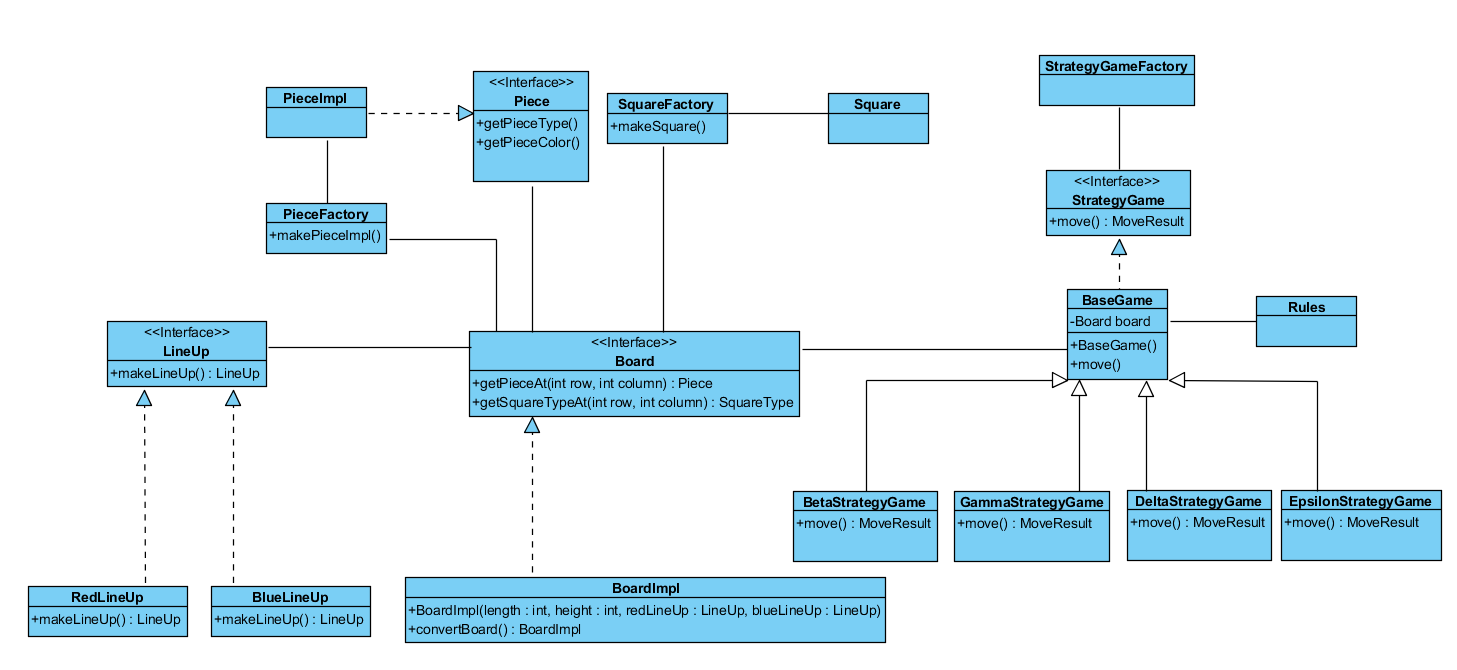
FINAL STRATEGY Design Document:

Zonglin Peng

**Design**



As shown in the UML above, the Strategy pattern, Factory, and Base Game pattern are applied in the strategy game across the versions.

* Strategy pattern: An interface named GameStrategy is created in the code template, and I implemented three concrete strategy class, corresponding to Beta, Gamma, Delta, and Epsilon. They individually represent different behaviors/ rules in the game, Delta has the most completed contents, but Beta has its own unique rules, but Epsilon have some rules that are different from those of Beta, Gamma, and Delta. To use implemt rules in the same classes (i.e. BaseGame and Rules), different versions of the same methods are implemented by passing in different parameters.
* Factory: Factories are created for the objects that can have instances. In my design they are the PieceImpl which implements Piece, Square (aka the coordinate), and the strategy game. The factory helps the other classes to create the instances of the objects without accessing a specific implementing class.
* Abstract Base Game (Template Pattern): An abstract class name BaseGame is created to contain the basic logics in methods of a strategy game in general. When a specific version of game is implemented, the super methods are invoked to help finish the implementation, and the move method in each version overrides the abstract method in the abstract class. The method name Move calls all the abstract methods. They are designed for different phases of a move action. They are pre-game operations, pre-game state checking, checking validity of a move or an attack, post-game state checking, and post-game operations. By using this template pattern, the same implementations in the move method are concatenate into the same super methods, which removes all the duplicate codes. It also makes the code structure clean, and the purpose of each method obvious.

**Considerations**

1. When checking if a side has movable pieces or not, I presume the check should be validated at the start of a move; however, because of the design that a valid pair of coordinates must be provided when invoking a move, the check should be performed elsewhere. Therefore, it is place at the prior move where it checks if the opponent has any movable pieces. If not, then at the current move it will report the current side wins. If the current side has no movable pieces, then the winning will be reported at the end of the same turn after confirming that the opponent has movable pieces.

**Appendix:**

