* **Smells:**
  + **Coupler smell**: Inappropriate Intimacy - GameController used all MakeMove methods.
    - **Solution:** Move all MakeMove metods into GameController and remove MakeMove class
  + **Coupler smell:** Middle Man- GameDriver was acting as middle man between GameController and the UI.
    - **Solution:** Remove GameDriver and have a direct connection between GameController and UI
  + **Coupler smell:** Middle Man- Space was acting as a middle man between Piece and Board.
    - **Solution:** Remove Space and board consist of different type of Pieces including Rook, King, or None.
  + **Dispensable smell:** Dead Code - UniqueRandomNumbers was created with the intent to generate unique IDs, but database handles the creation of IDs.
    - **Solution:** Remove UniqueRandomNumbers
  + **Dispensable smell:** Dead Code – DatabaseDriver was created to facilitate communication between database and code, but was never used.
    - **Solution:** Remove DatabaseDriver
* **Design Patterns:**
  + **Controller Pattern:**

- **Problem**: Who should be responsible for handling an input system event?

**- Solution:** GameContoller defines all the methods for all system operations. It’s this class that’s responsible for receiving and handling system events made by the user via the GUI. It also receives signals from different sensors or triggers from the UI to maintain/control information about the state of the game.

* + **Observers Pattern:**

**Problem-**  Ifone object changes its state how will all classes that are dependent on that object be notified and updated automatically?

**-Soultion-** Our whole entire UI Package is a view that contains and represents the observers needed to make a MVC pattern work. In our game this usually comes in the form of buttons that handle event management. For example, when player hits “Refresh” the game object and all information pertaining to it (i.e board) is updated.

* + **Coming Soon**: Builder Design Pattern – translate database objects to backend objects.