CHARLES UNIVERSITY

Checkers (Draughts) Application Document

By Turan Isgandarli

Submitted to- Jirka Sejnoha

Table of Contents

List of Figures

Purpose

The purpose of developing the Checkers (Draughts) Windows application in .net is to create a game where users can entertain and increase their mental ability.

It is a strategy and tactic game where the user needs to think ahead, plan their moves, and anticipate their opponent's actions. It helps to improve problem-solving, critical thinking, and concentration skills.

Application Overview and Rules

The traditional two-player board game Checkers is also known as Draughts in some regions. It is played on an 8x8 grid. In order to capture their opponent's pieces and eventually their king, players must use skill and strategy.

Objective:

- To capture all of your opponent's pieces or block them so they can't move legally is the main goal of the game of checkers.

Setup:

- 64 squares make up the 8x8 grid on which the game is played.
- Twelve pieces are placed on the dark squares of the first three rows for the player on the bottom and the last three rows for the player on top for each player at the beginning.
- Typically, players utilize pieces in pairs of two distinct colors, such as red and black or white and black.

Moves for Regular pieces:

- Regular pieces are allowed to diagonally into an empty adjacent square while moving forward one square at a time.
- If there is an empty square immediately after an enemy piece, pieces can grab it by leaping over it diagonally. If there are still jumps remaining, you can keep capturing in the same move.
- An ordinary piece turns into a "king" when it enters the opponent's rear row.

Moves for Regular pieces:

- A king can move diagonally, one square at a time, either forward or backward.
- Kings can also jump over pieces and continue to jump if more captures are possible. They can also grab pieces diagonally forward or backward.

Prerequisites

End users:

1. Windows operating system. (As a user can run directly from the exe file.)
2. Minimum I3 processor with 4 GB RAM for smooth run.

Developer:

3. .NET Framework 4.8.
4. Windows operating system.
5. Integrated development environment (Prefer- Visual studio)
6. Minimum I3 processor with 4 GB RAM for smooth build and run.

Developer Guide

PlayerColor.cs

Setting an enum class to set a group of constants as Red and black, which will be used in other classes.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace CheckersMinimax
8   {
        27 references
9       public enum PlayerColor
10      {
11          Red,
12          Black
13      }
14  }
15
```

Figure 1:PlayerColor.cs File.

Creating a board (CheckerBoard.cs)

- Inside CheckerBoard.cs class, I have created a function MakeBoard which sets the main playing area of an 8*8 grid.

- Setting alternative red and black boxes using Brushes.Red and Brushes.Black by putting even odd conditions using the remainder(%) operator.

```
if (Row%2==0)
{
        Brushes.Red
}
Else
{
        Brushes.Black
}
```

- Adding black and red pieces for the first three rows of both sides.

```
51          public void MakeBoard(RoutedEventHandler routedEventHandler)
52          {
53              int count = 0;
54              for (int row = 0; row < 8; row++)
55              {
56                  BoardArray.Add(new List<CheckersSquareUserControl>());
57
58                  for (int column = 0; column < 8; column++)
59                  {
60                      CheckersSquareUserControl checkerSquareUC;
61                      if (row % 2 == 0)
62                      {
63                          if (column % 2 == 0)
64                          {
65                              checkerSquareUC = new CheckersSquareUserControl(
66                                  Brushes.Red,
67                                  new CheckersPoint(row, column, CheckerPieceType.nullPiece),
68                                  routedEventHandler);
69                          }
70                          else
71                          {
72                              if (row < 3)
73                              {
74                                  checkerSquareUC = new CheckersSquareUserControl(
75                                      Brushes.Black,
76                                      new CheckersPoint(row, column, CheckerPieceType.BlackPawn),
77                                      routedEventHandler);
78                              }
79                              else if (row > 4)
80                              {
81                                  checkerSquareUC = new CheckersSquareUserControl(
82                                      Brushes.Black,
83                                      new CheckersPoint(row, column, CheckerPieceType.RedPawn),
84                                      routedEventHandler);
85                              }
86                              else
87                              {
88                                  checkerSquareUC = new CheckersSquareUserControl(
89                                      Brushes.Black,
90                                      new CheckersPoint(row, column, CheckerPieceType.nullPiece),
91                                      routedEventHandler);
92                              }
93                          }
94                      }
```

```
95                      else
96                      {
97                          if (column % 2 == 0)
98                          {
99                              if (row < 3)
100                             {
101                                 checkerSquareUC = new CheckersSquareUserControl(
102                                     Brushes.Black,
103                                     new CheckersPoint(row, column, CheckerPieceType.BlackPawn),
104                                     routedEventHandler);
105                             }
106                             else if (row > 4)
107                             {
108                                 checkerSquareUC = new CheckersSquareUserControl(
109                                     Brushes.Black,
110                                     new CheckersPoint(row, column, CheckerPieceType.RedPawn),
111                                     routedEventHandler);
112                             }
113                             else
114                             {
115                                 //empty middle spot
116                                 checkerSquareUC = new CheckersSquareUserControl(
117                                     Brushes.Black,
118                                     new CheckersPoint(row, column, CheckerPieceType.nullPiece),
119                                     routedEventHandler);
120                             }
121                         }
122                         else
123                         {
124                             checkerSquareUC = new CheckersSquareUserControl(
125                                 Brushes.Red,
126                                 new CheckersPoint(row, column, CheckerPieceType.nullPiece),
127                                 routedEventHandler);
128                         }
129                     }
130
131                     count++;
132                     BoardArray[row].Add(checkerSquareUC);
133                 }
134             }
135         }
```

Figure 2: CheckerBoard.cs file.

Get winner function (CheckerBoard.cs)

GetWinner()- This function helps to return the winner at the end.

I have used the condition when blackCheckersPoint.Count is 0, which means all the pieces of black are out, so in that case, Red will be the winner. Same with the redCheckersPoint.Count ==0 then it returns Black as a winner.

```
141     public object GetWinner()
142     {
143         List<CheckersPoint> redCheckerPoints = GetPointsForColor<IRedPiece>();
144         List<CheckersPoint> blackCheckerPoints = GetPointsForColor<IBlackPiece>();
145
146         /// MessageBox.Show(blackCheckerPoints.Count.ToString());
147         if (blackCheckerPoints.Count == 0)
148         {
149             return PlayerColor.Red;
150         }
151         else if (redCheckerPoints.Count == 0)
152         {
153             return PlayerColor.Black;
154         }
155         else
156         {
157             return null;
158         }
159     }
160
```

Figure 3: GetWinner function from CheckerBoard.cs file.

SwapTurns (CheckerBoard.cs)

SwapTurns()- This function helps to swap the chance for the players.

I have used the if else condition, when the Current player is red, so in the next move the current player will be black and vice versa.

```
378         /// Swaps the turn.
379
                1 reference
380         public void SwapTurns()
381         {
382             if (CurrentPlayerTurn == PlayerColor.Red)
383             {
384                 CurrentPlayerTurn = PlayerColor.Black;
385             }
386             else
387             {
388                 CurrentPlayerTurn = PlayerColor.Red;
389             }
390         }
391
```

Figure 4: Swap function from CheckerBoard.cs file.

Move on the board (CheckerBoard.cs)

MakeMoveOnBoard()- This function helps to set and return the possible moves available. Multiple if and else conditions are added to make the moves.

Example- `if (!(realDestination.Checker is KingCheckerPiece)`

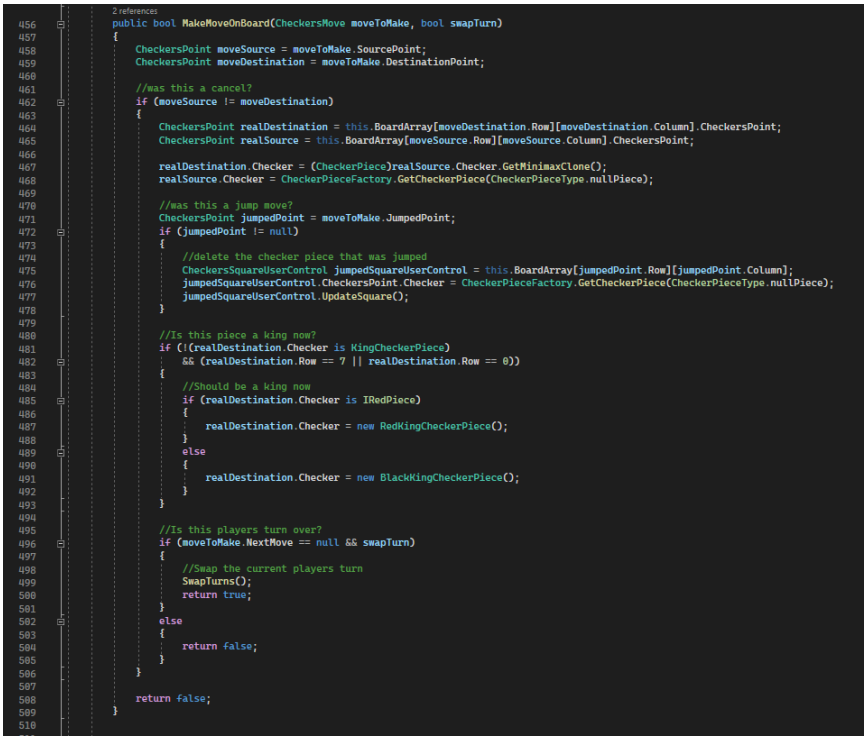
The above condition is used to set the king when the piece will move to the 1st row of the opponent.

```
if (realDestination.Checker is IRedPiece)
 {
     realDestination.Checker = new RedKingCheckerPiece();
 }
 else
 {
     realDestination.Checker = new BlackKingCheckerPiece();
 }
```

The above conditions are used to check if it is a red piece moved to another end then it will convert to red king and vice versa for black as well.

```
  if (moveToMake.NextMove == null && swapTurn)
  {
      //Swap the current players turn
      SwapTurns();
      return true;
  }
```

If the player's current move is completed, it will call the SwapTurn function. Which will change the move to the next colour.



```
2 references
public bool MakeMoveOnBoard(CheckersMove moveToMake, bool swapTurn)
{
    CheckersPoint moveSource = moveToMake.SourcePoint;
    CheckersPoint moveDestination = moveToMake.DestinationPoint;

    //was this a cancel?
    if (moveSource != moveDestination)
    {
        CheckersPoint realDestination = this.BoardArray[moveDestination.Row][moveDestination.Column].CheckersPoint;
        CheckersPoint realSource = this.BoardArray[moveSource.Row][moveSource.Column].CheckersPoint;

        realDestination.Checker = (CheckerPiece)realSource.Checker.GetMinimaxClone();
        realSource.Checker = CheckerPieceFactory.GetCheckerPiece(CheckerPieceType.nullPiece);

        //was this a jump move?
        CheckersPoint jumpedPoint = moveToMake.JumpedPoint;
        if (jumpedPoint != null)
        {
            //delete the checker piece that was jumped
            CheckersSquareUserControl jumpedSquareUserControl = this.BoardArray[jumpedPoint.Row][jumpedPoint.Column];
            jumpedSquareUserControl.CheckersPoint.Checker = CheckerPieceFactory.GetCheckerPiece(CheckerPieceType.nullPiece);
            jumpedSquareUserControl.UpdateSquare();
        }

        //Is this piece a king now?
        if (!(realDestination.Checker is KingCheckerPiece)
            && (realDestination.Row == 7 || realDestination.Row == 0))
        {
            //Should be a king now
            if (realDestination.Checker is IRedPiece)
            {
                realDestination.Checker = new RedKingCheckerPiece();
            }
            else
            {
                realDestination.Checker = new BlackKingCheckerPiece();
            }
        }

        //Is this players turn over?
        if (moveToMake.NextMove == null && swapTurn)
        {
            //Swap the current players turn
            SwapTurns();
            return true;
        }
        else
        {
            return false;
        }
    }

    return false;
}
```

Figure 5: MakeMoveOnBoard function from CheckerBoard.cs file.

Minimax algorithm (AIController.cs)

Minimax algorithm works on the recursive or backtracking algorithm which is used in decision-making. In the event that the opponent is likewise playing optimally, it offers the player an optimal move. Recursion is used by the Mini-Max algorithm to search across the game tree.

MinMaxStart()-

- This is the main minimax algorithm code. This method helps to trigger the algorithm and find the best move for the player.
- If two or more moves have having the same value, so in that case random() function will call and will pick any of the moves.
- I have used `Thread.Sleep(200);` method, in order to take a 2-second break for computer movement.
- `foreach (CheckersMove move in possibleMoves)` Using foreach loop, I am iterating all the possible moves, and inside this loop, there are more conditions to find the best move out of all possible moves.
- It will return the best move after doing all the computations like depth search.

```
public static CheckersMove MinimaxStart(CheckerBoard board)
{
    int alpha = int.MinValue;
    int beta = int.MaxValue;
    thinking = true;

    List<CheckersMove> possibleMoves = board.GetMovesForPlayer();
    List<int> values = new List<int>();

    Logger.Info(string.Format("Max is {0}", board.CurrentPlayerTurn));

    if (possibleMoves.IsNullOrEmpty())
    {
        return null;
    }

    foreach (CheckersMove move in possibleMoves)
    {
        Thread.Sleep(200);

        CheckersMove moveToMake = move;
        CheckerBoard boardToMakeMoveOn = board;
        do
        {
            Logger.Debug("Board Before");
            Logger.Debug(boardToMakeMoveOn.ToString());

            boardToMakeMoveOn = (CheckerBoard)boardToMakeMoveOn.GetMinimaxClone();
            boardToMakeMoveOn.MakeMoveOnBoard((CheckersMove)moveToMake.GetMinimaxClone());
            moveToMake = moveToMake.NextMove;

            Logger.Debug("Board After");
            Logger.Debug(boardToMakeMoveOn.ToString());
        }
        while (moveToMake != null);

        values.Add(Minimax(boardToMakeMoveOn, Settings.AIDepth - 1, alpha, beta, false, board.CurrentPlayerTurn));
    }

    int maxHeuristics = int.MinValue;
    foreach (int value in values)
    {

        if (value >= maxHeuristics)
        {
            maxHeuristics = value;
        }
    }

    //filter the list of moves based on max value
    List<CheckersMove> bestMoves = new List<CheckersMove>();
    for (int i = 0; i < values.Count; i++)
    {
        if (values[i] == maxHeuristics)
        {
            bestMoves.Add(possibleMoves[i]);
        }
    }

    counter = 0;
    thinking = false;
    Logger.Info("Node Values: " + string.Join(",", values.Select(x => x.ToString()).ToArray()));
    return bestMoves[Rng.Next(bestMoves.Count)];
}
```

Figure 6: Minimax algorithm code from AIController.cs file.

Menu functions (MainWindow.xaml.cs)

RestartGame()- This function helps to abort the current game using *Abort()* method. Later it will reset the game to the initial condition using *InitializeCheckers()* function.

Quit()- This function helps to quit(resign) the game and it will show the message as the opponent wins using `MessageBox.Show("Black wins ");` to display the popup window. Apart from that, it will abort the current game and initialize the game as a new game.

DV()- This function helps to display the developer information using *messageBox.Show()* window form.

```
1 reference
482 private void RestartGame(object sender, RoutedEventArgs e)
483 {
484     if (aiThread != null)
485     {
486         aiThread.Abort();
487     }
488
489     InitializeCheckers();
490 }
491
492 /// Quits the game.
493
    1 reference
494 private void Quit(object sender, RoutedEventArgs e)
495 {
496     MessageBox.Show("Black wins ");
497     if (aiThread != null)
498     {
499         aiThread.Abort();
500     }
501
502     InitializeCheckers();
503 }
504
505 /// Developer Information
506
    1 reference
507 private void DV(object sender, RoutedEventArgs e)
508 {
509     MessageBox.Show("Developed by Turan Isgandarli");
510
511 }
```

Figure 7: All menu functions from MainWindow.xaml.cs file.

User Guide (Graphical User Interface)

Users can run the application by just clicking on the .exe file available in the debug/release folder of the application.

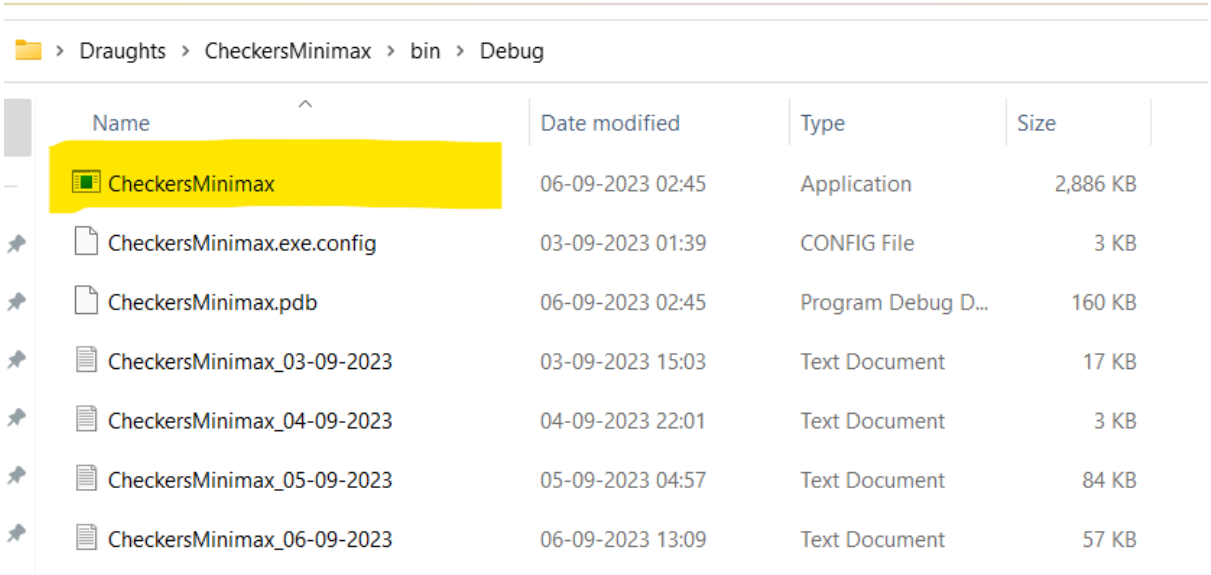


Figure 8: Open application using exe file.

The main window contains:

- **Menu-** Menu has 3 buttons placed horizontally:
    1. **New Game-** New Game will start.
    2. **Developer information-** This tab will show the developer information.
    3. **Quit-** This button helps to quit (Resign) the game. By clicking it, another player will automatically win.

- **8*8 checkers board-** It is the main screen to play the checkers game, where Black is the computer side and red is the player 1 side.

- **Title bar-** The title bar will show the dynamic content like which player needs to move the piece that is red or black.

- **Textblock:** Textblock is the area on the right side of the main screen where I have added the text “Welcome to Draughts”

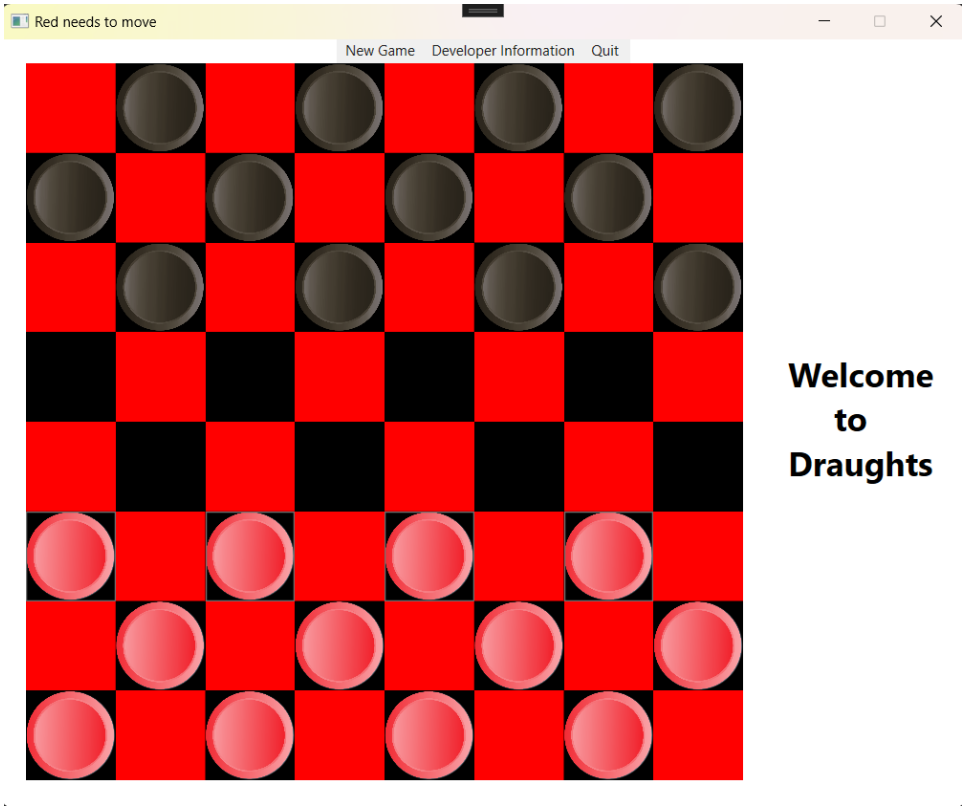


Figure 9: Main application window.

Selected Piece:

If users select and hold any piece then, that box will be highlighted in the green border, in order to distinguish between the selected piece and the non-selected piece.

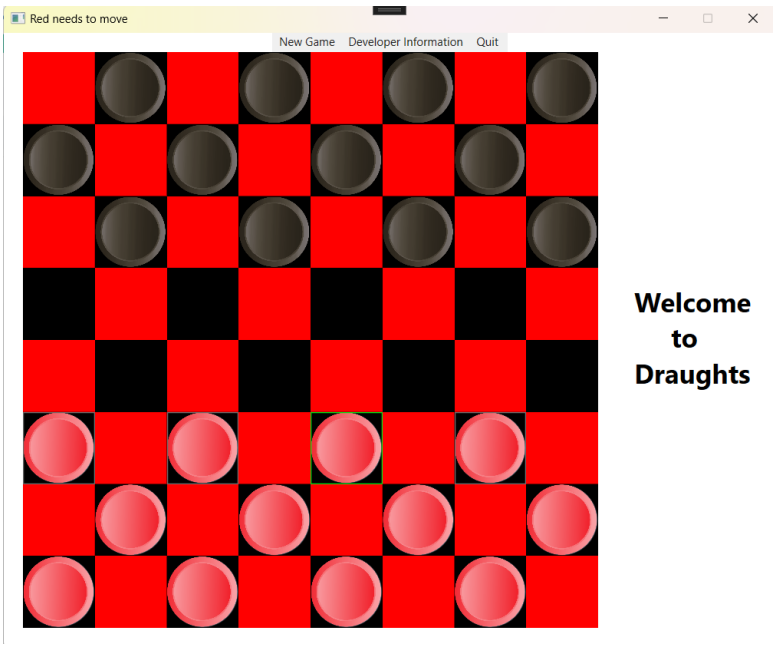


Figure 10: Selected piece having green background.

Possible Moves:

When users release the hold on the selected piece then possible moves will be highlighted in the aqua colour.

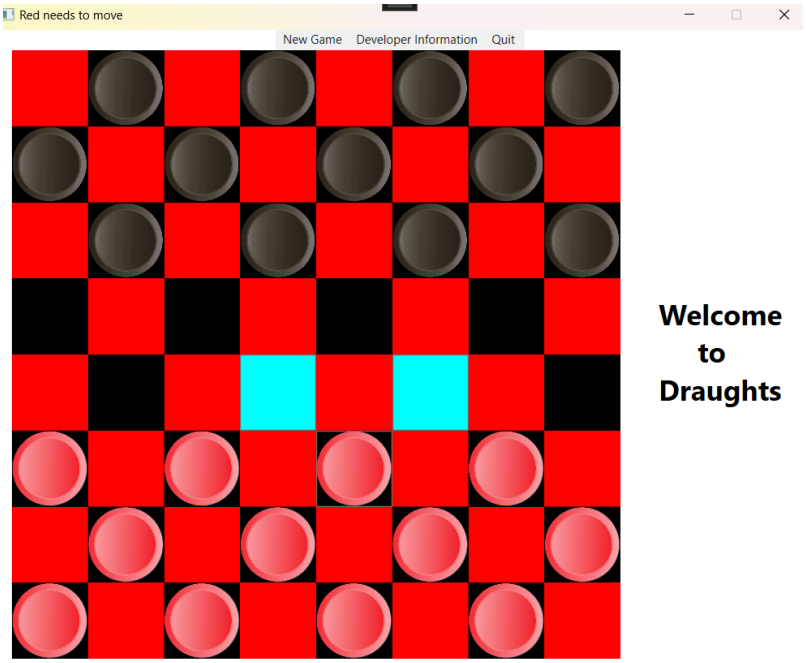


Figure 11: Possible moves for the selected piece.

Diagonally Cut:

As per the rules of checkers, the Player can remove the opponent's piece from the board by jumping them if empty space is available next to the diagonal. Please check the below image.

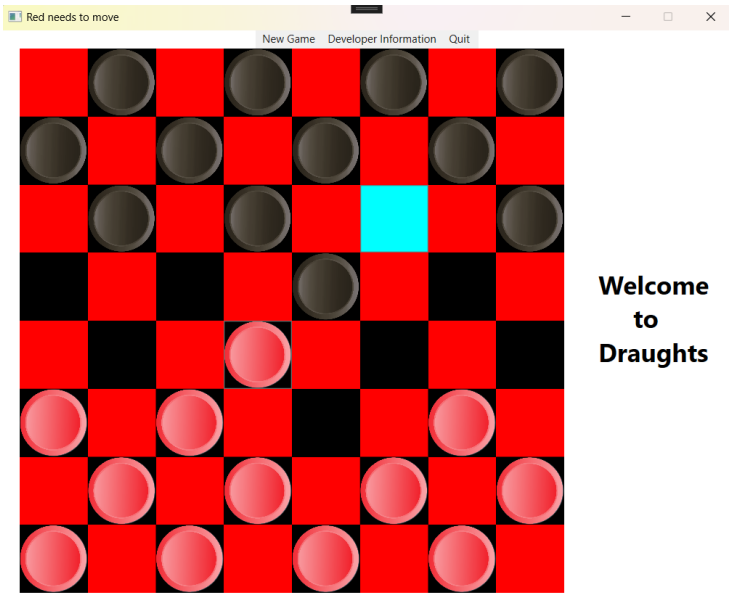


Figure 12: Diagonally cut to take others piece.

King:

As per the rules of checkers, if the opponent moves to the other side, normal piece will convert into king.

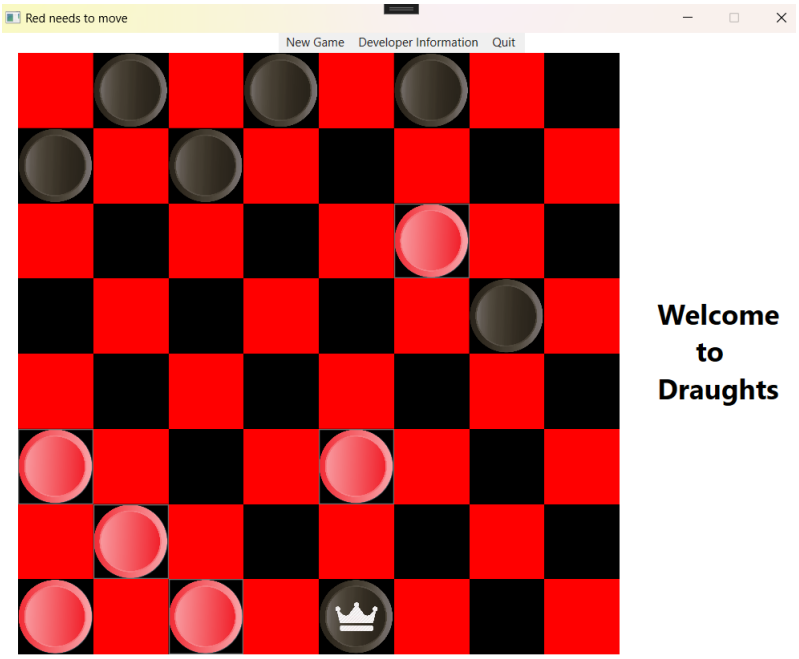


Figure 13: King view, once piece move to other side.

Quit:

If the Player clicks on the quit, then the opponent will win the game automatically.

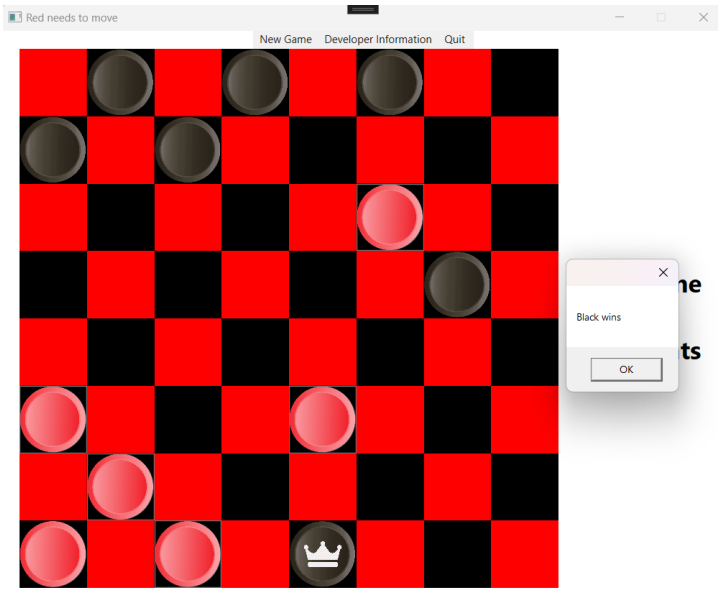


Figure 14: Quit game.

New Game:

If the Player clicks on the New Game button, then the board will reset all the pieces and new game will start.

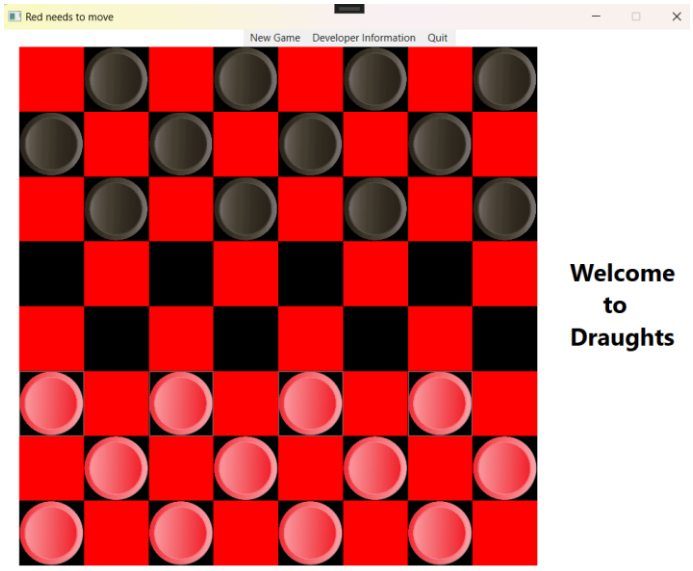


Figure 15: New game, sets to initial game.

Developer Information:

If the Player clicks on the Developer information button, then one dialog box will open and it will show the developer information.

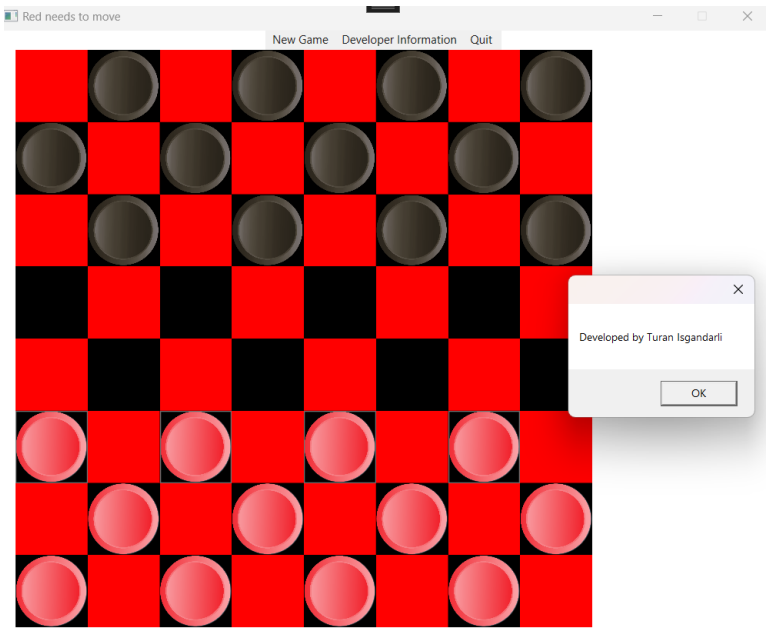


Figure 16: Popup showing developer's information.