```
**************************************************************
*                    PROGRAM  SPECIFICATION                  *
**************************************************************
```
@PURPOSE

This program implement a human-computer interactive game of 6*6 Russian Checkers. In particular, the alpha-beta pruning algorithm is used here to compute the best move.

@CONSTRAINTS

This game is based on a restricted rule comparing to the original version. There will be no King. Any checker will not be able to move once it reaches the ending row.

@DESIGN

Following is a brief specification of the design. Please see the files in documentation directory and the inline comments in the sourse code for more details.

1. AI algorithm

- A separate class Agent provides all static methods used for computation of moves.

    bestMove,: Compute and return the best move based on a given checkerboard status (a 6*6 matrix) and the role of player.

    findPath: Compute and return a full path starting from the given start checker to the target checker.

    goalTest: Test whether the given player wins the game based on a checker board status.

2. Graphic User Interface

This program uses Java Swing to create the user interface.

- Checker: Each checker is a customized JComponent which is paint according to its status. The status of a checker can be:

    unavailable (not supposed to contain a chess)

    empty (can but do not contain a chess)

    black (contain a black chess)

    white (contain a white chess)

- CheckerBoard: A checker board class manages a matrix of checkers.
- CheckerFrame: A checker frame constitutes of a checker board, a panel to show statistical information about a search and a panel to configure the game settings.

3. Players

There are two players in this game, a human player and a computer player.

They both inherit from the Player class. The only difference between the two players is their ways to take a move. To achieve an more efficient and neat design, it takes advantage of concurrency by sharing a turn lock between the two players. Each player starts taking/computing a move if only the turn lock is available for it.

- Player:

    Player is created with a role represented by the status of checkers (either black or white) owned by it. This role can also be changed.

    The takeTurn method would be called either when the derived class thinks it's time to take a move, or in the run method when the derived class is designed to be a Thread/Runnable.

    An abstract method move is called in takeTurn so that the derived class is able to customize its own way to take a move.

- Human:

    The overridden move method for a human is to use the Agent to find a full path from the two selected checkers by user. This path could extend more than just moving from the first selected checker to the second one. If there is more jumping possible, it will automatically jump to the end.

- Computer:

    The overridden move method for a computer is to use the Agent to find the best move by performing an alpha-beta pruning search.

4. Threads
- The main thread creates the UI and launches a thread for the computer player.
- The computer thread keeps waiting until it's his turn to make a move and then computes and shows a move.
- All the UI events and responses take place and are handles in the Event Dispatch Thread of Swing.
- Every time a move is worked out by computation, there will be a SwingWorker thread launched to perform the animation of moving a chess to the target checker. (Captures may take place.)

```
*****************************************************************
*                        HOW TO USE                             *
*****************************************************************
```

@COMPILER NVIRONMENT

OS: No restriction.

Java Virtual Machine: Java SE6 or later version.

@COMPILE THE GAME

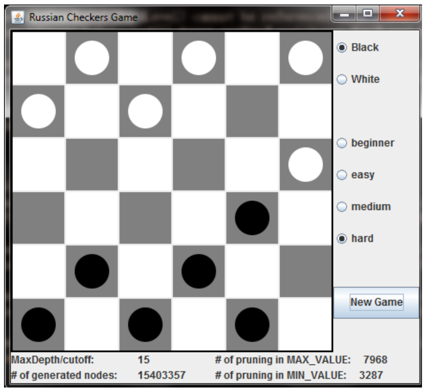
Go to the directory of "source code", execute the following command in command line:

```
javac *.java
```

@RUN THE GAME

Go to the directory of "executable", execute the following command in command line:

```
java CheckersGame
```

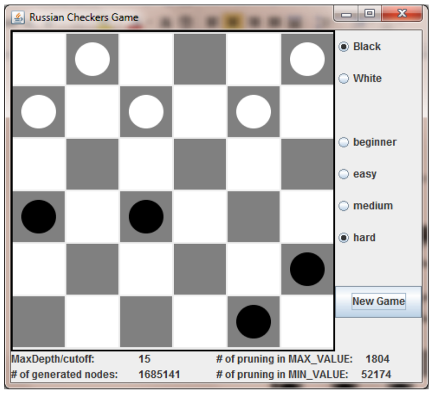


[✓ Black] White

beginner easy medium [✓ hard]

```
MaxDepth/cutoff:         15           # of pruning in MAX_VALUE:   7968
# of generated nodes:    15403357     # of pruning in MIN_VALUE:   3287
```

Figure 1. run the game



[✓ Black] White

beginner easy medium [✓ hard]

```
MaxDepth/cutoff:         15           # of pruning in MAX_VALUE:   1804
# of generated nodes:    1685141      # of pruning in MIN_VALUE:   52174
```

Figure 2. run the game

```
*****************************************************************
*                        PERFORMANCE                            *
*****************************************************************
```

On average, it takes no more than 30 seconds to complete a search deep into the 15 levels. See figure 1, in this case, a move is computed by a search of 15 levels which generates more than 15 million nodes and during which 3287 pruning take place in minValue procedure and 7968 in maxValue.

Normally it will take just a few seconds to complete a move if a goal is found at a max depth less than the cutoff level. The case in fugure 2 just takes about 2 seconds to finish a search of 1.7 million nodes in which more than 52 thousands pruning happen in minValue procedure and 1804 in maxValue.