Use of design patterns.

Design patterns are important part of software development irrespective of programming languages. They are recommended highly to use in development to avoid clutter, make the code more flexible and future proof. We have decided to use GRASP design patterns in our project which are discussed at length in our course. Patterns made our code better in terms of usability, updatability for iterations, and to work in groups.

The first pattern we used in our program is Information expert. We designed most of the classes in such a way that a particular responsibility is assigned to the class which has the most information which is required for that task. We divided the responsibilities across all the classes depending on the information they have. For instance, we have assigned the responsibility for the game play to the mainBoard because mainBoard has all the necessary information and fields required.

The second pattern we used is Protected Variation. For instance, in our program, we have created a method in New Game class named getPlayersDetails which provides details of all the players including player type, color, name, difficulty.

The third pattern we used in the project is Creator. We implemented this on basis of the information that a class have and how closely the class used the created object. We have used this pattern to create pawns and players. We have identified that mainBoard have the initializing data about players from getPlayersDetails from NewGame class and identified that mainBoard class closely uses Player and Pawn objects. This helped in thems of reusability for Pawn and Player.

We also used two design patterns which are low coupling and high cohesion. In our project we have implemented many classes. Each class is designed in such a way that it has minimal coupling or dependency with other classes and fairly independent. We have considerably reduced dependencies between classes which make the application suitable for further development and any future changes will not drastically after the program. This helped in making the program more manageable and easier to work in group.

We think we could have used more patterns or made better use of implemented patterns. We could have used polymorphism for pawn landing alternatives. For example, when a pawn in landed on opponents' pawn, or when a pawn is landed on barricade or an empty space or in the win position. We could also made better use of controller pattern to make it less complex by session to separate logic from interface layer which could enable us to re-use or plugin the logic wherever and whenever needed.