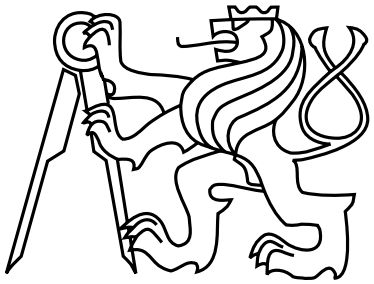
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor's Project

# Checkers with Artificial Intelligence

*Matěj Doležal*

Supervisor: prof. RNDr. Olga Štěpánková, CSc.

Study Programme: Open informatics, Bachelor programme

Field of Study: Computer and Information Science

July 8, 2015

# Aknowledgements

I would like to thank my supervisor prof. RNDr Olga Štěpánková, CSc for giving me the opportunity to develop this application under her guidance and valuable feedback. Furthermore, I would like to thank the university and others who encouraged me throughout the studies.

# Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací..

V Praze 1. 5. 2015 .....

# Abstract

1. Find out how artificial intelligence is used in checkers and draughts implementations currently available. 2. Think of the best possible ways of using artificial intelligence for helping the player to improve his/her skill and performance in the game of checkers. 3. Implement checkers/draughts with intuitive interface that allows player to: Play checkers with all different board setups (default board with 12+12 pieces) including also custom start with specific configuration. Set up time for each round. Provide help for next possible moves. Save and load unfinished game. Game control should be ready for simple utilization of alternative actuators (joystick, stepping carpet etc. - connecting the corresponding drivers is not part of this work). 4. Test the final implementation.

**Key words:** Checkers, Draughts, Java standalone application, Minimax, Artificial intelligence

# Abstrakt

1. Zjistěte, jaké základní cíle si kladou aktuálně dostupné programy, které umožňují hrát deskovou hru dáma a využívají umělou inteligenci. 2. Zvažte, jak lze nejlépe využít metody umělé inteligence pro zdokonalení hráčových herních schopností. 3. Implementujte hru dáma s intuitivním ovládáním, které umožňuje hráči následující: Hrát hru z výchozího herního stavu (12+12 herních kamenů), ale také umožňuje nestandartní rozložení herních kamenů. Nastavit čas pro jednotlivá kola. Získat nápovědu pro možné tahy. Načítat a ukládat rozehrané hry. Ovládání hry by mělo umožňovat snadné připojení nestandartních aktuátorů (dodaní ovladačů pro tyto aktuátory však není součástí této práce) 4. Výslednou implementaci otestujte.

**Klíčová slova:** Dáma, Java standalone aplikace, MiniMax, Umělá inteligence

# Contents

# List of Figures

# Chapter 1

# Introduction

Draughts is a British English word for a board game well known under the word Checkers that comes from American English. It is a strategy game for two players that involves diagonal moves of uniform game pieces and mandatory captures by jumping over opponent's pieces. The game has a long history. I discovered a sources with short descriptions [1] and also with detailed description of this game's history[2].

## 1.1 Rules

I did my own little research and discovered there are many different sets of rules for Checkers. Basically every country has different rules and that gets confusing. I have decided to use rules for English Draughts [4]. Exact description of rules is quite long so let me explain only the most important parts.

- THE MOVES
  There are fundamentally 4 types of move: the ordinary move of a man, the ordinary move of a king, the capturing move of a man and the capturing move of a king.
- Ordinary Move Of A Man 1.1
  - An ordinary move of a man is its transfer diagonally forward left or right from one square to an immediately neighbouring vacant square.
  - When a man reaches the farthest row forward (known as the "king-row" or "crown-head") it becomes a king, and this completes the turn of play.
- Ordinary Move Of A King 1.1
  - An ordinary move of a king (crowned man) is from one square diagonally forward or backward, left or right, to an immediately neighbouring vacant square.
- Capturing Move Of A Man 1.2
  - A capturing move of a man is its transfer from one square over a diagonally adjacent and forward square occupied by an opponent's piece (man or king) and

on to a vacant square immediately beyond it. A capturing move is called a "jump". On completion of the jump the captured piece is removed from the board.

- Capturing In General 1.2
  - If a jump creates an immediate further capturing opportunity, then the capturing move of the piece (man or king) is continued until all the jumps are completed. The only exception is that if a man reaches the king- row by means of a capturing move it then becomes a king but may not make any further jumps until their opponent has moved. At the end of the capturing sequence, all captured pieces are removed from the board.
  - All capturing moves are compulsory, whether offered actively or passively. If there are two or more ways to jump, a player may select any one that they wish, not necessarily move which gains the most pieces. Once started, a multiple jump must be carried through to completion. A man can only be jumped once during a multiple jumping sequence.
- Capturing Move Of A King 1.2
  - A capturing move of a king is similar to that of a man, but may be in a forward or backward direction.
- Definition of win
  The game is won by the player who can make the last move; that is, no move is available to the opponent on their turn to play, either because all their pieces have been captured or their remaining pieces are all blocked.

Rules in real life checkers are little bit different. For instance when player does not make his move before his time runs out he looses his game, in my application random move is played on his behalf. Also players in my application do not have to worry about overlooking possible jumps. It is impossible to play a move when jump is possible. My implementation does not include end of game by draw, since there are no player profiles, one can just close the game.

## 1.2 Purpose of the thesis

Purpose of this Bachelor project is to create a stand alone application that offers great learning opportunities for new or inexperienced players. Its controlling should be intuitive with graphical interface that is simple and easy to use.

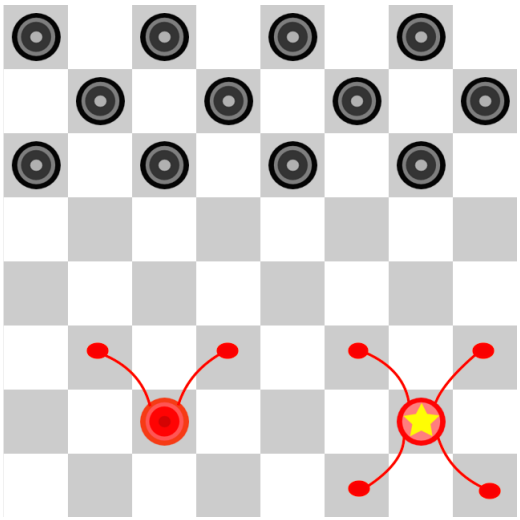


Figure 1.1: Move of a man and move of a king.

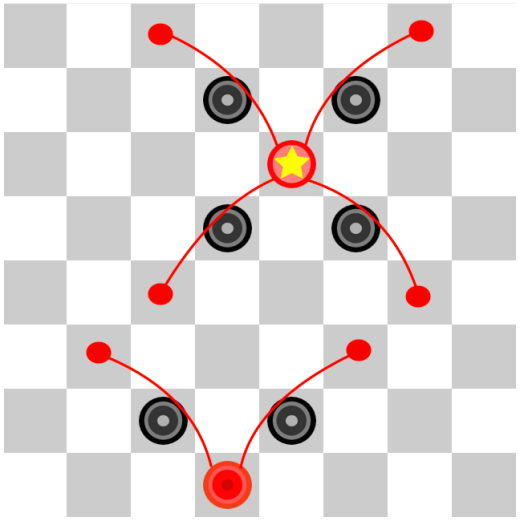


Figure 1.2: Capturing move of a man and capturing move of a king.

# Chapter 2

# Tools

## 2.1 Software

Before the development itself, major decisions have to be made. Such as the programming language, integrated development environment, and the algorithms the developer needs for his project, in this case, algorithm represents the artificial intelligence player.

### 2.1.1 Language of choice

After creating a simple implementation model and knowing the exact requirements for my project I needed to choose a programming language. Language with libraries for sophisticated data structures (ArrayList, HashMap, etc.), that I use for storing all the possible moves for each round. Also libraries for easy to understand object code development and graphical user interface that looks good and is simple to create.

I was deciding between java and c++. At this point it, was mostly about sympathy. Since I am more experienced in java, my decision was simple.

### 2.1.2 Integrated development environment (IDE)

Before the development phase starts it is important to choose an IDE that is well designed and helpful. People usually start using some IDE and then they stick with it. I use Eclipse at work and Netbeans for school and private projects. Netbeans for me offers a better developing environment, easier debugging mode, and I was already familiarized with its GUI development tool. For my bachelor thesis I decided to use Netbeans 8.0 for Mac.

## 2.2 Artificial Intelligence

From the very beginning it was obvious that this project will offer the artificial intelligence opponent. Since the main goal of this project is to improve player's game skill I was sure I wanted to implement at least three different levels of difficulty. Reason for that is very simple. Implementing different levels of difficulty gives the player a better opportunity to

improve. I wanted the differences between each difficulty to be noticeable so I designed difficulty implementations as follows.

- Easy Player 2.2.1
- Medium Player 2.2.2
- Hard Player 2.2.3

### 2.2.1 Easy level player implementation

This level was designed to be very easy to defeat, to give the player the opportunity to familiarize himself with the rules. At this level, the player's move decisions are completely random. The implementation simply finds all the possible moves on the board and randomly chooses one. Randomization element is obtain by java function Math.random().

### 2.2.2 Medium level player implementation

Medium player is smarter than previous difficulty but not even close to artificial intelligence. It creates the pool of all possible moves on the board and evaluates every single move and choose the one with the highest score. The evaluation process is simple. The move can obtain points in four different sections. The sections are:

- Score obtained for superior moves such as double jump and triple jump
  - Double jump is scored by 20 points.
  - Triple jump is scored by 40 points.
- Score obtained for the final position of the move
  - For moves ending on the side of the board 4 points.
  - For moves ending on the opposite side of the board and thus becoming a king 20 points.

Evaluation was optimized with only couple of games and I have to admit that better evaluation systems might exist.

### 2.2.3 Hard level player implementation

Previous players were implemented without any use of artificial intelligence. Players were not really smart and did not use any advanced tactics mechanism to defeat their opponent. However this player is much more sophisticated. It is the smartest player offered in this game and its core is well known algorithm in the world of AI called MiniMax (sometimes MinMax or MM).

Minimax is an algorithm that is maximizing player's outcome in games of perfect information. Game of perfect information is a game where it is able to predict all the possible opponent's moves and the space that represents all these moves is finite.

With computers we have a great power in our hands. The human brain is capable of great things but most of us probably cannot imagine all the possible game board combinations after five or six rounds. However with the correct code we can generate all these possible combinations. What MiniMax algorithm does is that it creates all the possible board states after a certain amount of rounds, that number is considered as depth of the recursion, and at the lowest level we evaluate all the boards with a target function. This function assigns every game board a certain score. These evaluated game boards represent leafs in a tree structure. When all the leafs are generated and evaluated we are going back in the tree to its origin and now it is the part when the MinMax evaluation is the key. On every level of the tree we either want to maximize the score or minimize. If the current player is me, I want to mark the move that leads to a board with the highest score. If the current player is my opponent I want to mark the move that leads to a board with the lowest score. Using this pattern I evaluate every node in the tree. With this being done, I have a tree structure with its values and I simply follow the path that leads to a best game board layout. Simplified tree scture is in the picture 2.1.

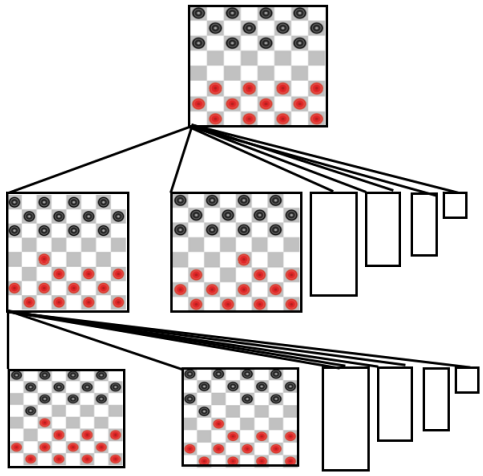


Figure 2.1: MiniMax tree structure

Since checkers is a finite game it is possible to set the depth of the algorithm until the game is finished and then we would have the information of which move is leading to a possible victory. Despite the fact how great this might sounds, it is crucial to consider the speed of the algorithm. With this knowledge in your mind we want to set the depth to a reasonable number. I have set mine to five. It was a great compromise between the speed and cleverness. With a depth of five it takes some time to return the best possible move but the waiting time is short enough to not let the player get bored by waiting.

#### 2.2.3.1 Testing

Target Function is the key for a good working MiniMax algorithm. Target function is a function that takes the board and returns a number that represents score. I have find an inspiration in book about AI [3], that describes one of the target functions for Checkers. It is not always obvious that some target function is wrong or correct. However some might give better performance outcome than the others. My target function is called evaulateBoard(char[][] board,String player) and works like this:
It is a double for cycle that iterates through every position on the board and every position grades with a certain score that adds up to an outcome number. Evaluation :

- Score based on piece itself
  - Empty square -> 0
  - Your piece -> 10
  - Your king -> 20
  - Enemy's piece -> -10
  - Enemy's king -> -20
- Score based on piece's position
  - Your piece or queen on the side of the board -> 4
  - Enemy's piece or queen on the side of the board -> -4
- When the board contains no other moves
  - For maximizing player -> 100
  - For minimizing player -> -100
- Score based on how many pieces are threatened by opponent's piece
  - For every piece that is threatened by current player -> +10
  - For every piece that is in threat by its opponent -> -10

These values are implemented as constants that are easily modifiable. I have decided to choose values, like this, based on my personal understanding of the game. I do not claim that these specific values offer the best performance possible. Obtaining the perfect values or weights is not an easy task. It is more of a task for machine learning algorithm that is capable of obtaining these values after learning on a training set that was designed for this purpose or it is large enough to capture all the important factors of the game. It is very important to understand that setting up the right weights is a crucial thing for correct MiniMax functionality. I did my best with setting up these values, but it is a tricky task to be done by the human mind. Sometimes it looks like the computer is making a wrong move but unlike human players, it can see five moves bellow in the tree structure. Maybe a move that seems like a wrong move can create a board situation that can offer the player a great advantage after couple moves. I tried to optimize these values by playing the game and creating special board layouts for testing. Let me show an example of such a board 2.2.

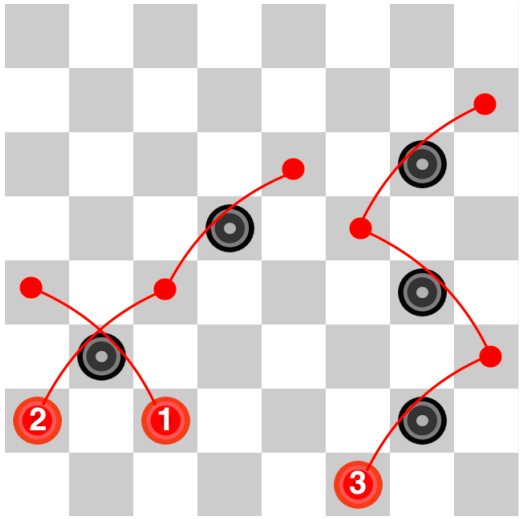



Figure 2.2: Game board with single jump, double jump and even triple jump

2.2 I created this board to test at least the very basic decisions of my AI player and more importantly to see algorithm behavior without complicated code debugging. When looking at the board 2.2 with probably no further examination the triple jump with piece number three looks like the best move. However my algorithm was always going for a double jump and that was confusing for me. Line by line I started examining the code looking for a possible bug that might be causing this unreasonable decision. Nevertheless, the code seemed correct to me. My second idea was to modify the constants in target function. The target function can be always modified in a way that works the best for given task, but it should also be in balance to work efficiently. Let me explain. Raising the value/score for triple jump twice for example, would fix this problem probably, but not every time the triple jump is the best move 2.3.

Setting up a high value for triple jump is a bad strategy. In the picture 2.3 the piece number one can play the triple jump and piece number two can play single jump. The way the algorithm is set up, is always go for triple jumps. Triple jump on this game board allows the opponent to play double jump in return and thus taking all your pieces and wining the game. By now it should be obvious why this fast fix might bring more harm than good. Fortunately, before I started with desperate code rewriting I realized it may not be a bug at all.

The problem is that I was too much focusing on what I could see and not realizing that the computer sees further in the game. Luckily testing of this hypothesis was quite simple, change the depth of MiniMax algorithm and run it on the same board 2.2 again. When I changed the depth of MiniMax to one, which means the player does one move and

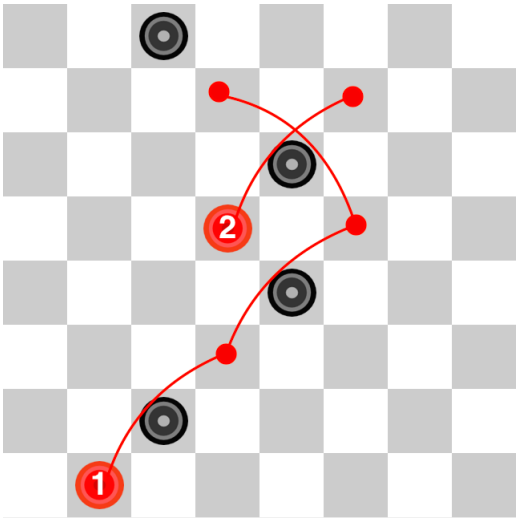



Figure 2.3: Not every time triple jump is the best move.

straight after that evaluates the board. With this depth my algorithm was deciding to go for the triple jump. When I changed back to depth of five it was going for the double jump. Taking a closer look at the board 2.2 reveals that playing the double jump with piece two prevents the player from loosing a piece one and giving the opponent a king. Even in this scenario opponent can get a king with piece in the right down corner, but it is trapped in the corner always threatened by the piece three 2.4. I have tried all different testing to approve the AI player has at least some level of intelligence and I honestly hope that it would not lose the game so easily.

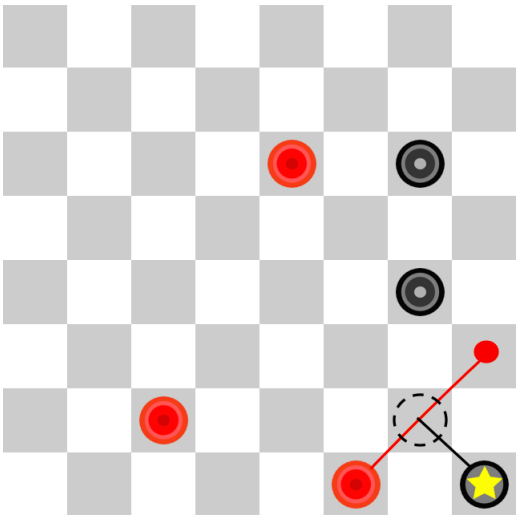


Figure 2.4: King in the corner is blocked.

#### 2.2.3.2 Results

Throughout the testing I was able to not only find and fix some bugs, but also modify the algorithm so that it gives the best performance. I did my very best with the testing part, but there still could be improvements done. For example adding more values in the target function and even find better values for the current values.

# Chapter 3

# Implementation

## 3.1 Project structure

The project's structure is the first important step towards a successful project. Creating a structure that is easily extendable and self-explanatory could be difficult and tricky. Having said that, my project is not really big or that complicated to make it hard. I created two main packages. One contains all the graphical classes and the second one contains all the functional code.

### 3.1.1 Graphical interface

Creating the graphics is much easier by using the GUI (Graphic User Interface) tool in Netbeans. It offers a solid graphical interface for editing the GUI of your application. Java language provides SWING and AWT graphics. I was using only SWING. It is more rounded and it looks modern. There are few disadvantages using the interface that is built in Neatbeans. Some parts of the code that are automatically generated cannot be modified and I found it extremely irritating. The positioning of objects is so much easier with this interface, but not flawless. The layout sometimes looks quite different in real run. well

### 3.1.2 Structure description

In this section I want to explain my project's structure. There is a list of classes I will not talk about at all for its irrelevance. By irrelevance I mean that their contribution to this application is small. I am talking about classes that create the window for displaying author's information etc. I will only describe the relations between six classes that creates the core of this application3.1.

In next section I will take every one of these six classes and tell little more about them.

## 3.2 Game control

All the game's controling is done by cursor and keyboard for saving and names. You can use mouse or any actuatuor that is able to perform the mouse tasks with cursor, that is moving

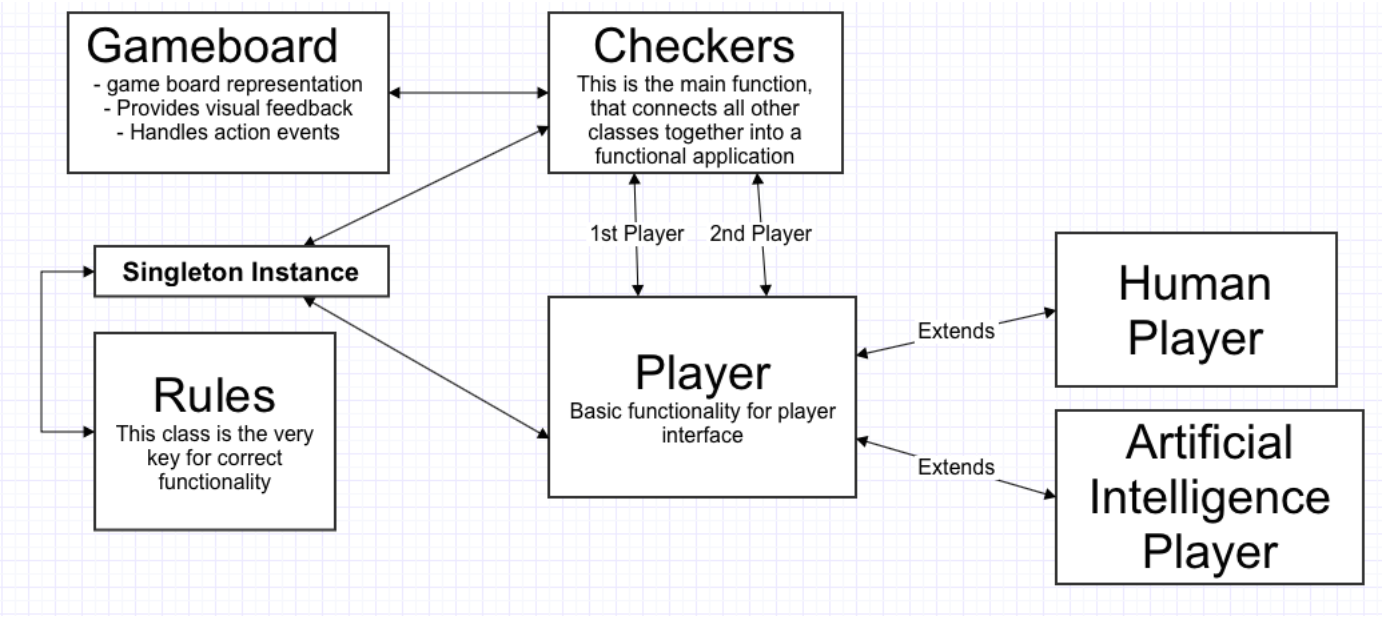



Figure 3.1: Class diagram

and left-click.

The game itself is very easy to control.

- First Click
  The First click on the piece lifts it up. Cursor is changed and the square is marked.
- Second Click For placing the piece simply place your cursor above desired square and perform click. For jumps place the piece on its final position. Multiple jumps are performed sequentally by single jumps. If another jump is possible, then the cursor will not change back to normal and requires another positioning, until no other jumps are possible. It is possible to place the piece back on its original position. Application does not allow plays that are against the rules, when that happens an informative dialog pops out.

First click on the piece lifts it up. Second click places the lifted piece on desired square.

## 3.3 Classes closer description

### 3.3.1 Gameboard

Length of this class is quite substantial, it has almost two thousand lines. However, most of it is the code for graphics. When graphic object is added, the initialization part is quite big and since there are over seventy different graphical objects the lines of code starts pilling

up fast. There are about five hundred lines of code that implement the functionality of this class. Great parts of this section are obviously methods that modify the graphic contents. Methods for initializing the board, where the start board setup could be default or custom. When I say custom, I mean either save and load the game or game board created in editor mode. Other methods are for setting up the values of different labels, like displaying the pieces the player has lost and the one he still has in the game, the player that is currently playing and also the remaining time for each round. A great part of this code is obviously for allowing the possibility of moving with pieces on the board. I have used a slightly different attitude towards the board implementation than other developers might use. The board is a group of sixty-four Jbuttons. I store all the buttons in two dimensional array.

```
private Jbutton[][] _buttField = new Jbutton[8][8];
```

When I have it all in the array it is easy to target the game board position I want. I use the field navigation by indexing through the field buttfield[row][column]. For instance the code below is for setting up the default game board for new game.

```
public void setPiecesNewGame(){
        for(int i=0;i<8;i++){
            _buttField[0][i].setIcon(_blackMan);
            _buttField[2][i].setIcon(_blackMan);
            _buttField[6][i++].setIcon(_redMan);
        }
        for(int i=1;i<8;i++){
            _buttField[1][i].setIcon(_blackMan);
            _buttField[5][i].setIcon(_redMan);
            _buttField[7][i++].setIcon(_redMan);
        }
```

This implementation has only two major disadvantages. First one, is that I am not able to create a graphic move of the piece. I can only set up an icon on the top of the button, but I cannot create a motion of that piece. I am trying to make it up a little bit by setting the cursor's icon to a piece that was painted on the button and I feel like it creates a nice visual feedback.

Second disadvantage is that since pieces can only move diagonal, half of the buttons are disabled and that to me feels wrong. I tried to create a Jpanel with colored background that should create the optical illusion of squares player cannot move to and then place only thirty-two buttons that create the game board. I honestly felt like that is a better solution. Unfortunately, I was not able to create it in a way that was graphically acceptable. Simply, the layout was not able to fit all the buttons and the gaps precisely. It was very noticeable that there are thirty-two buttons placed on a background. Based on these attempts and observations I decided to use sixty-four buttons and disable half of them. For those that are enabled I generated event handlers.

```
private void uxButton_1_7ActionPerformed
(java.awt.event.ActionEvent evt) {
        handleClickAction(1,7);
    }
```

All of the game board button handlers call the same method ("handleClickAction") with parameters that are the row and column of the button. Indexing convention is explained in picture 3.2.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| [0,0] | [0,1] | [0,2] | [0,3] | [0,4] | [0,5] | [0,6] | [0,7] |
| [1,0] | [1,1] | [1,2] | [1,3] | [1,4] | [1,5] | [1,6] | [1,7] |
| [2,0] | [2,1] | [2,2] | [2,3] | [2,4] | [2,5] | [2,6] | [2,7] |
| [3,0] | [3,1] | [3,2] | [3,3] | [3,4] | [3,5] | [3,6] | [3,7] |
| [4,0] | [4,1] | [4,2] | [4,3] | [4,4] | [4,5] | [4,6] | [4,7] |
| [5,0] | [5,1] | [5,2] | [5,3] | [5,4] | [5,5] | [5,6] | [5,7] |
| [6,0] | [6,1] | [6,2] | [6,3] | [6,4] | [6,5] | [6,6] | [6,7] |
| [7,0] | [7,1] | [7,2] | [7,3] | [7,4] | [7,5] | [7,6] | [7,7] |

Figure 3.2: Game board with indexes [row,column]

Method for handling the click event is the most complicated method in this class. Not only that it has to work for all the buttons, it also has to work for starting the move as well as finishing it. Let me summarize all the functionalities of this method:

- Handling the action for choosing the piece to be played with (pick-up phase).
  - Checking the timer (rewrite)
  - Recognizing if color of the piece and the current player are equal.
  - Controlling if player is allowed to play with that specific piece.

  - Setting the cursor icon.
  - Changing the button's icon.
  - Providing the help. Highlight the squares where the player can play with the given piece.
  - Deleting the highlighted move of your opponent.

- Handling the action for placing the piece (put-down phase).
  - Checking the put down position of the move. It has to be according to rules of the game. If player puts down the piece on the original place, It must not count as a move.
  - Providing the possibility to allow multiple jumps.
  - If move was valid, stopping the timer if active.
  - Making the game board changes.
  - Test if the game is over.
  - Test if any pieces on the board should become kings.
  - Passing the token of game control to opponent.
  - Turn on the opponent's timer for the round.

There was a lot done in this method and its implementation was complicated, but crucial for this project and proper behavior.

### 3.3.2 Game

Game is a class that not only connects all the classes together but it also provides very important functionality. When the instance of this class is created all the important information about each player is handed to this class constructor that creates all the necessities for a new game. In the picture 3.1 all the connections with this class are represented by a line. In the constructor it creates two different instances of Player class, it retrieves the singleton instance for Rules and it also creates the actual game board. When this is done the game itself starts. Functions in this class are mostly triggered from the Gameboard class where are all the event handlers for buttons. For instance, when action for moving with a specific piece is triggered it goes from GameBoard to Game. Game decides which player should play and call its instance that answers whether the move is possible (Human) or return the exact move that should be played (AI).

Besides all the functions that usually just handle the results from other classes, there is something special about this class that I want to talk about. Class Game has one inner class called Time that gives player the opportunity to set a timer for each round. If the timer is set and player does not make his move in time, the random move is played on his behalf. This class implements Runnable. That means that Time's method run is running in a independent thread. When the timer is ticking this application is double threaded. One thread is waiting for interruption from user, while the second one is counting seconds.

### 3.3.3 Rules

Rules was the very first class that needed to be implemented. I spent a lot of time writing it because it was very important to have not just a correct code that works but have a code that makes writing of other classes easier for the developer. Lets go deeper to understand what I am talking about.
The purpose of this class is to have a method that is able to go through the game board and return all the possible moves that are not just feasible but also according to rules 3.3. It might sound easy but let me show a tricky situation. The given game board offers five

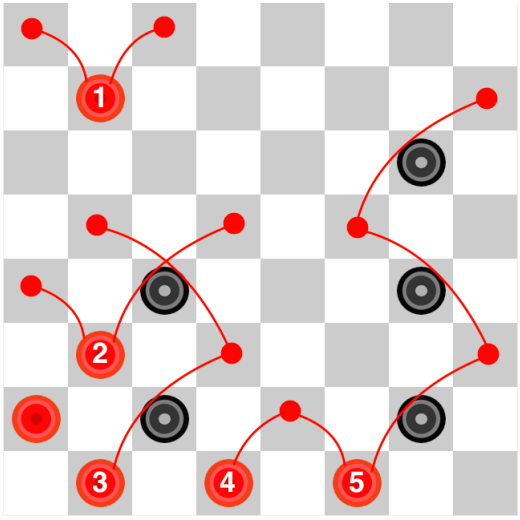



Figure 3.3: Special situation for describing the rules proper implementation

pieces to play with and eight possible moves. The player must play by rules and that means. If there is a piece that allows jump or multiple jump then player has to jump. That means the only moves possible by the rules are these.

- piece nr.2 [[5,1],[3,3]] - single jump
- piece nr.3 [[7,1],[5,3],[3,1]] - double jump
- piece nr.5 [[7,5],[5,7],[3,5],[1,7]] - triple jump

For better board orientation and indexing style check the picture with index explanation 3.2.

### 3.3.4 Player

Player class provides the basic structure for a player. It contains all the information about the player.

```
protected String _name;
protected String _color;
protected boolean _help;
protected int _pieces;
protected int _kings;
protected int _timeLimit;
protected char[][] _gameBoard;
protected Rules _rules;
protected int _lastMove;
```

On the top of player's information it also contains the game board state and instance of a singleton Rules class.

### 3.3.5 Human Player

HumanPlayer extends the Player class. It offers various verification functions and methods, that are used to force human player to play by rules.

### 3.3.6 AI Player

AIPlayer like previous class extends Player. Besides its public constructor, this class provides only one other public method and its name is playAI. This method calls the appropriate player(EASY2.2.1, MEDIUM2.2.2, HARD2.2.3). First two levels were not so difficult to implement, but the third one was difficult. Hard level is using real AI algorithm. The algorithm's name is MiniMax [5]. Let me show my implementation on next page.

```
private int miniMax(int depth,char[][] board,String player){
 1      _rules.setPropertiesForTesting(board, player);
 2      ArrayList<int[]> localArrayList = _rules.bestMoves();
 3      if(depth == _TOP_RECURSION_LEVEL){
 4          _bestMovesList = localArrayList;
 5      }
 6      boolean max = (player.equals(_color));
 7      if(localArrayList==null){
 8          return (max)?_POINT_FOR_WIN:-_POINT_FOR_WIN;
 9      }else if(depth<=0){
10          int evalForDebug = EvaluateBoard(board,player);
11          return evalForDebug;
12      }else{
13          int treshold = (max)?Integer.MIN_VALUE:Integer.MAX_VALUE;
14          for(int i=0;i<localArrayList.size();i++){
15              int[] each = localArrayList.get(i);
16              char[][] boardDC = copyBoard(board);
17              makeTheMove(each,boardDC);
18              int val = miniMax(depth-1, boardDC,(max)?"red":"black");
19              if(max){
20                  if(depth == _TOP_RECURSION_LEVEL){
21                      _decisionTree.add(val);
22                  }
23                  if(val > treshold)treshold = val;
24              }else{
25                  if(val < treshold)treshold = val;
26              }
27          }
28          return treshold;
29      }
30  }
```

This function is the body for the MiniMax algorithm. It is a function that is called recursively. Its parameters is the depth, the current board game layout, and the color of the active player.

- lines(1-2) This section sets the Rules's class properties and asks for all the possible moves on given game board.
- lines(3-4) If I just started the function, I store the reference to all the possible first moves. From this list I will later decide what move is the best to play on the real game board.
- lines(7-8) localArrayList is the ArrayList of all the possible moves and if it is null then there are no other moves and the game ends on that specific level .
- lines(9-11) The key to every recursive function. There has to be a condition for the end of the recursion. In this case, it is when the depth equals zero. When that happens I

want to evaluate the board on this level and start back propagation of this score and deciding what values are important based on MiniMax principal.

- lines(12-27)
  - (14) Initializing the for cycle for iterating through the possible moves.
  - (15-16) Get the move sequence and create deep copy of the game board.
  - (17) Make the move on the board.
  - (18) Call the recursion with depth minus one and switch the player.
  - (19-26) This section is storing the values and modifying the value for min and max player .

# Chapter 4

# Functionality

## 4.1 Main menu option


Figure 4.1: Main Menu

This screen is the start of this application. It offers the most basic and easy to understand directions.

- New game -> Creates a new game.
- Load game -> Loads up a game you previously saved.
- Rules -> The list of rules this game follows.
- Author -> Basic information about application's author.
- Editor -> Interface that allows to create a custom board set up.

## 4.2 New game

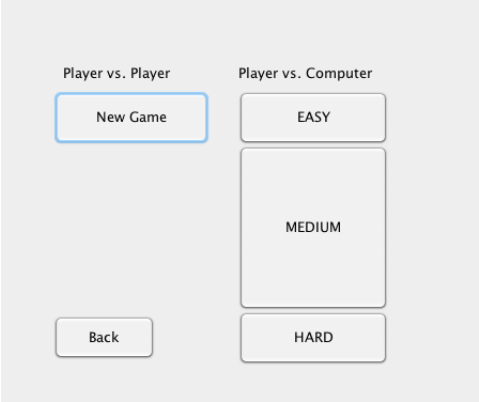



Figure 4.2: Game Option

This application state offers self explainatory buttons. You decide if you want to play human player vs. human player on one computer or if you are alone you can play against computer that offers three levels of difficulty, that are closer explained in section implementation.

## 4.3 Player's properties


Figure 4.3: Player's Info

Player's info application state is the part where you can set up all the different game properties.

- 1) Player's name.
- 2) Player's nature (Computer,Human).
- 3) This check box enables to set up a specific time for each round. This time setup is individual for each player. You can only set up time limit for human player. For unlimited time for your round, simply leave the check box blank.
- 4) The check box with the label help is a special option that allows player to see where you can move with your piece.
- 5) Group switch sets the color of a starting player. Rules specifically say which player should play first, this option gives you the opportunity to change it.
- 6) This list box is very interesting feature and also very important. Not every time you want to play the whole game. Sometimes you just want to train a specific skill with a specific game board setup. In this section you can load all the previously created game board layouts in the editor mode. If you want to play the whole game just leave the selection IGNORE.

## 4.4 Game control

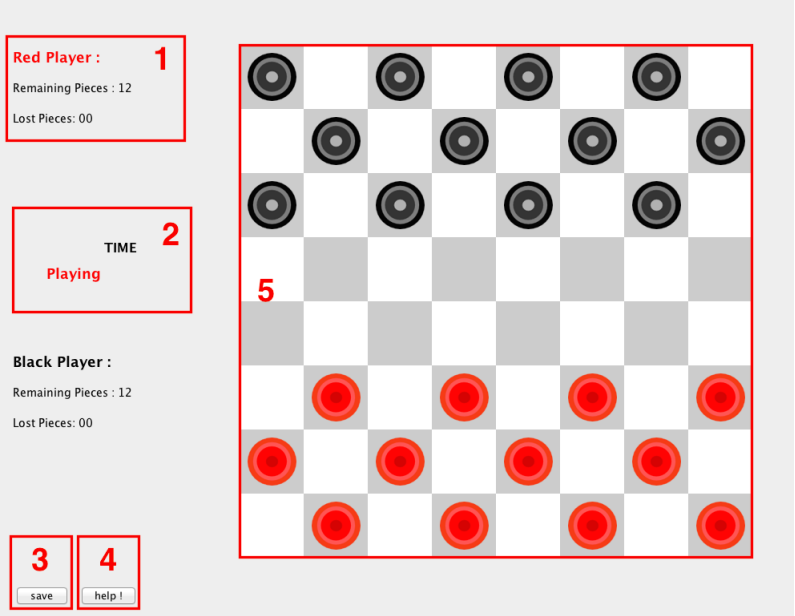



Figure 4.4: Default New Game Setup

- 1) Players information such as player's name, remaining pieces and pieces he lost.
- 2) The word playing is changing color based on which player's turn is it. The label with the word Time is a place where the counter appears if a time limit was set for a round.
- 3) This button gives you the possibility to leave the game and return to it later.
- 4) This button gives you advice what move you should play, according to AI algorithm. Help me 4.5!
- 5) Representation of the board.

In the picture 4.5 you can see how it looks like when double jump is possible. This is a screenshot from actual game. Starting position is in the frame number 1. Blue square is symbolizing help, place where you can move to. In this case only double jump is possible with given piece.

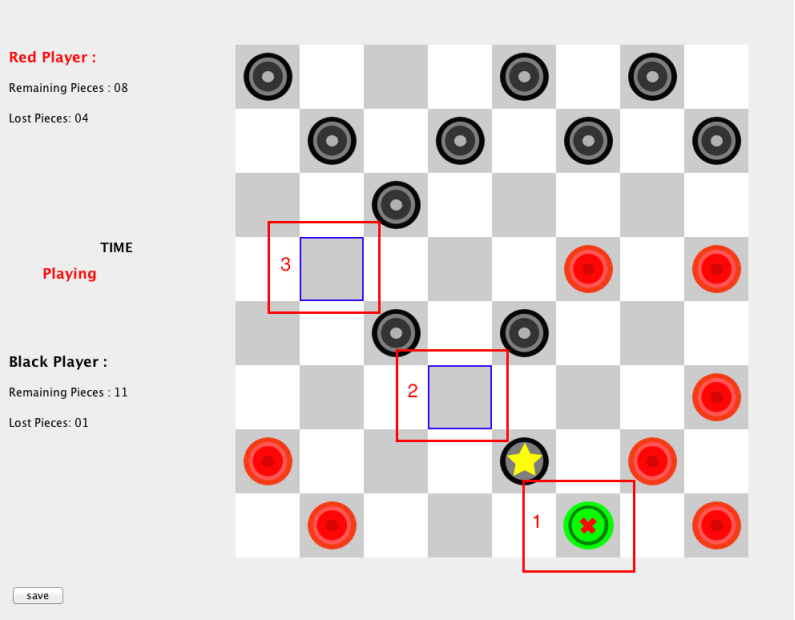



Figure 4.5: Double Jump

## 4.5 Help Me

In this project there are two different help possibilities.

First one you have to allow in player's properties 4.2 under the number 4. Whenever a player clicks on a piece it colors the squares that are possible to move to with blue color 4.5.

Second functionality help me was added at the very end of this project's development. Luckily the project was well designed and it was not difficult to include this functionality among others. It is a way to help the player find out what move the computer will play in his situation. The help algorithm is using the same algorithm like the AI hard difficulty player. It is a great way to help players who cannot see the right move on the game board. Using this help is very easy. Click the button with the label help and you will see what move to play. Example in the picture 4.6. I used same picture when I was describing the AI and its implementation 2.2.3. I also mentioned why is the algorithm picking this move.

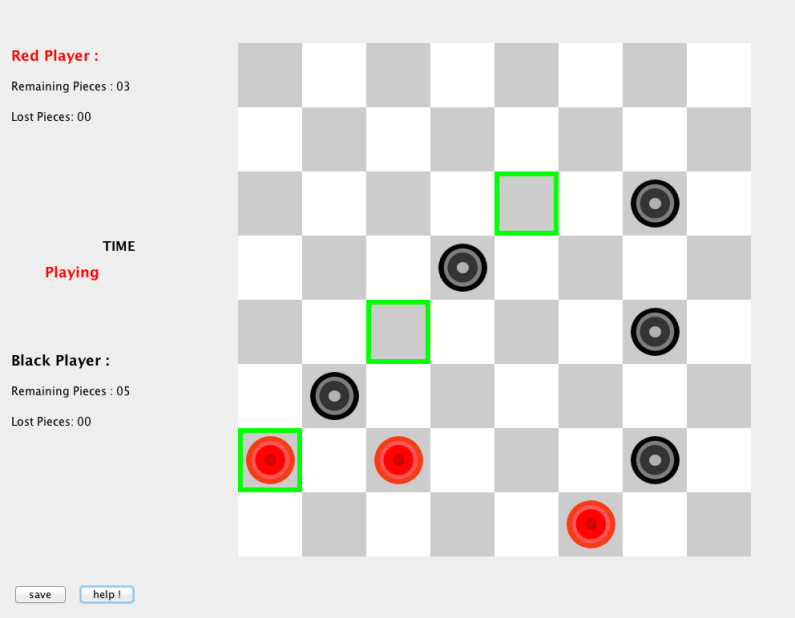



Figure 4.6: What move should I play?

## 4.6 Save and load

There are all different save and load possibilities. One of the very basic functionality is to save and load the game. Loading the game is in the main menu and saving the game is possible during the game itself, using the save button at the down-left corner.

Another functionality is to create a game board in the editor mode and save the board under any name. The application itself modifies the name, so that it is easy to recognize saved games and game boards. After this being done you can always load a game board when creating a new game 4.2.

## 4.7 Editor

In this application state you can create your own game board layout, that you can later load when creating a new game. Its graphic interface is very simple 4.7.

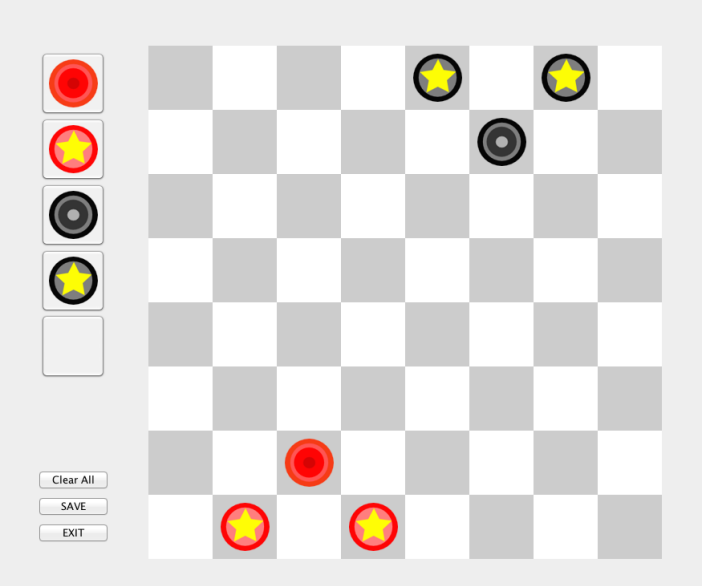



Figure 4.7: Create your own game board layout

On the left side of the editor you change your cursor to any icon on those five buttons (red man, red king, black man, black king, default). After changing the cursor, you simply place the pieces on game board or delete them if you cursor's icon is default. When you are done, you use the button save. This functionality can be very useful for players, that either want to play some concrete board layouts or train some specific skills by solving concrete situations.

# Chapter 5

# Summary

## 5.1 Conclusion

The goal of this Bachelor thesis was to create a stand alone application that uses the artificial intelligence in the best way for helping player to improve his/her skills and performance in the game of checkers.

Created application meets all the requirements. I find some of the points quite subjective, but I tried to always find a solution that meets the given criteria in the best way possible. Testing of this application is not that easy, because the space of all the possible board layouts is quite big, but I tried my best fixing all the various bugs. However it is possible that some bugs might occur during the game, but I hope the number is quite close to zero.

## 5.2 Future development

There are always thinks you can improve on any project. This project obviously has some future development tasks that might significantly improve the application and I am not talking about code polishing, refactoring and maybe even rewriting some of the working code. I had in mind four major features that I might add in the future.

### 5.2.1 Algorithm improvement

Implementation of the basic MiniSwing is a GUI widget toolkit for JavaSwing is a GUI widget toolkit for JavaMax algorithm is not very difficult, but there is a lot of work that can be done to improve it.

- Modify and improve the target function.
- Use a set of well know games to train the target function.
- Speed up the algorithm and search deeper in the tree structure.
- Use some kind of pruning and search even deeper.

- Reduce the memory usage if possible.
- Find board similarities in already played games by great players.
- Keep track of opponent's move and learn from it.

This is a list of some of the possible min max algorithm improvements that would certainly make player Hard harder to defeat.

### 5.2.2 Connection to actuators

Actuators are often used for fun to create a better experience from playing the games, like Xbox 360 Kinect or Nintendo Wii. Fact is they can be used for much more. In today's society one of the biggest problems amongst kids is obesity. Actuators like stepping carpet [6] that gives you the control over a game, forces the child to move in order to play the game. The kid might not even realize it and by playing the game it could help him not only to lose weight but to create the connection between movement and fun. With this idea in mind it can be used for helping with the obesity or to help recover body parts damaged by accidents. In this very example it might be any leg injuries and by jumping on the carpet the legs recover it's former strength.

### 5.2.3 Rollback to winnable state

Due to lack of time I was not able to implement this functionality that I believe is very interesting function and a great learning feature. This implementation will offer the possibility to "undo last moves". If you lose a game at the end a pop up window will ask if you want return the game board in a state that is still possible to win. In other words game board layout right before the move that made you lose this game.

### 5.2.4 Online mode

The possibility to play over internet would offer even better learning opportunities. Instead of playing against computer or against human player on the same computer you will have the possibility to connect with specific player or challenge any player that is willing to play against you.

# Bibliography


[1] *WIKIPEDIA*:**[Online, Accessed: 2015-04-16]**
Available from: http://en.wikipedia.org/wiki/Draughts

[2] *DRAUGHT HISTORY*:**[Online, Accessed: 2015-04-19]**
Available from: http://www.draughtshistory.nl/origin22.htm

[3] *Mitchell, Tom M. Machine Learning. New York: McGraw-Hill, 1997. Print.*

[4] *WCDF-World Checkers Draughts Federation***[Online, Accessed: 2015-04-19]**
Available: http://www.wcdf.net/rules.htm

[5] *Marik,V. at al.: Artificial intelligence (1), Academia, Prague 1993, 2003, in Czech*:**(see Chapter 2, section 2,4,1)**

[6] *HOPSCOTCH:* :**[Online, Accessed: 2015-04-19]**
Available: http://www.idmt.fraunhofer.de/en/Service_Offerings/products_and_technologies/e_h/hopscotch.html

# Appendix A

# Acronyms

**IDE** Integrated Development Environment

**GUI** Graphical User Interface

**AI** Artificial intelligence

**MM** MiniMax (game algorithm)

**AWT** Abstract Window Toolkit

**Others**

**SWING** Swing is a GUI widget toolkit for Java.

# Appendix B

# Contents of enclosed CD

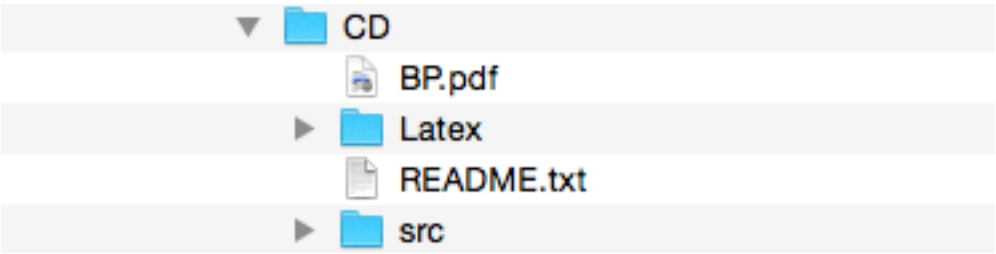



Figure B.1: Move of a man and move of a king.

- BP.pdf - Final version of bachelor thesis
- Latex - All the Latex source code, pictures and macros.
- README.txt - Explanation of cd content and step-by-step description, how to run the application.
- src - Application