Ročníkový projekt - Dáma
Gymnázium Arabská

Felix Navrátil
Vyučující: Mgr. Jan Lána
Předmět programování

Duben 2022

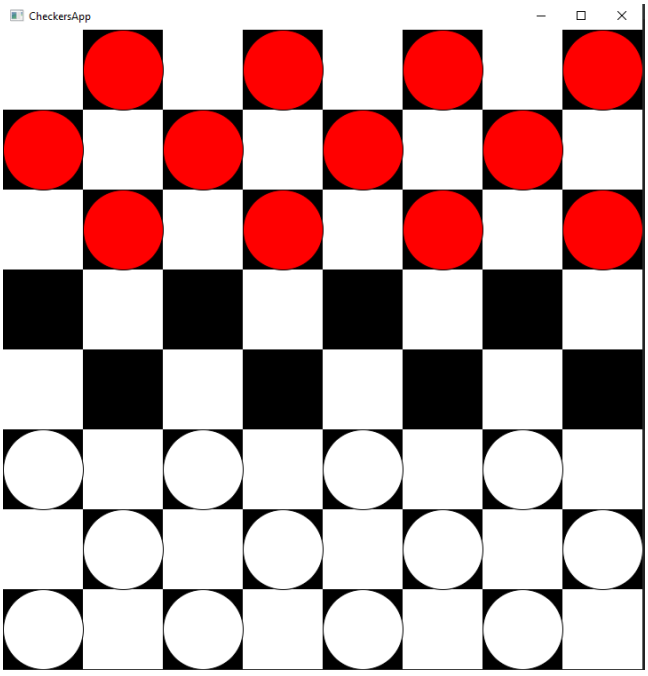
Contents

1 Anotace

Tento ročníkový projekt zahrnuje, jak jsem naprogramoval zjednodušenou dámu. Zde vás seznámím s problémy např.: jak vyřešit pravidla pomocí počítače, pohyb figurek nebo střídání hráčů. A jak jsem problémy vyřešil nebo nevyřešil.

2 Úvod

Za ročníkový projekt jsem si vybral dámu, protože mě baví ji hrát a také jsem si myslel že to nebude tak složité. Cíl mého ročníkového projektu byl naprogramovat dámy. Jako IDE jsem používal IntelliJ idea a jako programovací jazyk jsem používal Java a na grafiku jsem používal JavaFX. Při programování jsem narazil na mnoho problémů(např.: Rozmístění figurek, střídání hráčů nebo pohyb figurek) a zde vás seznámím jak jsem problémy vyřešil nebo nevyřešil.



3 Pravidla a obecné informace

Dáma je hra pro dva hráče a hraje na šachovnici 8x8. Každý hráč má vlastní sadu figurek, jeden má bílé a druhý má červené. Figurky se mohou pohybovat diagonálně o jedno pole, nebo můžou přeskočit nepřátelskou figurku a tím ji zabijí a vyřadí ze hry. V normální dámě hra končí, když jednomu hráči nezbývají žádné figurky, ale podle mých pravidel hra končí, když se jeden hráč dostane na poslední pole relativně z pohledu hráče.

4 Vytváření šachovnice

Jako první jsem definoval 3 parametry: 1. *tileSize* tento parametr značí jak velké je jedno pole šachovnice, 2. *width* značí kolik polí bude šachovnice mít na šířku, 3. *height* značí kolik polí bude mít šachovnice na výšku. Potom jsem Vytvořil třídu *Tile*, která je potomkem třídy *Rectangle*. Tam jsem nadefinoval konstruktor a metody *setPiece*, *hasPiece*, *getPIece*. Metoda *setPiece* připíše danou figurku na dané pole, metoda *hasPiece* zjistí jestli dané pole má figurkunebo ne a metoda *getPiece* se použije tehdy, když chceme nakládat i s metodama, které jsou ve třídě *Piece*. Konstruktor:

```
setWidth(tileSize);
    setHeight(tileSize);
    setLayoutX(x * tileSize);
    setLayoutY(y * tileSize);
    setFill(light ? Color.WHITE : Color.BLACK);
```

Šachovnici jsem potom vytvořil pomocí dvou for smyček. Jestli je součet X a Y dělitelný 2 tak je pole bíle jestli ne tak je pole červené. src:

```
for (int y = 0; y < 8; y++) {
    for (int x = 0; x < 8; x++) {
        Tile tile = new Tile((x + y) % 2 == 0, x, y);
        board[x][y] = tile;
        tileGroup.getChildren().add(tile);


    }
}
```

5 Vytváření figurek

Vytvořil jsem třídu *Piece* a enum *PieceType*[1], v enumu PieceType jsem nadefinoval typ figurky(bíla a červená) a int *moveDir*, který definuje jak se jaka figurka bude pohybovat po vertikální ose.

```
RED(1), WHITE(-1);

final int moveDir;

PieceType(int moveDir) {
    this.moveDir = moveDir;
}
```

V třídě *Piece* jsem nadefinoval konstruktor. V konstruktoru jsem vytvořil kruh o poloměru rovném *(1/2)*tileSize*, kterému jsem dal barvu podle typu figurky.

```
piece.setFill(type == PieceType.RED ? Color.RED : Color.WHITE);
```

Také jsem našel[1] pravidla co se bude dít, když se na figurku klikne nebo s ní bude chtít někdo pohnout. V kódu to vypadá takto

```
setOnMousePressed(e -> {
        mouseX = e.getSceneX();
        mouseY = e.getSceneY();
        //e.getSceneY/X zjistí momentalni polohu figurky když se na ní klikne
    });

    setOnMouseDragged(e -> {
        setLayoutY(e.getSceneY() - mouseY + oldY);
        setLayoutX(e.getSceneX() - mouseX + oldX);

        //oldY/X je zacatecni poloha figurky
    });
```

V třídě *Piece* jsem také napsal metody [1]: *move, getOldX, getOldY, abortMove, getType.* Metoda *move* přemístí figurku na její staré koordinace(*oldX/Y*, metoda *getOldX* vrátí hodnotu oldX stejně jako *getOldY*, která vrátí hodnotu *oldY*. Metoda *abortMove* vrátí figurku na její staré koordinace(*oldX,OldY*) a metoda *getType* zjistí jaký je typ figurky.

5.1 Rozmístění figurek

Figurky jsem rozmístil pomocí stejné for smyčky, kterou jsem použil na vytvoření šachovnice. Figurky jsem položil na černá pole vždy na 3 první řádky relativně od pohledu hráče.

6 Pohyb[1]

Pohyb figurky je rozdělen do dvou metod, *tryMove* a *makePiece*. Metoda *tryMove*, určuje podmínky kam se figurky mohou pohybovat a kam ne a metoda *makePiece* jenom připíše danou figurku na dané pole a odstraní ji z předešlého pole a to jsem udělal pomocí funkce *switch*.

6.1 Žádný

Žádný pohyb může nastat jedině když se nesplní podmínka pro pohyb *normální* nebo *zabití*. V metodě *tryMove* vypada kód takto:

```
if (board[newX][newY].hasPiece() || (newX + newY) % 2 == 0) {
        return new MoveResult(MoveType.NONE);
    }
```

Tento kód zjistí jestli pole, na které chce hráč figurku upustit, má už k sobě připsanou figurku nebo má bílou barvu. Jestli figurka tyto podmínky splňuje tak se zavolá tahle část kódu:

```
case NONE:
    piece.abortMove();
    break;
```

6.2 Normální

Normální pohyb figurky je o jedno místo diagonálně dopředu. Pomocí kódu je to napsáno takhle

```
\cite{AlmasBaimagambetov}!!!!
    int startX = boardCoords(piece.getOldX());
    int startY = boardCoords(piece.getOldY());

    ((newX - startX) == 1 || (newX - startX) == -1 )
    && newY - startY == piece.getType().moveDir

    //startX/Y jsou koordinace na šachovnici typu a1 b2 nebo f6
    //getOldX/Y je metoda, která vrátí hodnotu oldX/Y
    //movedir je podle buď -1 nebo 1 podle typu figurky
    //newX/Y je poloha ve které se figurka právě nachází
```

Tento kód udělá přesně to, co je v pravidlech. Zjistí jestli je nová poloha figurky vzdálená od staré polohy figurky -1 nebo 1 pole a jestli je vertikalní poloha -1 nebo 1 podle typu figurky. Jestli tyto podmínky figurka splňuje tak se zavolá tahle část kódu:

```
case NORMAL:
    piece.move(newX, newY);
    board[startX][startY].setPiece(null);
    board[newX][newY].setPiece(piece);
    currentPlayer = redPlayer;
    break;
```

6.3 Zabití

Figurka "zabije" nepřátelskou figurku, když jí přeskočí. To znamená, že se pohne o 2 místa po ose X a o 2 místa po ose Y. To je napsáno v metodě *tryMove* takto:

```
\cite{AlmasBaimagambetov}
    else if (((newX - startX) == 2 || (newX - startX) == -2 )
    && newY - startY == piece.getType().moveDir * 2) {

            int x1 = startX + (newX - startX) / 2;
            int y1 = startY + (newY - startY) / 2;

            if (board[x1][y1].hasPiece()
            && board[x1][y1].getPiece().getType() != piece.getType()) {
                return new MoveResult(MoveType.KILL, board[x1][y1].getPiece());
            }
        }
        //x1/y1 jsou souřadnice vyřazené figurky
```

První podmínka značí, kam se může figurka pohnout tedy rozdíl současných souřadnic figurky a počátečních souřadnic figurky se musí rovnat 2 nebo -2 a vertikální souřadnice se musí rovnat 2**moveDir* a to je buď -2 nebo 2. Druhá podmínka značí, kdy se první podmínka může uskutečnit, a to jen tehdy když je na souřadnicích *x1* a *y1* figurka jiné barvy. V metodě *makePiece* kód vypada takto:

```
case KILL:
    piece.move(newX, newY);
    board[startX][startY].setPiece(null);
    board[newX][newY].setPiece(piece);
    Piece otherPiece = result.getPiece();
    board[boardCoords(otherPiece.getOldX())][boardCoords(otherPiece.getOldY())]
    .setPiece(null);
    pieceGroup.getChildren().remove(otherPiece);
    currentPlayer = redPlayer;
    break;
```

7 Střídání hráčů

Vytvořil jsem si třídu *Player* a nadefinoval jsem v ní konstruktor:

```
PieceType allowedPieceType;

public Player(PieceType allowedPieceType){
    this.allowedPieceType=allowedPieceType;
}
```

Potom jsem si ve třídě *HelloApplication* vytvořil 3 objekty třídy *Player*:

```
public Player currentPlayer=null;
public Player redPlayer = new Player(PieceType.RED);
public Player whitePlayer = new Player(PieceType.WHITE);
```

Hrát může vždy jen *currentPlayer*, který je ze začátku roven objektu *whitePlayer*. Ale to se změní po tahu *normální* nebo *zabití*. A zde můžete vidět podmínky, podle kterých se určuje jaký typ figurek může být tažen.

```
if (currentPlayer == whitePlayer && PieceType.WHITE == type)
if (currentPlayer == redPlayer && type == PieceType.RED)
```

Jestli se nesplní tyto podmínky, tak funkce *switch*, v každém případě(*NORMAL, KILL, NONE*) bude volat metodu *abortMove*.

8 Výhra

Podle pravidel hráč vyhraje tehdy, když se dostane na poslední pole šachovnice. To jsem pomocí počítače vyřešil takto:

```
if (newY == 7 || newY == 0) {
    Label l = new Label("WIN");
    Label p = new Label(" ");
    p.setTranslateY(0);
    p.setTranslateX(0);
    p.setPrefSize(width * tileSize, height * tileSize);
    l.setTranslateX(tileSize);
    l.setTextFill(Color.BLUE);
    l.setFont(new Font("Arial", 300));
    System.out.println("vyhra");
    pieceGroup.getChildren().addAll(l, p);
}
```

Tento kód se zavolá pokaždé když se figurka pohne pohybem *NORMAL* nebo *KILL*. *Label l* má za účel napsal velkým písmem na okno *WIN* a *Label p* má za účel zabránit hráči aby klikal na figurky a pohyboval s nimi.

9 Slepé cesty

9.1 První pokus

Měl jsem nápad, že každé samostatné pole a každá figurka bude tlačítko. Všechny tlačítka, které sloužili jako pole jsem jsem vypnul pomocí metody *setDisable(true)*, aby je hráč nemohl zmáčknout . Když by hráč klikl na figurku tak by se změnila metoda z *setDisable(true)* na *setDisable(false)*, ale jen u polí, které jsou jedno pole diagonálně od figurky. Když by hráč klikl na dané pole tak by se figurka přemístila na lokaci daného pole. Tento nápad jsem nepoužil, protože jsem nevěděl jak mám udělat podmínky pro přeskakování figurek a jak mám poznat jestli pole má k sobě připsanou figurku.

9.2 Druhý pokus

Na rozdíl od prvního pokusu jsem šachovnici vytvořil stejně jako v kapitole *Vytváření šachovnice*, ale jako figurky jsem znova použil tlačítka. Pohyb jsem chtěl udělat podobně jako v kapitole *Pohyb*, ale nevytvořil jsem si k tomu metody, které byli jak jsem zjistil klíčové. A to byl problém, kvůli kterému jsem tento pokus vzdal.

10 Závěr

Jako téma ročníkové práce jsem si zvolil deskovou hru dámu, protože jsem si myslel, že to nebude až tak složité. Avšak když jsem začal programovat, tak jsem si uvědomil, že to co i dítě 2. třídy pochopí za několik minut, není tak jednoduché vysvětlit počítači. Při programování jsem narazil na mnoho problémů, a většina z nich mě velmi zaujala.

11 Zdroje

https://www.youtube.com/watch?v=6S6km5duBrM
https://www.tutorialspoint.com/javafx/layout_gridpane.htm
https://jenkov.com/tutorials/javafx/button.html
https://jenkov.com/tutorials/javafx/gridpane.html
http://www.java2s.com/ref/java/javafx-gridpane-create-chess-board.html
https://www.youtube.com/watch?v=YaDkj-bqcj8&t=106s
https://www.youtube.com/watch?v=8ECA5oowJhA

References

[1] Almas Baimagambetov. Javafx game: Checkers, 2016.