****

**COLD NITES**

Alpha-1

**TileMap Design Document**

Designed & Implemented by

Yash Chamria

**Table of Contents**

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

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

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

**C. Detailed Design of TileMap System**

1. **Process View 10**

**A. TileMap 'Regenerate Tiles' through the Editor**

**B.** **Actor Registering/UnRegistering to the TileMap**

**C.** **Actor Asking TileMap for Registered Actor**

**D.** **Actor Asking for desired TileCoord**

**E. Actor Asking any desired Tile's property**

1. **Use Case View and Practice 16**
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 TileMap. TileMap provides the base on which all the world actors are spawned and provides a grid-based environment for the game. It will also work as a catalyst for all the events on the Tile.

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

Below are interest points for the mentioned parties:

**Programmers** - TileMap will make the grid-based behaviour of the game easy to use and code around it.

\*It will also help the event base behaviour of the game (Event System will be implemented in Alpha 2).

**Artist** - TileMap properties make swapping or change the entire tile aesthetic at any movement a straightforward job from the editor itself.

**Level Designer** - One of the main targets of the TileMap is to achieve ease of level designing. It will provide nifty functionality of changing the size, style, type, properties, etc. for any given Tile.

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

* The design should minimize the complexity of the grid-based system.
* The design should allow the level designer to experiment on the fly and allow easy level creation.
* The design will also provide the base for all the future events taking place on the Tile.

**3. System Overview and Behaviour**

The TileMap actor is responsible for spawning the 'Tile' actors in a grid-based fashion in the world.

**Tile Actor -**

Tiles are the building block for the TileMap class. Tile inherits from an actor and is spawned by the TileMap.

Individual tiles hold a number of properties (Walkability, Destructibility, WinTile, etc.), their coordinates based on the 2D grid system, their world position, and most importantly an array of actors currently present on the tile. For this to work, all the actors need to register and unregister themselves to the Tilemap, on a particular Tile.

**TileMap Actor -**

TileMap will act as an intermediary manager between the world actors and the Tiles, which will provide ease of coding.

TileMap stores an array of Tile actors spawned in the world. It keeps track of all the TileCoordinates and assigns those to the individual Tiles.

With TileMap, any actor can get its coordinates on the grid, its neighbouring Tile or any desired Tile. This makes spawning and grid movement possible.

\*TileMap can also return the array of current actors on the particular Tile making the event base system possible for the game. (Alpha 2)

TileMap is also responsible to make level designing simple.

A Designer can change the TileMap Size from 5x5 to 15x10 on the fly without diving into the code.

Also, a designer can toggle TileProperties on or off from the editor itself. Tile also comes with some aesthetic customization settings.

**4. Logical View**

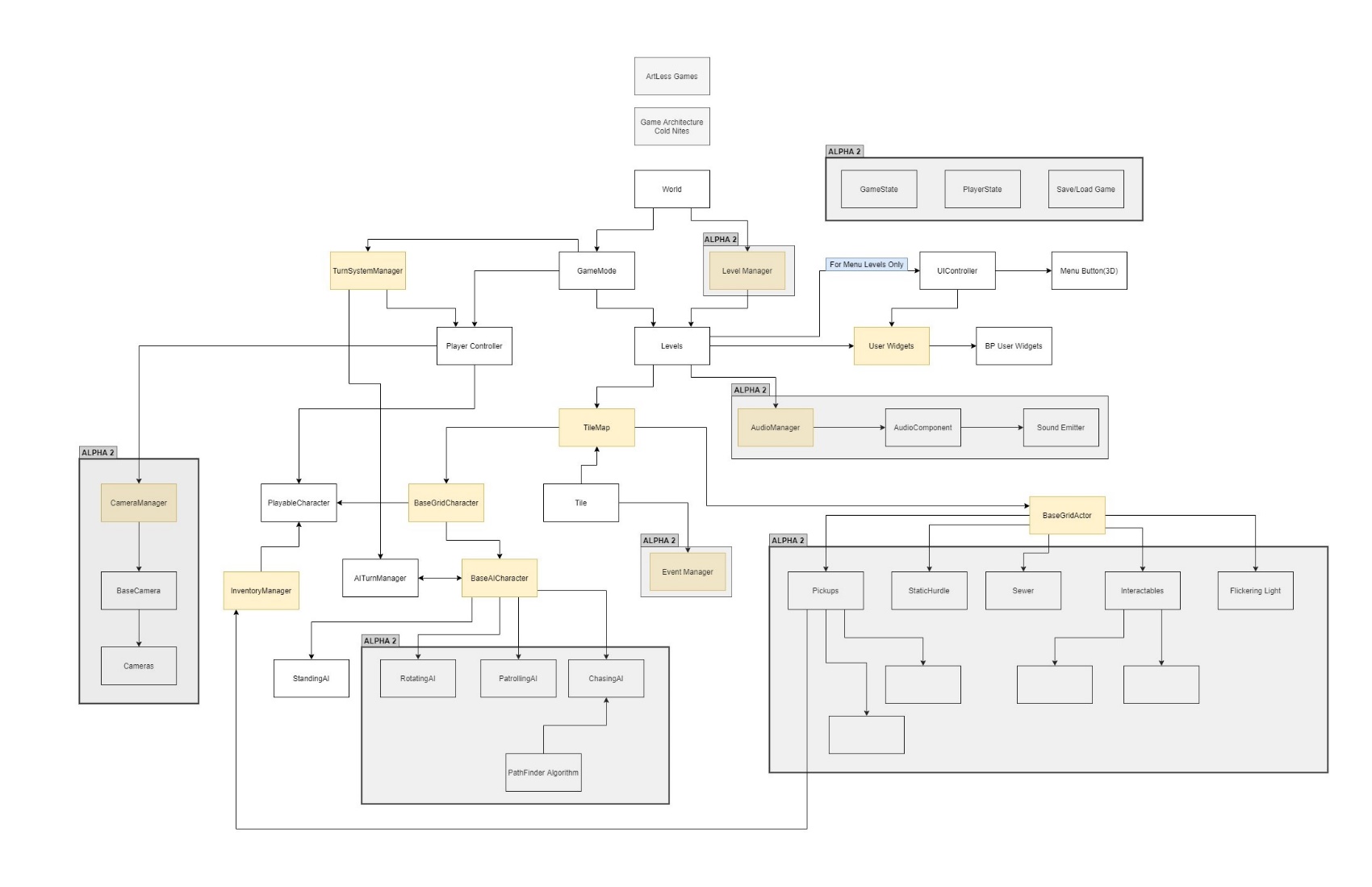
The logical view describes the high-level architecture for the entire game from all the core classes to high-level relations and interactions between them with a flow chart making it easy to read and understand.

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

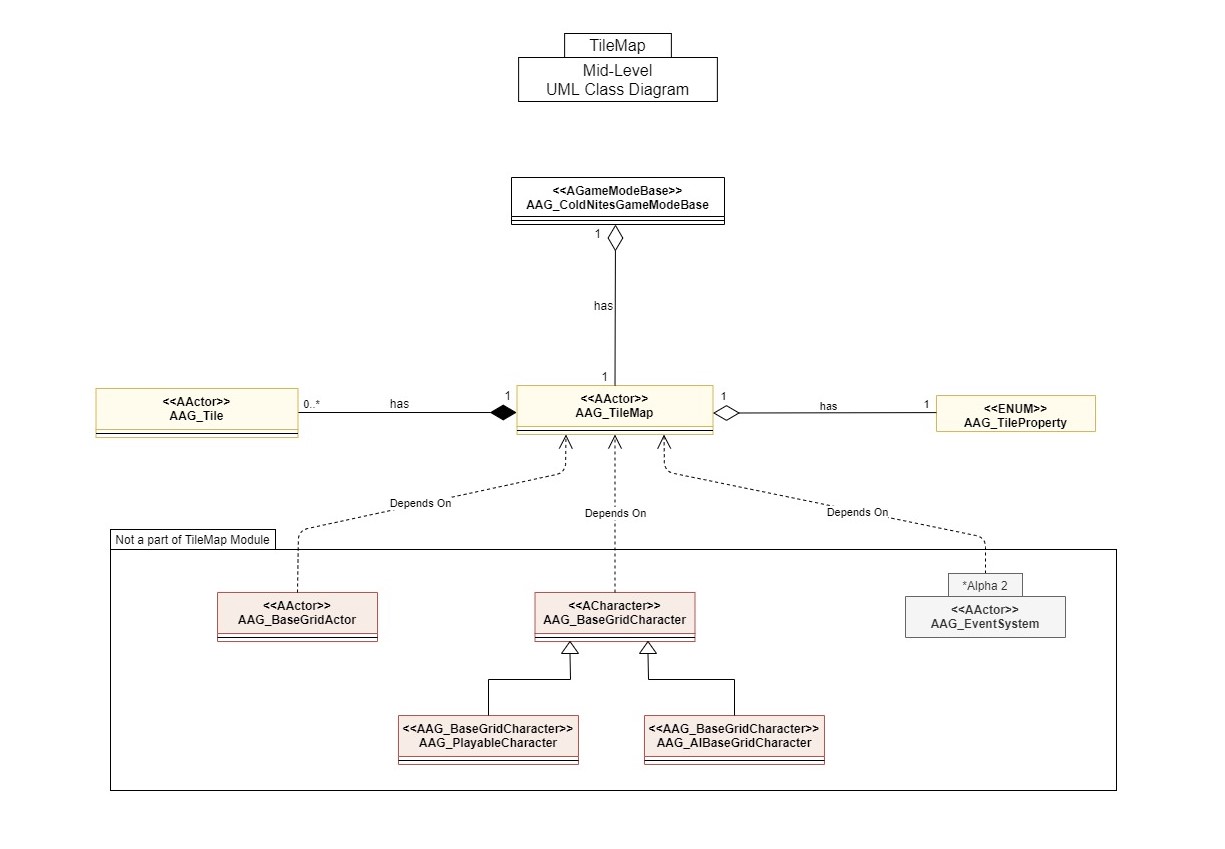
**A. High-Level Design Architecture of the Entire System**

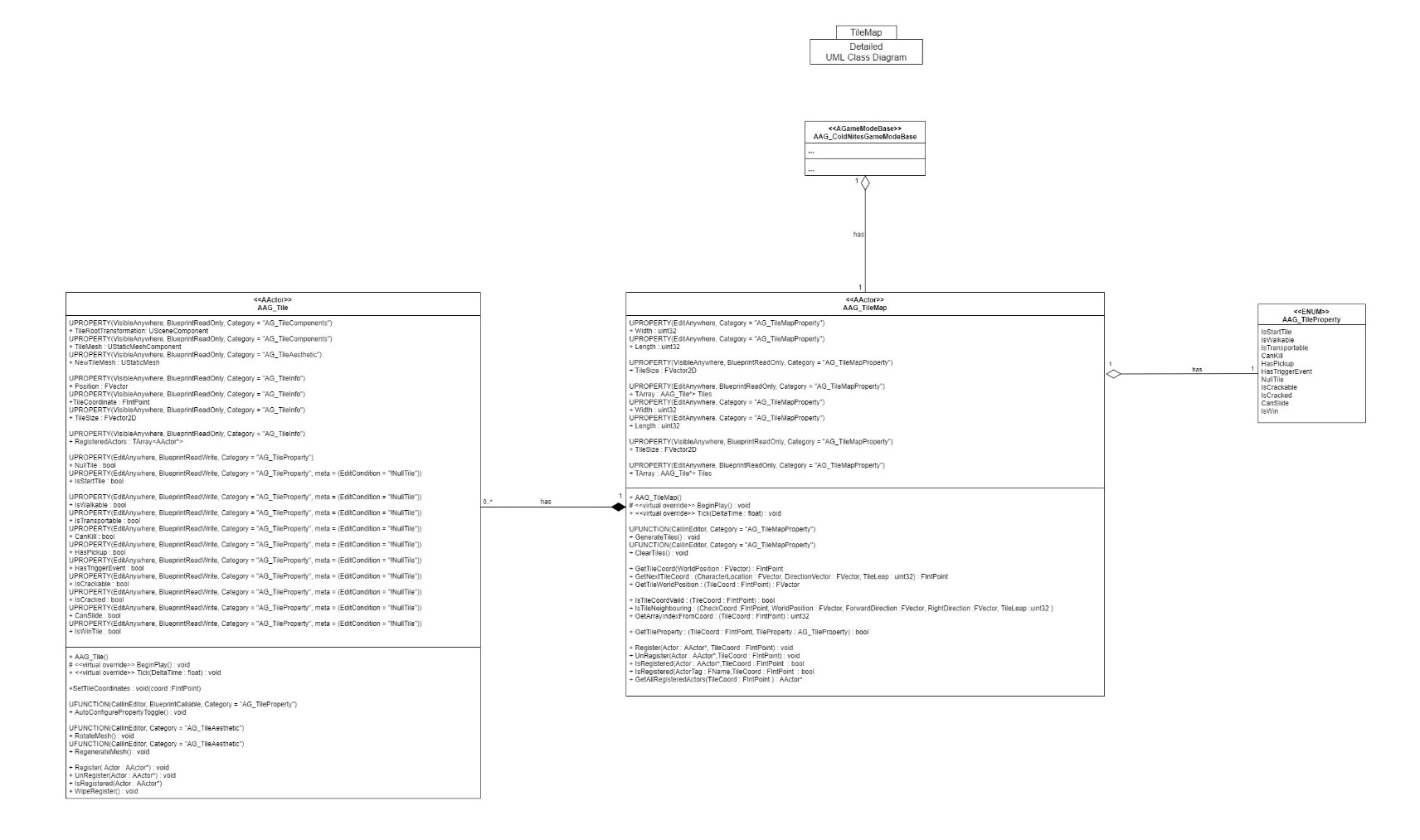
The primary features for the Alpha 1 release:

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. **Player** - Player is a controllable character that inherits from BaseGridCharacter, which takes user inputs to perform appropriate moves.
4. **Inventory System** - Inventory stores the item for the corresponding actor and will allow the player easy access to any collectible throughout the game.
5. **User Interface** - The User Interface will be responsible for Main Menu and any in-game HUD (or User Widget) with which the player can interact.



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



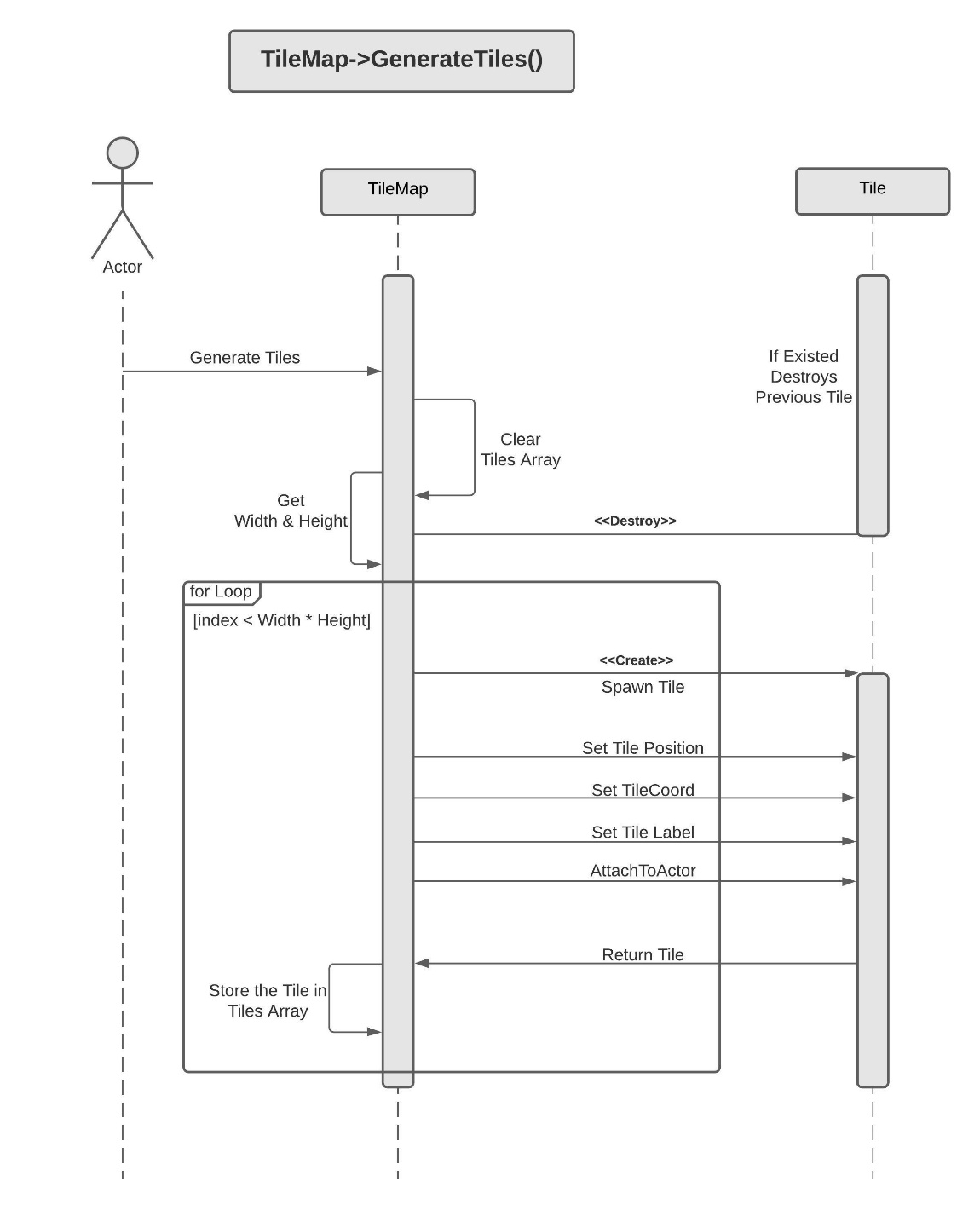
**C. Detailed Design of TileMap System**

**5. Process View**

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

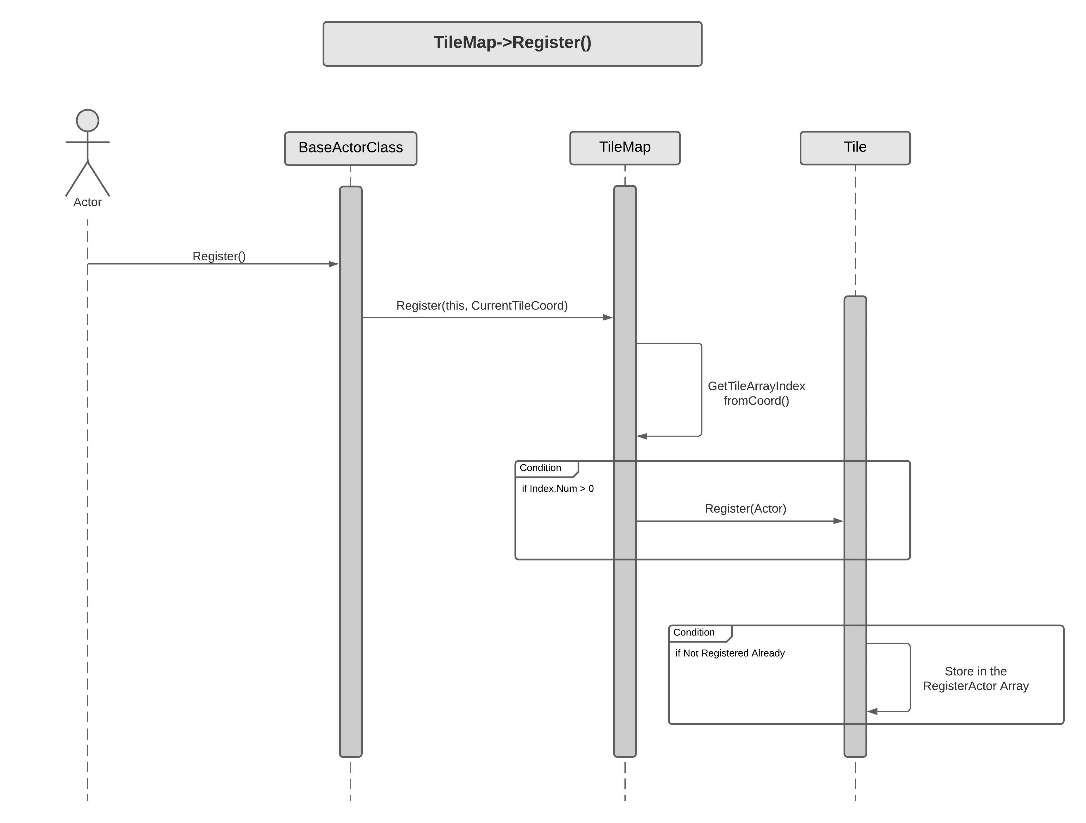
**TileMap 'Regenerate Tiles' through the Editor**

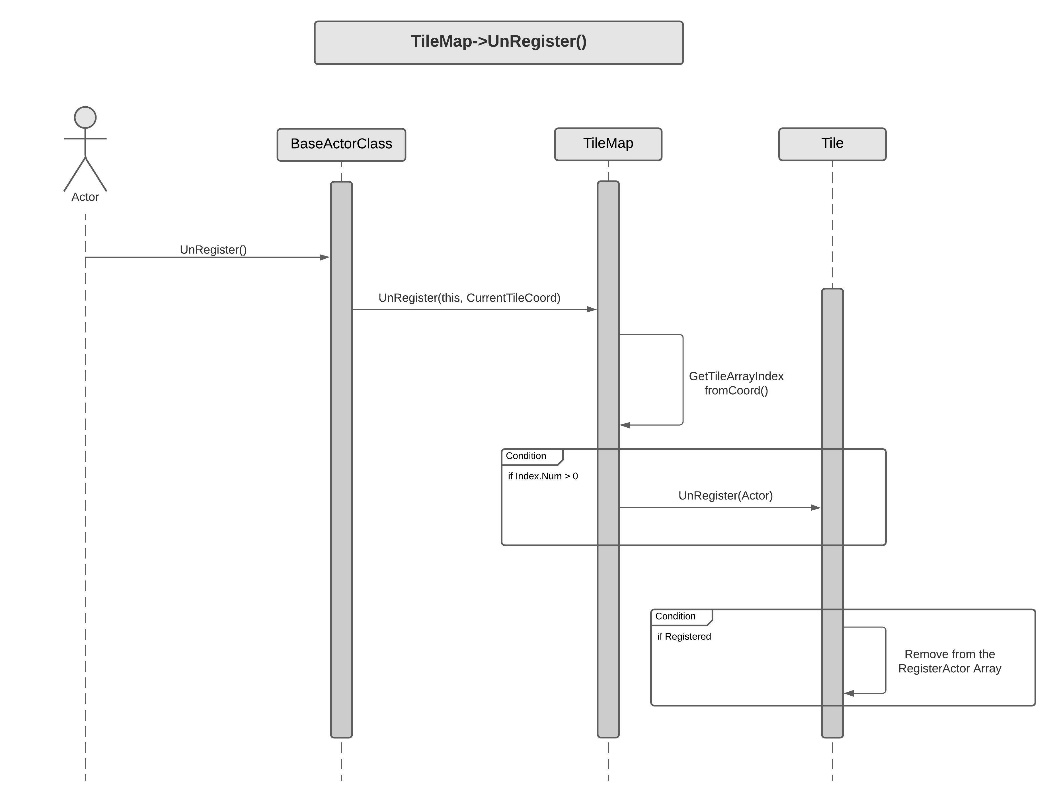
One can specify TileMap Width and Height and GenerateTiles() will clear any of the previous Tiles array, if existed, and will spawn new ones. It will also supply the location, coordinates and name for the Spawned Tile. Then it will store the new Tiles in the array and attach those to the TileMap.



**Actor Registering/UnRegistering to the TileMap (at Particular Tile)**

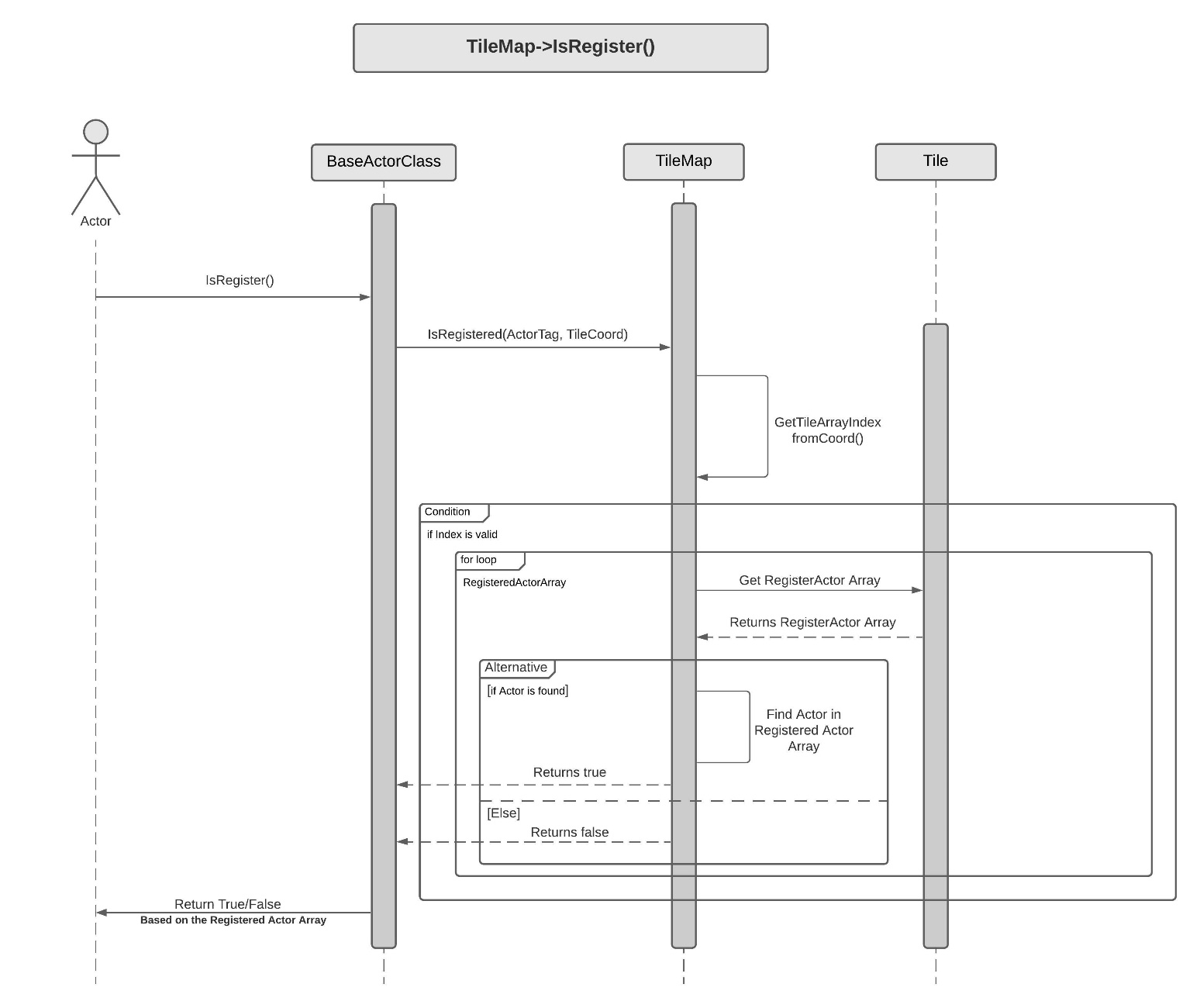
An actor can register or unregister to the TileMap by calling Register/UnRegister() and passing the address(this) and the desired TileCoord. This will add/remove the actor from the RegisteredActor for the corresponding tile.





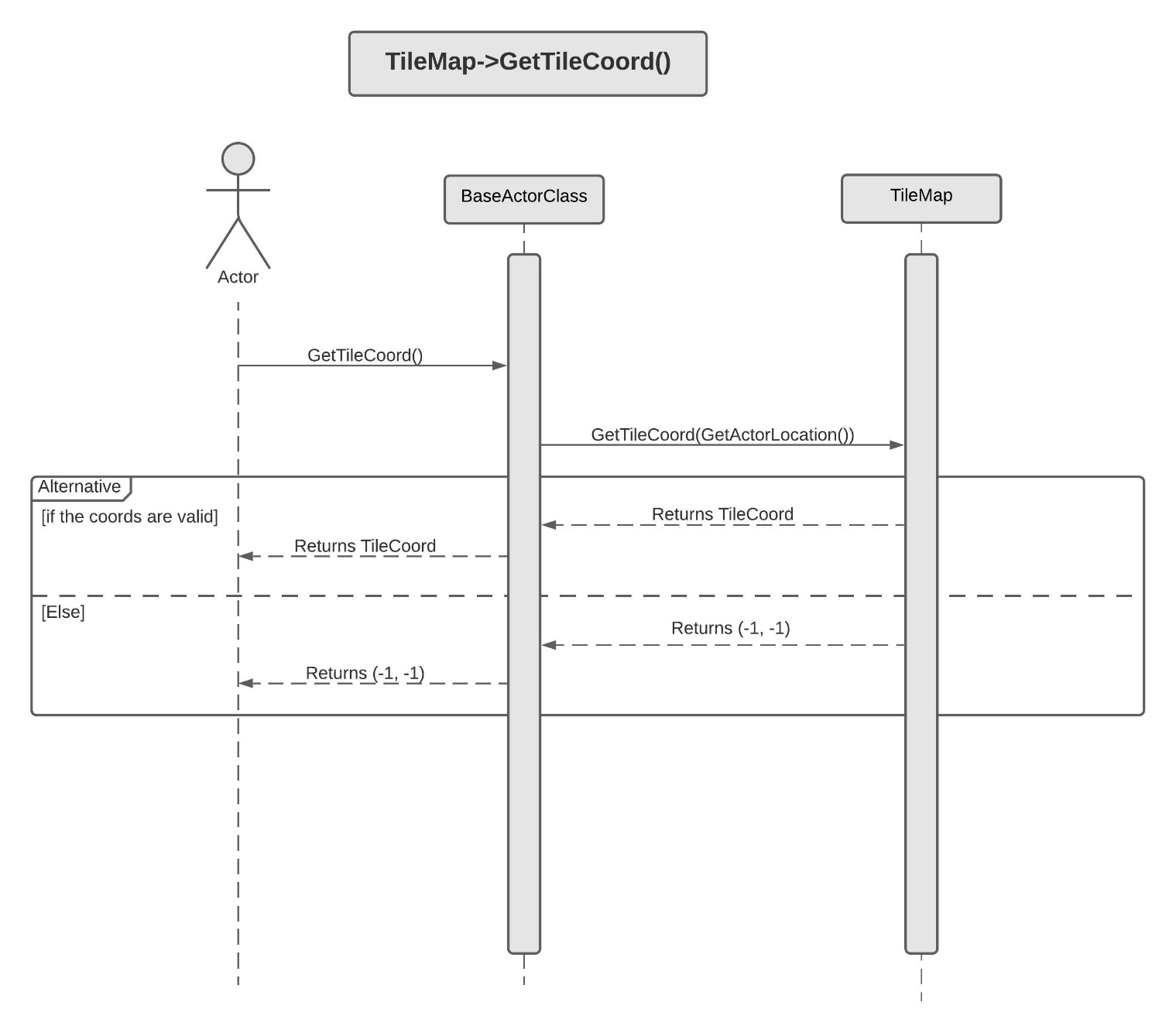
**Actor Asking TileMap for Registered Actor**

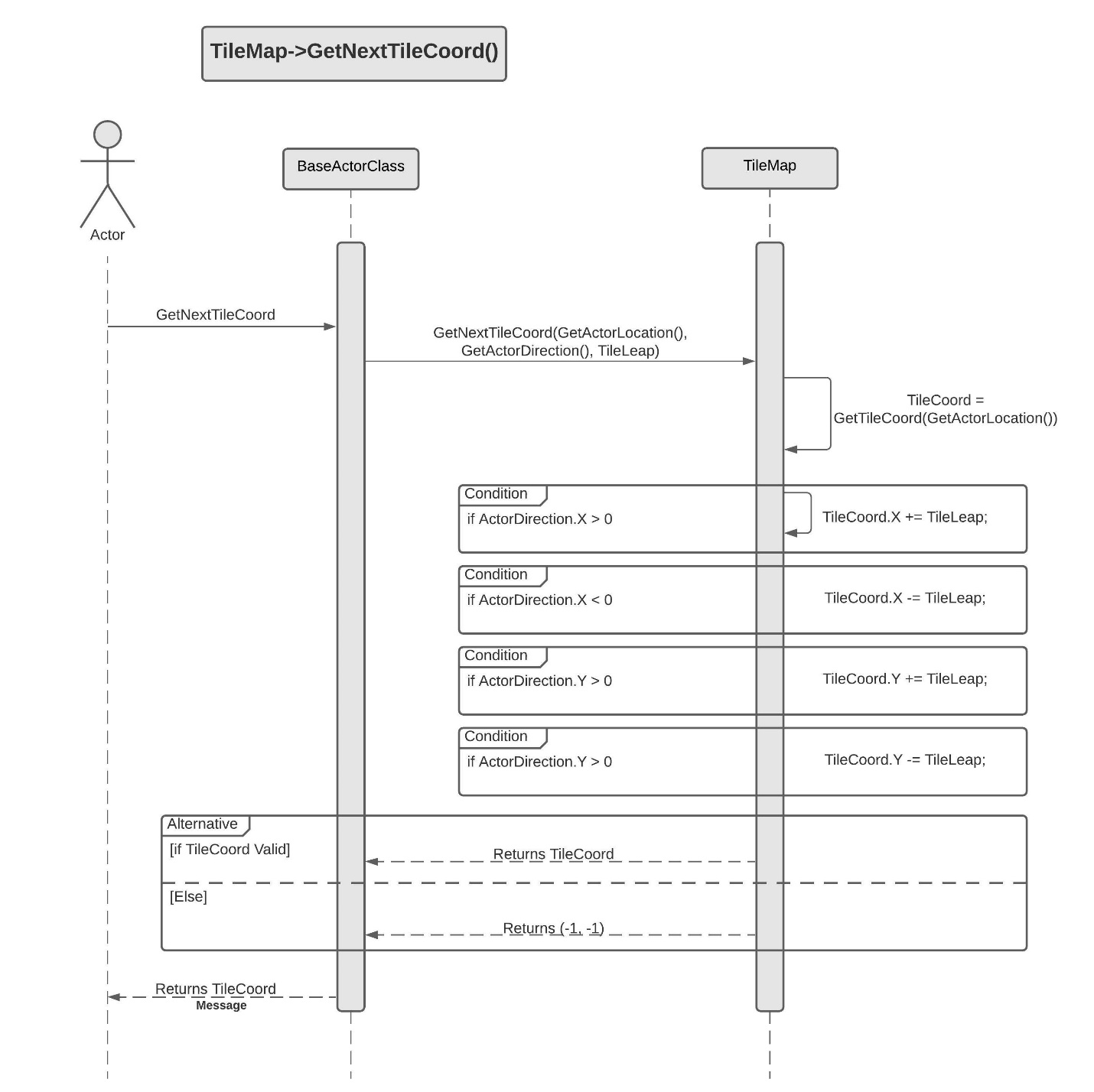
An actor can ask the TileMap which actors are present on a certain by calling IsActorRegister() and passing Actor or ActorTag and TileCoordinates. Internally TileMap will check the coordinate validity and will try to find the actor at a calculated index. If the given actor is found it will return true, else false.



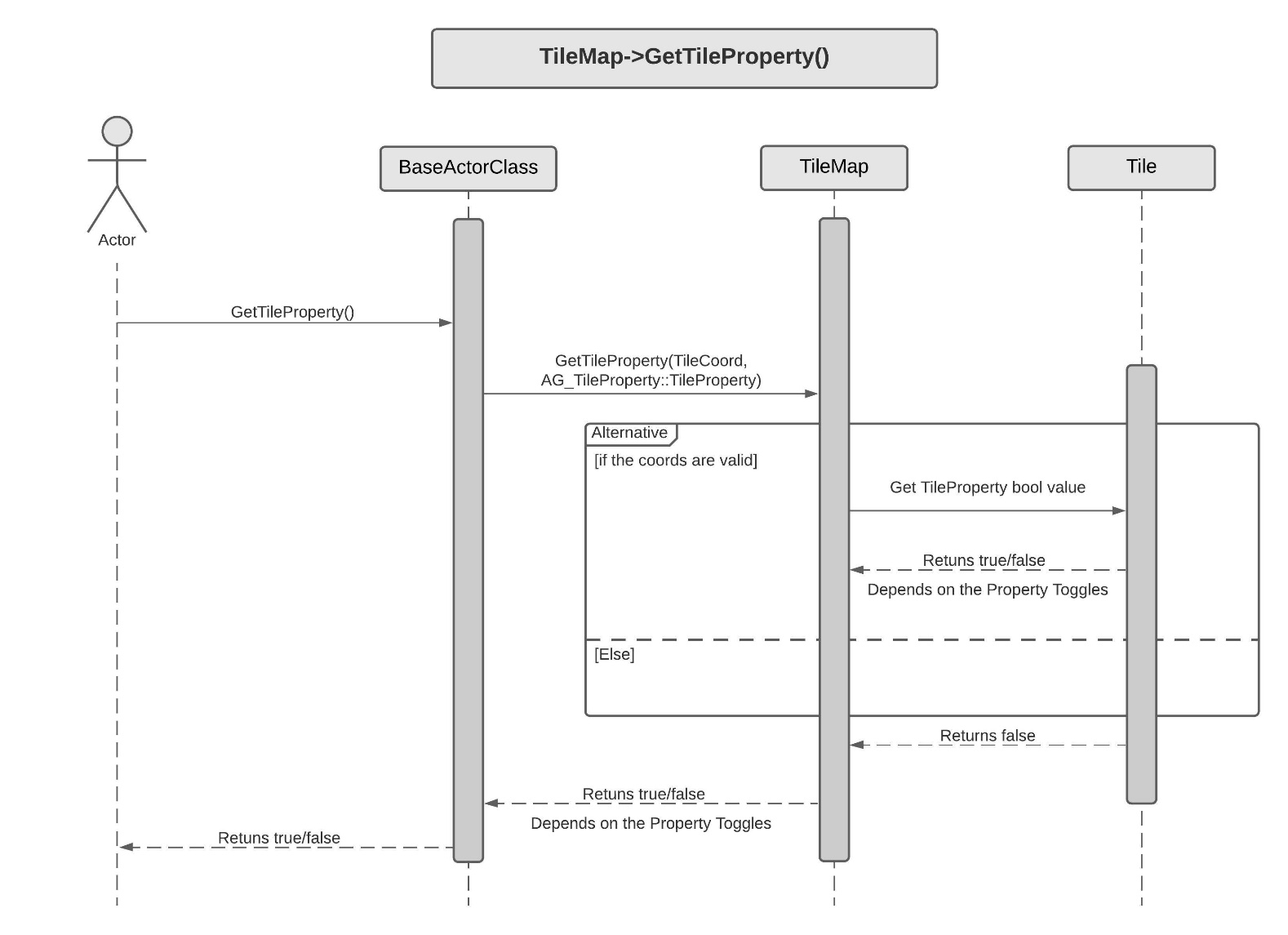
**Actor Asking for desired TileCoord**

An actor can get his TileCoord or NextTileCoord by calling GetTileCoord() or GetNextTileCoord() respectively and passing its location, Direction and TileLeap(how far is the tile, the actor wants? default is 1). TileCoord math is handled by TileMap internally so it can return the coordinates from World location math.





**Actor Asking any desired Tile's property**

****An actor can get any TileProperty by calling GetTileProperty() and passing the TileCoord and TileProperty Enum value for the respective property. Internally, TileMap will calculate the Tile index in the Tiles array and then will check if that Tile has the desired property toggled on or off.

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

The use case will focus on showing the uses of TileMap at different stages in the game and will explain its application so that the TileMap module can act as a guide/reference for someone not quite familiar with the TileMap codebase.

**A. In Code:**

**Actor Classes interaction with TileMap -**

* In the code, any of the classes can get their Tile Coordinates or any desired horizontal or vertical Tile Coordinates with a simple line of code –

*FIntPoint CurrentTileCoord = TileMap->GetTileCoord(GetActorLocation());*

*FIntPoint NextTileCoord = TileMap->GetNextTileCoord(GetActorLocation(), ActorDirectionVector, TileLeap);*

All the world coordinates to grid coordinates and visa versa are handled by the Tilemap.

* Also, Tile Properties are made quite accessible with just one function call and passing the TileProperty Enum –

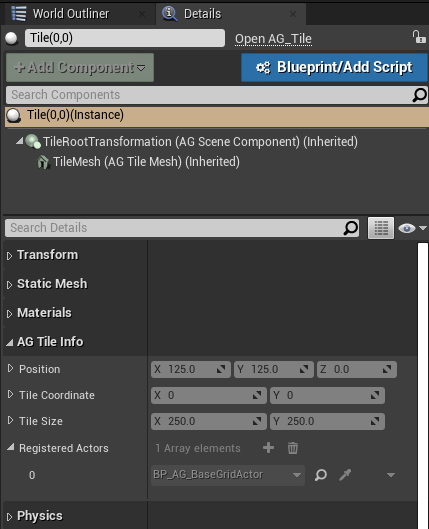
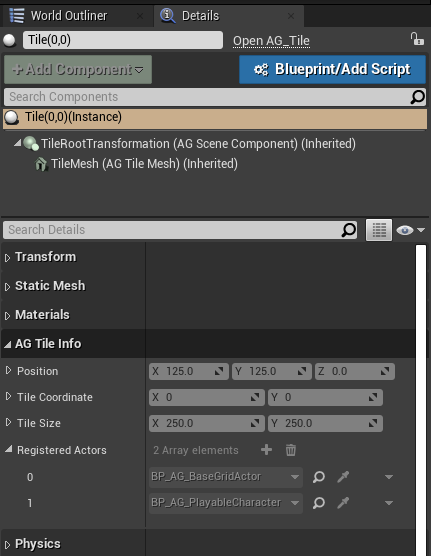
*bool IsThisTileWalkable = TileMap->GetTileProperty(ThisTileCoord, AG\_TileProperty::IsWalkable);*

*bool IsThisWinTile = TileMap->GetTileProperty(ThisTileCoord, AG\_TileProperty::IsWinTile);*

* All the interactable world actors will register to the Current Tile through the TileMap. And will unregister if they leave the Tile or get destroyed -

*TileMap->UnRegister(this, CurrentTileCoord);*

*TileMap->Register(this, NextTileCoord);*



* This allows us to ask TileMap which actors are present on the Tile –

*bool IsThisActorRegister = TileMap->IsRegistered(ActorTag, TileCoord);*

With this, we can instigate various check and supply event accordingly,

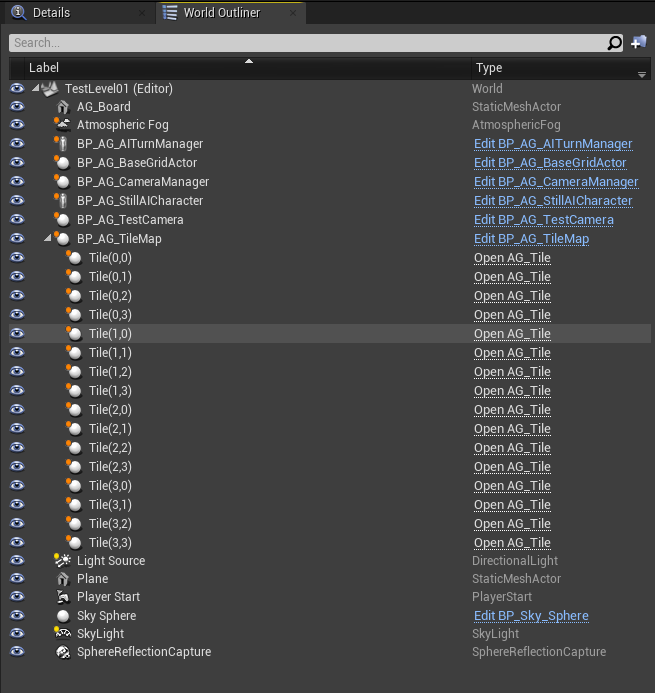
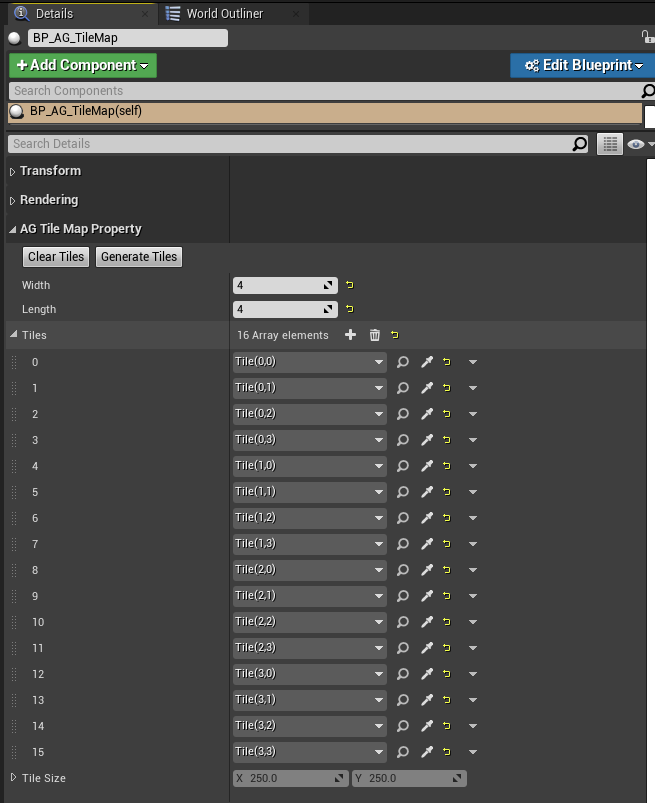
For instance, to see if Player and AI are on the same Tile, or if Player and Pickup are on the same Tile.

**B. In Editor:**

All the position, rotation and scaling are disabled for the TileMap and the Tiles.

**TileMap Editor Features -**

In the editor, Level Designer can specify TileMap 'Width' and 'Height' and then the 'Generate Tiles' will create a desired grid for the level.



**Tile Editor Features -**

Level Designer can quickly toggle on or off property for any Spawned Tile. Also, one can easily swap the mesh or rotate it.

