****

**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.

**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.**

**B.**

**C.**

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 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.

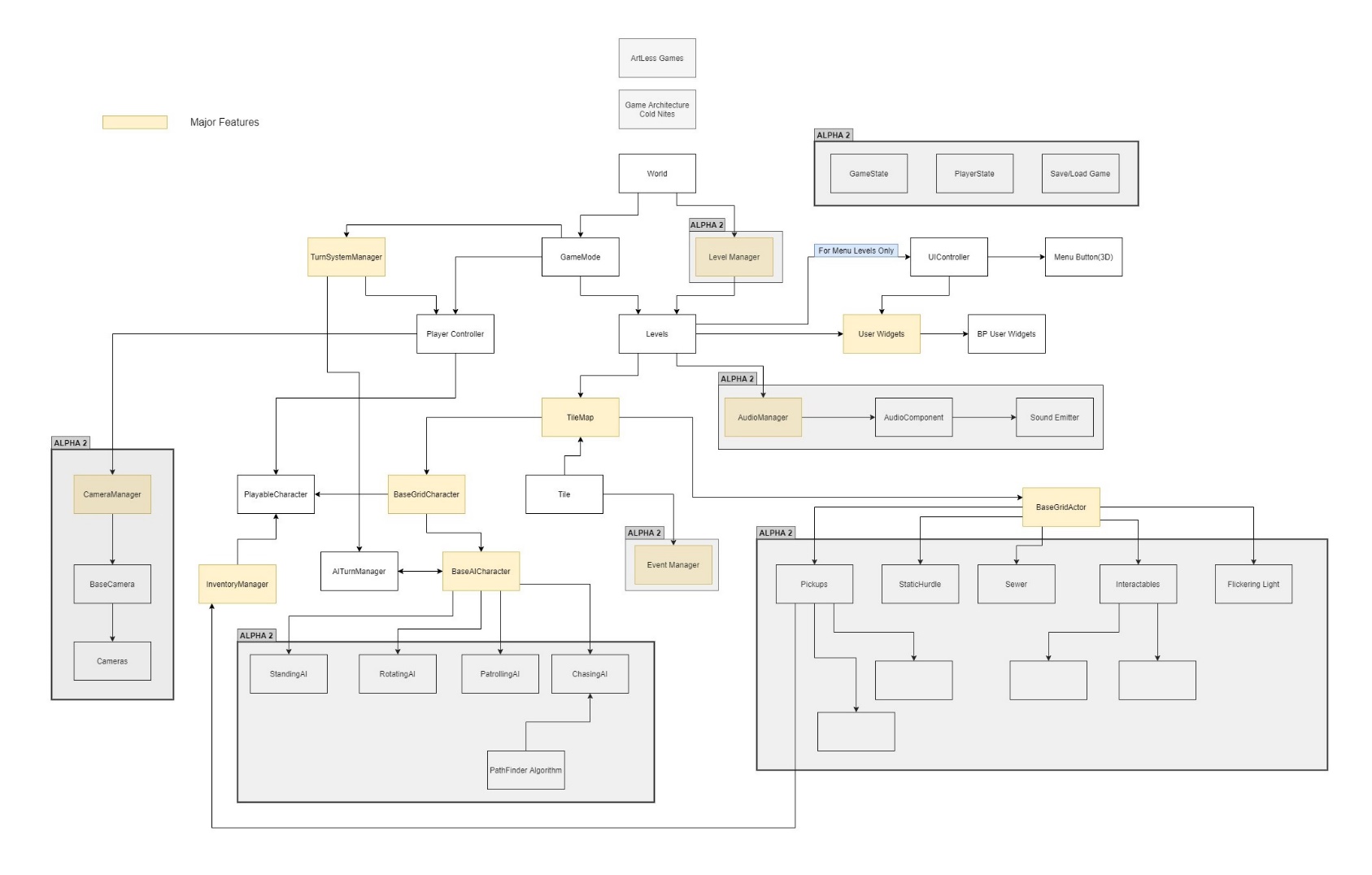
**A. High-Level Design Architecture of the Entire System**

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.

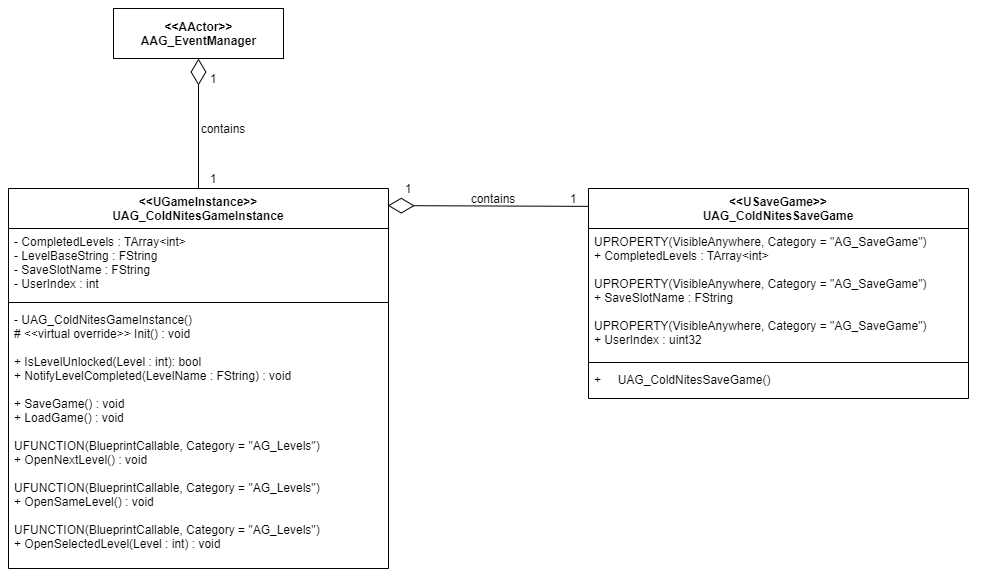
The primary features for the Alpha 2 release:

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** –
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 Level Management System**

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

****