****

**COLD NITES**

Alpha-2

**Level Management System Design Document**

Designed & Implemented by

Vrund Soni

**Change Log**

**Version** – 0.2

**Modifier** – Vrund Soni

**Date** – 12th April 2021

**Description** – Started on the document and completed introduction.

**Version** – 0.4

**Modifier** – Vrund Soni

**Date** – 13th April 2021

**Description** – Completed Design Goals.

**Version** – 0.6

**Modifier** – Vrund Soni

**Date** – 15th April 2021

**Description** – Completed System Overview and behavior and added UML Class diagram and High level architecture.

**Version** – 0.8

**Modifier** – Vrund Soni

**Date** – 16th April 2021

**Description** – Added Process Views.

**Version** – 0.9

**Modifier** – Vrund Soni

**Date** – 16th April 2021

**Description** Added Use Case View and practice.

**Table of Contents**

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

**A. ColdNitesGameInstance**

**B. ColdNitesGameSaveGame**

1. **Logical View 7**

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

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

**C. Detailed Design of Level Management System**

1. **Process View 11**

**A.** **Opening Next Level**

**B. Opening Same Level**

**C. Opening Selected Level from the LevelSelectionScreen**

**D. Notifying the game instance of level completion**

**E. Saving and Loading completed levels**

1. **Use Case View and Practice 17**
2. **In Code**
3. **In Editor**

**1. Introduction**

ColdNites 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 Level Management System.

This document will describe the architecture and design choices that make the Level Management System’s 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** – Level Management System allows the programmers to easily code around multiple levels and how to handle switching of levels in the game.

**Level Designer** – The Level Management System allows the level designers to easily work on as many levels as they want to by making small and effortless changes to the blueprints.

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

* The design should reduce the complexity of switching between levels throughout the game.
* The design should allow other programmers and designers to add and work on any number of levels with less or no effort of changing the existing code or blueprints.
* The design should allow the programmers to have easy control over saving and loading the level specific information.

**3. System Overview and Behaviour**

The Level Management System is made up of two classes:

1. ColdNitesGameInstance
2. ColdNitesSaveGame

**ColdNitesGameInstance -**

This class will handle 3 main processes:

1. Allowing the game to switch to next level on current level completion.
2. Making sure the game does not open levels that are yet not unlocked.
3. Allowing to replay the level.

This class keeps track of number of levels completed by storing the completed level number in an array of integers.

**ColdNitesSaveGame -**

When a level is completed by the player the GameInstance will save the game and store the level completion data in an instance of UAG\_ColdNitesSaveGame. On game start this class will give back the saved data to the GameInstance to load the game.

**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 Level Management System, using a UML Class Diagram.

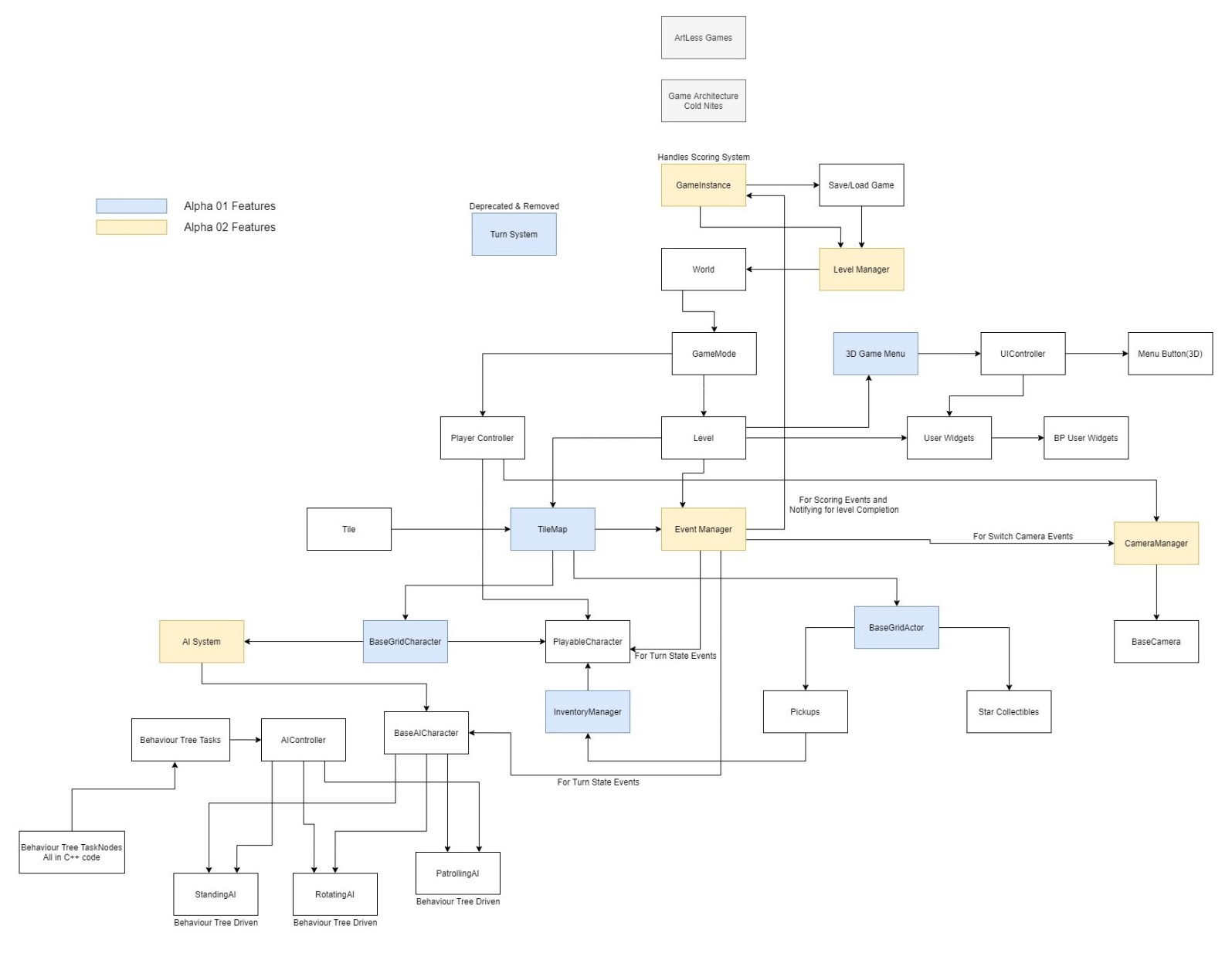
**A. High-Level Design Architecture of the Entire System**

The primary features for the Alpha 1 release:

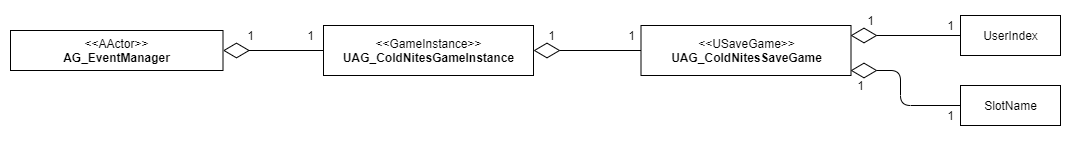
1. **TileMap** - TileMap provides the grid-based behaviour for the game and will facilitate the event system, based on the actor present on the Tiles.
2. **Turn-Based System** - This provides the turn-based aspect for the game. It is responsible for maintaining the turn order for all the world elements(actors)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

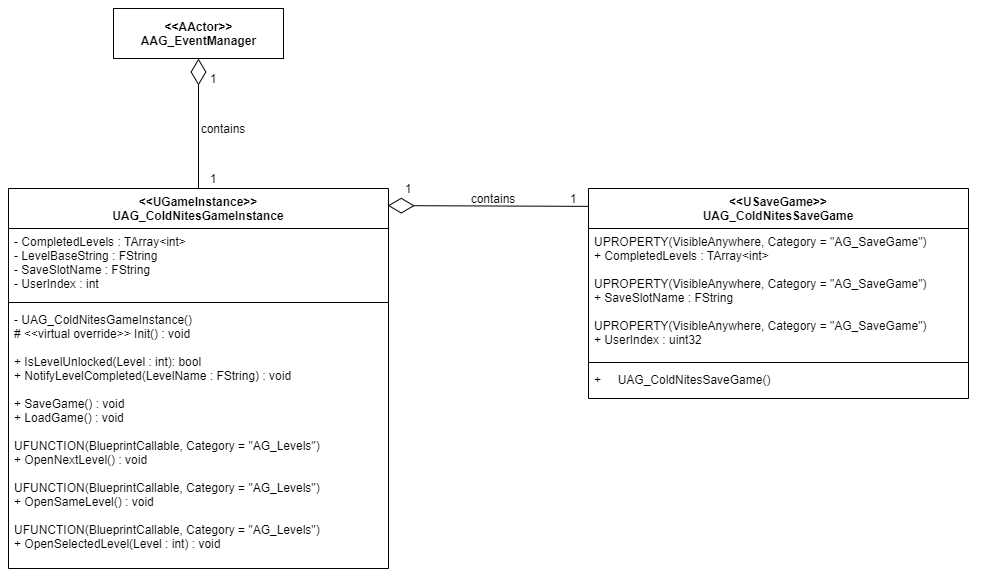
1. **AI System** - AI System is responsible for various enemy types in the game. The AI System allows the ease of creation using the single BaseAI class and Behaviour Tree Task Nodes.
2. **Event System** – Event Manager handles and centralizes level-specific events such as Turn System Events, Camera 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 of levels in the game and saving and loading 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 Level Management System**



**C. Detailed Design of Level Management System**

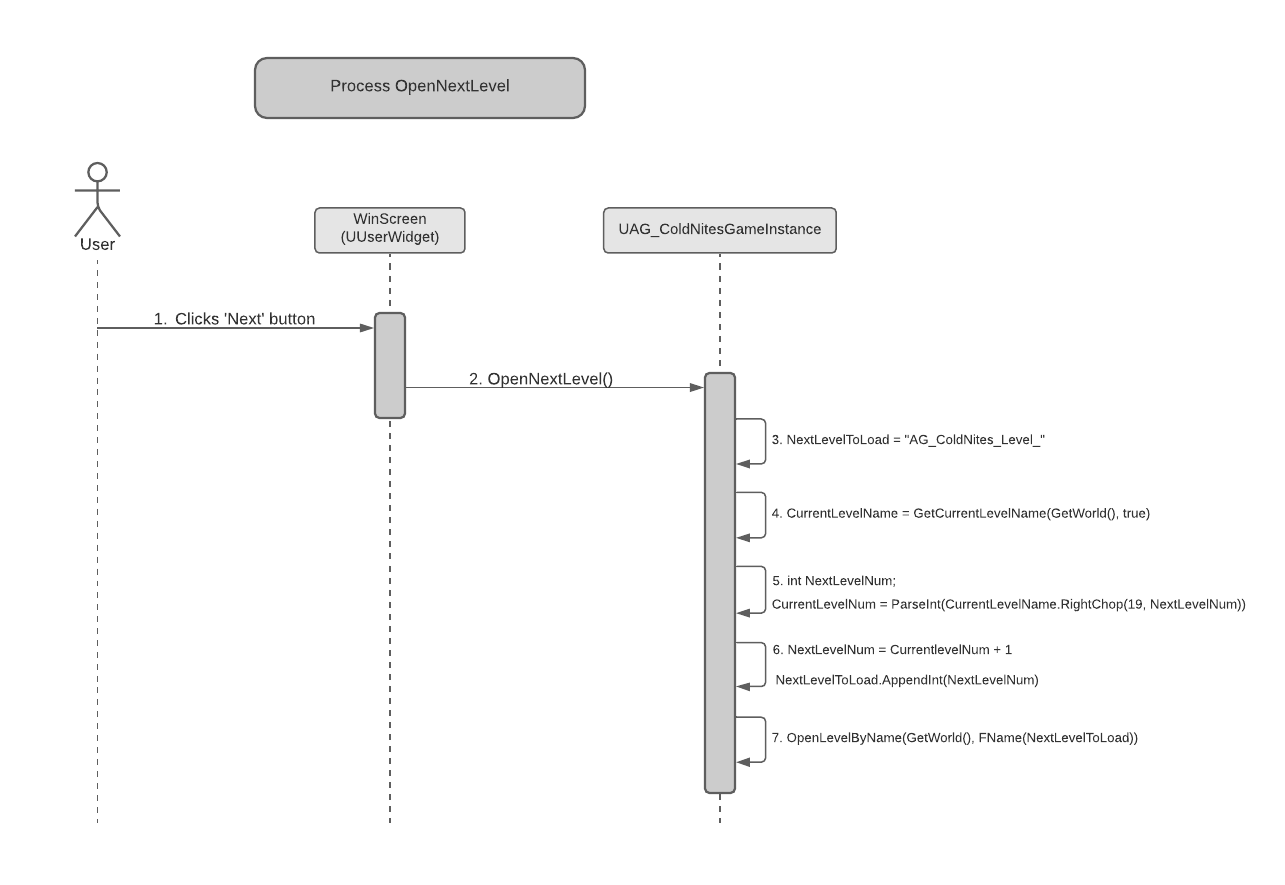
****

**5. Process View**

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

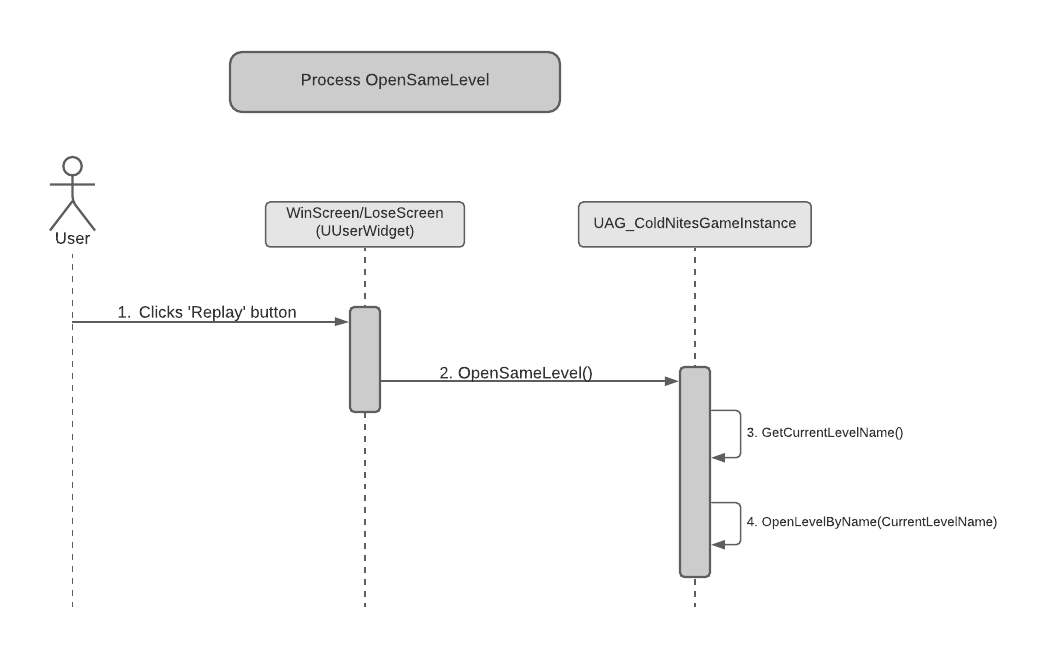
1. **Opening Next Level:**

When the player completes a level, they will have two options on the win screen either to go to next level or to play the same level again. When the player clicks the next level buttons, the button will call the C++ OpenNextLevel() function from the widget blueprint. This function basically gets the name of the level completed and extracts the level number from the name string and increments it to set it to the next level number. After getting the next level number, it appends that number to the base string which is present in all the level names. All the levels in the game are named “AG\_ColdNites\_Level\_” with the level number appended as the last character of the string.

****

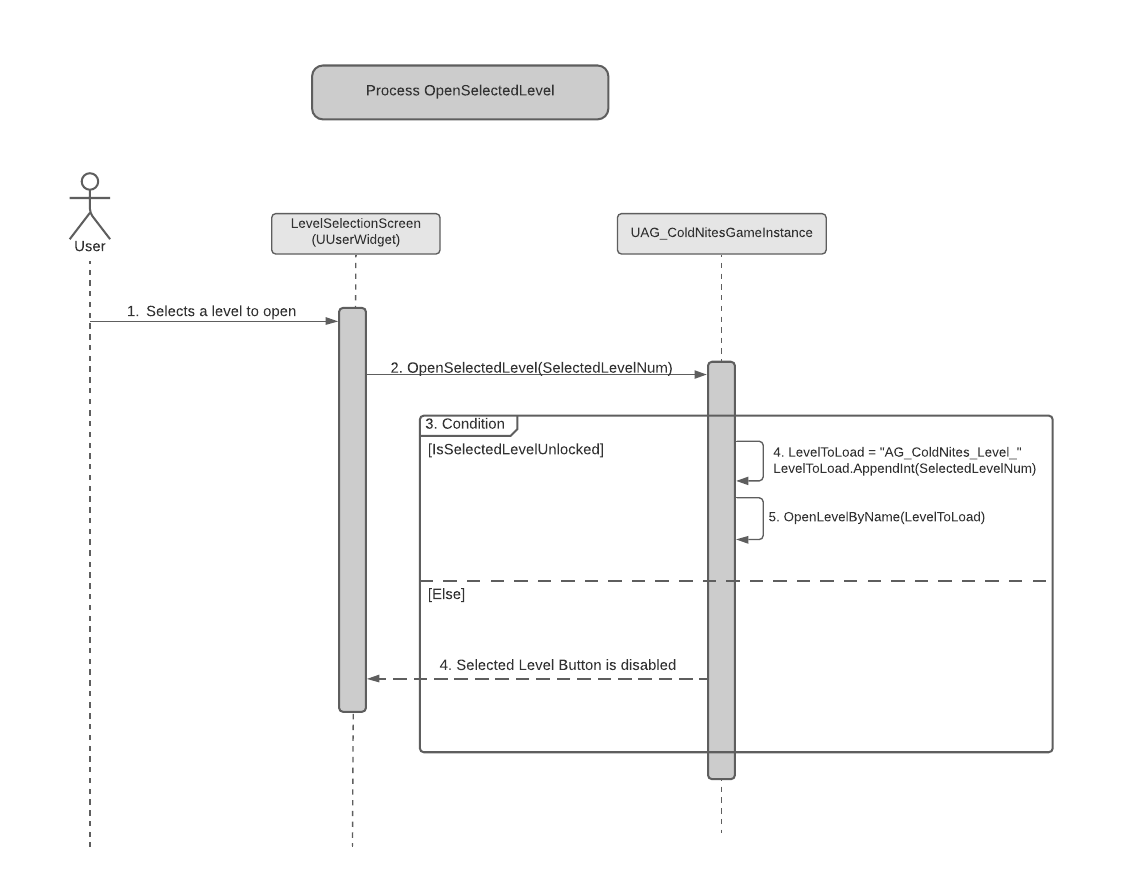
1. **Opening Same Level:**

If the player fails to complete a level or they want to play the completed level once again they will have a restart button on the screen. On clicking the restart button, the OpenSameLevel() on the C++ side will be called by the widget blueprint. This method will simply get the current level name and will feed the same name in the OpenLevelByName() function which is part of the Gameplay Statics file.

****

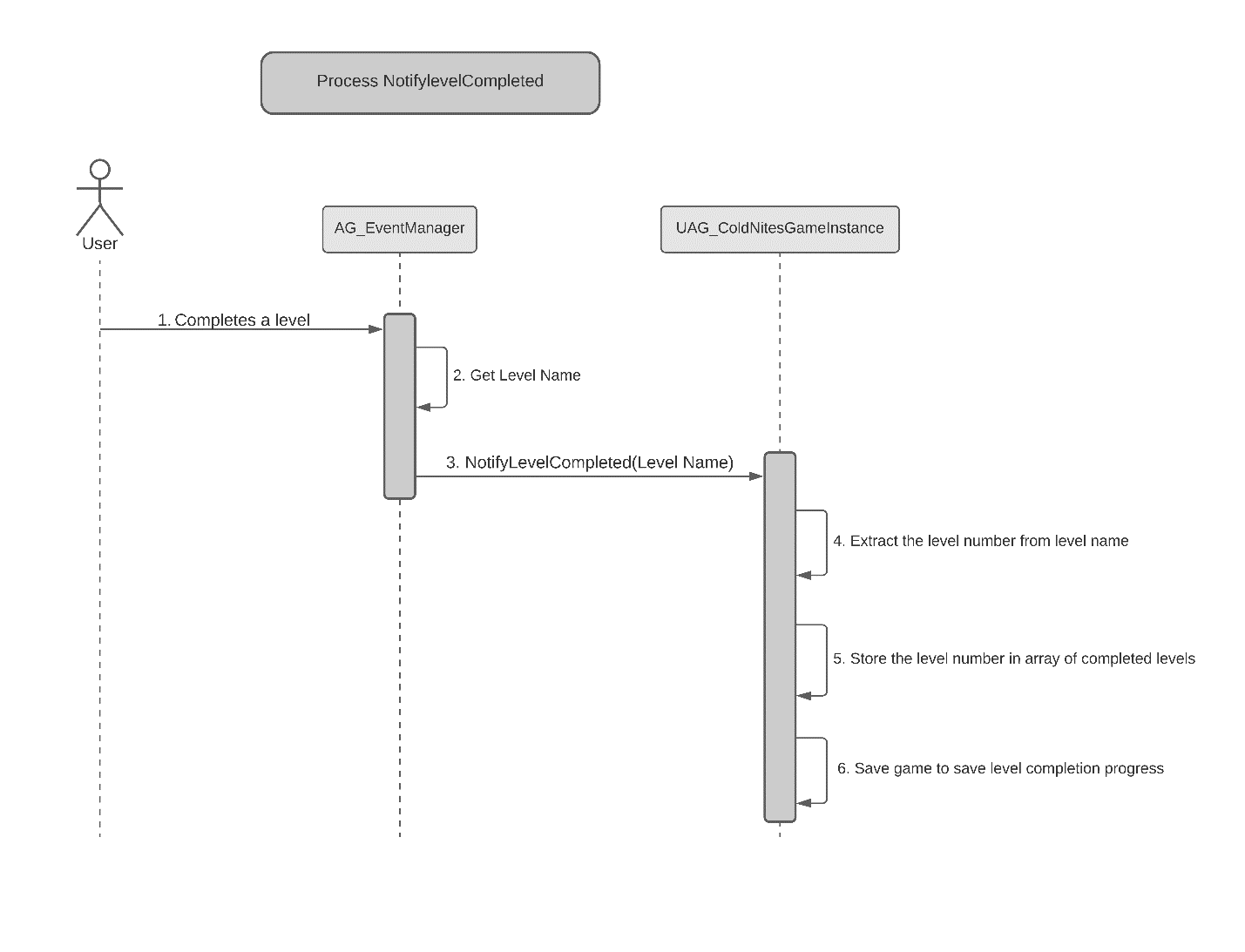
1. **Opening Selected Level from the LevelSelectionScreen:**

When the player first starts the game, they will be directed to the level selection screen from where they select which level they want to play. For the levels that are not yet unlocked, their associated buttons will be disabled on the screen. When the player presses the particular level button, the OpenSelectedLevel() function on the C++ side will be called by the widget blueprint. The function takes an argument in the integer form which is the level number of the level the player is trying to open, will be passed from the blueprint. This function will take the level number and will append it to the base level string “AG\_ColdNites\_Level\_”. The appended string will then be passed to the OpenLevelByName() function which will actually open the level.



1. **Notifying the game instance of level completion:**

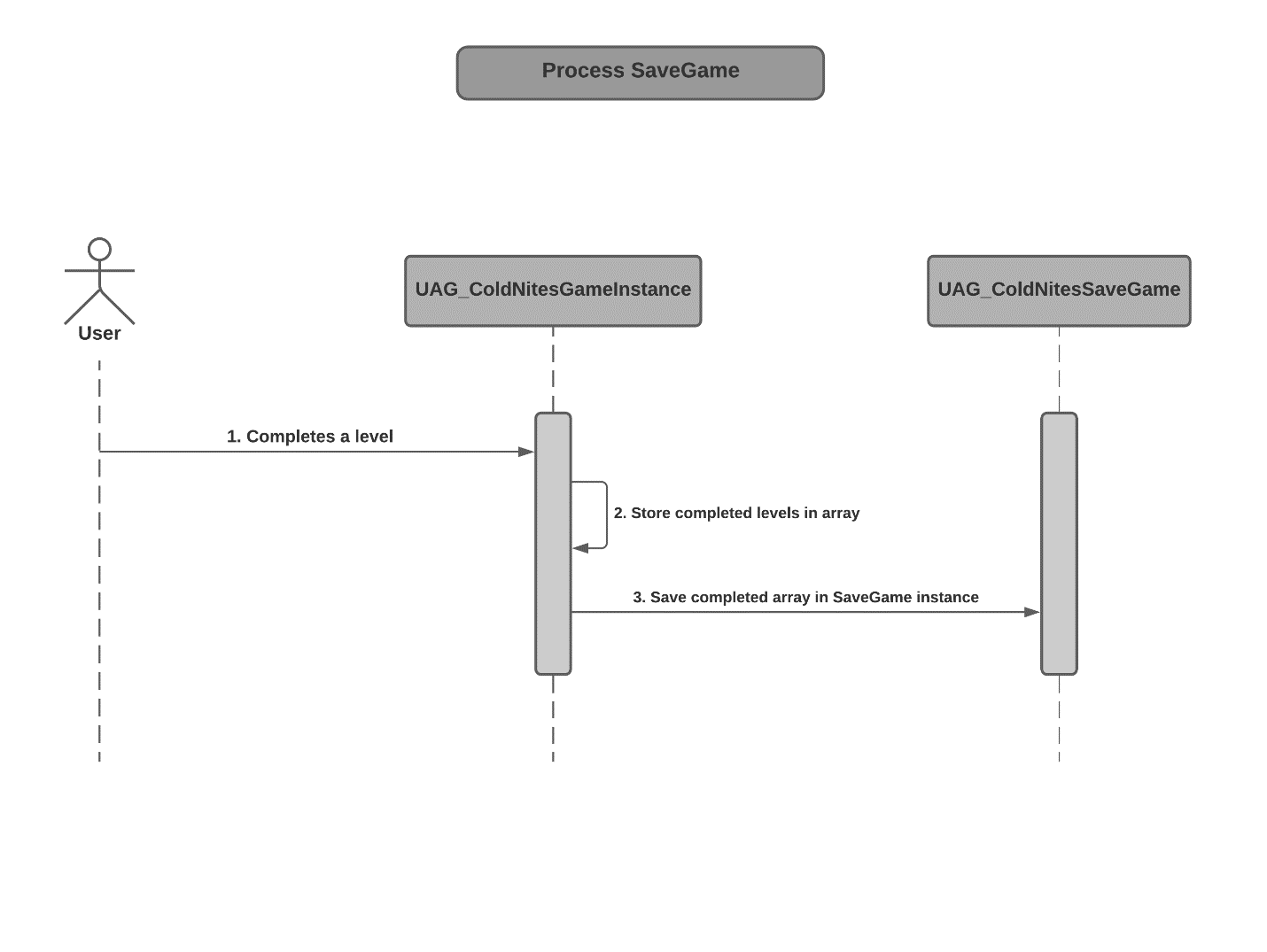
When the player completes a level, the GameInstance need to know that a level has been completed and thus it can allow the player to go to the next level. When a level is completed the EventManager will get a Level Win event. On the level win event the EventManager will get the name of the completed level and will pass it on to the game instance by calling the NotifyLevelCompleted() function. This function will extract the level number from the level name received from the EventManager and it will the number into an array of CompletedLevels.

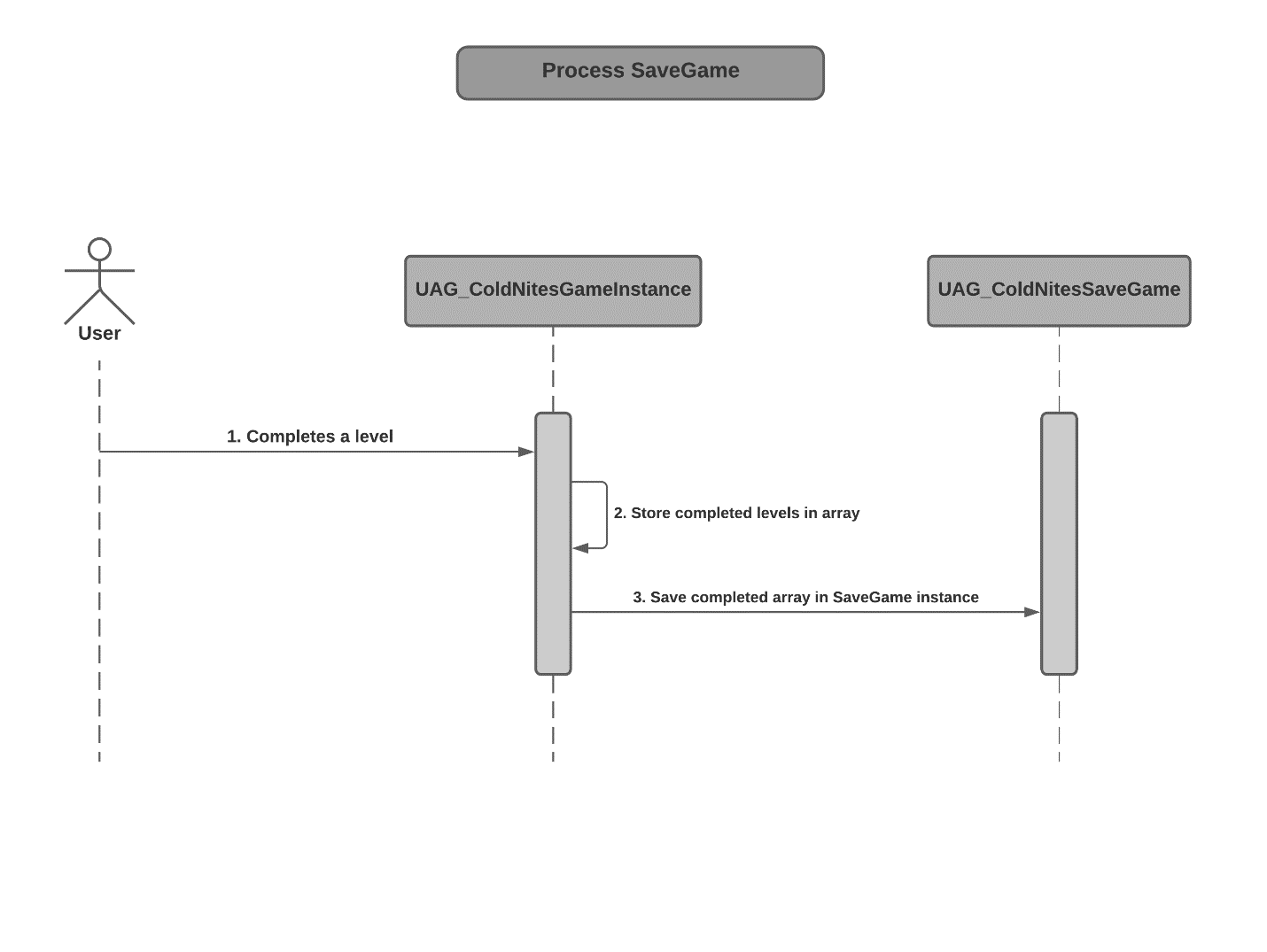


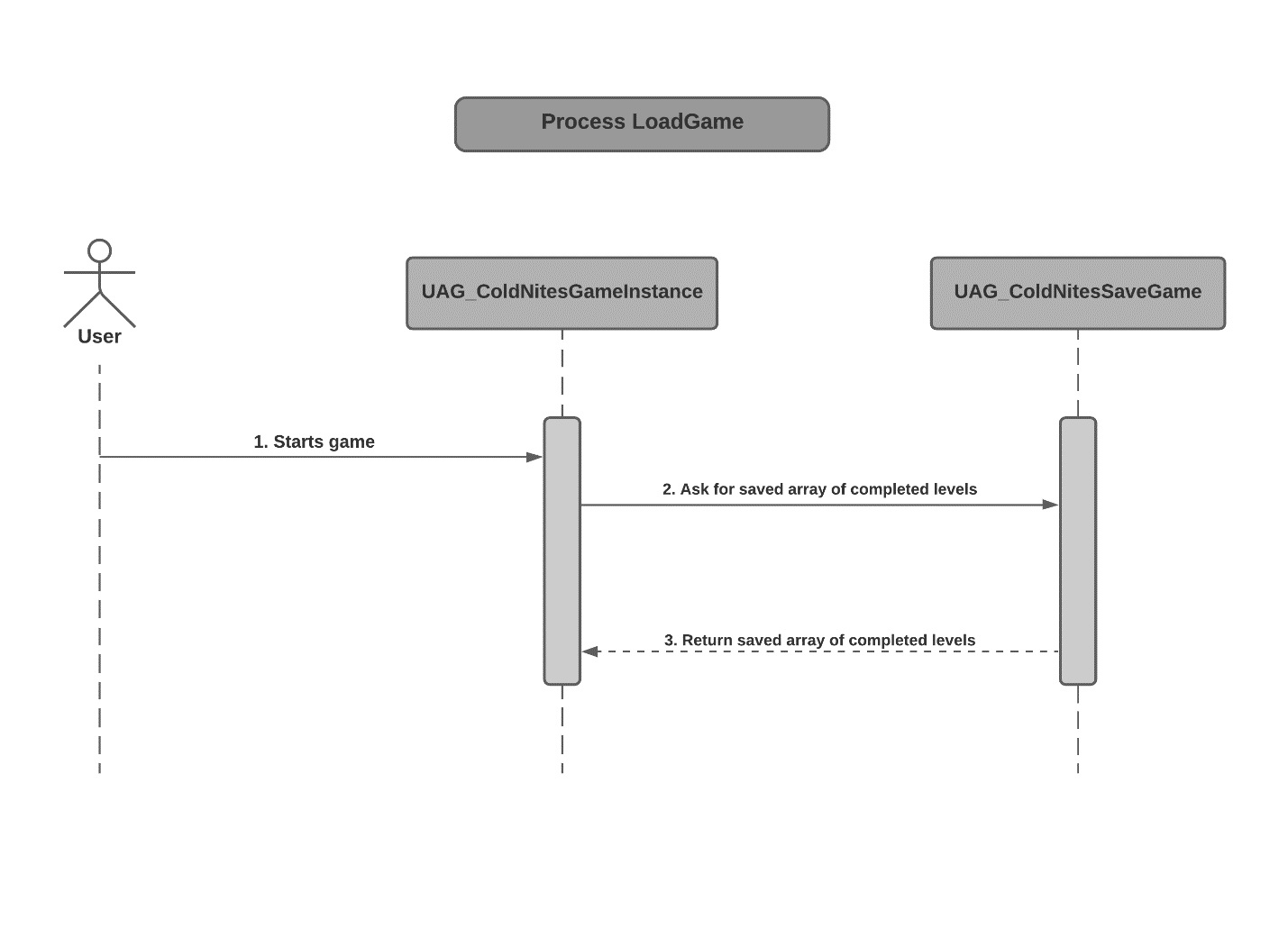
1. **Saving and Loading completed levels:**

The game instance will save the completed levels array in the SaveGame instance after a level is finished by the player.

When the player will start the game again, the SaveGame instance will load this saved array of completed levels back in the GameInstance so the player can continue playing from whichever level they left off.







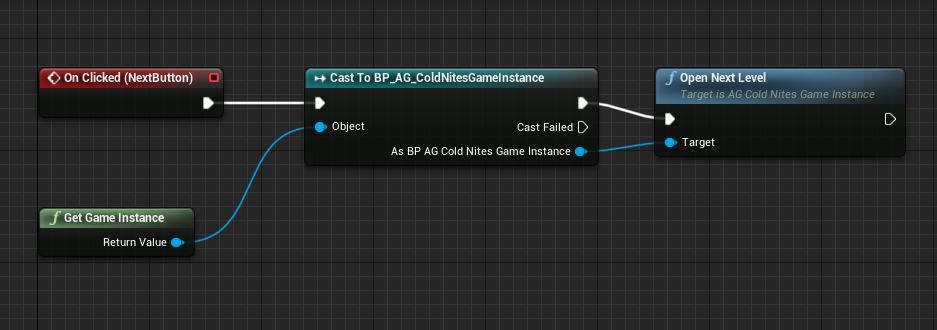
**6. Use Case View and Practice**

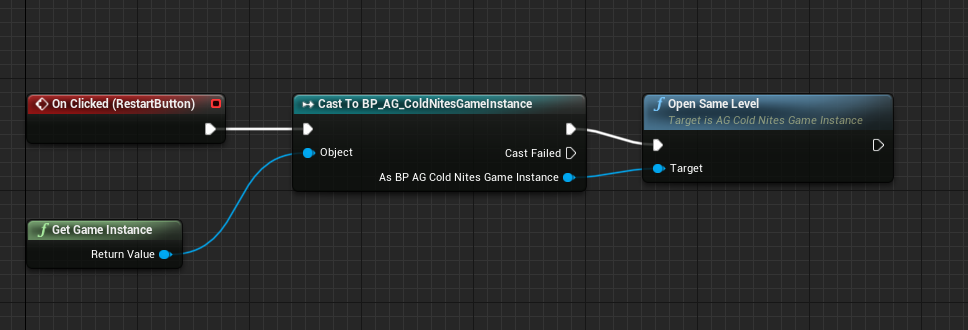
The use case will focus on showcasing the use of Level Management System in the level building for the game and will explain its application so that the Level Management System module can act as a guide/reference for someone not quite familiar with its codebase and implementation.

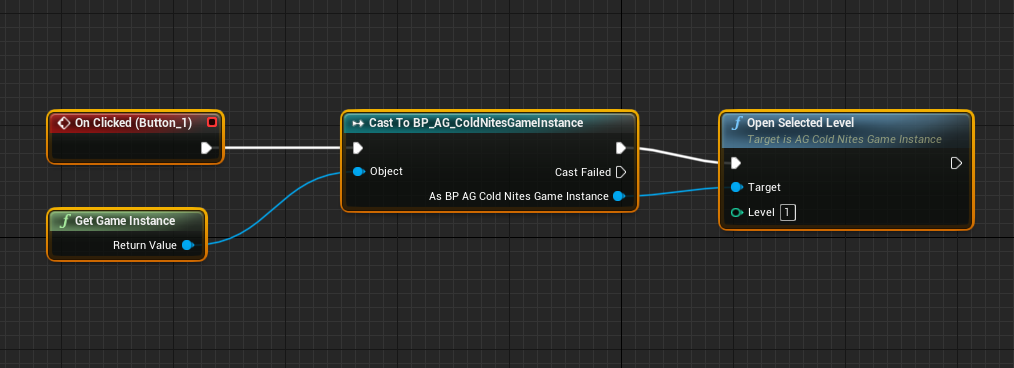
1. **In Code**

All the required functions for managing levels are written in C++ which can easily be called in blueprints in the editor.

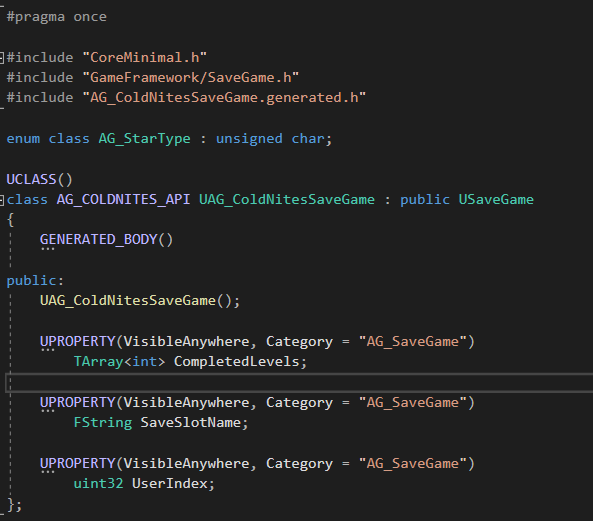
1. **In Editor**

* To open a next level after pressing on the next button, the OpenNextLevel function which is a BlueprintCallable function can be called as shown in the image below.
* To open same level after pressing on the Restart button in either Win or Lose widget blueprint, the function OpenSameLevel which is BlueprintCallable can be called as shown in the image below.

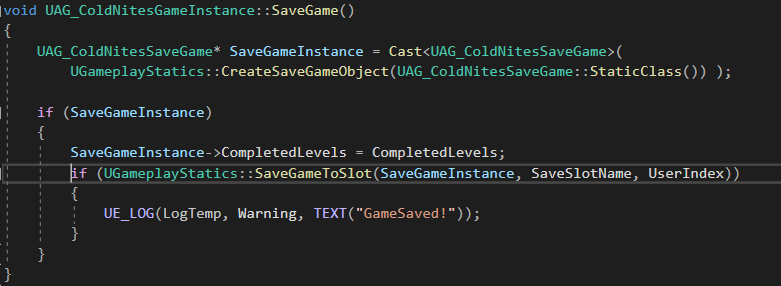
****

* ****To open a specific level from the LevelSelectionScreen. The function OpenSelectedLevel(int LevelNum), which is also a BlueprintCallable function can be called with passing the LevelNum as an argument to the function as shown in the image below.
* **Saving and Loading completed levels**:

To save completed levels array, declare a variable in the SaveGame class as shown in the image below.



To save the array, assign the completed levels array in the GameInstance to the completed levels array of the SaveGameInstance as shown in the image below.



To load the completed levels array, assign the array from the SaveGameInstance to the array in the GameInstance as shown in the image below.

