The four design patterns I used were Memento, Strategy, Template Method, and Abstract Factory. Firstly, the Memento pattern involves saving and restoring the previous state of an object without revealing the details of its implementation. It works by having the originator create a snapshot of its state and save it into a memento, which is then stored by the caretaker. I implemented Memento, Caretaker, and Originator classes to handle storing states, backing up every move, and enabling undo and redo functionality using a stack of mementos. Additionally, I used the Strategy pattern. This pattern defines a family of algorithms, encapsulates each one in a separate class, and makes them interchangeable. I applied this by creating separate classes for different strategies like Random and Heuristic, allowing different approaches to selecting valid moves. I also implemented the Template Method pattern, which defines the skeleton of an algorithm in a superclass while allowing subclasses to override specific steps without changing the algorithm's overall structure. In my project, the Human, Random, and Heuristic AI classes all inherit from the Player class and override methods as needed to fit their specific behavior. Finally, I used the Abstract Factory pattern, which allows the creation of families of related objects without specifying their concrete classes. I created a factory file that includes three factory classes, Human, RandomAI, and Heuristic, and based on user input, the correct factory is selected to create the appropriate object, reducing redundancy in the code. I also attempted to use the Command pattern, but I got a bit lost during implementation and ended up not including it.