Aidan Lewis

**Pattern name**: Singleton

**Class Name:** Player

**Purpose:** To begin, the Singleton pattern could be used with the Player class because at all times there will be only be a singular instance of the Player class. Further, using the Singleton pattern was necessary in that the player’s functionality must be accessed from multiple places. For example, when a Water Cell object is determining whether it will cause death, it checks whether the instance of the Player class currently has the ability to swim. In addition, in the Door Cell class, the keys a Player currently has must be checked, and if the necessary one is found, will be used. If the Singleton pattern were not used in this case, I would’ve had to pass the Player object through to the Water Cell and Door Cell class, and any others that use it (the Key Cell class, for example). This would’ve required significant additional work.

**Class Name:** Grid

**Purpose:** The Singleton pattern could also be used with the Grid class because at any given time, there will only be one instance of the Grid class. Additionally, aspects of the Grid class need to be accessible in various places, including the Game class, the Level class, the Chip Cell class, and the Portal Gate class. If I did not use Singleton, this would’ve required me to pass the Grid class in and out of all of these different classes. The Singleton pattern allows us to avoid doing this.

**Pattern name:** Composite

**Class Name:** Grid

**Purpose:** The Grid class consists of a 2D array of Cell class objects. In other words, this grid is composed of Cell objects. Composition helps here in that each cell can have its own, unique functionality while still be part of a larger, overarching object that allows us to know the location of each cell.

**Class Name:** Game

**Purpose:** The Game class is composed of a Grid object, a Scoreboard object, and a Player object. After all, the game consists of the game board (Grid), a player in this game (Player), and a scoreboard for tracking progress (Scoreboard).

**Pattern name:** State

**Class Name:** Cell

**Purpose:** The game consists of many different types of cells, which will each share some basic methods (while still each having their own unique functionality). Because of this, the State design pattern made sense, since we could have many different specific types of cells implement an overarching Cell interface that laid out the necessary functionality for all cells. Each specific type of cell would then implement these basic methods according to how that cell was supposed to function. In addition, any unique methods to a specific cell could be added to it without issue.