ChessGames Improvements

Tesina di ingegneria del software riguardante il miglioramento di un gioco degli scacchi.

Leonardo Nels

Matricola 139908

Dichiaro che questo elaborato è frutto del mio personale lavoro, svolto sostanzialmente in maniera individuale e autonoma.

Introduzione

Questo progetto è finalizzato a proseguire il lavoro di Khald64 riguardo al gioco "ChessGame". Nei paragrafi che seguiranno sono presenti una descrizione del lavoro di Khald64 e una analisi dei requisiti del prodotto finale. A seguire una analisi del codice. Successivamente una descrizione della struttura del gioco finale e del suo funzionamento, terminando con una disamina riguardo una possibile espandibilità futura.

ChessGame

Il gioco sviluppato da Khald64 prevede solo una scacchiera otto per otto caselle e trentadue pezzi del gioco degli scacchi. Tutti i pezzi hanno una immagine che li contraddistingue. Tutti i pezzi sono divisi per colore: bianchi e neri.
I giocatori possono, tramite uso del mouse, cliccare sulla pedina scelta e spostarla sulla casella desiderata. Al rilascio del mouse il pezzo spostato andrà automaticamente a posizionarsi al centro della casella scelta dal giocatore. Se la casella scelta è già occupata da un pezzo dello stesso colore tale mossa verrà annullata. Se la casella di destinazione invece contiene una pedina del colore opposto, questa verrà "catturata".
Prima di commentare il codice serve evidenziare che il programma di base non prevede altre regole al di fuori di quelle sopra menzionate, conseguentemente ogni pezzo viene trattato allo stesso modo, non vi sono limiti di mobilità o altre regole del gioco.

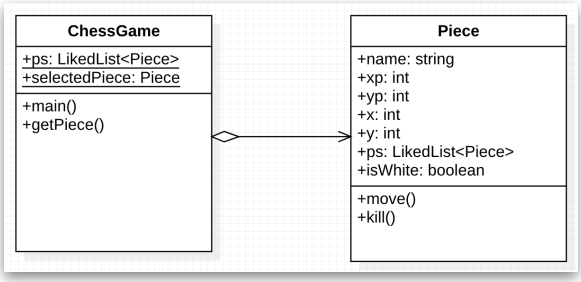
Requisiti del Nuovo ChessGame

Abbiamo già visto come il gioco ChessGame alla base sia molto semplice, ma manca di profondità. Per completarlo sarà necessario per prima cosa introdurre delle regole, come dei limiti di spostamento per i pezzi e regole che definiscano quando una partita finisce che sia vittoria o pareggio.
Inoltre si dovrà poter scegliere di invertire la disposizione delle pedine, come se la scacchiera venisse ruotata, infine si devono implementare più variazioni cromatiche della tavola da gioco.
Serviranno menu sia prima della partita che dopo. Il primo per decidere layout e colore della scacchiera come sopra e per scegliere se giocare o uscire dal gioco;

il secondo per mostrare il risultato della partita appena terminata, decidere se giocare ancora, se modificare la scacchiera o uscire dal gioco.

Il Codice Originale

Il codice di Khald64 prevede l'uso di java swing per il comparto grafico e comprende due sole classi: Piece e ChessGame. La prima prevede una Linkedlist di pezzi necessaria per tenere traccia di tutte le pedine in gioco, una variabile booleana per distinguere le pedine bianche da quelle nere e quattro altre variabili per gestire la posizione degli stessi. Due mantengo salvata la posizione sulle caselle della scacchiera (xp e yp), due gestiscono la posizione in termini di pixel a schermo (x e y). Inoltre la classe Piece presenta due metodi: move e kill. "Move" si occupa del movimento delle pedine mentre "kill" serve a "catturare" (o "mangiare") i pezzi dell'avversario, si tratta semplicemente di una funzione che rimuove tale pezzo dalla lista di pedine in gioco.



Tutto il resto del gioco viene gestito all'interno della classe ChessGame, più nello specifico nel metodo "main". Assegnare le icone da un file png ai trentadue pezzi della scacchiera, creare le trentadue pedine, la creazione dell'interfaccia grafica (scacchiera) tramite JPanel e JFrame, è tutto compreso nel metodo "ChessGame". Infine la selezione di un pezzo viene effettuata tramite un metodo getPiece, sempre di ChessGame, che compara le coordinate del mouse a quelle delle pedine cercando una corrispondenza della lista delle pedine in gioco ps.

Lo Studio del Nuovo Codice

Come anticipato, il codice di Khald64 è molto semplice, ma si appoggia eccessivamente sulla classe principale ChessGame, serve separare molte delle sue funzioni. Essa infatti si occupa di:

- creare le icone per i singoli pezzi
- creare le trentadue pedine
- creare la schermata del gioco tramite uso di JFrame

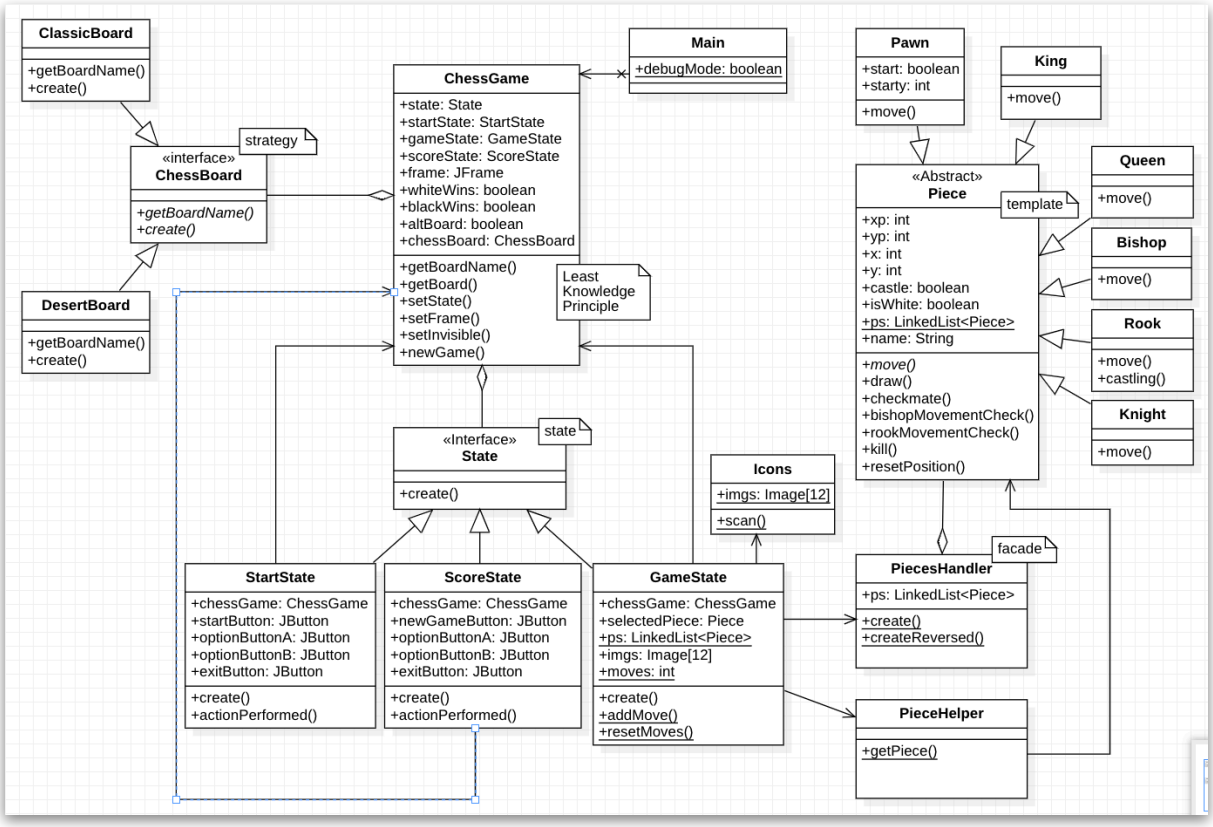
- creare la scacchiera come JPanel
- contenere il metodo getPiece

Ne consegue che la classe ChessGame avrebbe ben cinque ragioni per cambiare, se per esempio, volessimo cambiare le icone, l'ordine dei pezzi, i colori della scacchiera oppure apportare una modifica al metodo getPiece dovremmo sempre mentre mano alla classe ChessGame e questo è male.

La più piccola classe Piece, invece, si occupa solo delle logiche dei pezzi. Nello specifico si occupa dello spostamento con il metodo "move" e della conquista dei pezzi tramite "kill". Il metodo "move" si occupa del movimento per tutti i pezzi senza distinzioni. Se vogliamo limitare il movimento delle pedine secondo le regole servirà modificare qualcosa.

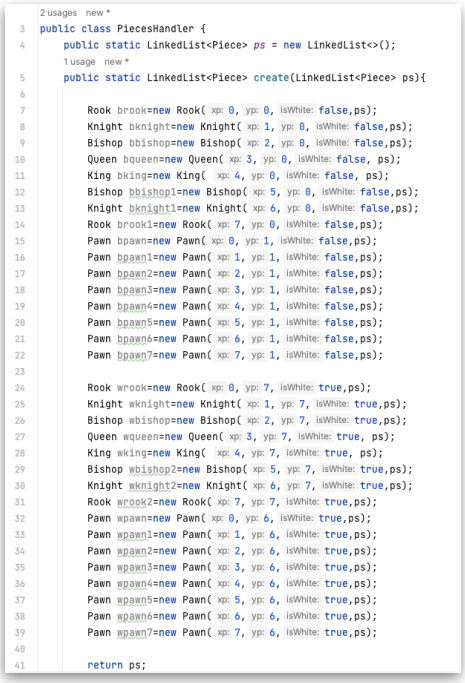
Un primo approccio potrebbe essere quello di implementare la logica per tutti i pezzi dentro a questo metodo. Dopotutto se la tipologia del pezzo (es re o pedone) è definita da una variabile di tipo stringa si potrebbero definire movimenti per ogni tipo di pezzo nello stesso metodo move (per esempio tramite switch case o if a cascata). Un secondo approccio sarebbe quello di specializzare la classe Piece in sette sottoclassi figlie, una per tipo di pezzo. All'interno della classe figlia è poi possibile ridefinire il metodo move per inserire le regole di movimento specifiche per ogni pezzo. Questo secondo approccio è più prolisso, ma il codice finale sarà più facile da comprendere e soprattutto modificare.

Il Nuovo ChessGame



I design pattern

Sono utilizzati quattro design pattern: facade, state, template e strategy.



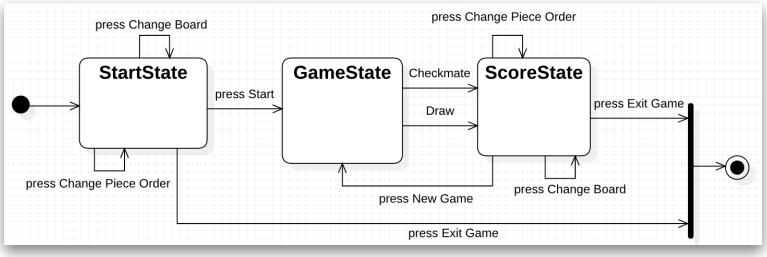
```
2 usages  new *
public class PiecesHandler {
    public static LinkedList<Piece> ps = new LinkedList<>();
    1 usage  new *
    public static LinkedList<Piece> create(LinkedList<Piece> ps){

        Rook brook=new Rook( x: 0, y: 0, isWhite: false,ps);
        Knight bknight=new Knight( x: 1, y: 0, isWhite: false,ps);
        Bishop bbishop=new Bishop( x: 2, y: 0, isWhite: false,ps);
        Queen bqueen=new Queen( x: 3, y: 0, isWhite: false, ps);
        King bking=new King( x: 4, y: 0, isWhite: false, ps);
        Bishop bbishop1=new Bishop( x: 5, y: 0, isWhite: false,ps);
        Knight bknight1=new Knight( x: 6, y: 0, isWhite: false,ps);
        Rook brook1=new Rook( x: 7, y: 0, isWhite: false,ps);
        Pawn bpawn=new Pawn( x: 0, y: 1, isWhite: false,ps);
        Pawn bpawn1=new Pawn( x: 1, y: 1, isWhite: false,ps);
        Pawn bpawn2=new Pawn( x: 2, y: 1, isWhite: false,ps);
        Pawn bpawn3=new Pawn( x: 3, y: 1, isWhite: false,ps);
        Pawn bpawn4=new Pawn( x: 4, y: 1, isWhite: false,ps);
        Pawn bpawn5=new Pawn( x: 5, y: 1, isWhite: false,ps);
        Pawn bpawn6=new Pawn( x: 6, y: 1, isWhite: false,ps);
        Pawn bpawn7=new Pawn( x: 7, y: 1, isWhite: false,ps);

        Rook wrook=new Rook( x: 0, y: 7, isWhite: true,ps);
        Knight wknight=new Knight( x: 1, y: 7, isWhite: true,ps);
        Bishop wbishop=new Bishop( x: 2, y: 7, isWhite: true,ps);
        Queen wqueen=new Queen( x: 3, y: 7, isWhite: true, ps);
        King wking=new King( x: 4, y: 7, isWhite: true, ps);
        Bishop wbishop2=new Bishop( x: 5, y: 7, isWhite: true,ps);
        Knight wknight2=new Knight( x: 6, y: 7, isWhite: true,ps);
        Rook wrook2=new Rook( x: 7, y: 7, isWhite: true,ps);
        Pawn wpawn=new Pawn( x: 0, y: 6, isWhite: true,ps);
        Pawn wpawn1=new Pawn( x: 1, y: 6, isWhite: true,ps);
        Pawn wpawn2=new Pawn( x: 2, y: 6, isWhite: true,ps);
        Pawn wpawn3=new Pawn( x: 3, y: 6, isWhite: true,ps);
        Pawn wpawn4=new Pawn( x: 4, y: 6, isWhite: true,ps);
        Pawn wpawn5=new Pawn( x: 5, y: 6, isWhite: true,ps);
        Pawn wpawn6=new Pawn( x: 6, y: 6, isWhite: true,ps);
        Pawn wpawn7=new Pawn( x: 7, y: 6, isWhite: true,ps);

        return ps;
```

Facade viene implementato su una classe PiecesHandler e serve a incapsulare il "wall of text" del codice originale utilizzato per la creazione delle trentadue pedine del gioco degli scacchi.

La creazione dei pezzi all'interno del PiecesHandler accetta in ingresso una LinkedList di Piece (pezzi) e chiama il metodo di creazione dei singoli oggetti, specificando di volta in volta la posizione sulla scacchiera, se bianco o nero e la lista di pezzi nella quale questi vanno aggiunti.

È inoltre possibile aggiungere altri metodi simili che modifichino la disposizione dei pezzi o il numero degli stessi, in questo caso abbiamo anche un metodo createReversed che capovolge la disposizione delle pedine.

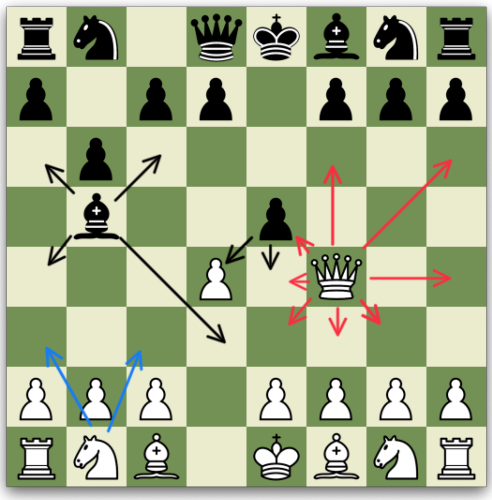
State pattern si rende utile per differenziare i singoli stati di gioco. È possibile in questa maniera dividere il gioco in tre stati: StartState, GameState e ScoreState. Il primo e l'ultimo sono dei menu che permettono all'interno di JFrame di mostrare informazioni come il risultato della partita e bottoni per scegliere tra varie opzioni.



Strategy serve a definire diverse scacchiere, è così possibile aggiungere diverse scacchiere di gioco senza modificare quelle preesistenti. L'interfaccia ChessBoard impone che siano implementati i metodi getBoardName per riconoscere univocamente la scacchiera e create. Create ritorna la scacchiera scelta come JPanel. In questo caso avremo due diverse palette cromatiche: verde (ClassicBoard) e bordeaux (DesertBoard).

Template è il modo più semplice per specializzare le singole pedine. La classe Piece originaria viene trasformata in una classe astratta con il metodo move

astratto. Le classi figlie dovranno specializzarsi implementando ciascuna il proprio metodo move.



Le Pedine

Come detto nel paragrafo precedente la classe originale Piece ora è una classe astratta priva di una implementazione per il metodo move. Ogni classe figlia eredita variabili e metodi da Piece e si preoccupa di implementare il proprio metodo move. Questo permette ai pezzi di muoversi in modo differente imponendo dei limiti, delle regole. Tra tutti i pezzi il più semplice è il cavallo, il quale si muove formando una L di tre caselle, due in una direzione non obliqua e una nella direzione normale alla prima. Il suo metodo move non fa altro che controllare la casella di destinazione, controlla la posizione di questa rispetto alla posizione di partenza e che questa non sia già occupata. Se la casella risulta già occupata controlla il colore di chi la occupa per decidere se annullare la mossa o catturare la pedina.

```
public class Knight extends Piece{
    8 usages  new *
    public Knight(int xp, int yp, boolean isWhite, LinkedList<Piece> ps) { super(xp, yp, isWhite,  n: "Knight", ps); }

    1 usage  new *
    @Override
    public void move(int xp,int yp){
        if(Main.debugMode)System.out.println("Knight "+this.xp+" "+this.yp+" : "+xp+" "+yp);

        if(!(Math.abs(xp-this.xp)==2&&Math.abs(yp-this.yp)==1||Math.abs(xp-this.xp)==1&&Math.abs(yp-this.yp)==2)) {
            xp=this.xp;
            yp=this.yp;
            x=xp*64;
            y=yp*64;
            return;

        }

        if(PieceHelper.getPiece( x: xp*64,  y: yp*64, ps)!=null){
            if(PieceHelper.getPiece( x: xp*64,  y: yp*64, ps).isWhite!=isWhite){
                PieceHelper.getPiece( x: xp*64,  y: yp*64, ps).kill(ps);

            }else{
                x=this.xp*64;
                y=this.yp*64;
                return;
            }
        }
        this.xp=xp;
        this.yp=yp;
        x=xp*64;
        y=yp*64;
    }
}
```

Nuovi metodi aggiunti alla classe astratta Piece aiutano l'implementazione di move per le altre pedine. Tutti i pezzi degli scacchi per muoversi fanno un controllo sulla casella di destinazione verificando che questa permetta un movimento "legale". Purtroppo nessuna pedina è a conoscenza della posizione delle altre. Le regole degli scacchi prevedono che la mossa sia considerata legale se nel percorso non ci sono altre pedine a bloccare la strada. Due nuovi metodi rookMovementCheck e

bishopMovementCheck effettuano tali controlli. La torre sfrutta il primo metodo, l'alfiere il secondo, la regina entrambi.

Il pedone richiede qualche passo in più, ma è comunque semplice. Esso si muove solo di una casella e solo avanti, se trova una pedina davanti a se non può procedere, può catturare solo mangiando in diagonale e non dietro di se. Tramite una variabile booleana permettiamo al pedone di spostarsi di due caselle in avanti se è la sua prima mossa.

Il re si può muovere di una casella in ogni direzione, inoltre, tramite il rookMovementCheck e un metodo castling presente nella classe della torre il re può effettuare l'arrocco (la sua pedina può essere invertita con quella di una torre, purché non vi siano pezzi tra i due e purché entrambi siano alla loro prima mossa, di ciò se ne tiene traccia tramite una variabile booleana castle).

```
23    public boolean draw(){...}
      2 usages  new *
33    public Piece checkmate(){...}
      2 usages  new *
47    public boolean bishopMovementCheck(int xp, int yp){...}
      3 usages  Leonardo Nels *
85    public boolean rookMovementCheck(int xp, int yp){
86        int xdif=xp-this.xp;
87        int ydif=yp-this.yp;
88        if (xdif>0&&ydif==0){
89            for(int i=1; i<xdif; i++){
90                if(PieceHelper.getPiece( x: (xp-i)*64,  y: yp*64, ps)!=null){
91                    return true;
92                }
93            }
94        }
95        if(xdif<0&&ydif==0){
96            for(int i=-1; i>xdif; i--){
97                if(PieceHelper.getPiece( x: (xp-i)*64,  y: yp*64, ps)!=null){
98                    return true;
99                }
100           }
101       }
102       if (xdif==0&&ydif>0){
103           for(int i=1; i<ydif; i++){
104               if(PieceHelper.getPiece( x: xp*64,  y: (yp-i)*64, ps)!=null){
105                   return true;
106               }
107           }
108       }
109       if(xdif==0&&ydif<0){
110           for(int i=-1; i>ydif; i--){
111               if(PieceHelper.getPiece( x: xp*64,  y: (yp-i)*64, ps)!=null){
112                   return true;
113               }
114           }
115       }
116       return false;
117   }
      7 usages  new *
118   public void kill(LinkedList<Piece> ps){...}
      1 usage  Leonardo Nels *
```

```
15    public String getBoardName(){
16        return chessBoard.getBoardName();
17    }
      1 usage  new *
18    public JPanel getBoard(LinkedList<Piece> ps, Image[] imgs){
19        JPanel pn = chessBoard.create(ps, imgs);
20        return pn;
21    }
      5 usages  new *
22    public void setState(State state){
23        this.state=state;
24    }
      6 usages  new *
25    public void setFrame(){
26        frame=state.create();
27    }
      5 usages  new *
28    public void setInvisible(){
29        frame.setVisible(false);
30    }
      1 usage  new *
31    public void newGame(){
32        gameState = new GameState( chessGame: this);
33        scoreState=new ScoreState( chessGame: this);
34        whiteWins=false;
35        blackWins=false;
36        setState(gameState);
37        setInvisible();
38        setFrame();
39    }
      1 usage  new *
40    public ChessGame() {
41        startState = new StartState( chessGame: this);
42        gameState = new GameState( chessGame: this);
43        scoreState = new ScoreState( chessGame: this);
44        if(chessBoard==null){
45            chessBoard=new ChessClassic();
46        }
47        state = startState;
48        setFrame();
```

Piece introduce infine due nuovi metodi, draw e checkmate che eseguono due controlli per dichiarare una vittoria o una patta (conclusione della partita in parità).

Least Knowing Principle

La classe ChessGame, centrale nel codice di Khald64, viene alleggerita di molte responsabilità e soprattutto di motivi per cui potrebbe cambiare. Ora la classe ChessGame gestisce e tiene traccia delle impostazioni globali del gioco. Fornisce metodi per accedere agli oggetti di cui tiene traccia, come la scacchiera chessBoard o i vari stati dello state pattern.

Gli Stati

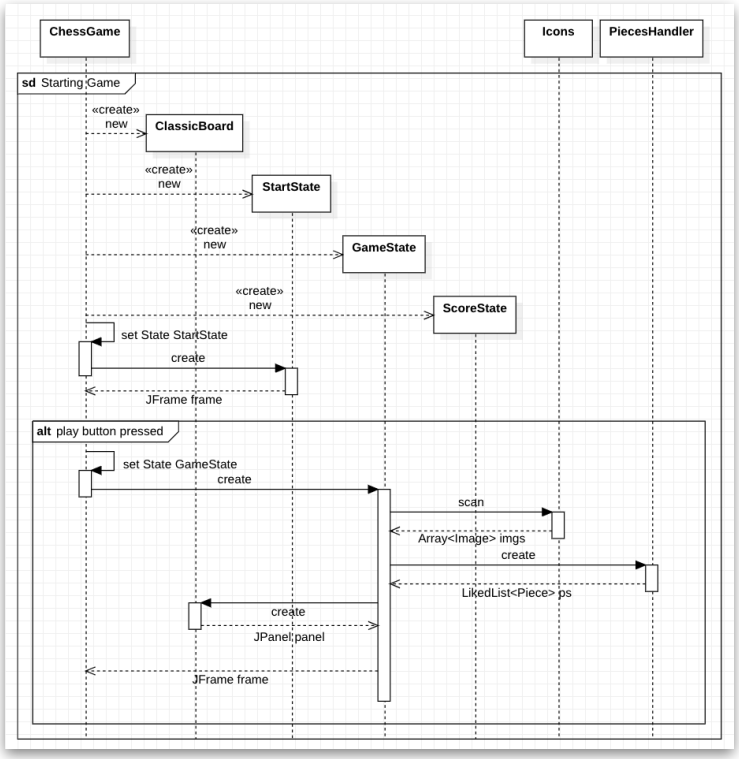
Attualmente gli stati del gioco sono solo tre: StartState, GameState e ScoreState. Tutti gli stati sono classi figlie della classe astratta State, la quale impone di implementare il metodo create. Create non necessita di parametri in ingresso e restituisce un frame JFrame.
Gli stati di inizio e di fine prevedono anche un metodo actionPerformer, chiamato ogni volta che l'utente clicca su un bottone del gioco nel frame attivo.

Come Funziona il Nuovo ChessGame

All'avvio la classe **Main** crea un oggetto di tipo ChessGame e controlla una variabile di debug che se impostata su false impedisce tutte le stampe di debug. Il gioco ha inizio.

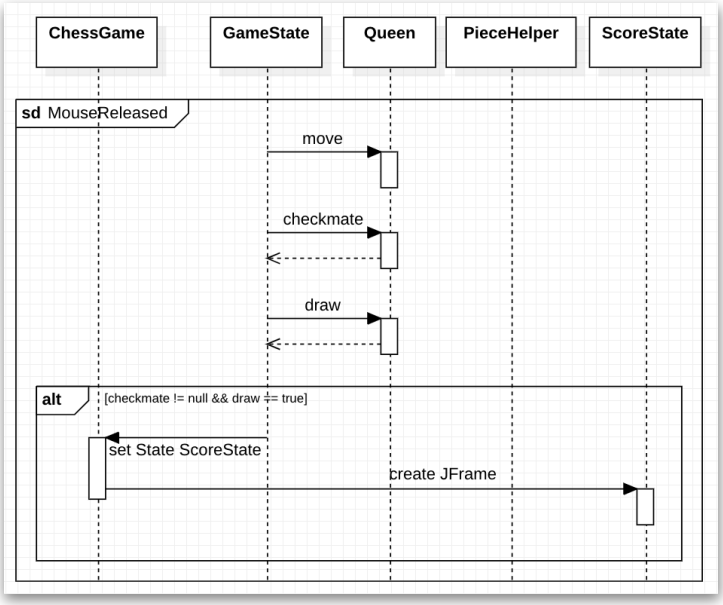
ChessGame crea un oggetto scacchiera e i singoli oggetti stato, imposta come attivo il primo stato, StartState, e invoca il metodo create di StartState per creare l'interfaccia grafica del menu iniziale.

StartState eventualmente cambia scacchiera e inverte i pezzi, poi, se il giocatore decide di iniziare la partita, preme il pulsante START e viene impostato GameState come stato attivo, subito dopo viene invocato il suo metodo create.



Sequence diagram dell'avvio di una partita

GameState, tramite il metodo create appena invocato, per prima cosa crea una lista su cui salvare i pezzi attivi sulla scacchiera e successivamente, tramite una seconda classe, Icons, salva su un array le immagini per le



Sequence diagram del rilascio del mouse per muovere una pedina

pedine; infine crea il frame di gioco da ritornare a ChessGame. Il frame di gioco si occupa di chiamare il PiecesHandler per la creazione dei pezzi di gioco e impone dei comportamenti al movimento del mouse, utili a selezionare, spostare e rilasciare le pedine sulla scacchiera. Quando una pedina viene selezionata il metodo contenuto dentro una classe PieceHelper cerca nella lista di pezzi la pedine che il giocatore ha selezionato. La posizione di tale pezzo viene costantemente aggiornata mentre il giocatore sposta la pedina. Quando il giocatore rilascia la pedina viene chiamato il metodo move specifico del pezzo spostato, infine vengono controllate le condizioni di vittoria o pareggio della partita. La partita termina con una vittoria quando nella lista di pezzi in gioco rimane un solo re. Si verifica un pareggio se nella lista di pezzi sono presenti solo i due re o se vengono effettuate cinquanta mosse consecutive senza catturare una pedina all'avversario. Se la partita si conclude lo stato attivo cambia da GameState a ScoreState.

ScoreState, molto simile a StartState, mostra al centro il risultato della partita appena conclusa e permette al giocatore di cambiare scacchiera, invertire le pedine o decidere se avviare una nuova partita o uscire dal gioco.

Espandibilità Futura

L'utilizzo di design pattern permette di espandere facilmente il gioco. Prendiamo ad esempio la dama. Per le pedine si possono creare ad esempio due classi, Dama e Damone, figlie di Piece. La scacchiera sarebbe la stessa, ma nulla ci impedirebbe di aumentare il numero di caselle all'interno di una nuova classe figlia di ChessBoard. Per la gestione del gioco stesso è sufficiente creare un

nuovo stato (esempio DamaState) e aggiungere ai menu i pulsanti necessari ad entrare in questo stato. La classe DamaState assieme alle classi pedine permettono di aggiungere tutte le dinamiche e le regole del gioco. Per altre funzioni come la creazione dei pezzi e delle icone si può appoggiare tranquillamente alle già presenti classi PiecesHandler e Icons. L'introduzione di nuovi stati per nuove modalità di gioco potrebbe avvenire applicando ai vari GameState un design pattern factory.
Lo stesso vale anche altri giochi come Othello o altre famose modalità degli scacchi come Blitz.