Design Patterns

Sprint1

CSC207

Aleksandra Kalas, Malhaar Kumar, Dhir Shukla, Jovana Spasojevic

Table of Contents:

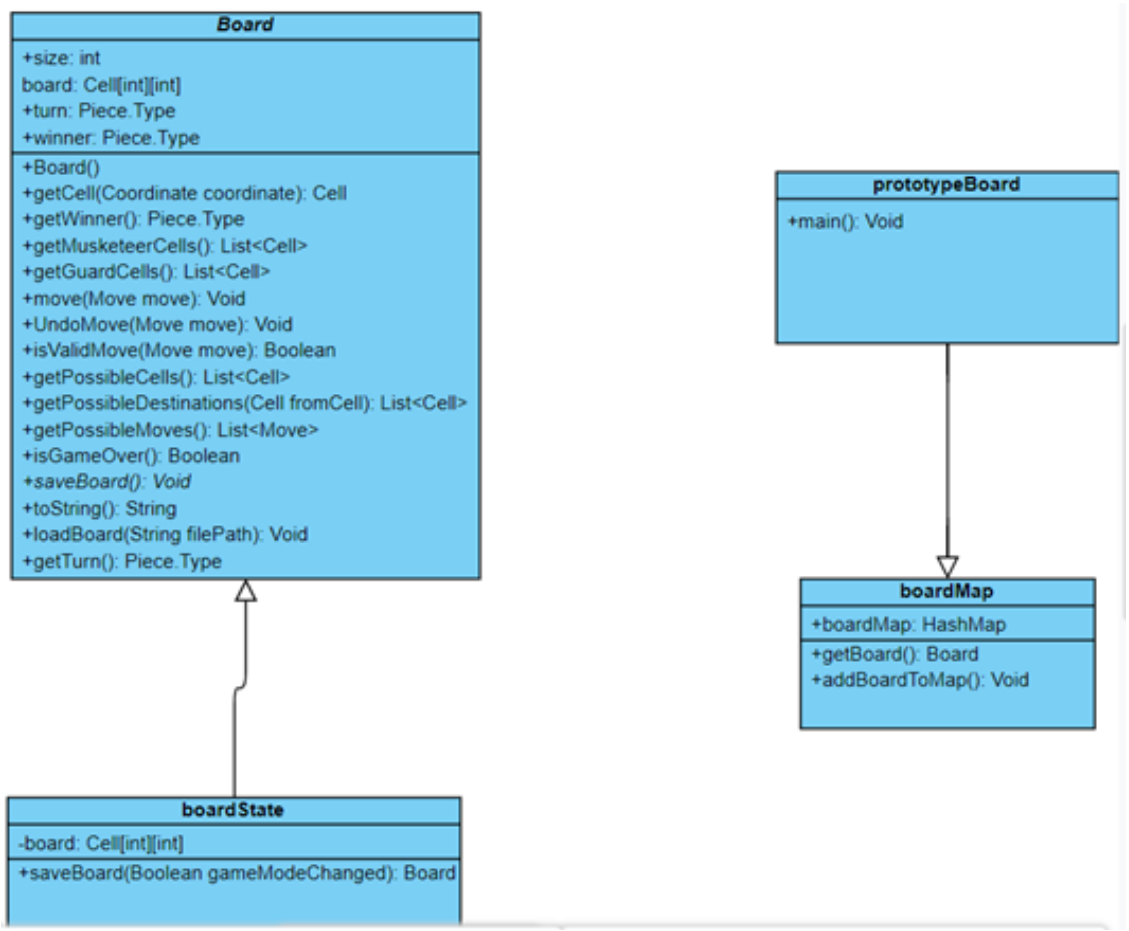
Prototype Pattern:

How it Works/What it Solves:

The prototype pattern is used to specify the kind of objects that are being created using a prototypical instance. Along with this it allows for creating new objects that are copies of the prototypical instance.

We will be implementing the prototype pattern in order to save the current board as a prototype. By using this prototype board the user will be allowed to switch from playing a game mode of Human vs. Human to Human vs. some version of computer whether that is Greedy or Random. We will be able to switch easily by loading the prototype board into the constructors of these various game modes.

UML Diagram:



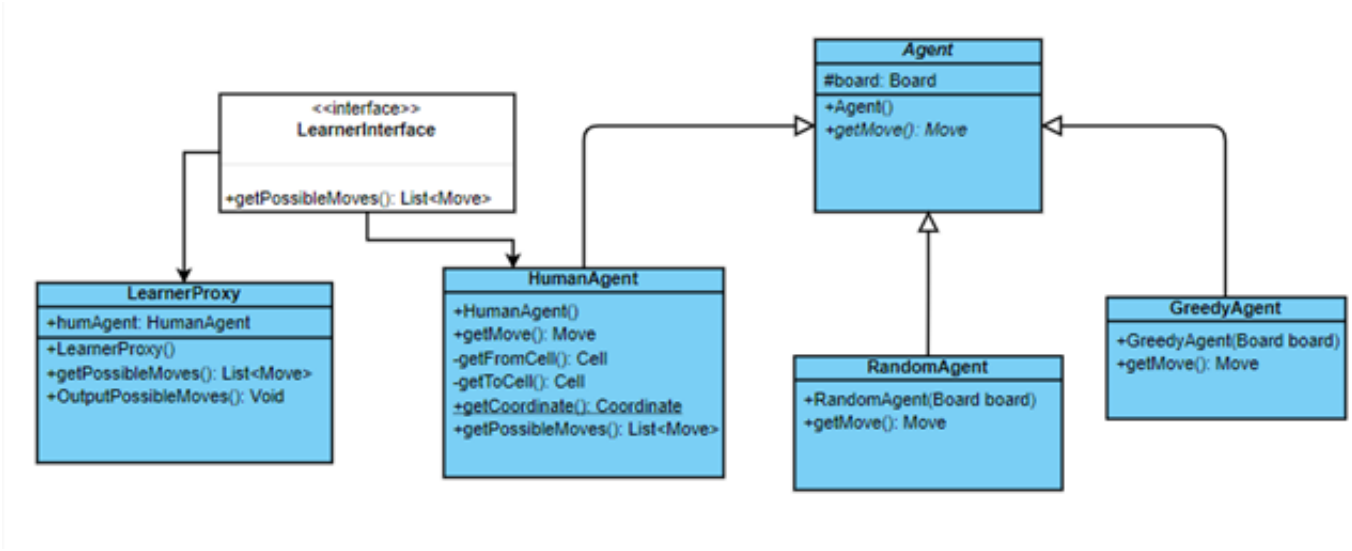
Proxy Pattern:

How it Works/What it Solves:

In general the proxy pattern provides a placeholder for an object so that the object can control some references to it. Along with this it will help in regards to accessing an object in this case our learning mode in a sophisticated way.

We will be using the proxy pattern to develop a learning mode of the game Three Musketeers. In this learning mode the user will be able to play as either the Musketeer or Guard and have the opportunity to see what possible moves they can make. By implementing the proxy pattern with this design in mind we will be able to access this temporary board and not make any changes to the starter board.

UML Diagram:



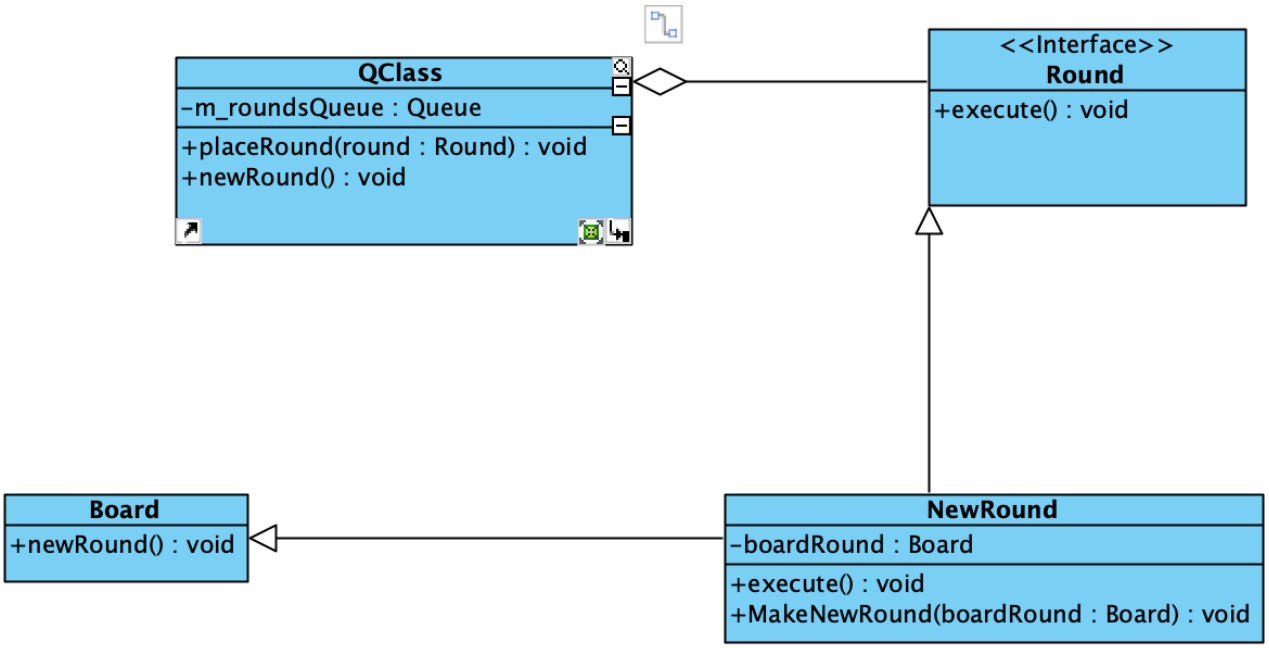
Command Pattern:

How it Works/What it Solves:

The command pattern is used to allow saving the requests that a user has. When a client is asked for commands these commands are first encapsulated into an object. These commands are then saved in a queue and then executed in an order that can be modified.

This patten will be implemented to hold account of how many games have been won by each player. Along with this the pattern will allow us to save the boards of each round and keep track of essential information such as the moves done on that board, the players, and who the winner of that round was.

UML Diagram:



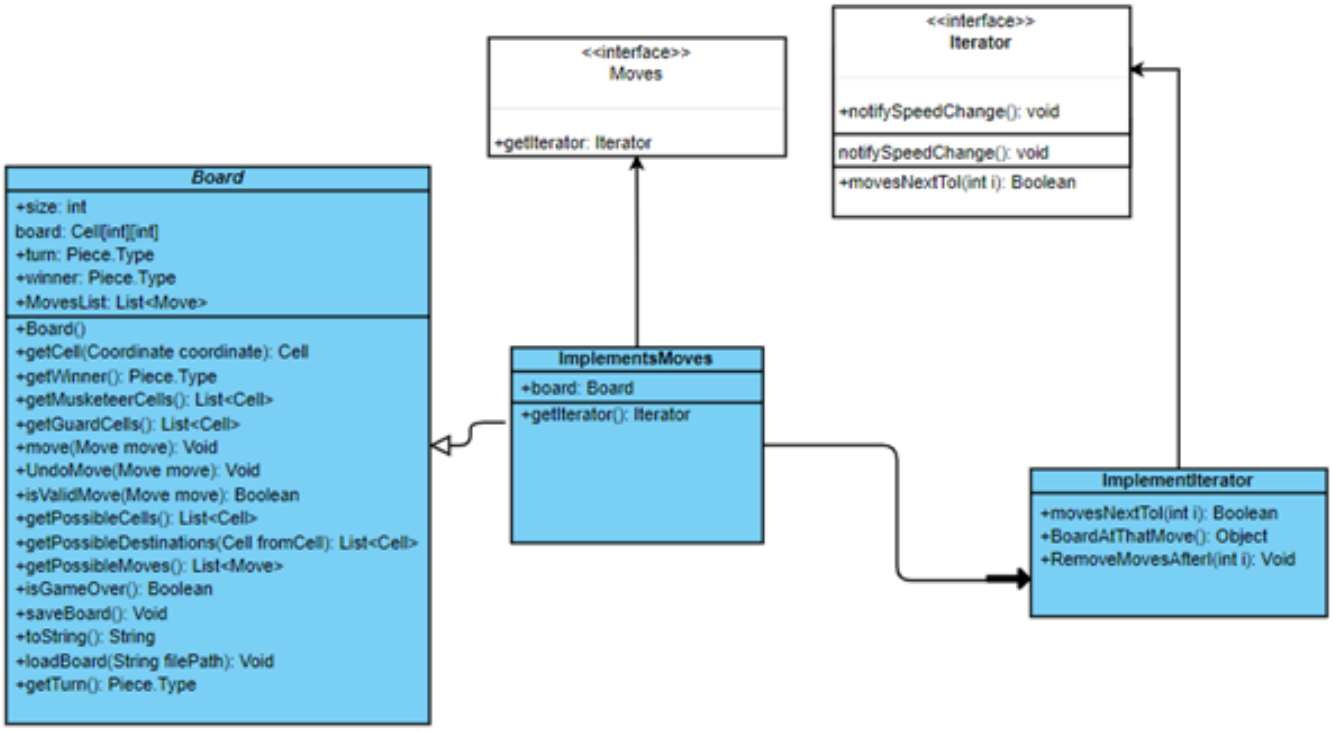
Iterator Pattern:

How it Works/What it Solves:

The iterator pattern provides a way for us to access objects sequentially without having to expose any underlying representation of that object. Due to the abstraction we are able to modify the collection without making changes to any items outside of that specific collection.

We will be using the iterator pattern to iterate through moves. This will allow us to implement the functionality of being able to resort back to any previous move you have made in this round of the game on the current board you are playing. It is essential to implement the iterator pattern here as we do not want to reveal the method in which we look for the move the client wishes to revert back to.

UML Diagram:



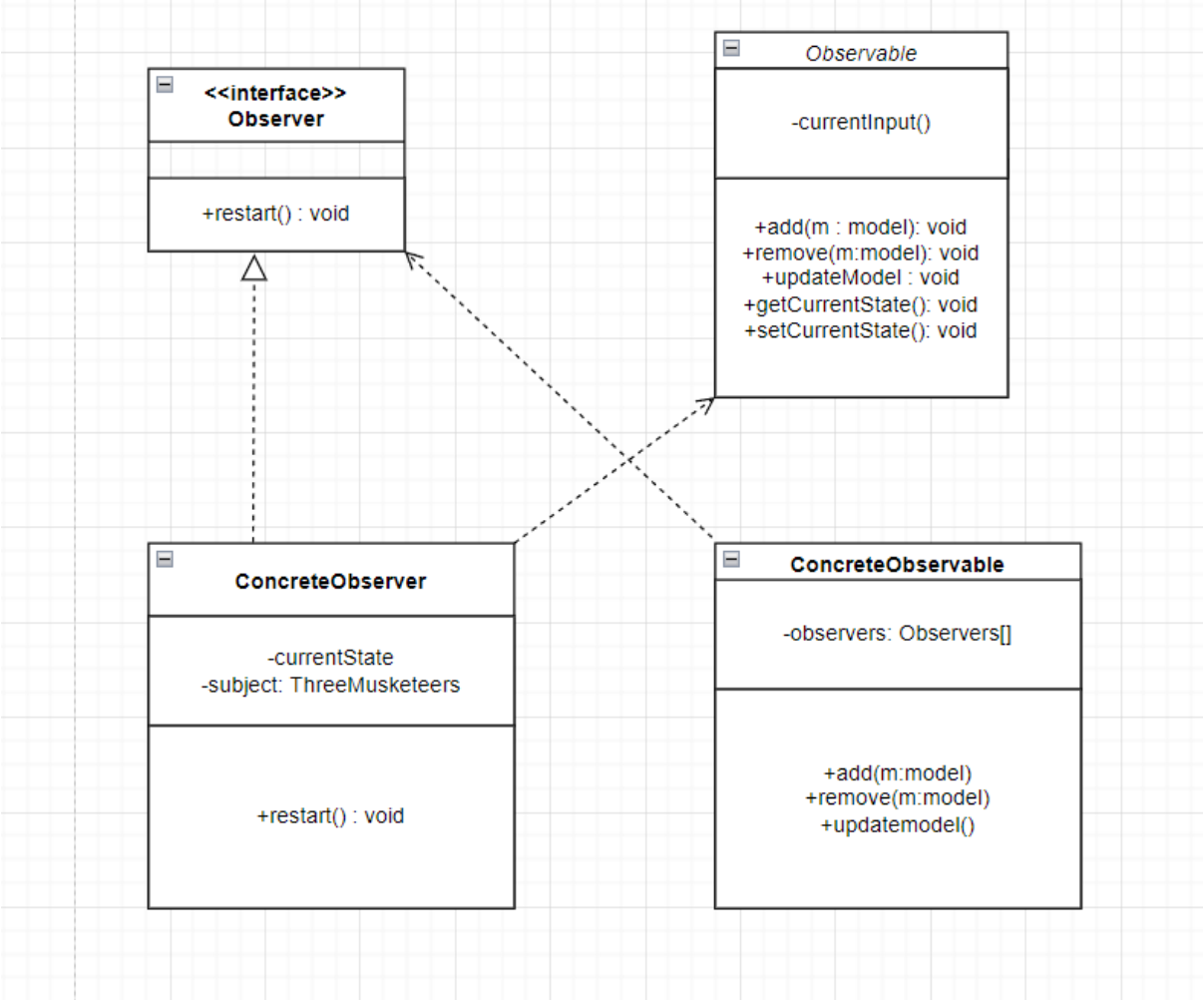
Observer Pattern:

How it Works/What it Solves:

The observer pattern defines a “one-to-many” dependency between objects so that once one object has been modified all of the objects that depend on that specific object are notified and automatically updated as a result.

We are going to implement the observer pattern in regards to restarting the game. The observer will observe the user inputs which are the observables and if it observes that the user wishes to restart the game it will notify the game to restart the board to the initial state.

UML Diagram:



Visitor Pattern:

How it Works/What it Solves:

The visitor pattern represents some operation that is to be performed on an element of an object structure. This pattern allows us to define new operations without changing any of the elements that these new operations would be accessing.

We will be using the visitor pattern to implement a feature for the user to have access to the probability of winning for the current player for the current state of the game. This will be done by accessing the board itself and the number of moves for musketeers remaining without modifying the board itself. Using values derived from these calculations the potential of winning will be calculated.

UML Diagram: