Advanced Software Engineering Topics

Team Name: WinTech

Harmish Patel (110008572) | Sahil Sangani (105170048) | Harsh Patel (110006131)

Lab Session 4

Objectives: To make student familiar with:

1. The concept of system analysis and design
2. UML
3. GRASP design patterns

Consider the following game: Chess

Chess is a two players board game of strategy. Each player has 16 pieces, each with particular rules controlling aspects of its movement on the board and points value. The AI player, (computer artificial intelligence player) employs different strategies for opening (Start of the game), mid game and end game as it makes its moves. Take a look at the "Monopoly" and "Recipe Book" applications (Available on Blackboard) to see how design patterns were considered in order to generate a "responsible" design.

In this assignment you are asked to do the following:

a) **The description of the AI functionality you are designing to play chess.**
Assuming that, a chess game has two players – Human and AI. Artificial Intelligent (The 2nd player) needs to know all about the chess game such as rules for each piece move, checkmate conditions, some special moves (e.g., pawn promotion, en passant, castling), and basic strategies as well as strategies for defeating the opponent. To teach the AI player about these all things, the FICS database can be used as it has about 97000 games, with 7.3 million moves made. This dataset can be used to train our AI player (model) to learn all about the chess game.

The ‘board evolution function’ is used for picking up the highest/lowest value as per the player role. It can be -∞ to ∞, and base on the concept of min-max. Basically, ‘board evolution function’ is a decision tree that keeps track of all moves of each piece of each player. For better performance, we merge with another algorithm, such as ‘alpha-beta pruning,’ which provides a cutoff system to decide whether it should be searched down a branch or not. It optimizes the resources and restricts unnecessary computation. A ‘Winner’ variable is used to determine the winner player (1 for the white win, -1 for black, and 0 for a draw).

b) The 5 GRASP patterns used

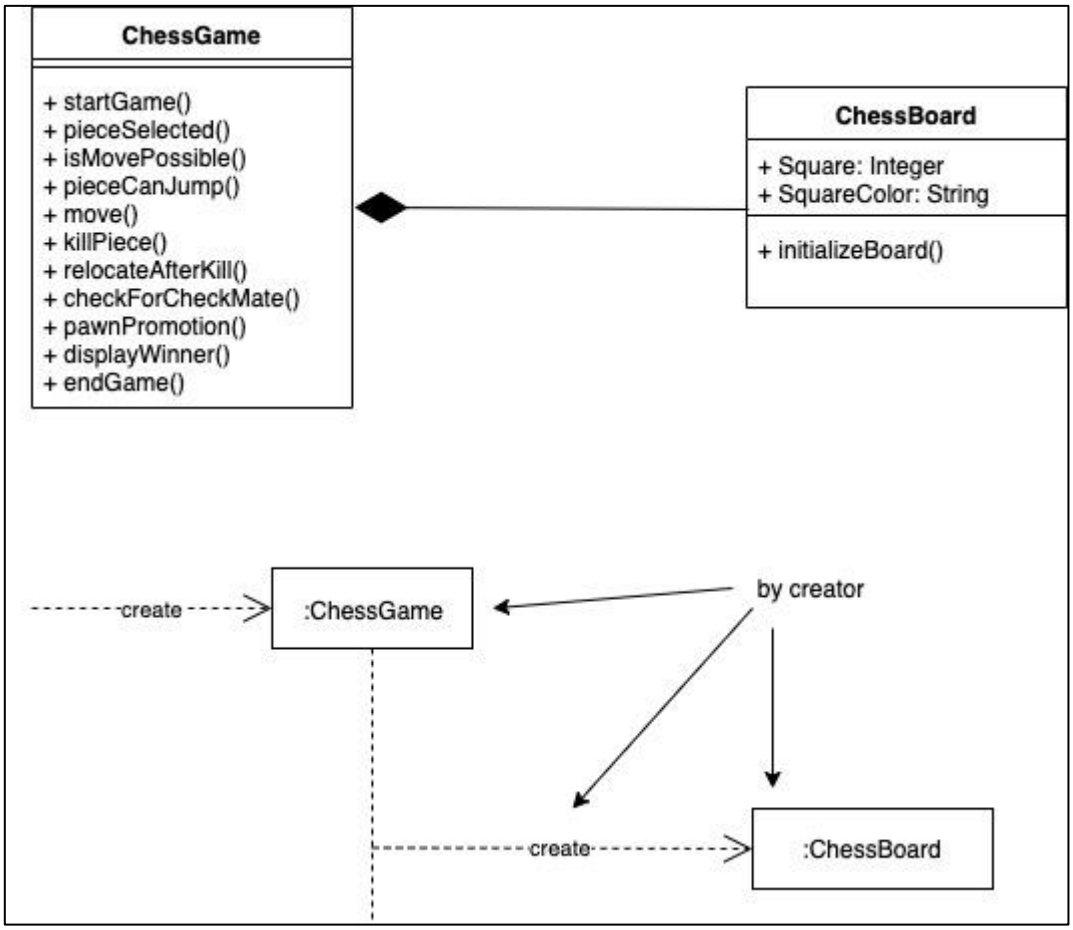

Figure 1: Creator for Chess game

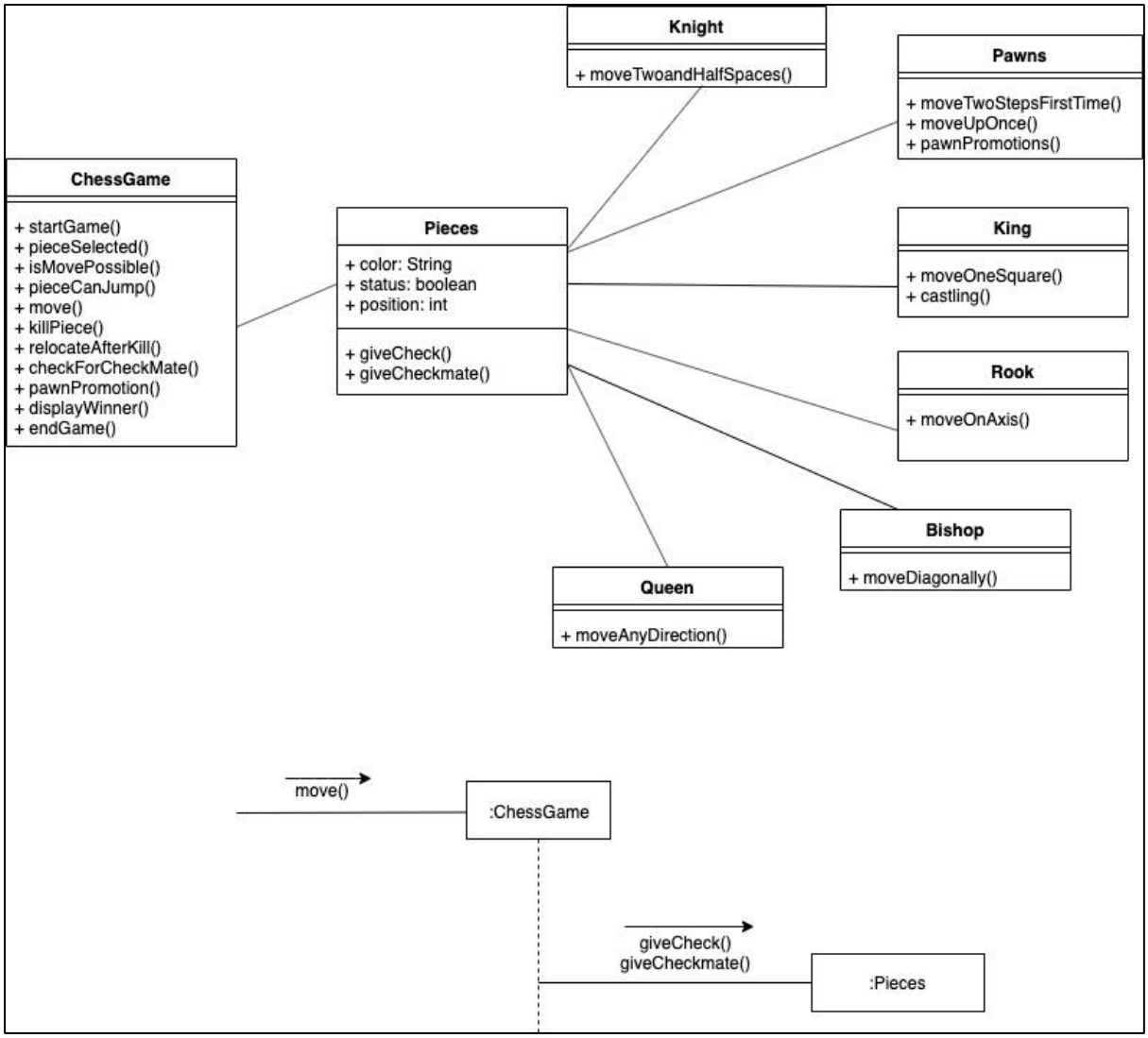

Figure 2: Low Coupling for Chess game

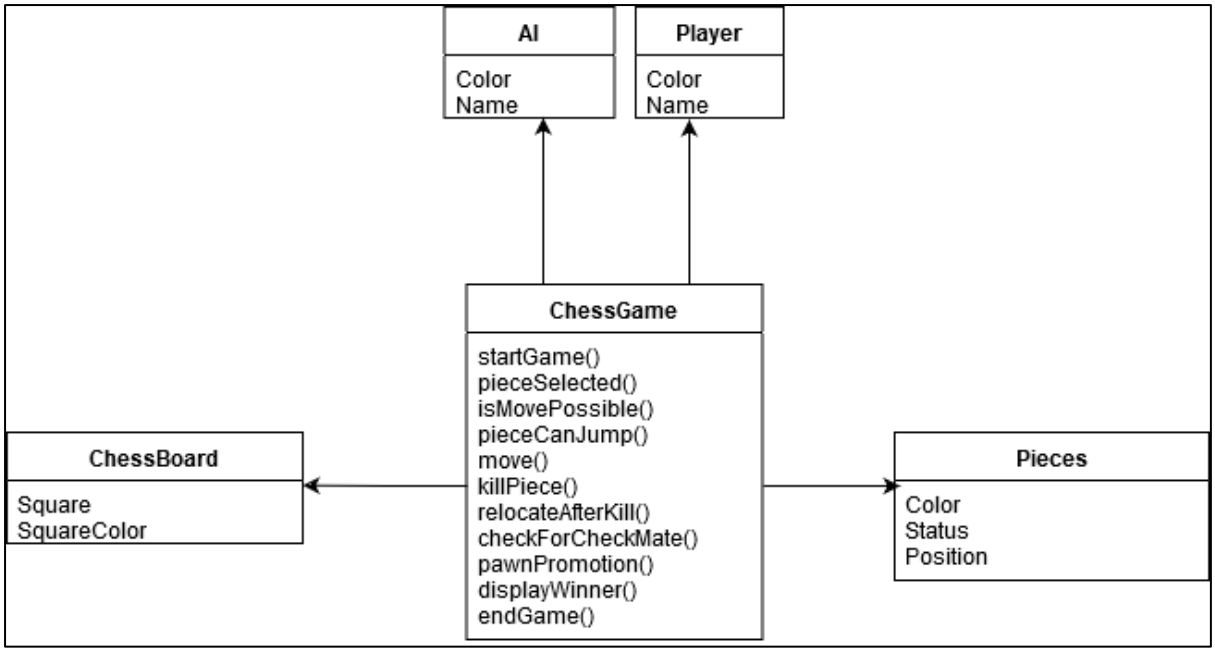

Figure 3: Information Expert for chess game

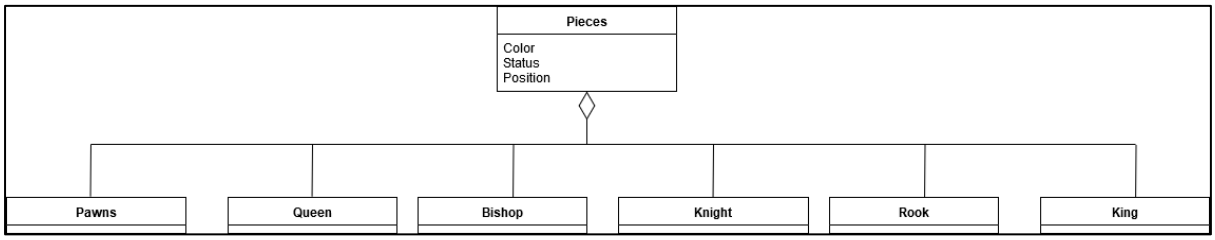

Figure 4: Polymorphism for chess game

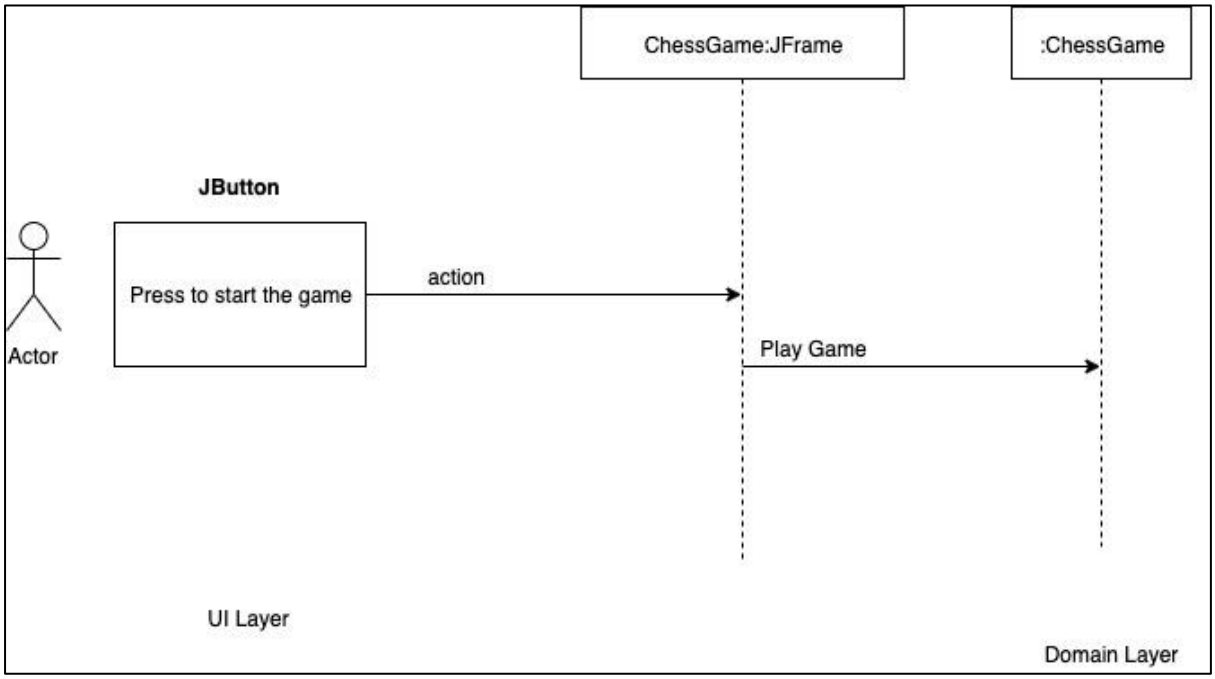

Figure 5: Controller for chess game

c) The complete UML diagram for the AI Player (and associated classes)

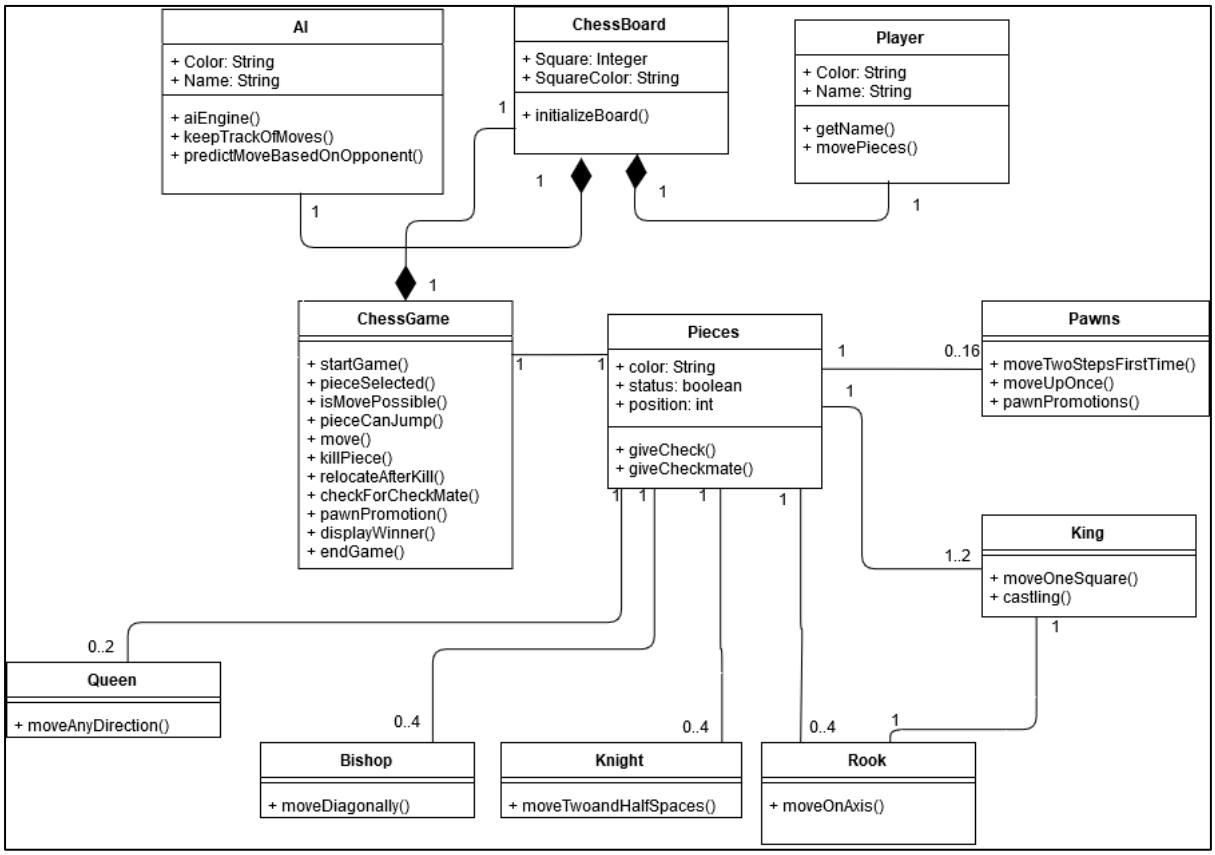

Figure 6: UML diagram for ChessGame

d) The java(skeleton) code files for each class

```
1  package chess;
2
3  public class AI {
4      public String color;
5      public String name;
6
7      public static void aiEngine() {
8
9      }
10
11     public static void keepTrackOfMoves() {
12
13     }
14
15     public static void predictMoveBasedOnOpponent() {
16
17     }
18 }
19
```

Figure 7: AI.java

```java
package chess;

public class chessBoard {
    public int square;
    public String SquareColor;

    public static void initializeBoard() {

    }
}

```

Figure 8: chessBoard.java

```java
package chess;

public class ChessGame extends chessBoard {
    public static void startGame() { }
    public static void pieceSelectd() { }
    public static void isMovePossible() { }
    public static void pieceCanJump() { }
    public static void move() { }
    public static void killPiece() { }
    public static void relocateAfterKill() { }
    public static void chechForCheckmate() { }
    public static void pawnPromotions() { }
    public static void displayWinner() { }
    public static void endGame() { }
}

```

Figure 9: ChessGame.java

```java
package chess;

public class Player {

    public String color;
    public String name;

    public static void name() {

    }

    public static void movePiece() {

    }
}

```

Figure 10: Player.java

```java
package chess;

public class Pieces {
    public static String color;
    public static boolean status;
    public static int position;

    public static void giveCheck() {

    }

    public static void giveCheckmate() {

    }
}
```

Figure 11: Pieces.java

```java
package chess;

public class Pawns extends Pieces {

    public static void moveTwoStepsFirstTime() {

    }

    public static void moveUpOnce() {

    }

    public static void pawnPromotions() {

    }
}
```

Figure 12: Pawns.java

```java
package chess;

public class King extends Pieces{

    public static void moveOneSquare() {

    }

    public static void castling() {

    }

}
```

Figure 13: King.java

```java
package chess;

public class Queen {

    public static void moveAnyDirection() {

    }
}
```

Figure 14: Queen.java

```java
package chess;

public class Knight {
    public static void moveTwoandHalfSpaces() {

    }
}
```

Figure 15: Knight.java

```java
package chess;

public class Rook extends Pieces {

    public static void moveOnAxis() {

    }
}
```

Figure 16: Rook.java

```java
package chess;

public class Bishop {
    public static void moveDiagonally() {

    }
}
```

Figure 17: Bishop.java