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 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 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 and Door class, and any others that use it (the Key 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 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:** Factory Method

**Class Name:** CellFactory

**Purpose:** The CellFactory class allows cells to easily be generated when necessary. By passing in the necessary cell type, CellFactory will return a cell with the requested cell behavior. This class made it simpler to add cells to the grid when creating each level.

**Pattern name:** Strategy

**Class Name:** CellBehavior

**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 Strategy design pattern made sense, since we could have many different specific types of cells implement an overarching CellBehavior 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 to simulate that specific cell’s behavior.