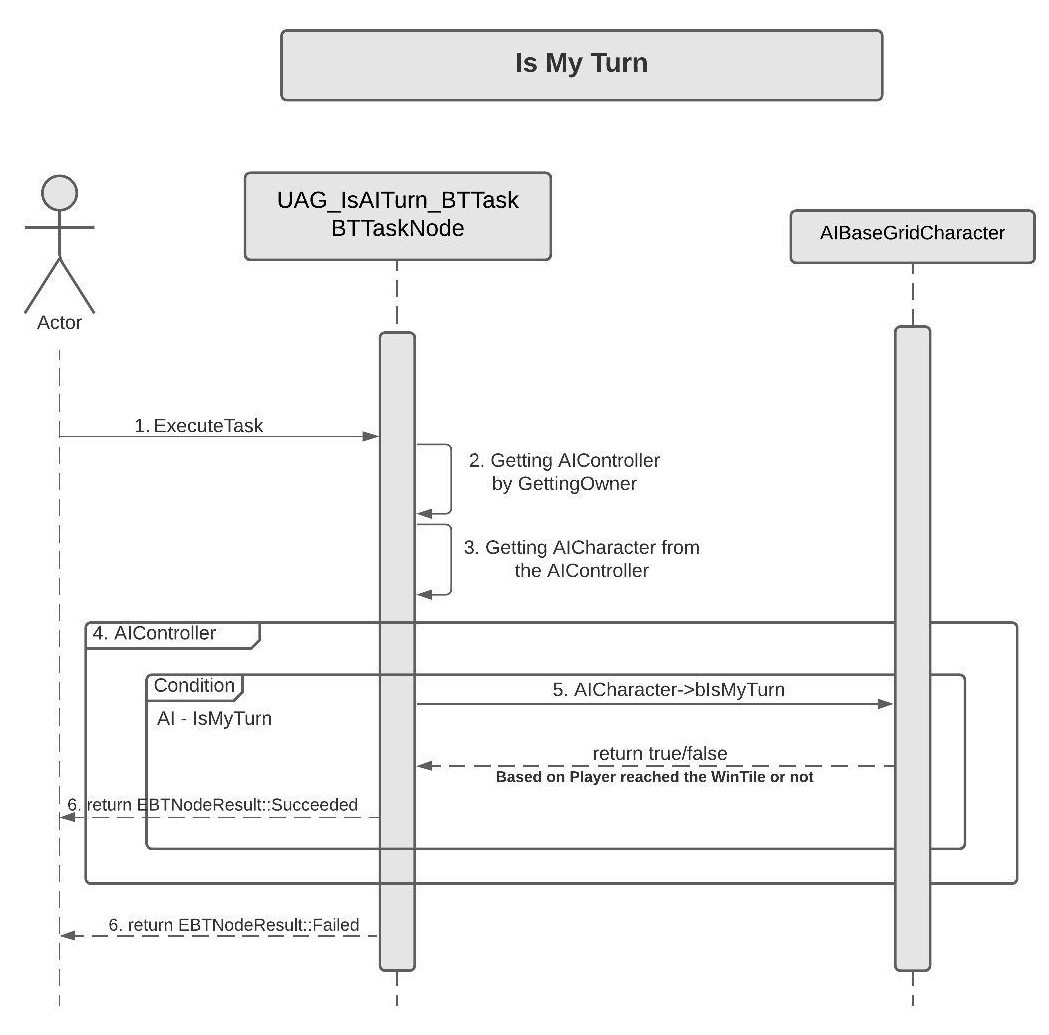
This node is responsible for checking if it is appropriate to move. It will check situations such as "is player knocked out" or "has player won" or "is AI knocked out". If it's true AI will not take its turn.

****For this AI asks the Event Manager about "HasPlayerWon". And to check if the player "IsKnockedOut", AI asks the player directly. If both return false, then only this node is considered "Succeeded".

**C. Is My Turn**

IsMyTurn's only responsibility is to check a boolean "bIsMyTurn", if true, AI can move.

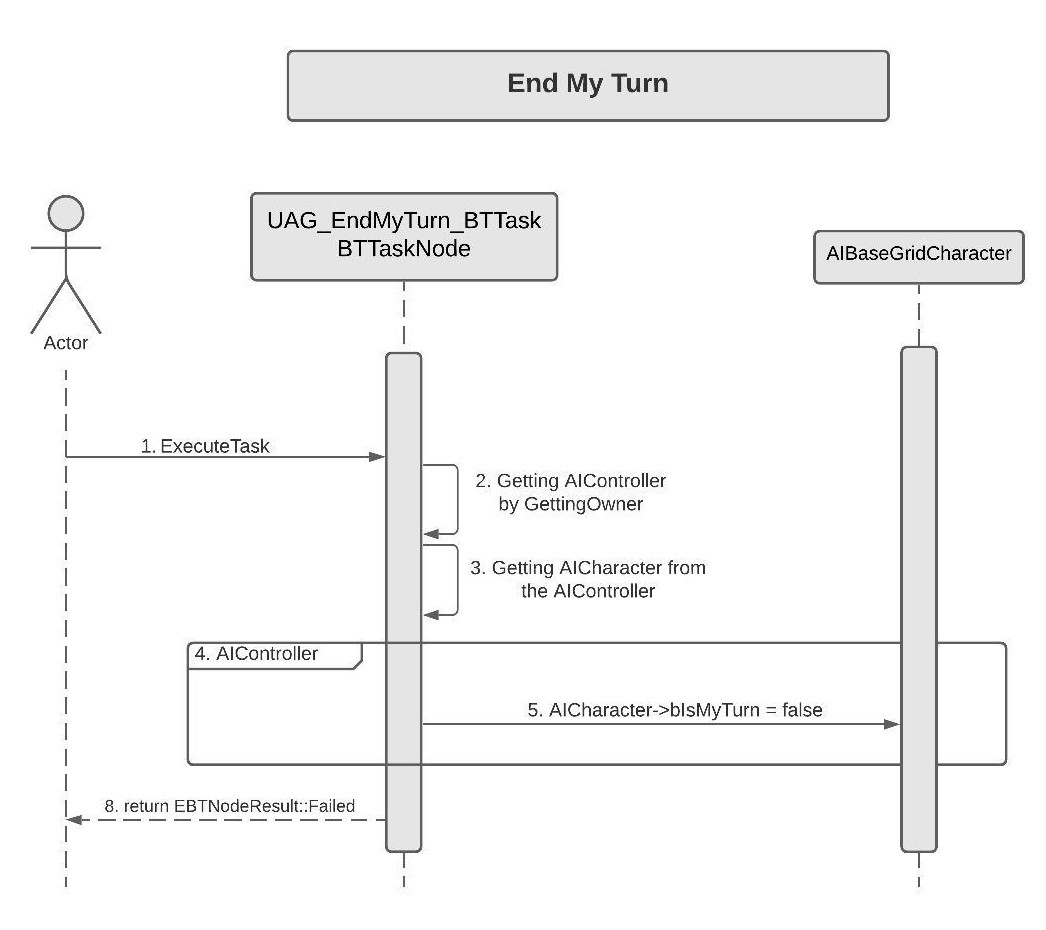
Firstly, the task gets the Owning Controller. And Controller has a member variable to store the AI it possesses. So, this node asks the AI "IsMyTurn" true or false and returns "Succeeded" or "Failed" accordingly.

****

**D. End My Turn**

EndMyTurn is in charge of ending the owning AI's Turn. First, it gets the AICharacter from the owning controller. Then it just sets the "BIsMyTurn" to false. Later, the Event Manager is the one that decides if all the AI is done with their Turns, then it will change the state to PlayerTurn.

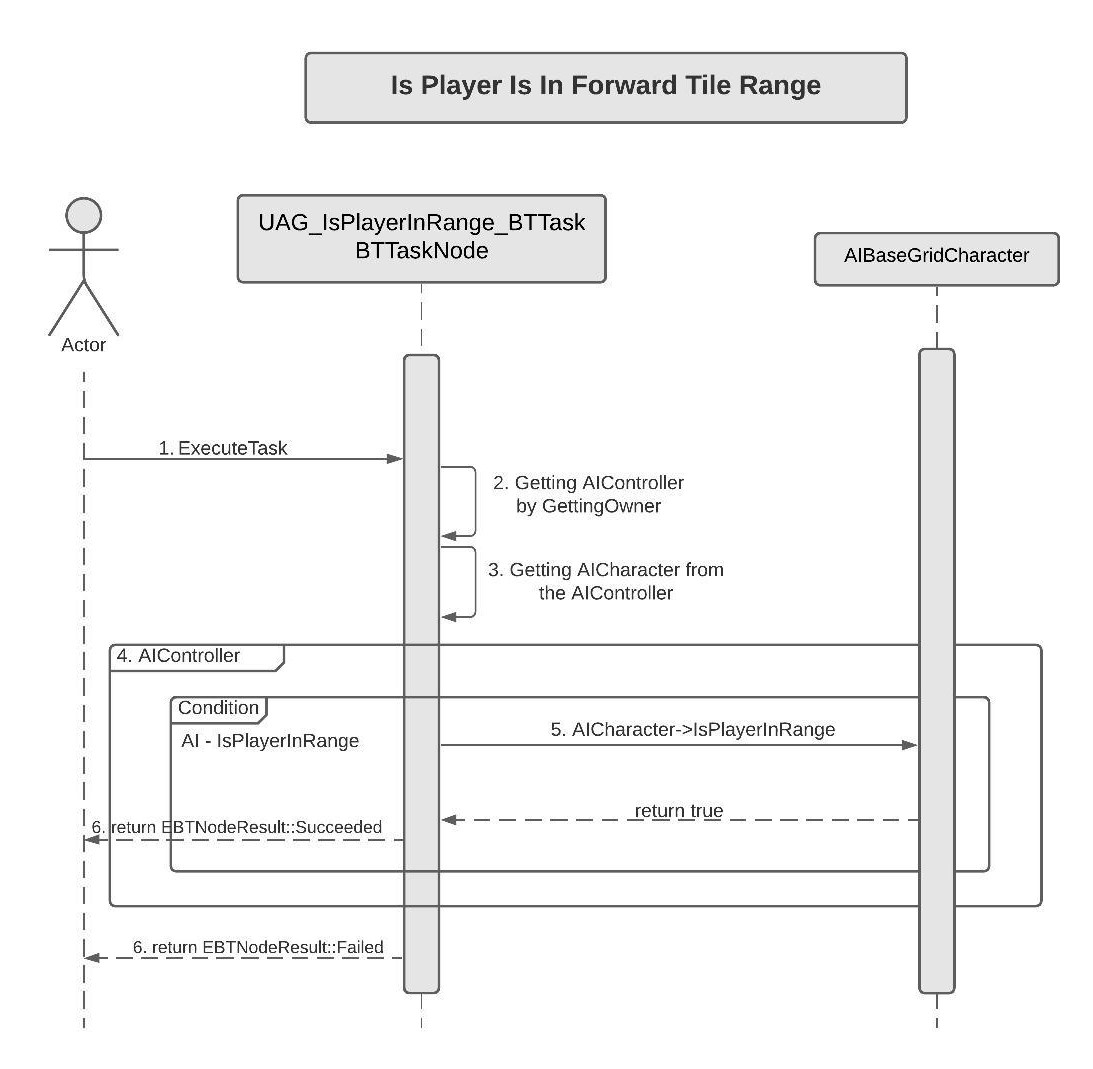
Note - More detailed description about how to turn works is provided in Event Manager.



**E. Player Detection**

The only duty of this node is to get the AICharacter from the AIController and calling the actual IsPlayerInRange function on AICharacter. If the functions return true, the node returns "Succeeded" or vice versa.

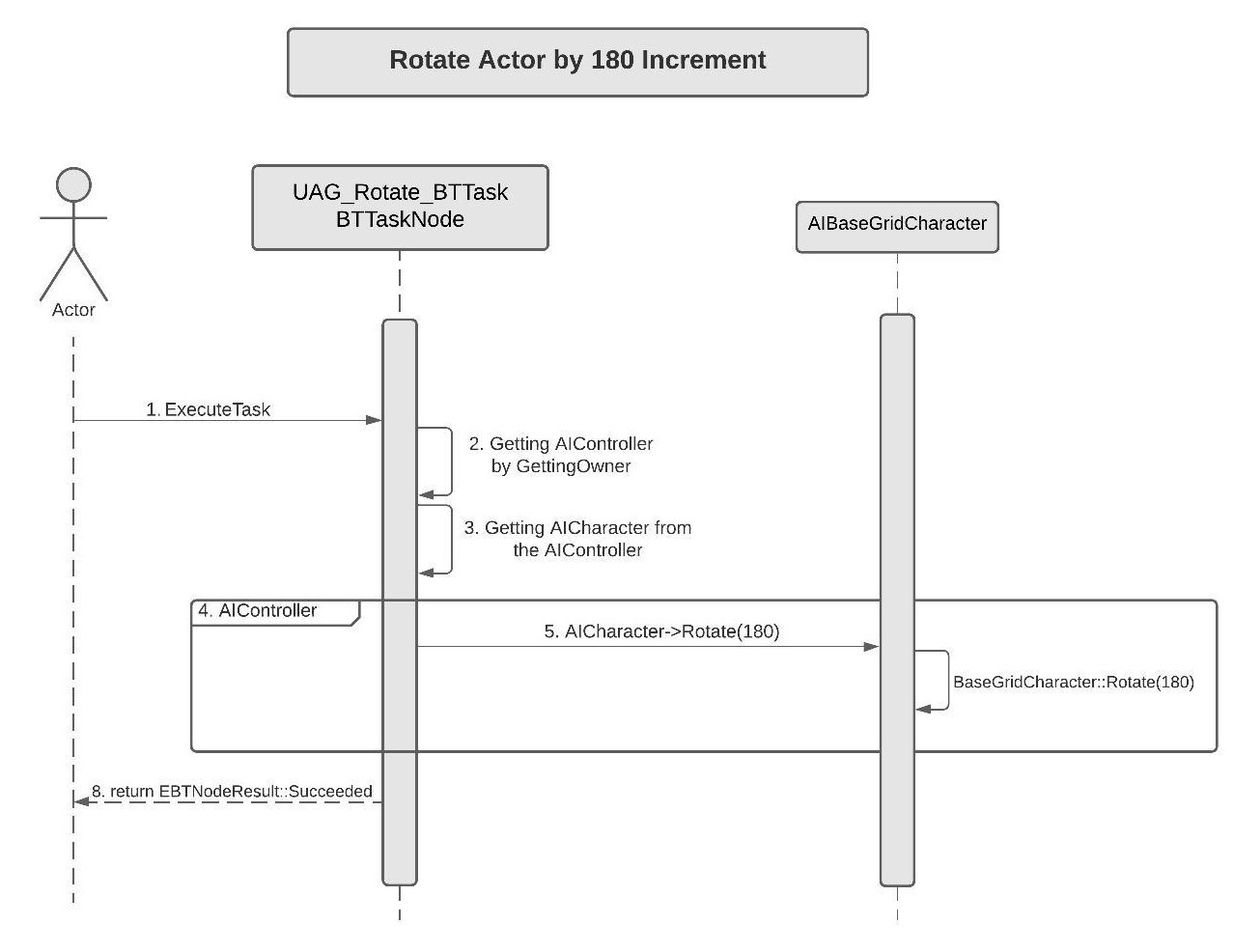
Note - First Sequence Diagrams cover the actual "IsPlayerINRange" function working.

****

**F. Rotate AI by 180**

This node rotates the AI by 180 in Yaw. It is mainly used for rotating AI. For this node gets the AICharacter from the Controller and calls the rotate function passing in 180. After this, AICharacter calls Rotate(180) function of the parent BaseGridCharacter.

Note - Rotate functions actual working in covered in BaseGridClasses Module.

****

**G. Move To Next Patrol Point**

This is node is mainly created for Patrol AI. On Execute, it gets the Owning AIController and then AICharacter from the AIController.

Then it checks if the AI has PatrolTileCoord Array filled with anything or not.

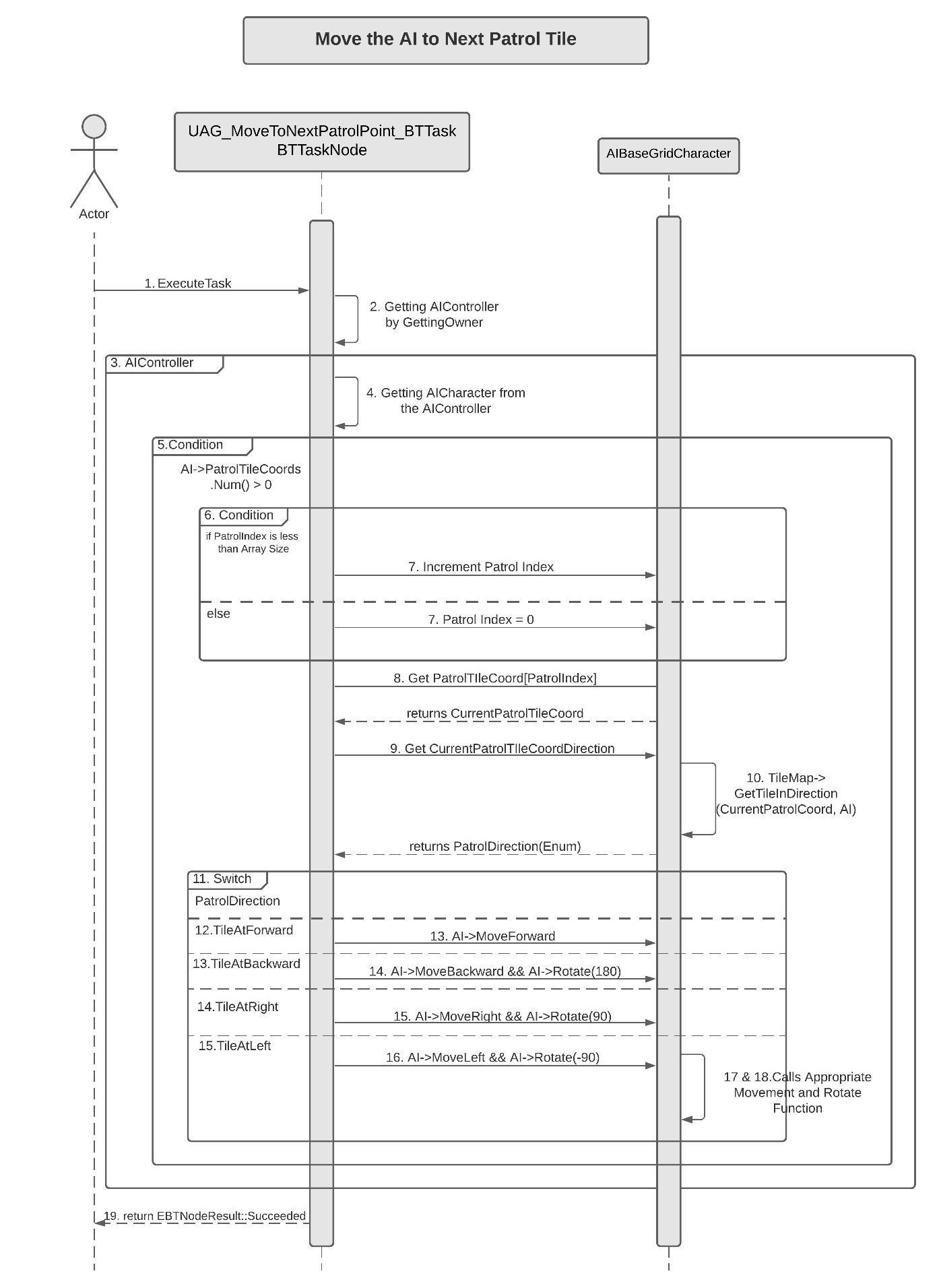
Note - For the setup of Patrolling AI, we can set some Patrol Tile Coord in the editor, on which we want the AI to patrol on. So if we set some points the PatrolTileCoord.Num() will be greater than zero for any patrol AI.

Then it increments the index member variable for the AI if it's less than the Array size or else sets it to zero. This creates a loop of PatrolTileCoords. With this, the node gets the CurrentPatrolTileCoord.

After this, it passes the PatrolTileCoord and the AICharacter to the TileMap, which, based on the actor's current tile and passed tile, returns the direction enum, that tells in which direction the tile is in.

Lastly, a switch statement based on the direction the TileMap returns.

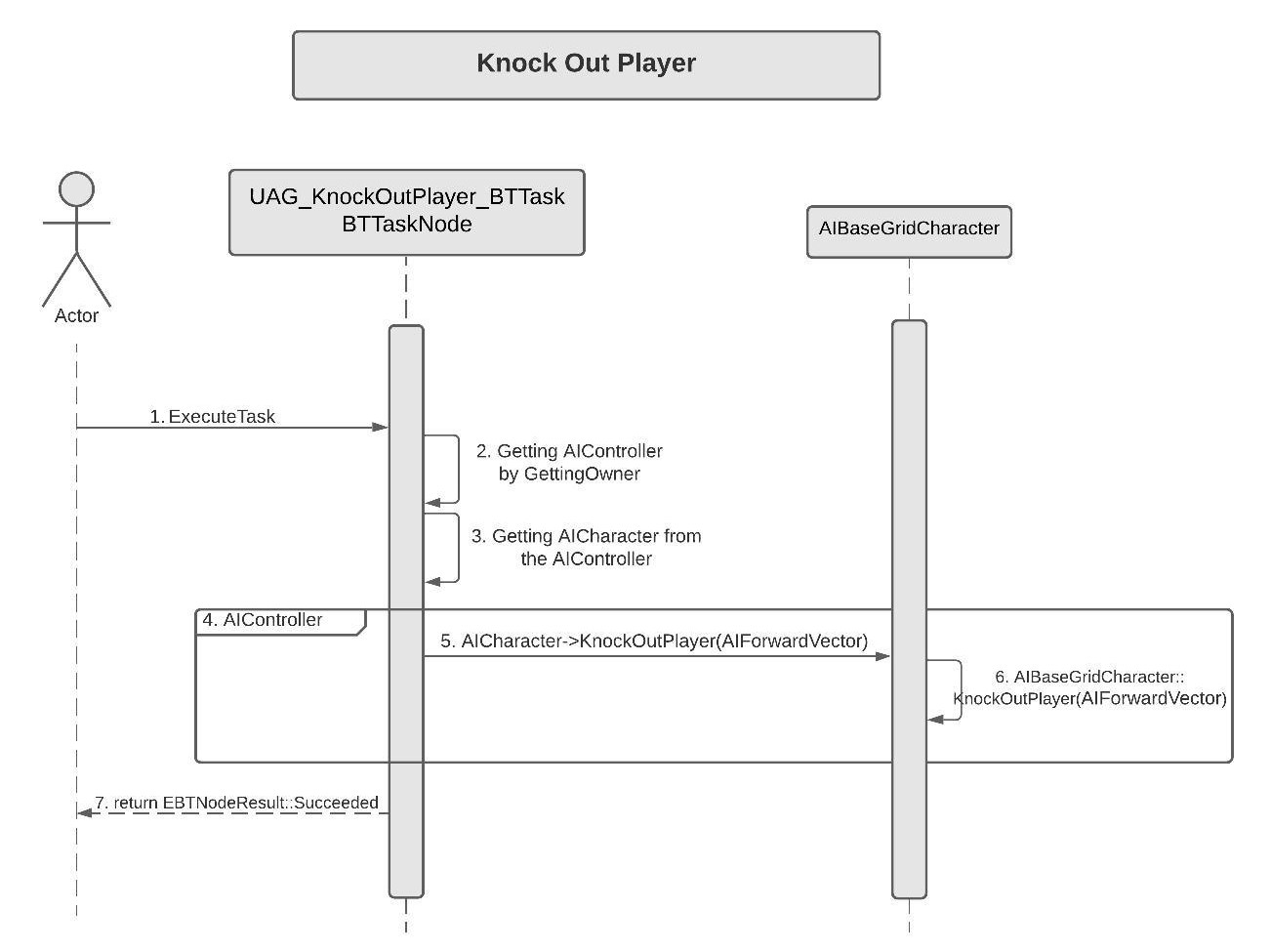
In the switch statement, it calls the respective move and rotate functions according to the direction. For Instance, if the direction is Forward, it will call MoveForward() on AI. For Backward, it will call MoveBackward() and Rotate by 180. For Right, it will call MoveRight() with Rotate(90) and for left MoveLeft() with Rotate(-90).

****

**H. Knock Out Player**

This node handles the player knockout. On Execute, the node gets the AICharacter from the owning controller and calls the KnockOutPlayer() function on AICharacter passing in the AICharacters forward vector. This internally calls the base AIBaseGridCharacter:knockOutPlayer and returns "Succeeded".

Note - The KnockOutPlayer function is covered in BaseGridClasses Module in more detail.



**Sequence View for Behaviour Trees**

This section will cover the construction of the actual Behaviour Tree used for the in-game enemies. All the tree has a selector attached to a root, which then leads to two sequences. The first sequence is "Attack Sequence" which is the same for all the AI. The second Sequence is the "Behaviour Sequence", which is responsible for the unique behaviour of all the enemies when idle.

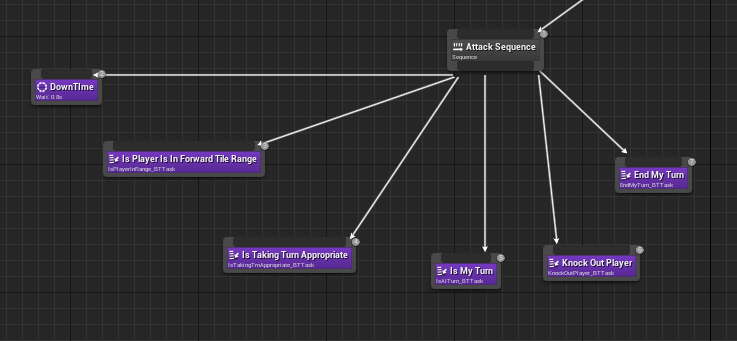
**I. Attack Sequence(Common Among All the enemies)**

The Sequence begins with a Downtime of 0.8s(Wait Node) which lets the player settle down a bit once his/her turn is over.

After that, the next node "Is Player Is In Forward Tile Range" checks if the player is standing in the forward Tile of the AI.

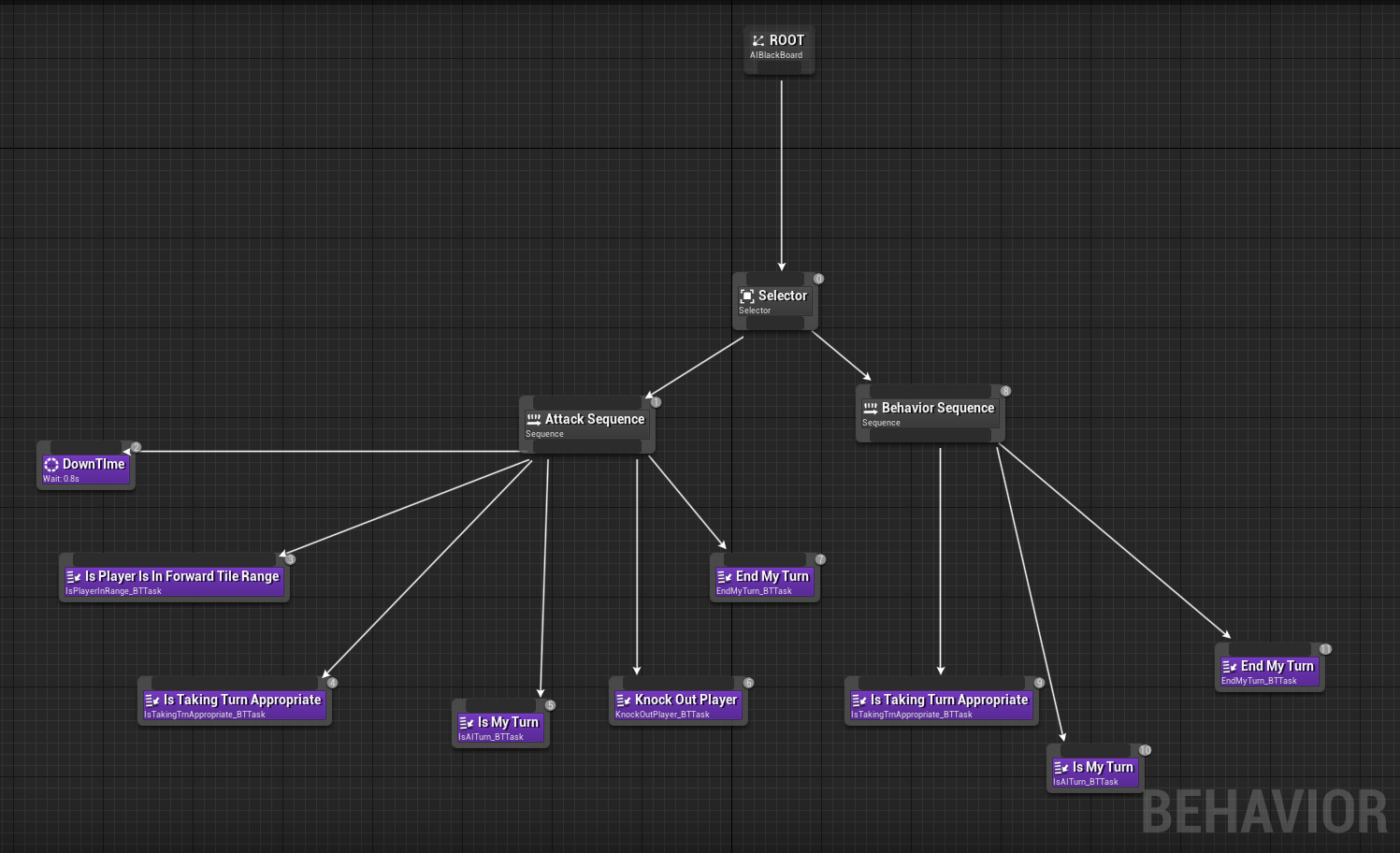
Next Nodes are "Is Moving Appropriate" and "IsMyTurn" in order which checks, if the AI is allowed to move or not.

If all the above nodes succeed "Knock Out Player" Node is called which knocks out the player and then AI can call "End My Turn" which stops him from moving once the player is down.

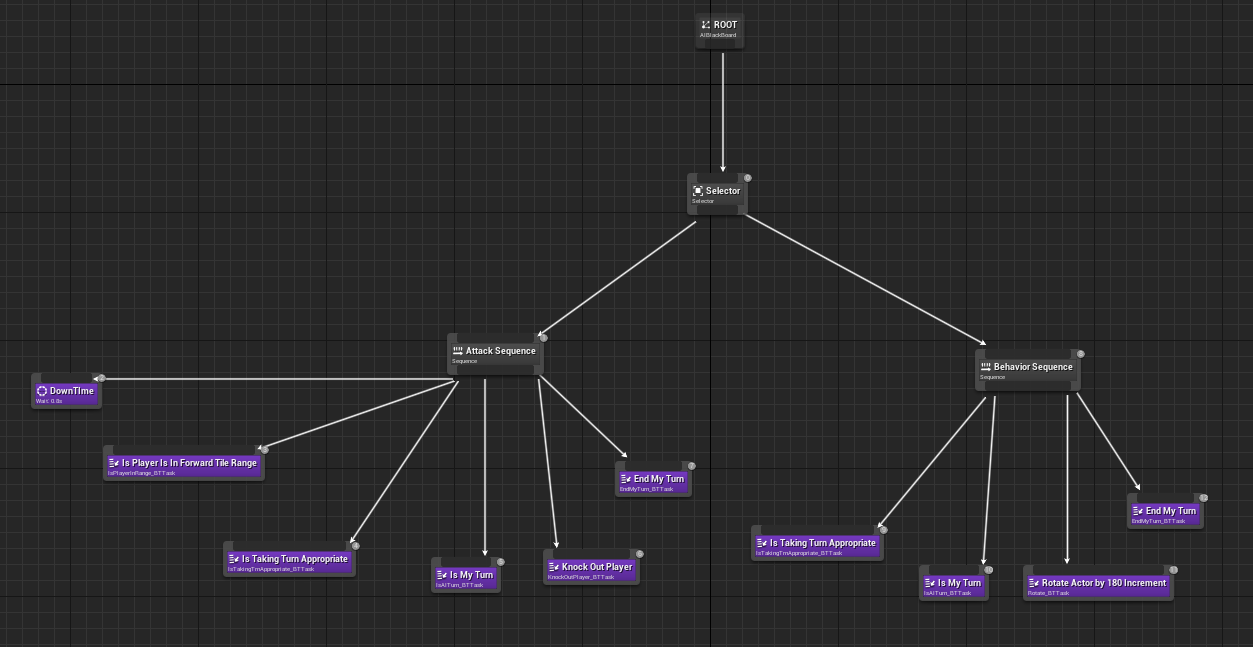


**J. Still AI Behaviour Tree**

The Still AI Tree starts with the Attack Sequence(explained above) followed by Behaviour Sequence. In terms of Behaviour, StillAI does nothing other than ending his turn. The First two nodes "Is Moving Appropriate" and "Is My Turn" check if the AI is supposed to move. After this, StillAI ends its turn right away by calling "End My Turn".

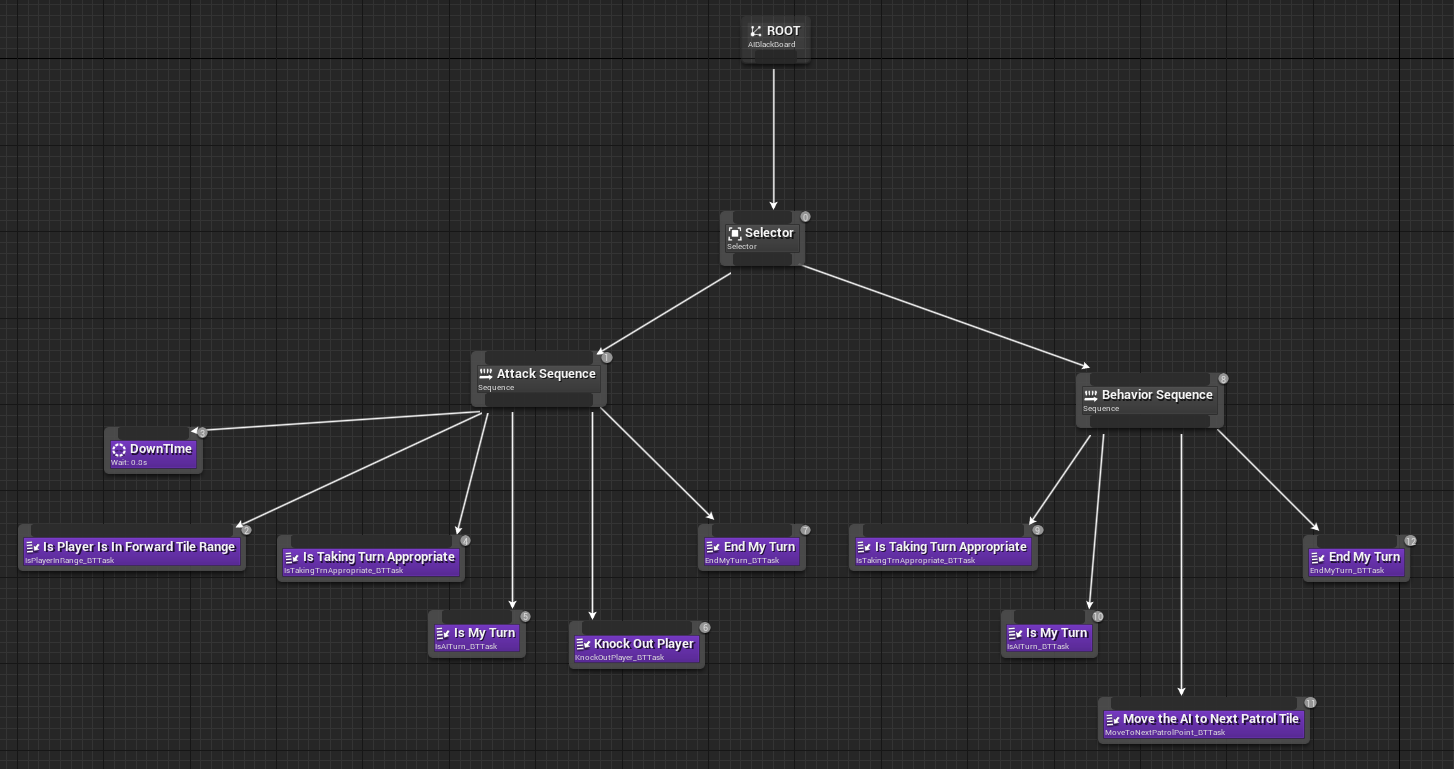
****

**K. Rotating AI Behaviour Tree**

****Once the Attack Sequence Fails, Rotating AI moves to Behaviour Sequence. First, it checks if he is allowed to move or not using "Is Moving Appropriate" and "Is My Turn" in order. Then it calls the "Rotate Actor by 180 Increment" node which rotates the AI by 180 Degree in Yaw. And finally, it calls "End My Turn" to finish its turn.

**L. Patrolling AI Behaviour Tree**

Following the same pattern, Patrolling AI checks if he can move by calling "Is Moving Appropriate" and "Is My Turn" in order. If Succeeded, it calls "Move the AI to Next Patrol Tile" which picks the next patrol tile for the AI and moves the AI. Lastly, it calls "End My Turn" wrapping up its Turn.

****

**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.

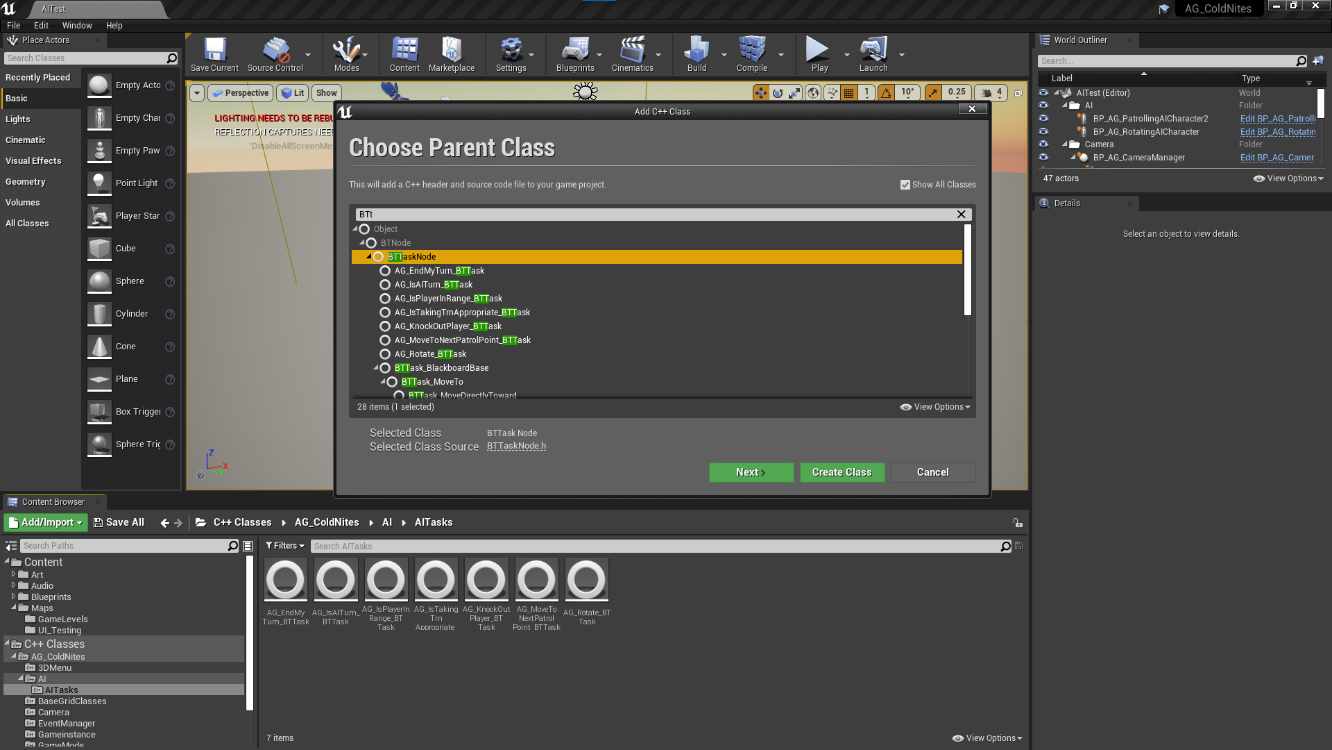
Note - For a detailed description of the BaseGrid classes refer to the previous Base Grid Classes Module.

**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)

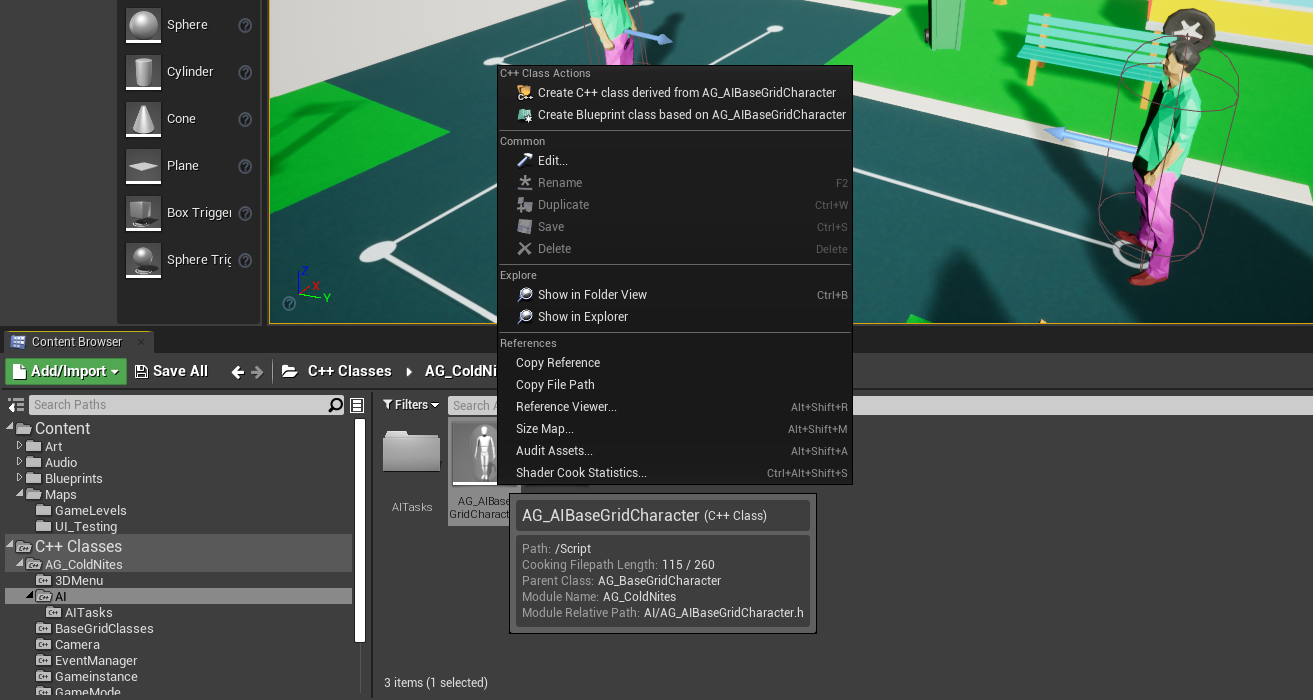
AAG\_AIBaseGridCharacter\* AICharacter = AIController->AICharacter;

**B. In Editor:**

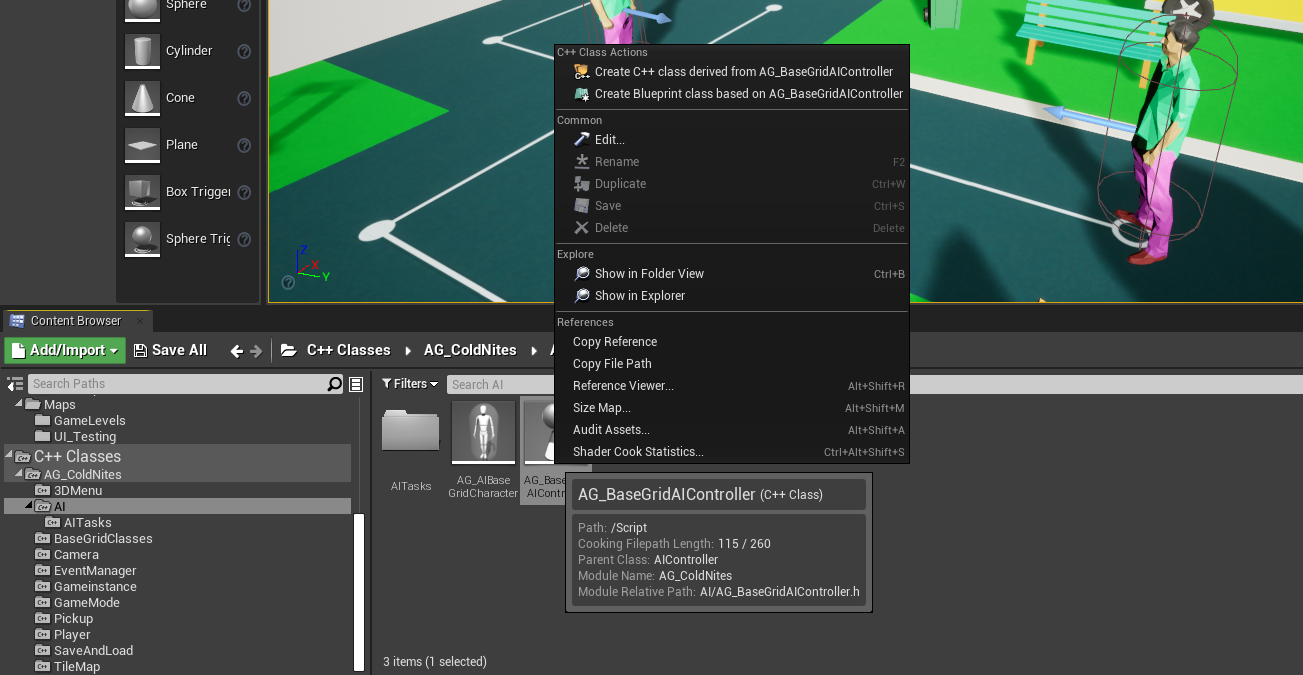
AI System setup for AI creation –

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

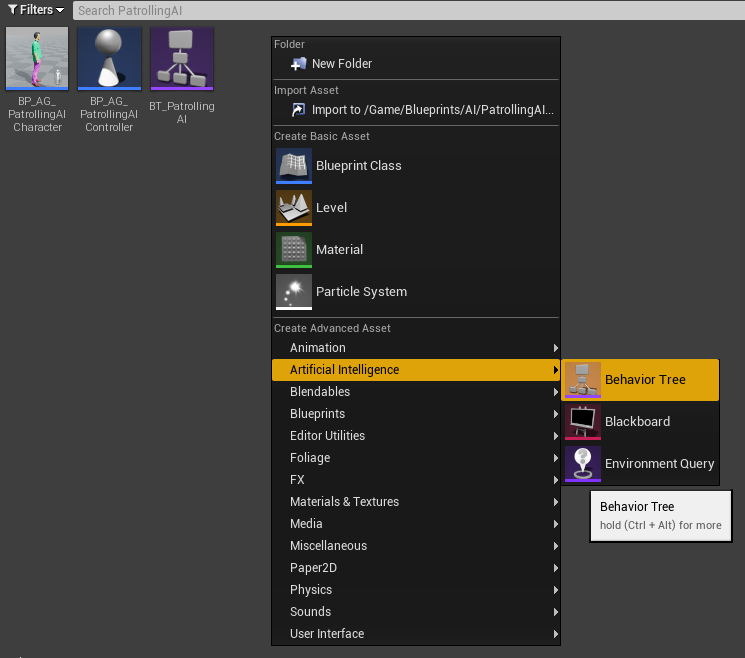
To create the AI Character, simply create a blueprint class based on AG\_AIBaseGridCharacter. Then rename the AI. Set any desired mesh.



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

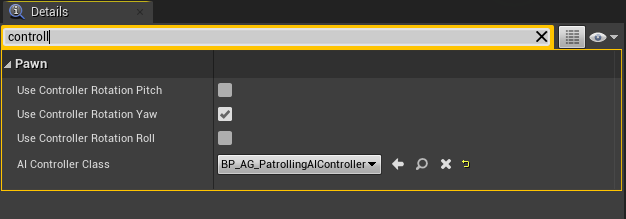


Finally, create a Behaviour tree for the AI and again rename it accordingly.

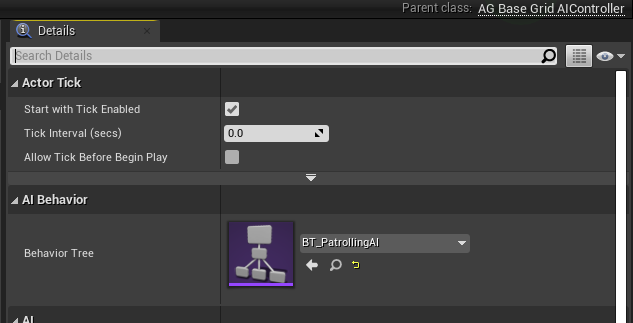


**Hooking up the AI Character, Controller and Behaviour Tree**

Firstly, hook up AI Character with the AI Controller.

****

Then, feed the Behaviour Tree to the Controller.

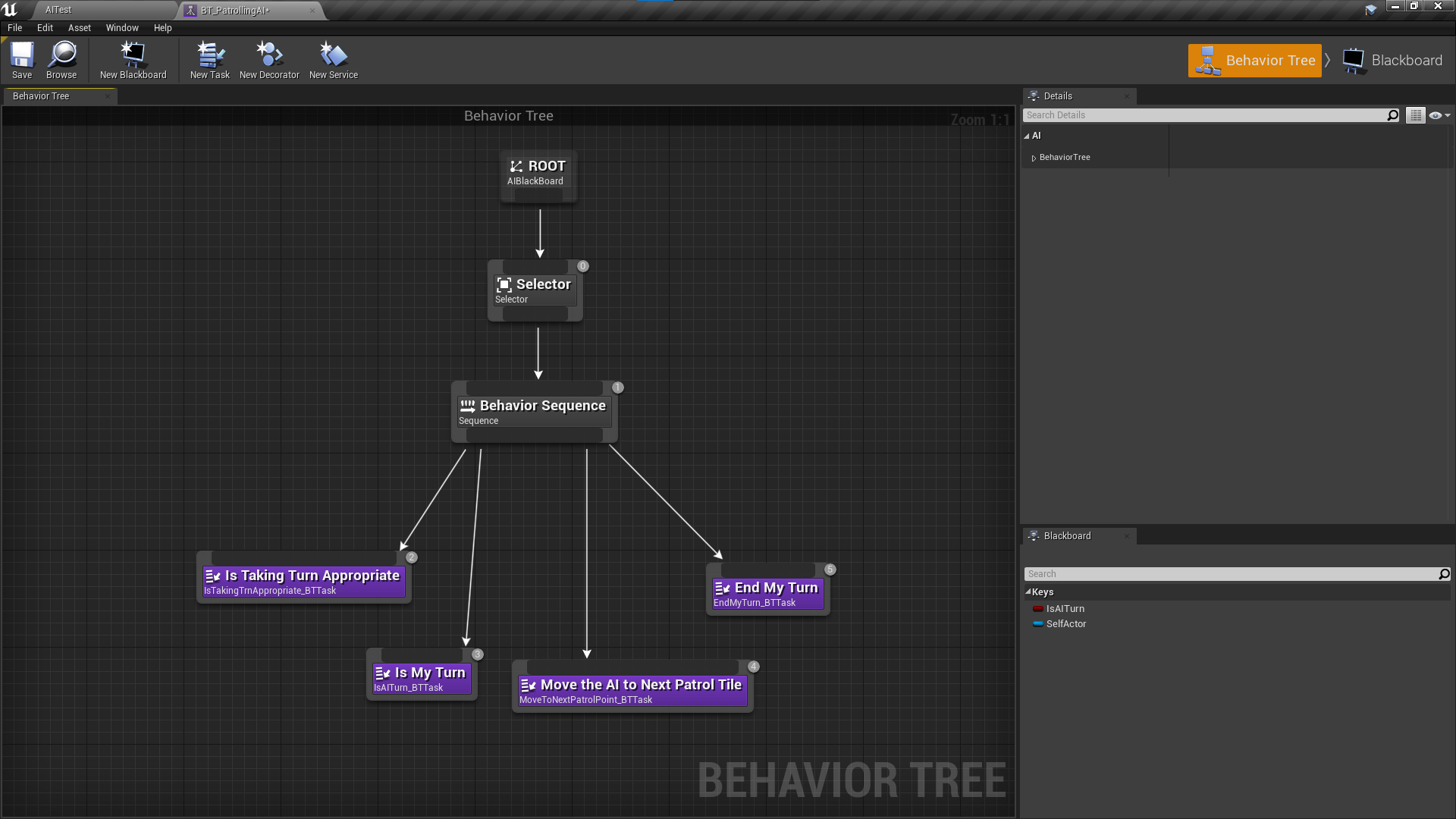
****

**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.