AI PROJECT

Name: Rahul Purushottam Gaonkar(rpg283)

CS6613

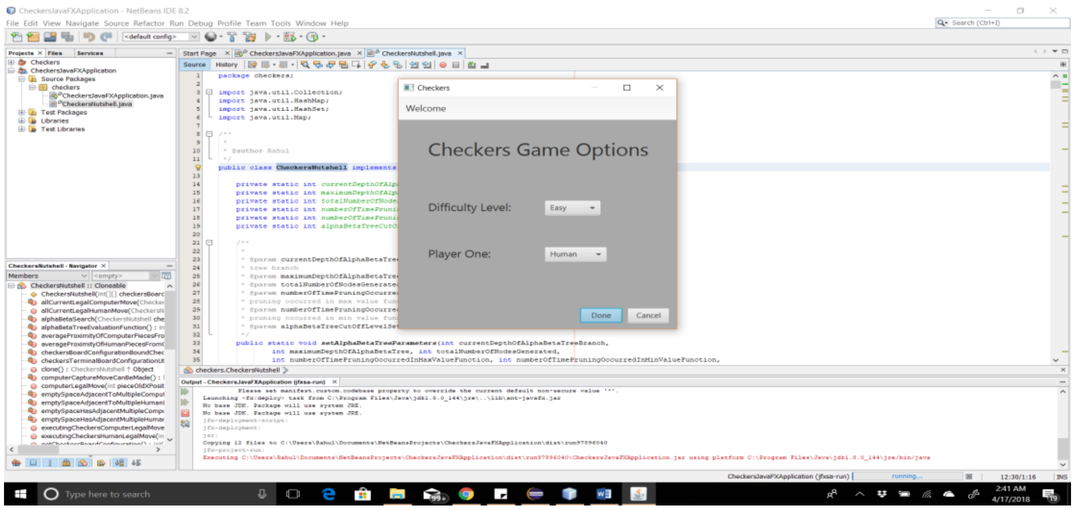
Instructions On How To Compile And Run:

- The checkers game is developed using Java and JavaFX for the GUI.
- The Checkers Java Project has two Java classes namely **CheckersJavaFXApplication** and **CheckersNutshell**.
- The main method is in class **CheckersJavaFXApplication** which can be used to run the Java project.

High Level Description Of the Checkers Game Design And Program:

Checkers Game Design:

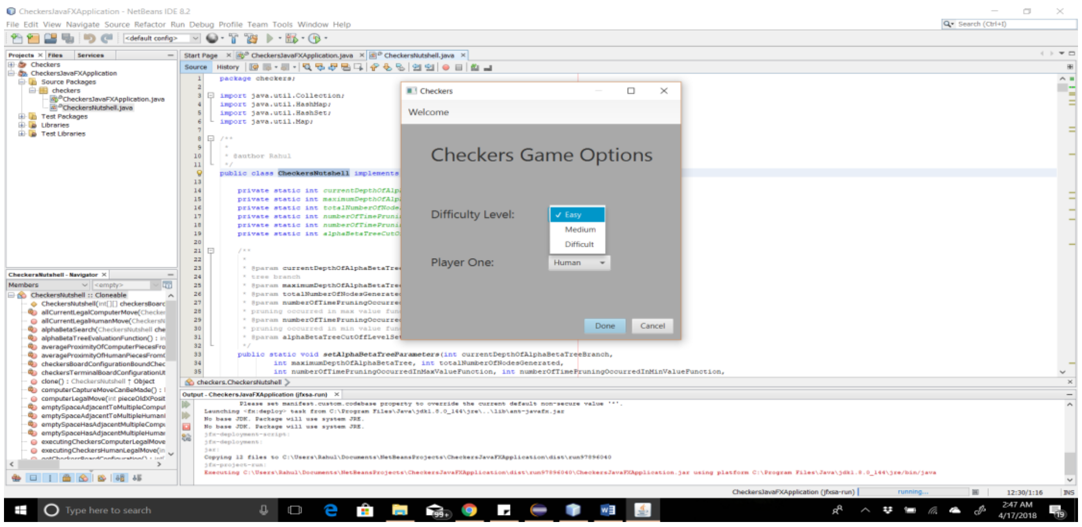
- Once you run the Project you will get a Checkers Menu Dialog Box as shown below:

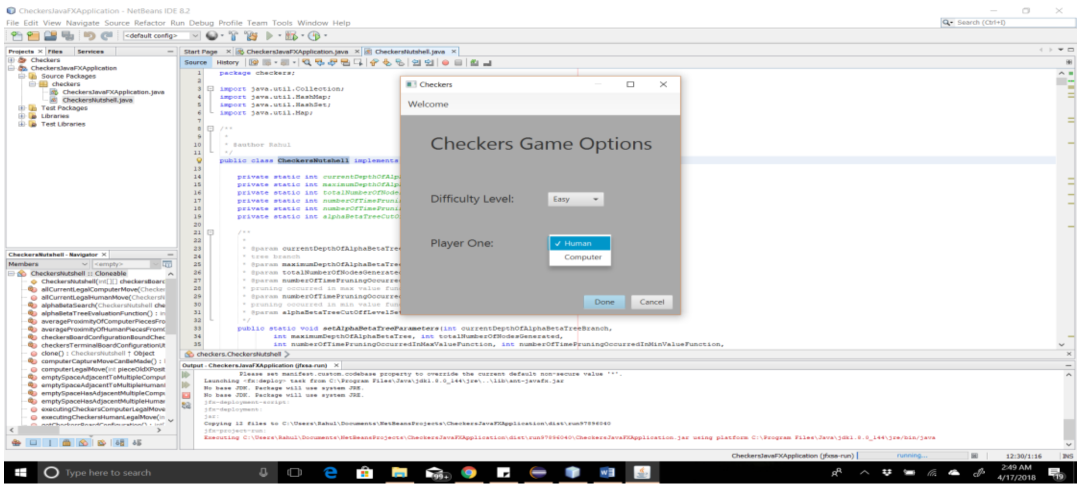


```
package checkers;

import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;

/**
 *
 * @author Rahul
 */
public class CheckersNutshell implements

    private static int currentDepthOfAlp
    private static int maximumDepthOfAlp
    private static int totalNumberOfNode
    private static int numberOfTimePruni
    private static int numberOfTimePruni
    private static int alphaBetaTreeCutO

    /**
     *
     * @param currentDepthOfAlphaBetaTre
     * tree branch
     * @param maximumDepthOfAlphaBetaTre
     * @param totalNumberOfNodesGenerate
     * @param numberOfTimePruningOccurre
     * pruning occurred in max value fun
     * @param numberOfTimePruningOccurre
     * pruning occurred in min value fun
     * @param alphaBetaTreeCutOffLevelSe
     */
    public static void setAlphaBetaTreeParameters(int currentDepthOfAlphaBetaTreeBranch,
            int maximumDepthOfAlphaBetaTree, int totalNumberOfNodesGenerated,
            int numberOfTimePruningOccurredInMaxValueFunction, int numberOfTimePruningOccurredInMinValueFunction,
```

```
        Please set manifest.custom.codebase property to override the current default non-secure value '*'.
Launching <fx:deploy> task from C:\Program Files\Java\jdk1.8.0_144\jre\..\lib\ant-javafx.jar
No base JDK. Package will use system JRE.
No base JDK. Package will use system JRE.
jfx-deployment-script:
jfx-deployment:
jar:
Copying 12 files to C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run97896040
jfx-project-run:
Executing C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run97896040\CheckersJavaFXApplication.jar using platform C:\Program Files\Java\jdk1.8.0_144\jre/bin/java
```

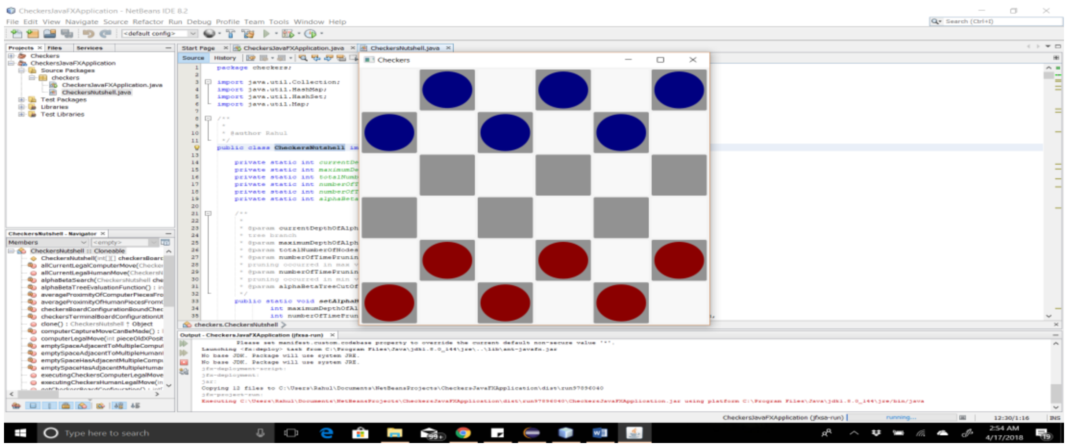
- Please select the **Difficulty Level** and the player who will play first (Human or Computer) i.e. **Player One** and click on **Done**. You can also close the application by clicking **Cancel** or closing the dialog box by clicking on the cross **(X)**.





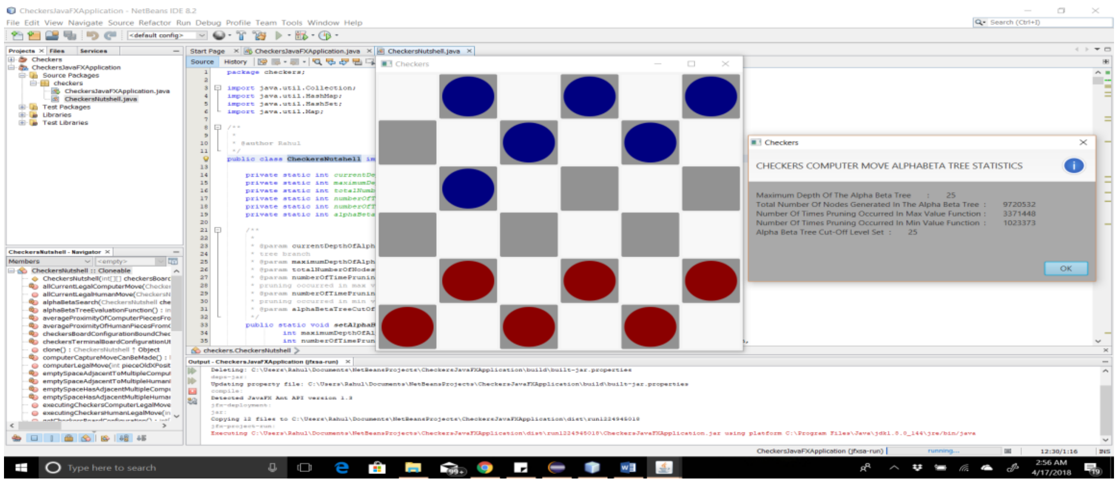
- Once you select the **Difficulty Level** (Easy Selected for the below checkers game demo) and the **Player One**, the checkers board will be displayed as show below: (Red Piece: Human Piece and Blue Piece: Computer Piece)

1. If **Player One** is **Human:**



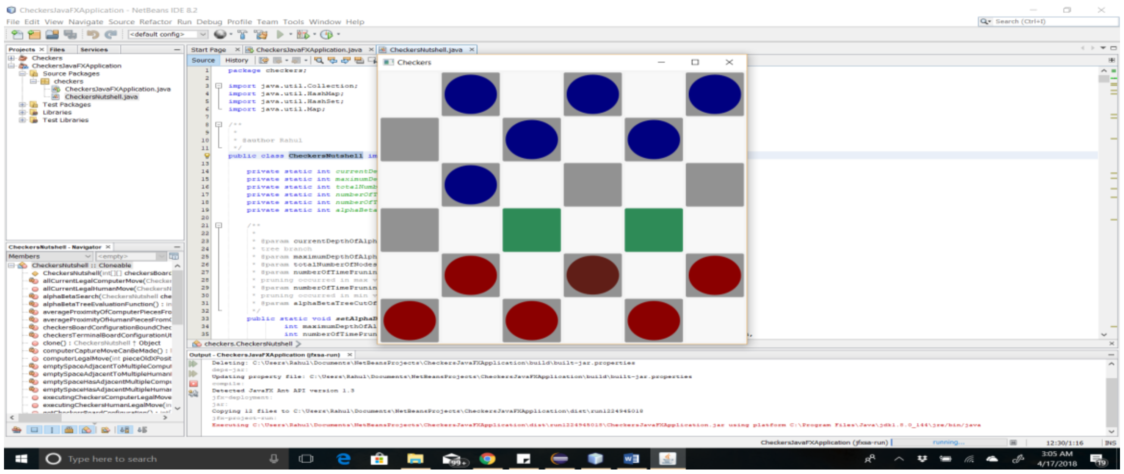
2. If **Player One** is **Computer:**

Note: After every Computer Move, the Checkers Alpha Beta Tree Statistics will be displayed in a different Dialog box as shown below.

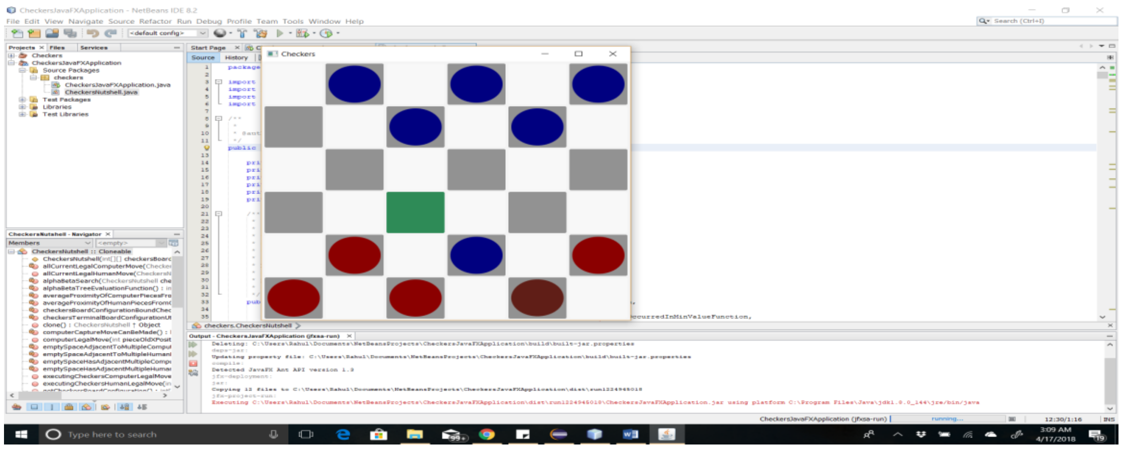


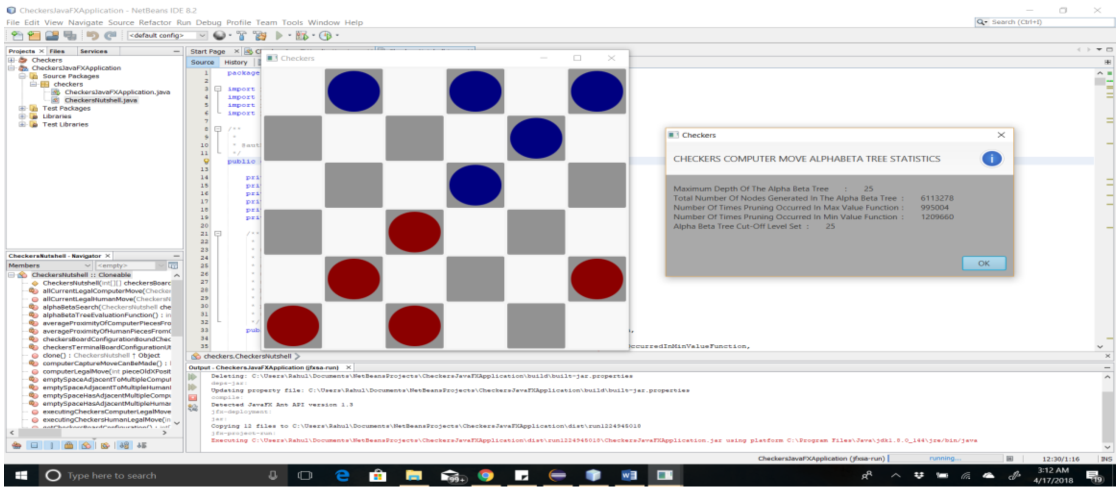
```
CHECKERS COMPUTER MOVE ALPHABETA TREE STATISTICS

Maximum Depth Of The Alpha Beta Tree       :       25
Total Number Of Nodes Generated In The Alpha Beta Tree  :       9720532
Number Of Times Pruning Occurred In Max Value Function :       3371448
Number Of Times Pruning Occurred In Min Value Function  :       1023373
Alpha Beta Tree Cut-Off Level Set  :       25
```

- The legal Move squares of the clicked Human Piece are highlighted by Green color as shown below. If a clicked Human Piece doesn't have a legal move no square is highlighted.



- Some Intermediate Checkers Board Configuration are shown below:





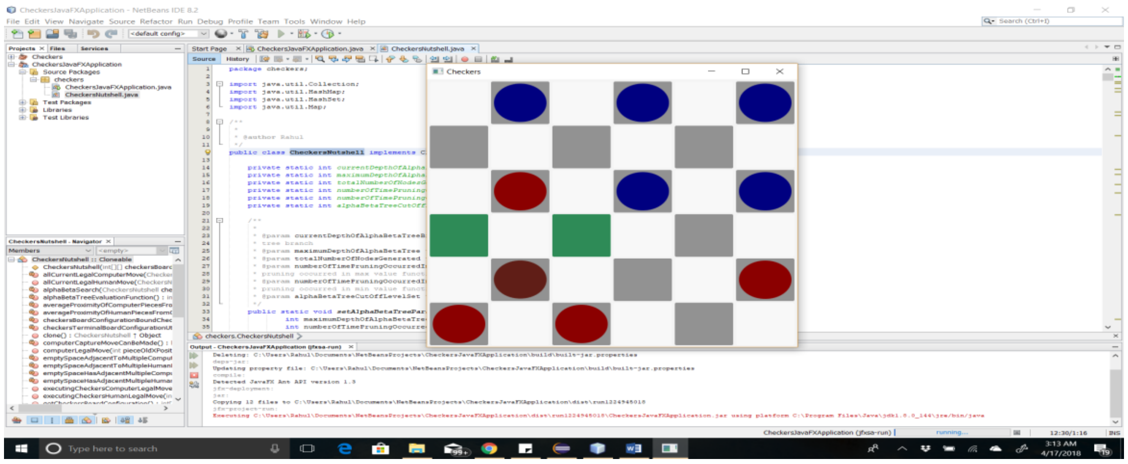
Checkers

CHECKERS COMPUTER MOVE ALPHABETA TREE STATISTICS

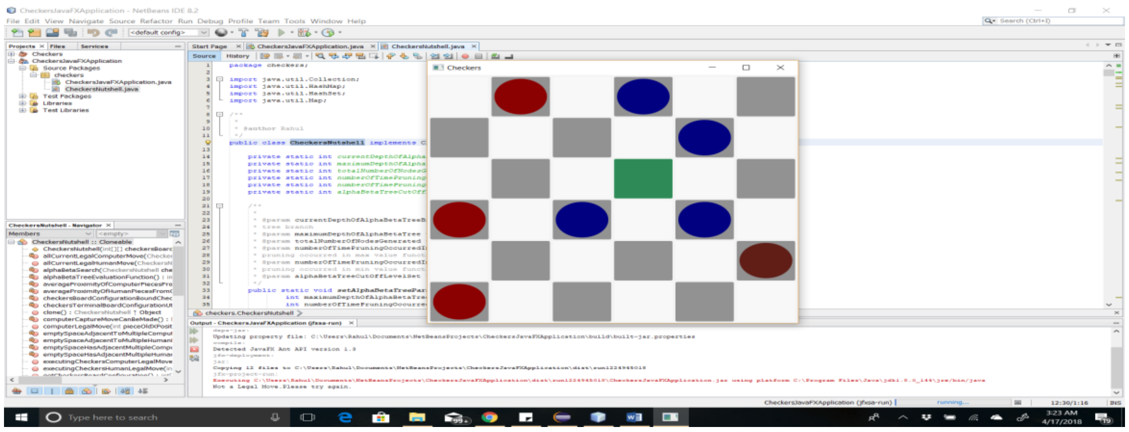
```
Maximum Depth Of The Alpha Beta Tree      :     25
Total Number Of Nodes Generated In The Alpha Beta Tree  :      6113278
Number Of Times Pruning Occurred In Max Value Function :      995004
Number Of Times Pruning Occurred In Min Value Function  :      1209660
Alpha Beta Tree Cut-Off Level Set  :     25
```

OK

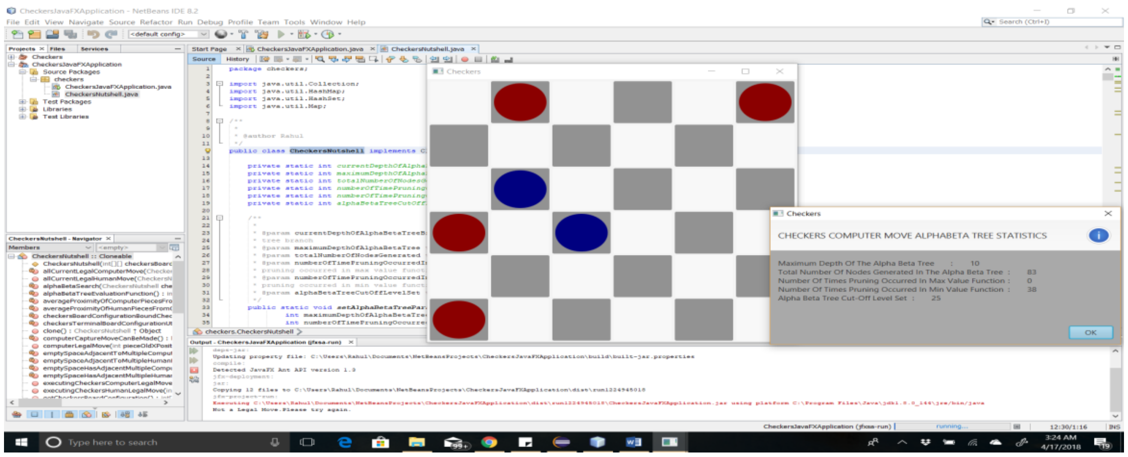
```
Deleting: C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\build\built-jar.properties
compile:
Detected JavaFX Ant API version 1.3
jfx-deployment:
jar:
Copying 12 files to C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945010
jfx-project-run:
Executing C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945010\CheckersJavaFXApplication.jar using platform C:\Program Files\Java\jdk1.8.0_144\jre/bin/java
```



```
Deleting: C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\build\built-jar.properties
compile:
Detected JavaFX Ant API version 1.3
jfx-deployment:
jar:
Copying 12 files to C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945010
jfx-project-run:
Executing C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945010\CheckersJavaFXApplication.jar using platform C:\Program Files\Java\jdk1.8.0_144\jre/bin/java
```



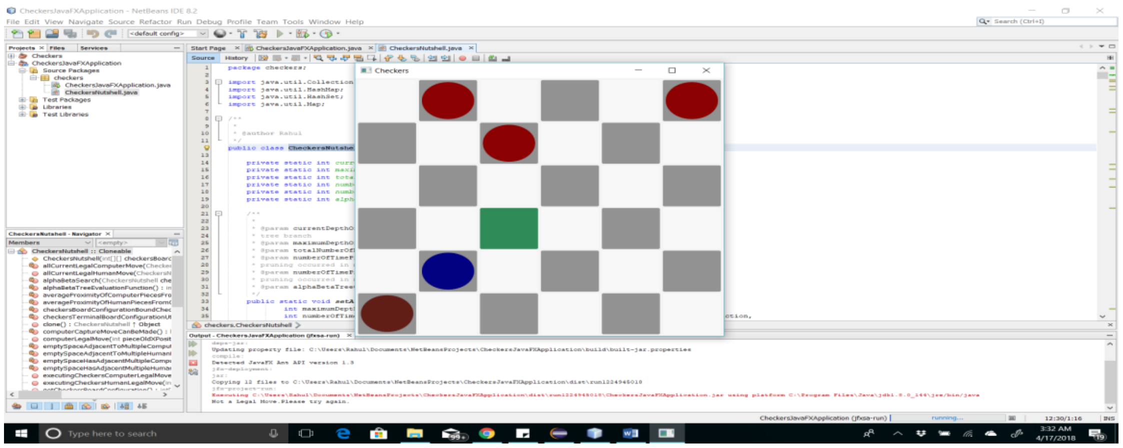
```
deps-jar:
Updating property file: C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\build\built-jar.properties
compile:
Detected JavaFX Ant API version 1.3
jfx-deployment:
jar:
Copying 12 files to C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945010
jfx-project-run:
Executing C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945010\CheckersJavaFXApplication.jar using platform C:\Program Files\Java\jdk1.8.0_144\jre/bin/java
Not a Legal Move.Please try again.
```



```
deps-jar:
Updating property file: C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\build\built-jar.properties
compile:
Detected JavaFX Ant API version 1.3
jfx-deployment:
jar:
Copying 12 files to C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945018
jfx-project-run:
Executing C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945018\CheckersJavaFXApplication.jar using platform C:\Program Files\Java\jdk1.8.0_144\jre/bin/java
Not a Legal Move.Please try again.
```

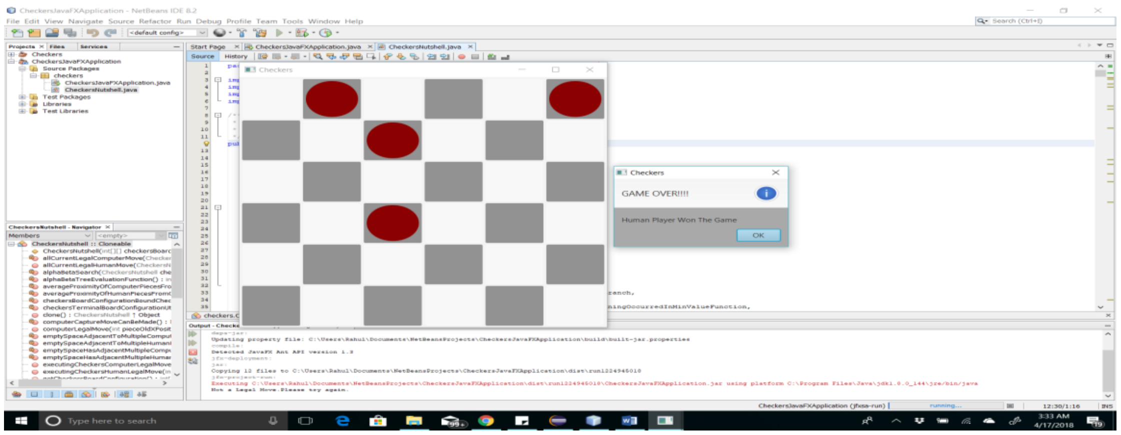
CHECKERS COMPUTER MOVE ALPHABETA TREE STATISTICS

```
Maximum Depth Of The Alpha Beta Tree     :     10
Total Number Of Nodes Generated In The Alpha Beta Tree :     83
Number Of Times Pruning Occurred In Max Value Function :     0
Number Of Times Pruning Occurred In Min Value Function :     38
Alpha Beta Tree Cut-Off Level Set  :     25
```



```
deps-jar:
Updating property file: C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\build\built-jar.properties
compile:
Detected JavaFX Ant API version 1.3
jfx-deployment:
jar:
Copying 12 files to C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945018
jfx-project-run:
Executing C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945018\CheckersJavaFXApplication.jar using platform C:\Program Files\Java\jdk1.8.0_144\jre/bin/java
Not a Legal Move.Please try again.
```

- Once the game is over, the result of the game is shown in a different dialog box. The Human Player won the game as shown in the below dialog box because the difficulty level was **Easy**:



GAME OVER!!!!

Human Player Won The Game

```
deps-jar:
Updating property file: C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\build\built-jar.properties
compile:
Detected JavaFX Ant API version 1.3
jfx-deployment:
jar:
Copying 12 files to C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945018
jfx-project-run:
Executing C:\Users\Rahul\Documents\NetBeansProjects\CheckersJavaFXApplication\dist\run1224945018\CheckersJavaFXApplication.jar using platform C:\Program Files\Java\jdk1.8.0_144\jre/bin/java
Not a Legal Move.Please try again.
```

<u>Checkers Game Program:</u>

1. CheckersJavaFXApplication Class

Helper Methods:

i. **computerPlayerMove:** It is used to call the "nextCheckersComputerLegalMoveBoardConfiguration" method of the CheckersNutshell class and display the CheckersBoard after the computer move. It also sets the alpha beta tree cut off level for every alpha beta tree search of the computer move executed depending on the difficulty level.

ii. **hardDifficultyLevelalphaBetaTreeCutOff:** helps to assign the alpha beta tree cut off level for the hard difficulty level based on which computer move number we are searching using the alpha beta tree search i.e. first computer move, second computer move, etc.

iii. **mediumDifficultyLevelalphaBetaTreeCutOff**: helps to assign the alpha beta tree cut off level for the medium difficulty level based on which computer move number we are searching using the alpha beta tree search i.e. first computer move, second computer move, etc.

iv **checkersGameOverAfterComputerMove** : It is used to check if the game is over after every computer move and display the appropriate game result as who won the game if the game is over and if the game is not over it also checks if a human player legal move is left and forfeits the human player move if no human player legal move is left.

v. **checkersGameOverAfterHumanMove**: It is used to check if the game is over after every human move and display the appropriate game result as who won the game if the game is over and if the game is not over it also checks if a computer player legal move is left and either forfeits the computer player move if no computer player legal move is left or performs the computer player move using the alpha beta tree search.

2. CheckersNutshell Class

Static Variables:

currentDepthOfAlphaBetaTreeBranch: Used to keep track of the depth of the current alpha beta tree branch while performing the alpha beta tree search.

maximumDepthOfAlphaBetaTree: Used to keep track of the maximum depth of any branch while performing the alpha beta tree search. This value is updated by currentDepthOfAlphaBetaTreeBranch depending on which value is greater (maximumDepthOfAlphaBetaTree or currentDepthOfAlphaBetaTreeBranch) when either the terminal state is reached or the cut off occurs.

totalNumberOfNodesGenerated: Used to keep a track of the total number of nodes generated while performing the alpha beta tree search.

numberOfTimePruningOccurredInMaxValueFunction: Used to keep a track of the number of times pruning occurred in the max value function.

numberOfTimePruningOccurredInMinValueFunction: Used to keep a track of the number of times pruning occurred in the min value function.

alphaBetaTreeCutOffLevelSet: Used to keep a track of the alpha beta tree cut off level set while performing the alpha beta tree search.

Helper Methods:

setAlphaBetaTreeParameters: Used to set the alpha beta tree parameters described above every time before starting the alpha beta tree search.

nextCheckersHumanLegalMoveBoardConfiguration: This method checks if the human player move is legal based on the checkers game rules by calling “humanLegalMove” and executes the human player move if the move is legal by calling “executingCheckersHumanLegalMove”.

humanCaptureMoveCanBeMade: This method checks if a human player checkers piece capture move is possible. It is used by humanLegalMove to determine if the human player move is legal or not.

nextCheckersComputerLegalMoveBoardConfiguration: This method checks if the computer player move is legal based on the checkers game rules by calling “computerLegalMove” and executes the computer player move if the move is legal by calling “executingCheckersComputerLegalMove”.

computerCaptureMoveCanBeMade: This method checks if a computer player checkers piece capture move is possible. It is used by computerLegalMove to determine if the computer player move is legal or not.

isCheckersGameOver: This method is used to check if the checkers game is over.
i. This method calls “isAHumanPieceLeft” to check if any human piece is left. If no human piece is left the computer won the game and a utility value of 100 is returned.

ii. This method calls “isAComputerPieceLeft” to check if any computer piece is left. If no computer piece is left the human won the game and a utility value of -100 is returned.

iii. This method calls “isHumanPieceLegalMoveLeft” and “isComputerPieceLegalMoveLeft” to check if no human and computer legal move are left respectively and if both are true then it calls “checkersTerminalBoardConfigurationUtilityValue” which returns a utility value of 100, -100 or 0 depending on the number of human and computer checkers pieces left on the checkers board.

Statements (i), (ii), (iii) are executed by using conditional statements.

allCurrentLegalHumanMove: This returns a Set of all legal human moves for a human checkers piece. This method is used to highlight the legal moves of a clicked human checkers piece by green color on the GUI.

allCurrentLegalComputerMove: This returns a 2-D array of the legal computer moves of all the computer pieces on the checkers board. This method is used by “alphaBetaSearch” while performing the alpha beta search to execute a computer move.

alphaBetaSearch: Used to perform the alpha beta search to execute the best possible computer move. “maxValue”, “minValue” are used with “alphaBetaSearch” method to perform the alpha beta search.

printAlphaBetaTreeStatistics: Used to print the alpha beta tree paramters after the alpha beta search is completed.

Terminal states And Their Utility Values:

1. **Human Player wins the game when there are no Computer Player Checkers Piece left on the board:**
   Utility Value: -100
2. **Computer Player wins the game when there are no Human Player Checkers Piece left on the board:**
   Utility Value: 100
3. **Human Player wins the game when there are no Computer and Human Player legal moves left and the number of Human Player Checkers pieces are greater than the number of Computer Player Checkers pieces.**
   Utility Value: -100
4. **Computer Player wins the game when there are no Computer and Human Player legal moves left and the number of Computer Player Checkers pieces are greater than the number of Human Player Checkers pieces.**
   Utility Value: 100
5. **The game is drawn when there are no Computer and Human Player legal moves left and the number of Computer Player Checkers pieces is equal to the number of Human Player Checkers pieces.**
   Utility Value: 0

Evaluation Function Used:

The method **"alphaBetaTreeEvaluationFunction"** in CheckersNutshell class is used as an evaluation function to calculate the utility value of the checkers board state based on **12** different heuristic functions when a cut off occurs.

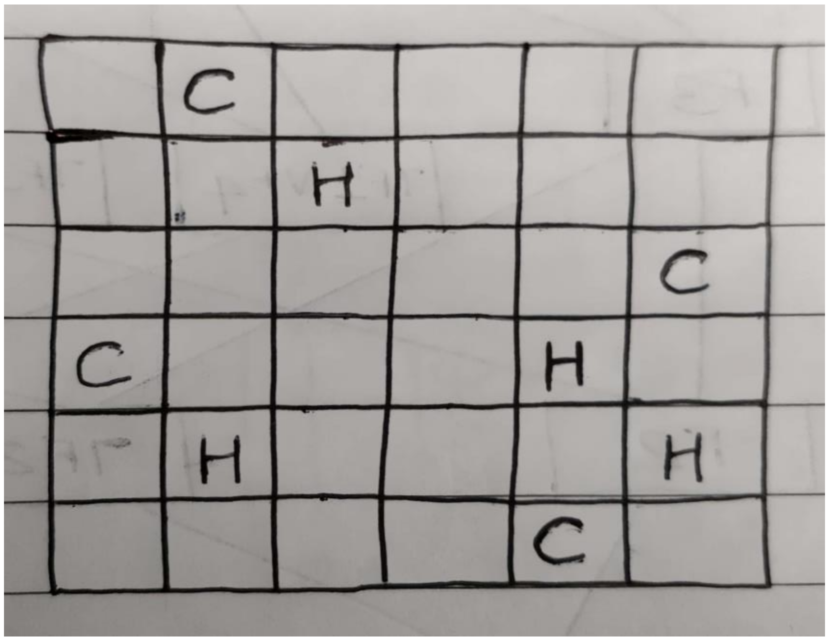
The evaluation function is the weighted sum of 12 different features (heuristic functions) and returns a utility value in the range between -100 and 100.

```
private int alphaBetaTreeEvaluationFunction() {

  return (numberOfComputerPieceLeftHeuristic() - numberOfHumanPieceLeftHeuristic()
           + averageProximityOfComputerPiecesFromOtherSideHeuristic()
           -  averageProximityOfHumanPiecesFromOtherSideHeuristic()
           + numberOfComputerPieceCaptureMovePossibleHeuristic()
           -  numberOfHumanPieceCaptureMovePossibleHeuristic()
          +  numberOfComputerPieceRegularMovePossibleHeuristic()
           -  numberOfHumanPieceRegularMovePossibleHeuristic()
          +  safeComputerPiecesHeuristic() - safeHumanPiecesHeuristic()
          +  emptySpaceAdjacentToMultipleComputerPieceHeuristic()
          -   emptySpaceAdjacentToMultipleHumanPieceHeuristic());

}
```

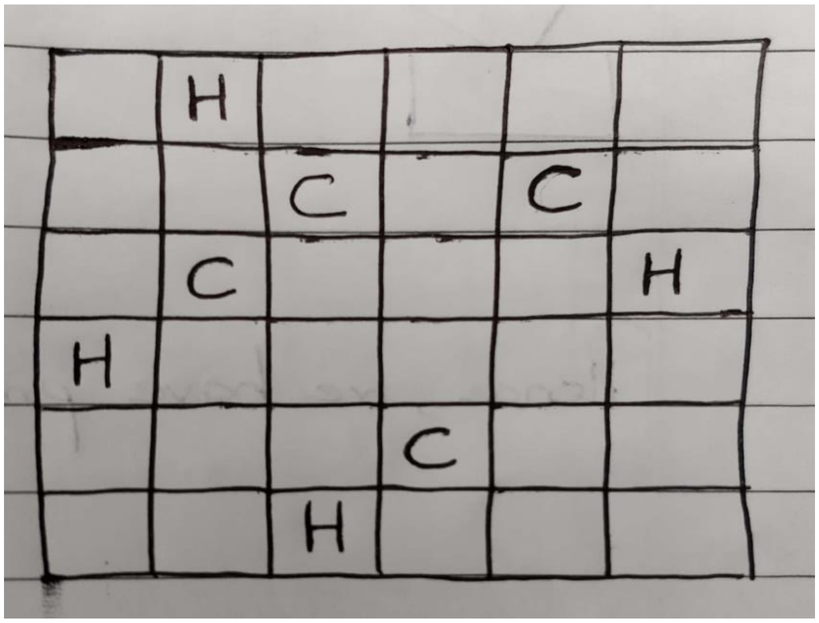
Heuristic Functions:

1. **numberOfComputerPieceLeftHeuristic()**: returns the total number of Computer Pieces left on the checkers board. **Weight assigned: 1**
2. **numberOfHumanPieceLeftHeuristic()**: returns the total number of Human Pieces left on the checkers board. **Weight assigned: -1**
3. **averageProximityOfComputerPiecesFromOtherSideHeuristic()**: returns the average of the vertical distances of all the checkers computer pieces from the other side of the board. The significance of using this heuristic is that the closer the pieces are to the other side of the board the lesser are the number of legal moves possible in future for that piece. **Weight assigned: 1**
4. **averageProximityOfHumanPiecesFromOtherSideHeuristic()**: returns the average of the vertical distances of all the checkers human pieces from the other side of the board. The significance of using this heuristic is that the closer the pieces are to the other side of the board the lesser are the number of legal moves possible in future for that piece. **Weight assigned: -1**
5. **numberOfComputerPieceCaptureMovePossibleHeuristic():** returns the number of capture moves that can be made by all the computer pieces on the current checkers board configuration. **Weight assigned: 1**
6. **numberOfHumanPieceCaptureMovePossibleHeuristic()**: returns the number of capture moves that can be made by all the human pieces on the current checkers board configuration. **Weight assigned: -1**
7. **numberOfComputerPieceRegularMovePossibleHeuristic()**: returns the number of regular moves that can be made by all the computer pieces on the current checkers board configuration. The significance of using this heuristic along with "numberOfComputerPieceCaptureMovePossibleHeuristic()" is that we get an idea of the total number of legal computer player moves possible. **Weight assigned: 1**
8. **numberOfHumanPieceRegularMovePossibleHeuristic():** returns the number of regular moves that can be made by all the human pieces on the current checkers board configuration. The significance of using this heuristic along with "numberOfHumanPieceCaptureMovePossibleHeuristic()" is that we get an idea of the total number of legal human player moves possible. **Weight assigned: -1**
9. **safeComputerPiecesHeuristic()**: returns the number of computer checkers piece along the four edges of the checkers board. The significance of using this heuristic is that the checkers piece along the edges are safe since they cannot be attacked. **Weight assigned: 1.**

   PFB different positions of checkers computer and human pieces where C is the computer piece and H is the human piece. It is evident that the computer piece along any of the four edges cannot be attacked by the human piece.

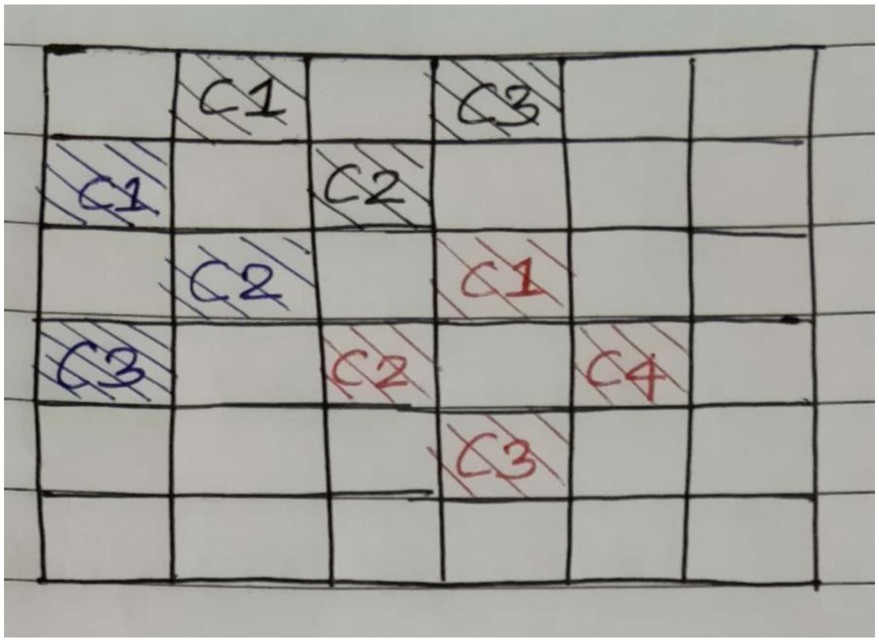


10. **safeHumanPiecesHeuristic():** returns the number of human checkers piece along the four edges of the checkers board. The significance of using this heuristic is that the checkers piece along the edges are safe since they cannot be attacked. **Weight assigned: -1.**
    PFB different positions of checkers computer and human pieces where C is the computer piece and H is the human piece. It is evident that the human piece along any of the four edges cannot be attacked by the computer piece.

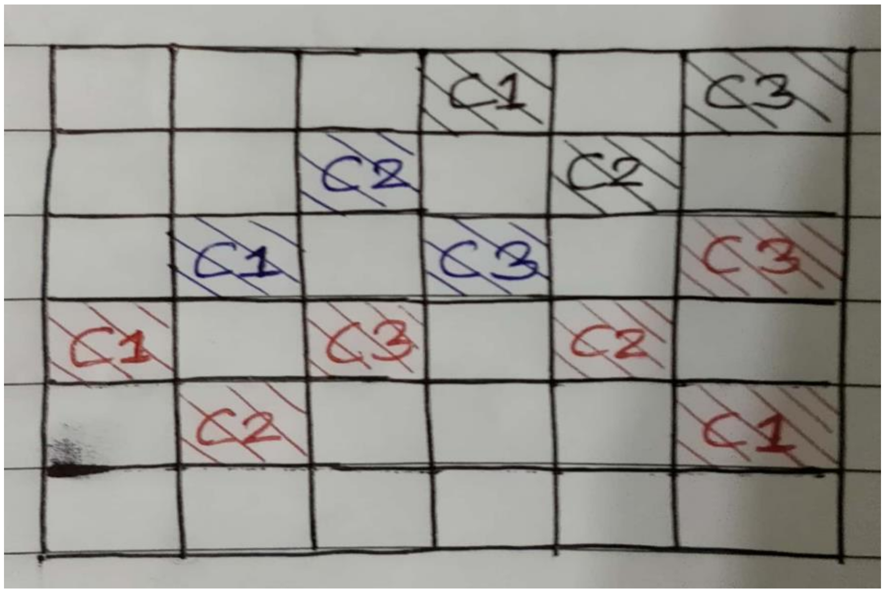


11. **emptySpaceAdjacentToMultipleComputerPieceHeuristic()**: returns the number of patterns where a blank space is surrounded by three or four computer checkers pieces. The significance of using this heuristic is that it ensures some checkers computer piece are safe depending on the pattern. **Weight assigned: 1**

    In the figure below there are three different patterns that can be identified by the "**emptySpaceAdjacentToMultipleComputerPieceHeuristic()"** heuristic. The three patterns are marked with red, blue and black color. In the pattern marked by red, all the computer pieces C1, C2, C3 and C4 cannot be attacked by the human pieces. In the pattern marked by black, all the computer pieces C1, C2 and C3 cannot be attacked by the human pieces. In the pattern marked by blue, all the computer pieces C1, C2 and C3 cannot be attacked by the human pieces.

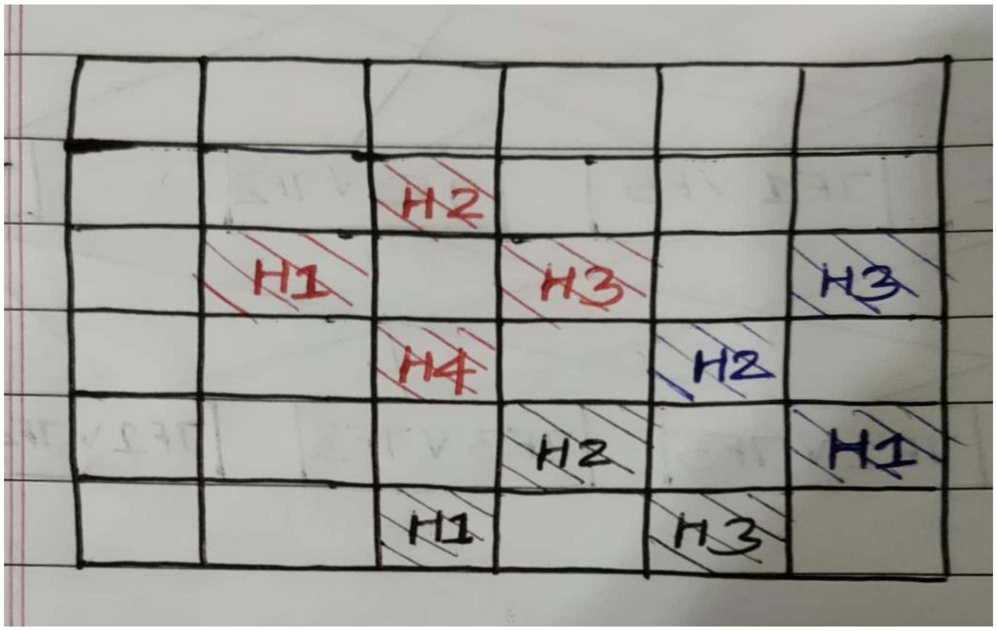


In the figure below there are four different patterns that can be identified by the "**emptySpaceAdjacentToMultipleComputerPieceHeuristic()"** heuristic. Two patterns are marked with red, one with blue and one with black color. In the rightmost pattern marked by red, all the computer pieces C1, C2, C3 cannot be attacked by the human pieces. In the other pattern marked by red, computer pieces C2 and C1 cannot be attacked by the human pieces but maybe computer piece C3 can be attacked. In the pattern marked by black, all the computer pieces C1, C2 and C3 cannot be attacked by the human pieces. In the pattern marked by blue, computer piece C2 cannot be attacked by the human pieces but computer piece C1 and C3 may or may not be safe depending on which human piece is attacking it.

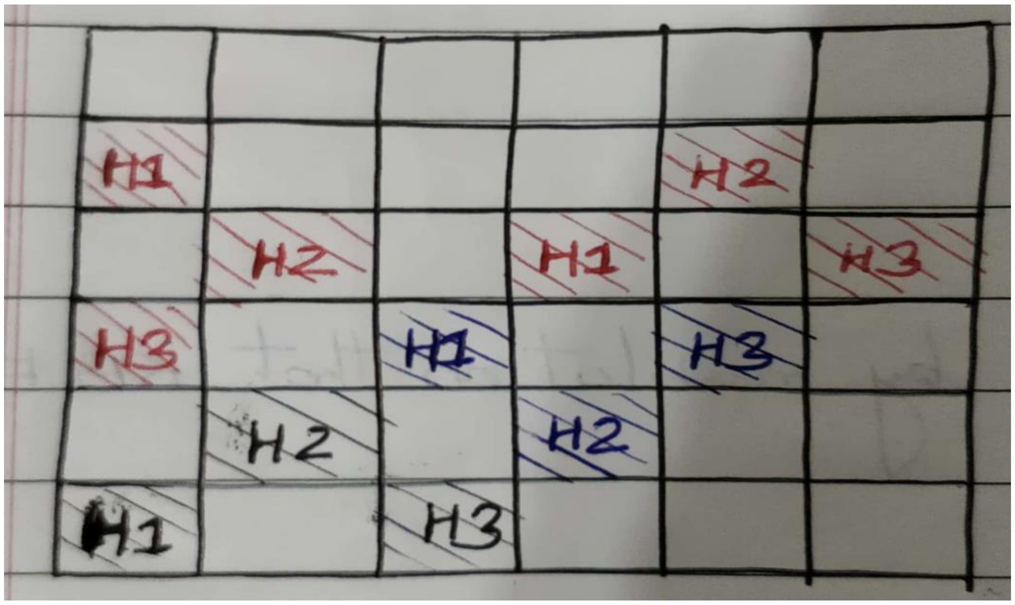


12. **emptySpaceAdjacentToMultipleHumanPieceHeuristic():** returns the number of patterns where a blank space is surrounded by three or four human checkers pieces. The significance of using this heuristic is that it ensures some checkers human piece are safe depending on the pattern. **Weight assigned: -1**

In the figure below there are three different patterns that can be identified by the "**emptySpaceAdjacentToMultipleHumanPieceHeuristic()"** heuristic. The three patterns are marked with red, blue and black color. In the pattern marked by red, all the human pieces H1, H2, H3 and H4 cannot be attacked by the computer pieces. In the pattern marked by black, all the human pieces H1, H2 and H3 cannot be attacked by the computer pieces. In the pattern marked by blue, all the human pieces H1, H2 and H3 cannot be attacked by the computer pieces.



In the figure below there are four different patterns that can be identified by the "**emptySpaceAdjacentToMultipleHumanPieceHeuristic()"** heuristic. Two patterns are marked with red, one with blue and one with black color. In the leftmost pattern marked by red, all the human pieces H1, H2, H3 cannot be attacked by the computer pieces. In the other pattern marked by red, human pieces H2 and H3 cannot be attacked by the computer pieces but maybe human piece H1 can be attacked. In the pattern marked by black, all the human pieces H1, H2 and H3 cannot be attacked by the computer pieces. In the pattern marked by blue, human pieces H2 cannot be attacked by the computer pieces but human piece H1 and H3 may or may not be safe depending on which computer piece is attacking it.



Different Levels Of Checkers Game Difficulty:

1. **Easy:** The checkers alpha beta tree cut off level is set to 10 for every computer move executed using the alpha beta tree search. The evaluation function is used everytime when the cut-off occurs to return the utility value of the checkers board configuration.
2. **Medium:** The checkers alpha beta tree cut off level is set to 20 for the first computer move executed using the alpha beta tree search and then the alpha beta tree cut off level is incremented by 2 for every next computer move executed. The evaluation function is used everytime when the cut-off occurs to return the utility value of the checkers board configuration.
3. **Hard:** The checkers alpha beta tree cut off level is set to 25 for the first computer move executed using the alpha beta tree search. From, the second computer move onwards no cut -off level is set and the entire alpha beta tree is searched for the best computer move to be executed. The evaluation function is used everytime when the cut-off occurs to return the utility value of the checkers board configuration.