****

**COLD NITES**

Alpha-2

**Scoring System Design Document**

Designed & Implemented by

Vrund Soni

**Change Log**

**Version** – 0.2

**Modifier** – Vrund Soni

**Date** – 17th April 2021

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

**Version** – 0.4

**Modifier** – Vrund Soni

**Date** – 17th April 2021

**Description** – Completed Design Goals.

**Version** – 0.6

**Modifier** – Vrund Soni

**Date** – 18th April 2021

**Description** – Completed System Overview and behavior and High-level Architecture.

**Version** – 0.8

**Modifier** – Vrund Soni

**Date** – 18th April 2021

**Description** – Added UML Diagram.

**Version** – 1.0

**Modifier** – Vrund Soni

**Date** – 18th April 2021

**Description** Added Process View and 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**

**C. EventManager**

1. **Logical View 7**

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

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

**C. Detailed Design of Scoring System**

1. **Process View 11**

**A.** **Updating the Turn Count**

**B. Handling Earned Stars**

1. **Use Case View and Practice 14**
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 Scoring System.

This document will describe the architecture and design choices that make the Scoring 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** – The Scoring System allows the programmers to have easy control over how the player can earn stars in the game and in what different ways.

**Level Designers –** The scoring system allows the Level Designers to have an easy control over objectives in the game which can earn a star to the player.

**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 managing how earning stars is handled.
* The design should allow other programmers to easily add an earned star by the player to the total collected stars through out the game and stars earned per level and display them on the screen.
* The design should allow level designers to easily change objective parameters for earning stars without worrying about the code behind it.

**3. System Overview and Behaviour**

In the Scoring system the player can earn stars in 3 ways:

* The first is simple, the player can get 1 star just by winning the level.
* The second simplest way is to collect a prop which will give a star.
* The third one and a bit hard, is to complete the level in a given maximum number of turns. If the player does so, they will earn a star.

The Level Management System is made up of three classes:

1. ColdNitesGameInstance
2. ColdNitesSaveGame
3. EventManager

**ColdNitesGameInstance -**

This class holds all the necessary variables and functions to keep track of stars earned on per level basis and throughout the game.

**ColdNitesSaveGame -**

This class holds the number total earned stars throughout the game.

NOTE: This data is stored in TMultiMap. The current engine version (4.26) does not support TMultiMap as UPROPERTY. As a result, CollectedTotalStars will not work for the new Instance of the Game and will be reset on every new launch.

Though CollectedTotalStars will work perfectly with the same running Instance of the Game.

**EventManager –**

This class is responsible for keeping track of stars earned in an individual level.

**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 Scoring System, using a UML Class Diagram.

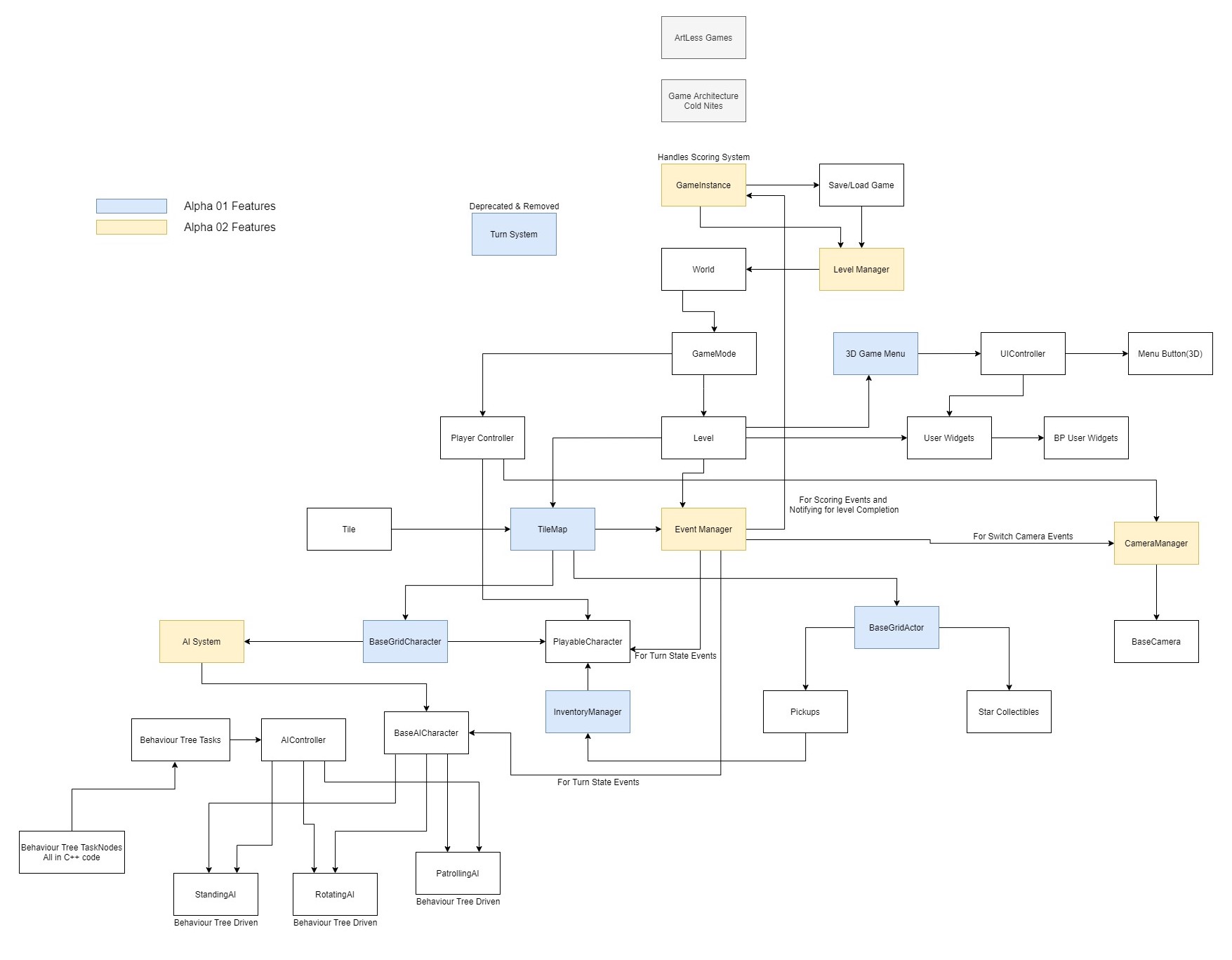
**A. High-Level Design Architecture of the Entire System**

The primary features for the Alpha 1 release:

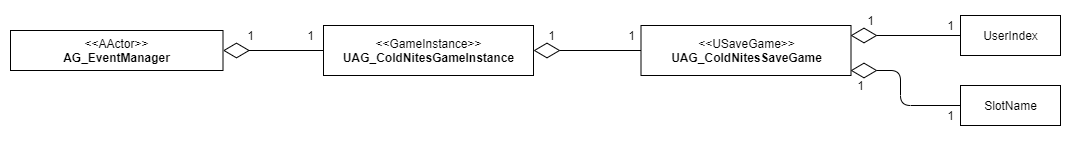
1. **TileMap** - TileMap provides the grid-based behaviour for the game and will facilitate the event system, based on the actor present on the Tiles.
2. **Turn-Based System** - This provides the turn-based aspect for the game. It is responsible for maintaining the turn order for all the world elements(actors)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

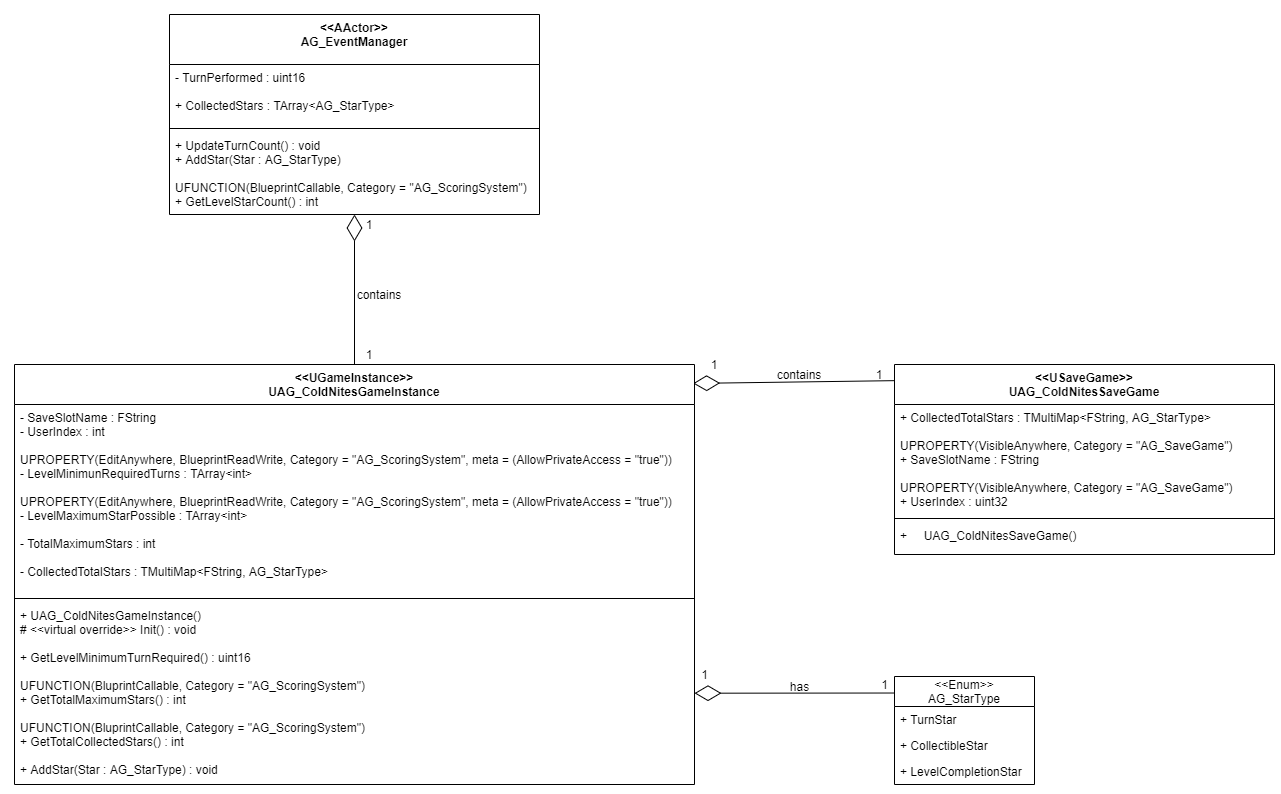
1. **AI System** - AI System is responsible for various enemy types in the game. The AI System allows the ease of creation using the single BaseAI class and Behaviour Tree Task Nodes.
2. **Event System** – Event Manager handles and centralizes level-specific events such as Turn System Events, Camera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



**C. Detailed Design of Scoring System**

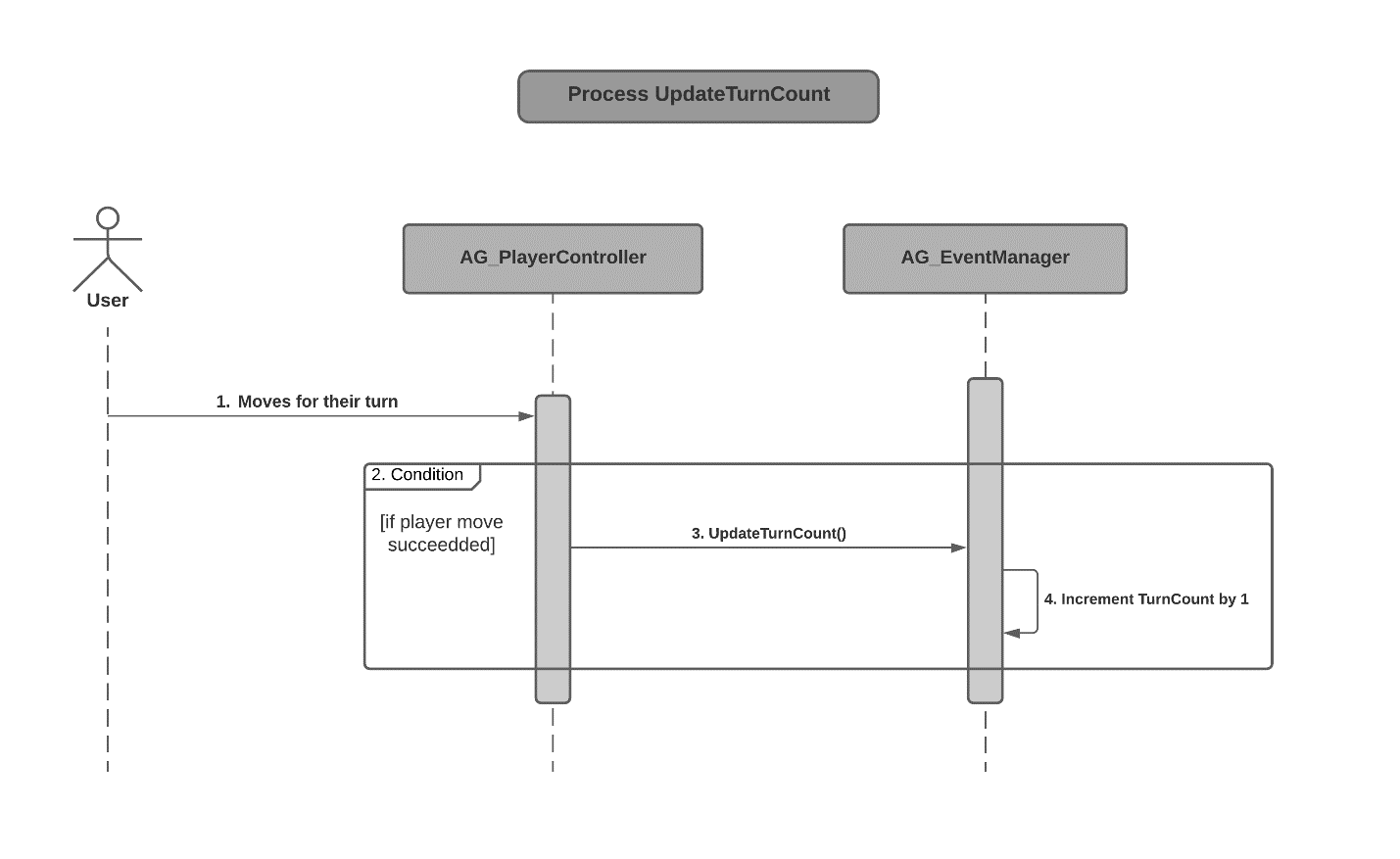
****

**5. Process View**

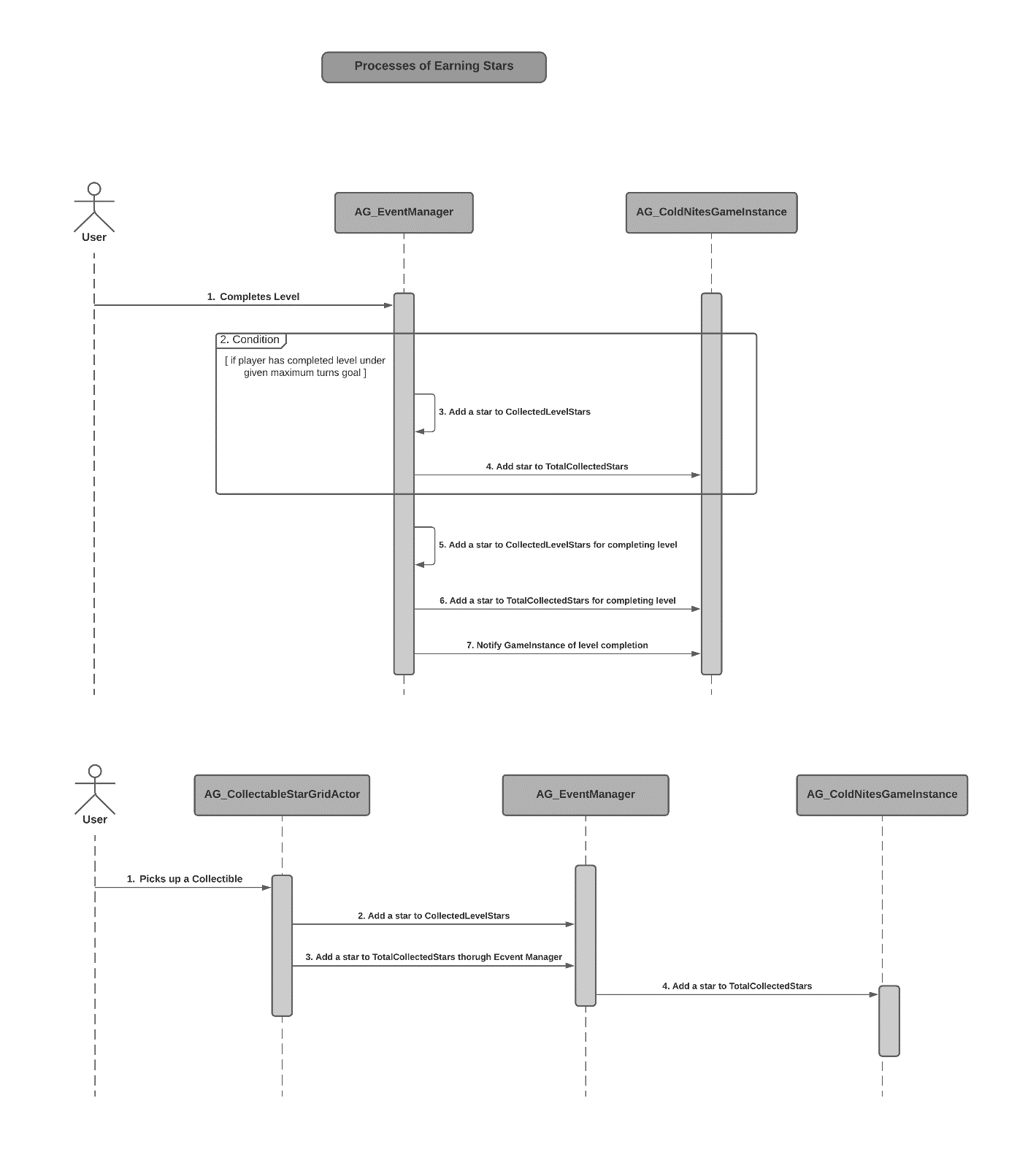
The process view will explain the relation and interaction between various cases using Sequence and Collaboration Diagrams.

1. **Updating the Turn Count:**

To give a star to the player for completing level under given maximum turns the game will store a turn count and the turn count will be updated every time when the player completes their turn.

****

1. **Handling Earned Stars:**

****

* 1. When the player completes a level under given maximum turns. The player earns a star. The GameInstance holds an array of number of required turns for each level. So, when a level is completed, the event managers checks if that level is completed within the given maximum turn goals. If that condition is true, then the event manager adds a AG\_StarType:  
  :TurnStar to the CollectedLevelStars. Furthermore, it will add this star to the GameInstance’s TotalCollectedStars to keep track of total collected stars throughout the game.
* 2. The player also earns a star for just winning a level. Even if they haven’t completed it under the given turns goal or if they haven’t collected a collectible. When the level is completed the event manager will simply add a star of type AG\_StarType::LevelStar to the CollectedLevelStars and GameInstance’s TotalCollectedStars.
* 3. The player can also earn a star on collecting a collectible item. When the player picks up the collectible item the, the actor will add a star of type AG\_StarType::CollectibleStar to the CollectedLevelStars of the event manager and will also add the same star to the GameInstance’s TotalCollectedStars.

**6. Use Case View and Practice**

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

1. **In Code**

* To add a star the to the CollectedLevelStars the, the AddUnique() function can be used by passing in the star type as argument as shown below:

CollectedLevelStars.AddUnique(AG\_StarType::CollectibleStar);

* To add a star to the TotalCollectedStars in the GameInstance the AddStar() function can be used by passing the star type as an argument. For example:

GameInstance->AddStar(AG\_StarType::LevelStar);

* To add a star on collecting the a collectible item, the EventManager’s AddStar can be called from the CollectibleActor’s file, which will in turn call the GameInstances’s AddStar():

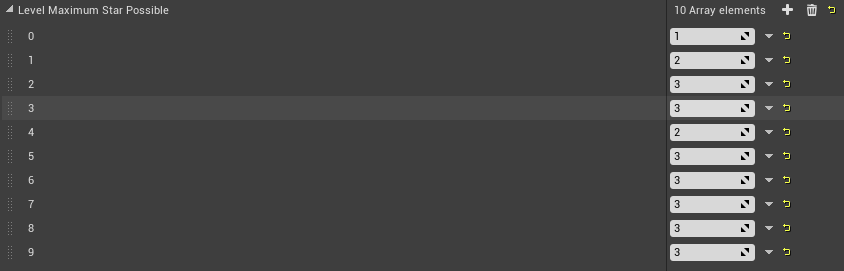
EventManager->AddStar(AG\_StarType::CollectibleStar);

1. **In Editor**

* You can set the maximum number of turns required to complete a level to earn a star in the blueprint of ColdNitesGameInstance as shown below:



* You can set the number of maximum stars the player can earn in each level as shown below:



This array is used to calculate and show how many stars the player can earn throughout the game on the LevelSelectionScreen.