****

**COLD NITES**

**Alpha-1**

**Turn Design Document**

**Designed & Implemented by**

**Jonathan Sime**

**Table of Contents**

**1 Introduction 4**

**2 Design Goals 4**

**3 System Behaviour 4**

**4 Logical View 5**

**4.1 High-Level Design 5**

**4.2 Mid-Level Design 6**

**4.3 Detailed Class Design 6**

**5 Process View 6**

**6 Use Case View 11**

**Change History**

**Version:** 0.1

**Modifier:** Jonathan Sime

**Date:** 03/08/2021

**Description of Change:** Created Design document and modified contents of items 1 and 2

**Version:** 0.2

**Modifier:** Jonathan Sime

**Date:** 03/09/2021

**Description of Change:** Modified contents of items 3 and 4

**Version:** 0.3

**Modifier:** Jonathan Sime

**Date:** 03/13/2021

**Description of Change:** Finished modifying contents

**Version:** 1.0

**Modifier:** Jonathan Sime

**Date:** 03/15/2021

**Description of Change:** Formatted to be consistent with other team documents

**1 Introduction**

This document describes the architecture and design for the Cold Nites application being developed for GAM1528 High Level Development. 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.

The purpose of this document is to describe the architecture and design of the Turn system in a way that addresses the interests and concerns of all major stakeholders. For this application, the major stakeholders are:

* Developers - For an architecture that minimizes complexity and development effort
* Project Manager - For an architecture that divides the system into components of roughly equal size and complexity that can be developed simultaneously with minimal dependencies.
* Maintenance Programmers - For assurance that the system will be easy to evolve and maintain in the future.

**2 Design Goals**

The main design priorities for the Turn system are shown below:

* A design that should minimize complexity and development effort.
* A design that allows the turns to change smoothly and keep the game flowing

**3 System Behaviour**

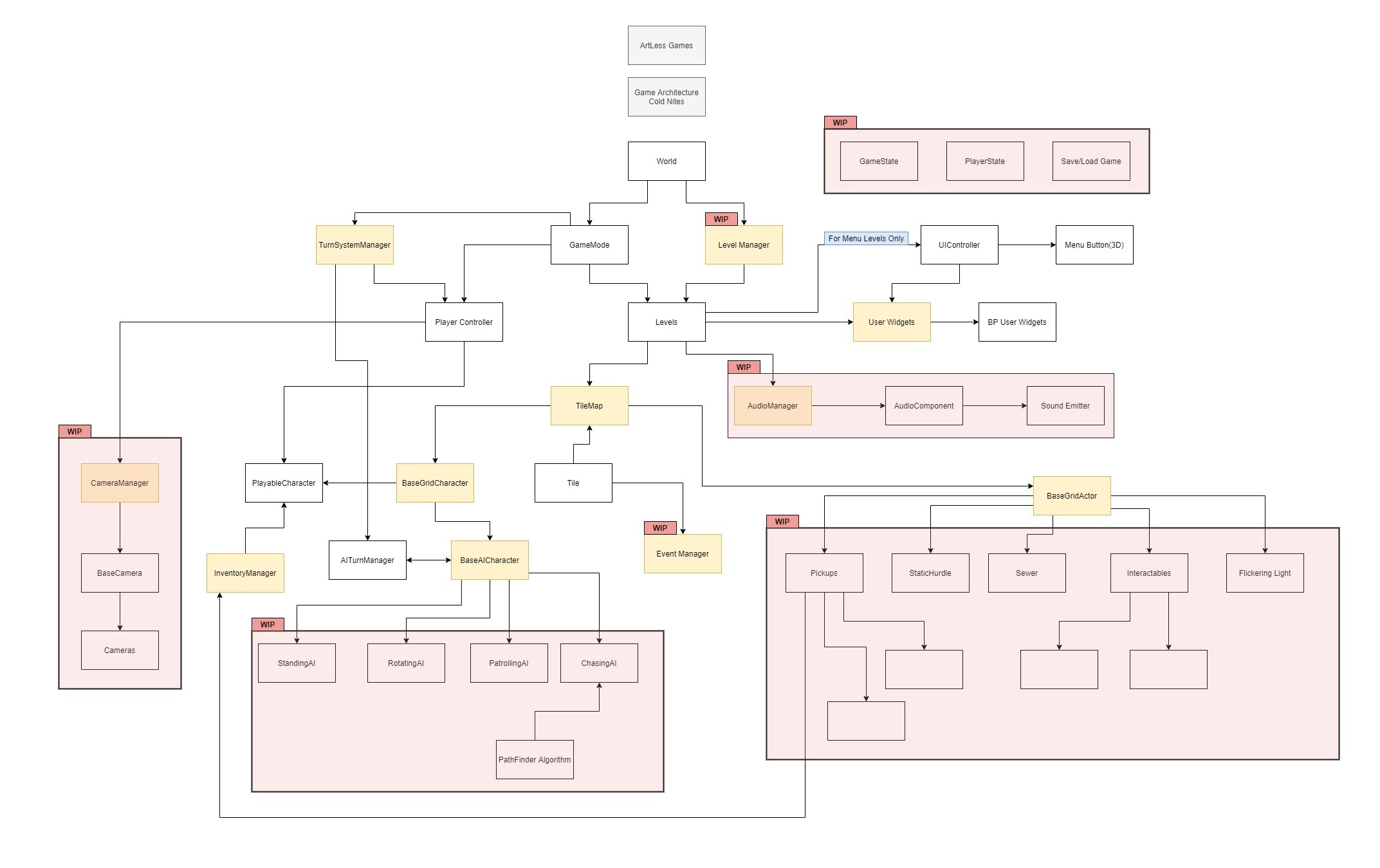
This system will switch between the player’s turn, the AI’s turn and neutral actors' turns, such as flickering lights which would turn on or off at the beginning or end of a turn, as well as make sure that the player and AI are able to move during their turn only.

**4 Logical View**

The logical view describes the module from a high concept to a detailed analysis of the interactions and possible use cases.

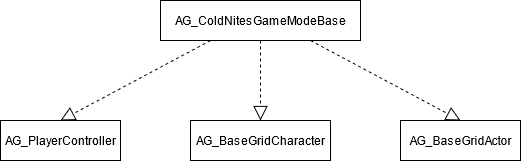
***4.1 High-Level Design***

The high-level view consists of 4 major components:

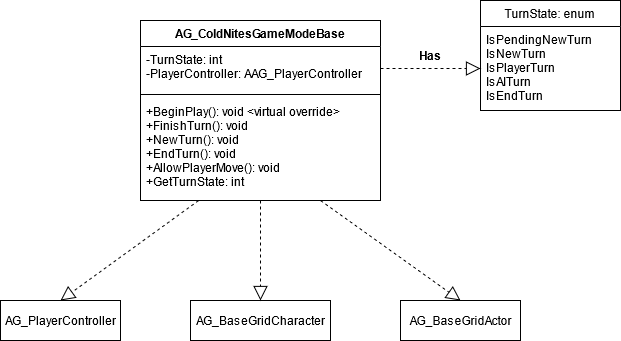


* Player system consists of a controllable character that inherits from BaseGridCharacter, which takes user inputs to perform appropriate moves.
* Tilemap system provides the grid-based behaviour for the game and will facilitate the event system, based on the actor present on the Tiles.
* Inventory system stores the item for the corresponding actor and will allow the player easy access to any collectible throughout the game.
* User Interface is responsible for Main Menu and any in-game HUD (or User Widget) with which the player can interact.

***4.2 Mid-Level Design***



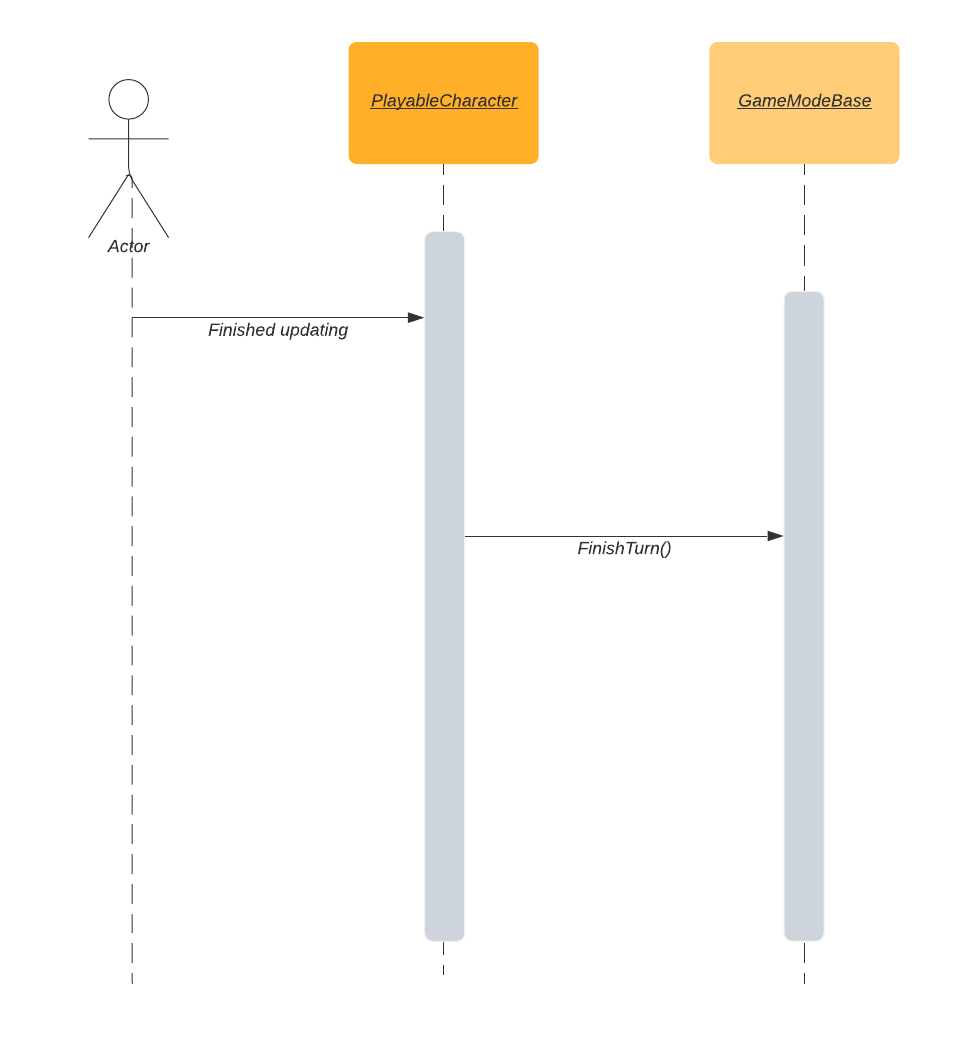
***4.3 Detailed Class Design***



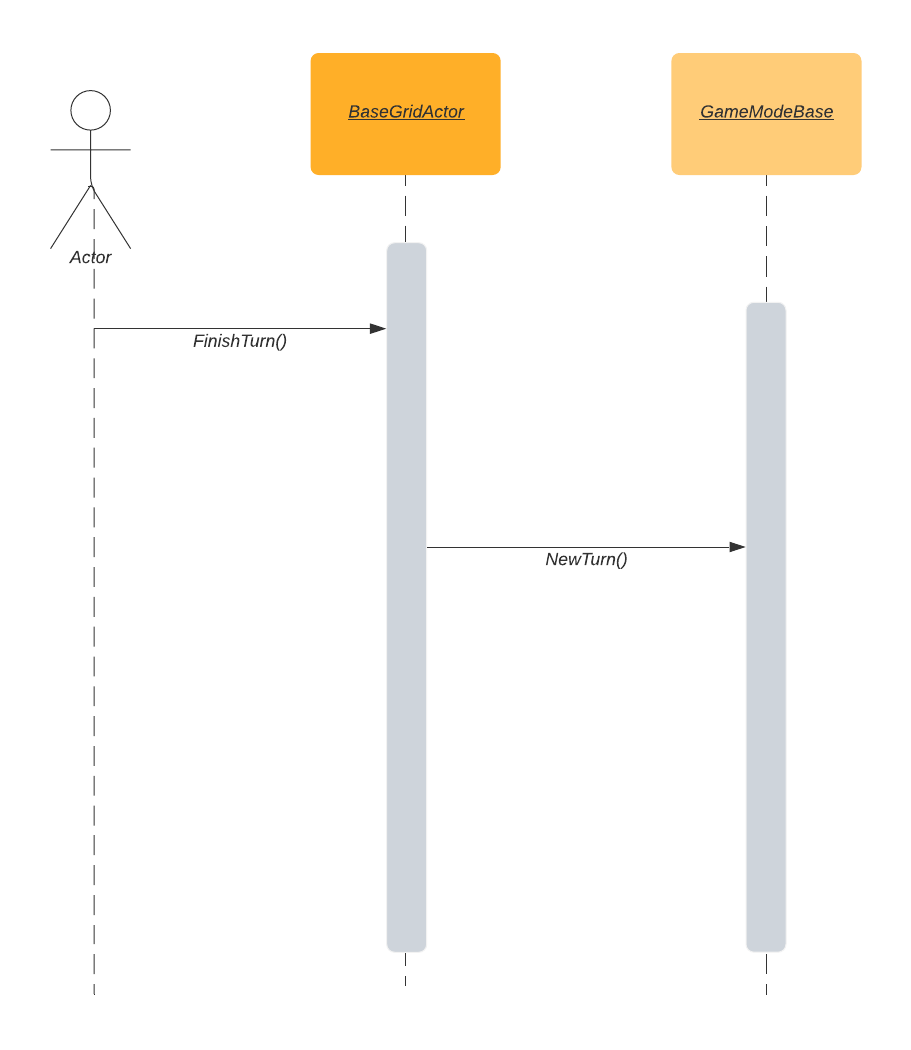
**5 Process View**

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

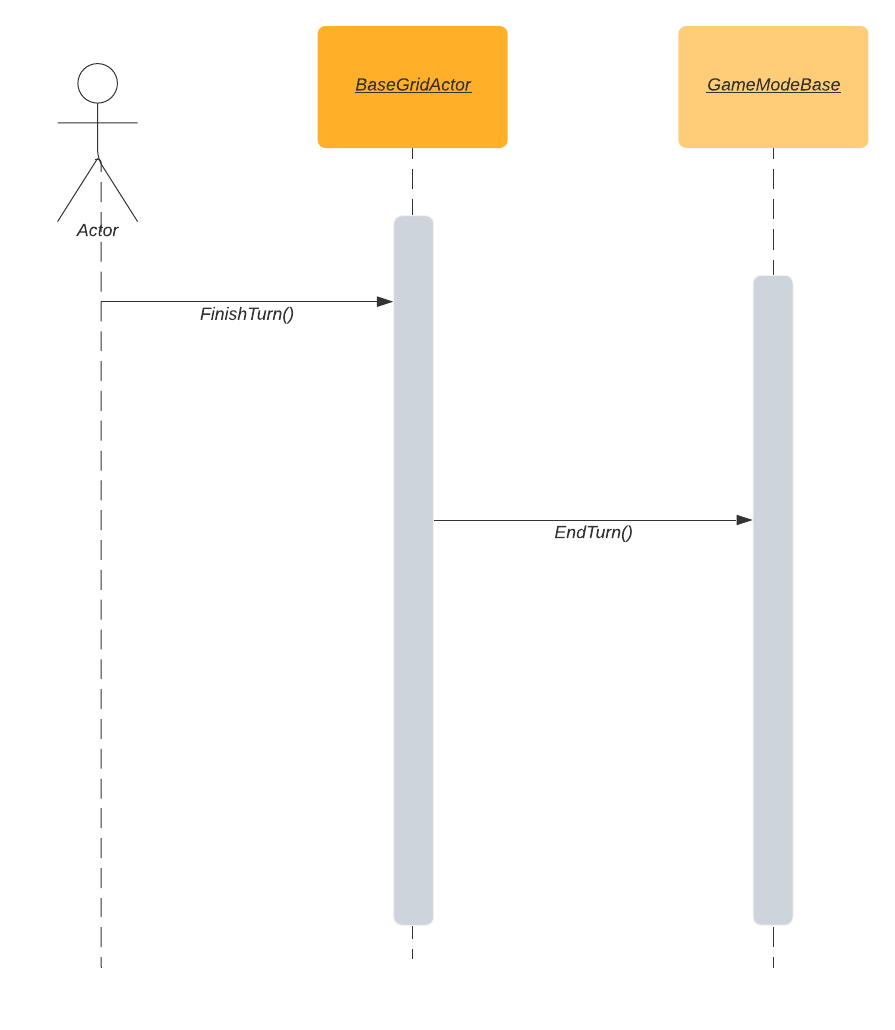
FinishTurn() is called by the last actor whose turn it is to move on to the next turn. This ensures that all objects get to update before the turn ends.



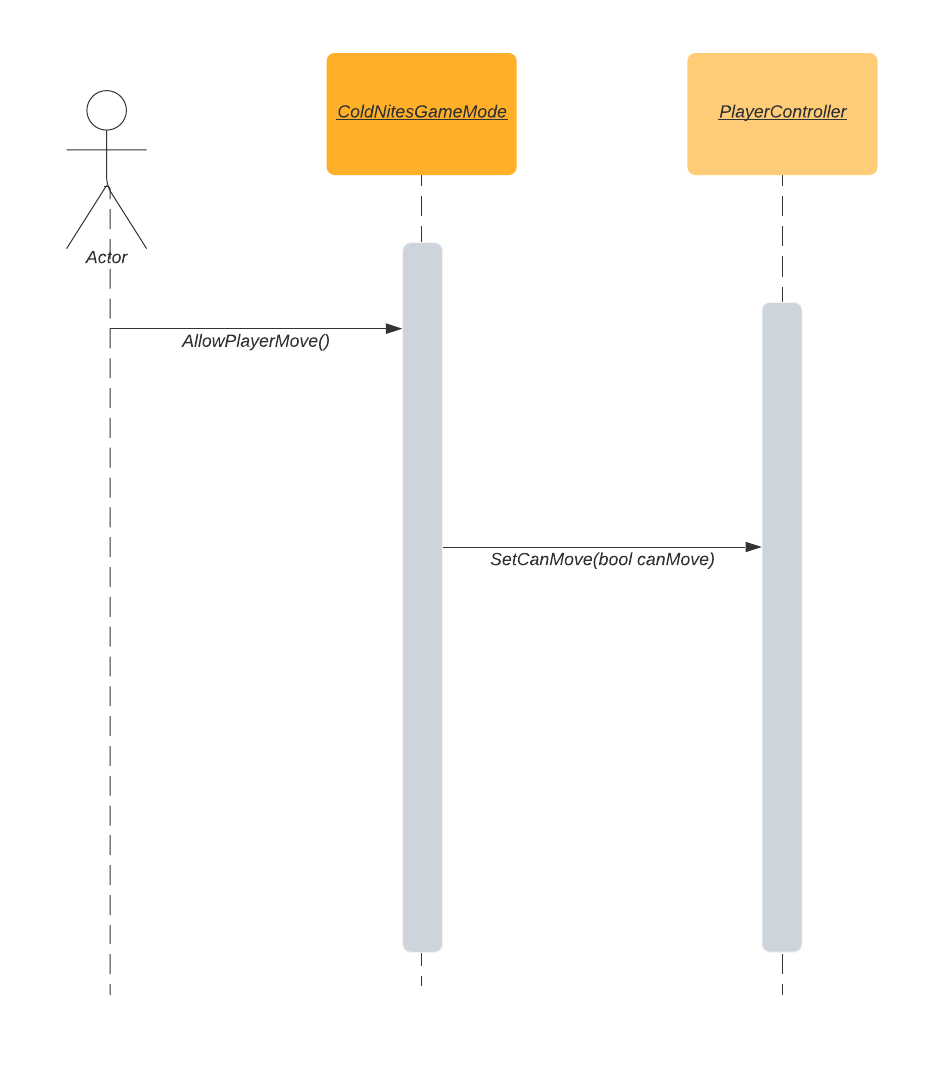
NewTurn() is used to update any actors or objects that should be updated before a new turn starts, such as flickering lights.



EndTurn() is used for the same purpose as NewTurn(), but for actors or objects that should be updated before the turn ends.



AllowPlayerMove() is called on the player’s turn so the player can receive inputs and move or interact with other actors.



**6 Use Case View**

**Turns**

The TurnState begins with IsNewTurn which allows the game to update any objects before the player as to show mechanics or to change actors in the level. TurnState moves to IsPlayerTurn so that the player can update. The AI updates during IsAITurn which is the state after IsPlayerTurn. Finally, IsEndTurn is the last state which is used to update any objects to be updated after both the player and AI have finished their turn. IsPendingNewTurn is to make sure the game continues into the NewTurn state.