Artificial Intelligence : Checkers

Prateek Mishra
4th Semester (I.T.) *Department of Information Technology,*
Indian Institute of Information Technology Allahabad.
iit2018199@iiita.ac.in

***Abstract*—In this paper I have described my solution to making a GUI based checkers and designing an artificial intelligent agent to play the same.**

I. Introduction

Checkers is a very old strategy board game. This game has been there since the 3000BC [1]. This game is of particular interest in the field of Artificial intelligence particularly because of its recursive structure and simple states. This game consists of two players playing alternatively, having similar moves, which can be very easily modelled as a computer program.

II. Game Description

A. *Starting[2]*

The players start by tossing a coin to decide who plays first. In my version of the game the Black always plays first.

B. *Objective*

The objective of the game is to, kill your opponent's pieces or block them to an extent that he/she cannot make any further move.

C. *Setup*

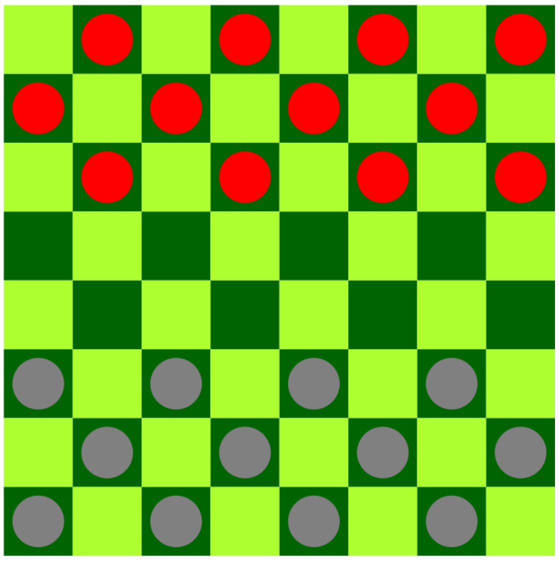


Fig 1 (A snap from the game, showing the pieces)

The board consists of alternative red and black squares like chess. The dimension of the board is 8 x 8. The board is set up in a peculiar way. There are two types of pieces – RED and BLACK. Both of these are placed on the dark squares on the board. Considering the board to be an 8 x 8 matrix with the top left corner being the origin (0, 0) and the bottom right corner being the point (7 x 7). The black pieces are arranged on the following coordinates – (5,0), (7,0), (6,1), (5,2), (7,2), (6,3), (5,4), (7,4), (6,5), (5,6), (7,6), (6, 7) and the red pieces are arranged on the following coordinates – (1, 0), (0, 1), (2, 1), (1, 2), (0, 3), (2, 3), (1, 4), (0, 5), (2, 5), (1, 6), (0, 7), (2, 7). Notice that all these positions are of black boxes as the first block on the top left is a red box.

D. *Rules*

Both the players take turns to move their respective pieces. The players can slide their pieces along the diagonals in forward direction (away from them) if the new location is not occupied by any other piece.[3]
A player can also capture (kill) the opponent's piece by jumping over it if the opponent's piece is on the adjacent diagonal square and the location across the opponent's piece (on the same diagonal line) is empty. This way the player who jumps over the opponent's piece is said to have captured the opponent's piece and is rewarded with a score of +1.
Even during a player turn to play, even one of his pieces are in a capturing position he must make the capturing move. However, if there are multiply pieces in capturing positions then the player is allowed to choose from the pieces but he must make a capture. If after a capture a player's piece lands on another piece from where he can capture another opponent's piece then he must capture that piece in the same turn.
If during playing a piece reaches the opposite end. (i.e. black pieces reaching the points (0, k): k ∈ [0, 8] or the red pieces reach the points (7, k): k ∈ [7, 8]) then that piece is said be become the king and now the piece can move in forward as well as backward directions (still along the diagonals though).

E. *Approach*

The game has been modelled using a state class that holds the present state (location of pieces, scores, whose turn) and the players alternatively take turns to get new states. The model the AI agent I have used Minimax algorithm, which used recursion to make the most favourable decision.

III. Classes

This section contains, the Modelling of the above proposed approach.

A. *Main.java*

This class is responsible managing the GUI of the game. The GUI works as follows.

1. As soon as the game starts, the user is asked to choose a mode to continue playing. There are three modes available viz a viz AI vs AI, AI vs Human, Human vs Human.

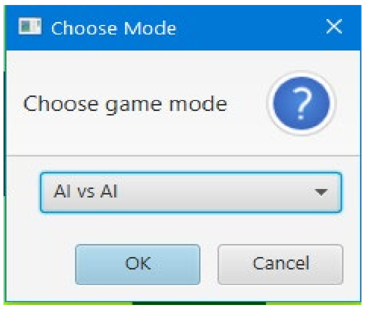


Fig 2 (A snap from game showing the choose game dialogue)

From the drop down if the user chooses the AI vs Human mode then the user is again given 5 options to choose the difficulty.

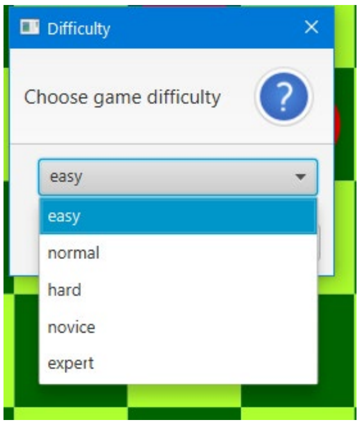


Fig 3(A snap from the game showing the choose difficulty mode)

Upon choosing the options the game begins as is shown above in Fig1. To play the user clicks on his respective pieces. Upon clicking the piece the possible moves applicable on the piece are reflected on the board in the form of pink circles.

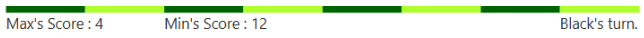


Fig9(A figure showing the scoreboard as featured in the game.)

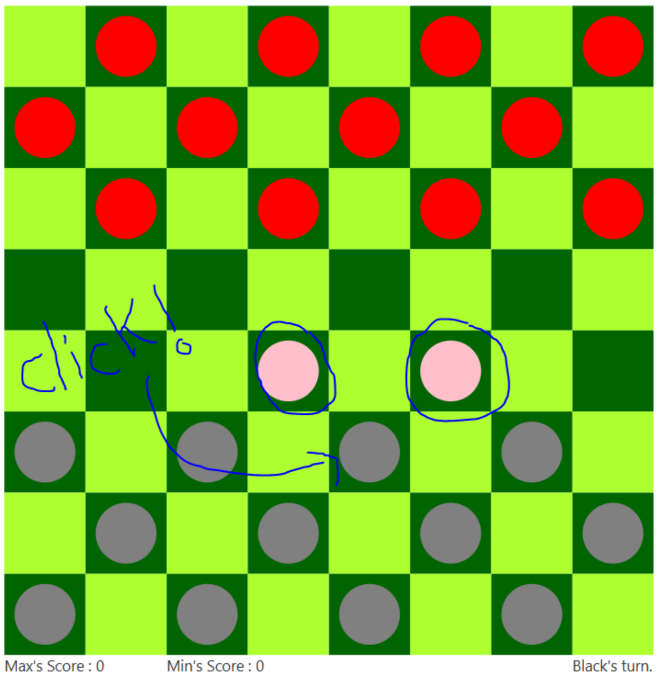
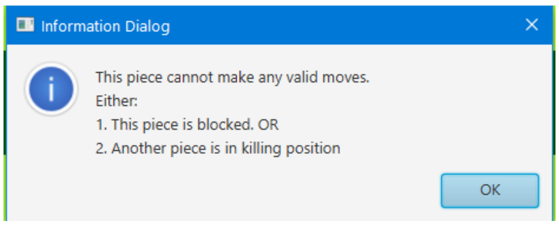


Fig 4 Showing the emergence of possible moves on clicking.

Upon clicking on the pink circles, the respective piece moves to the new location. Under the hood - the pink circles and the original circles have click listeners on them that respond when a click is made on the circles. Upon receiving a click on the pink circle, the respective coordinates of the circles are changed in the game class object and then the GUI is drawn again at the new position.
The rules of the game suggest that if a player in one of his chances can capture the other players piece, then he must do that. However, if there are multiple pieces in capturing position the player is free to choose any one of them. So, in such a situation the pink circles only show up for the pieces that are in a capturing position and if the user clicks on another piece, he is given the following persistent notification.



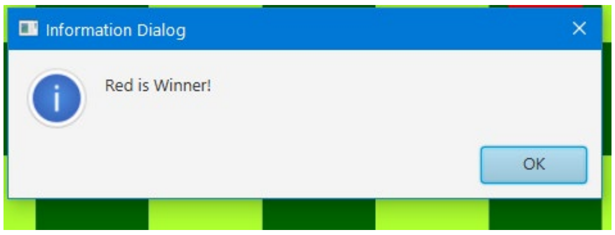
Information Dialog

This piece cannot make any valid moves.
Either:
1. This piece is blocked. OR
2. Another piece is in killing position

Fig 5 (A snap from the game signaling the user that the current piece cannot be moved.)

Note that a similar notification shows up if the players tries to move a piece that is blocked by the opponents or one of his own pieces.
As is shown in the previous images (Fig 4) and (Fig 9) The game GUI also features a score board at the bottom of the screen which shows the scores of the two players and the color of the player who plays next.

Last but not the least, upon completion of the game a dialogue is shown which tells the name of the winner.


Information Dialog

Red is Winner!

Fig 6 (The dialogue shown on game completion.)

This class has an object of the main class and upon detecting the click uses the playNextMove() function of the game class.

B. *Game.java*

This class contains the main playing logic of the game. It mainly holds the state variable that is the present state of the game.
This class contains a function :

```
void playNextMove(boolean oneHuman,
boolean twoHuman, int depth)
```

This function calls the minValue or the maxValue function depending upon which players turn it is. It receives the two Boolean variables which denote if the 1st players are human.
The other variable is the depth, till which the function should call recursion. This function is also responsible for changing the turn of every player after each move.
This class is also responsible for setting up the board initially.
This class has the minVal and the maxVal functions that are responsible for the artificial intelligence agent.
The pseudo code of the agents is as follows :

```
maxValue(State s, int alpha, int beta, int depth) {
if(depth == maxdepth) return guessUtility of state.
Int utility = Min value;
For all possible actions a on the present state
newState s = a(oldState);
int utilityNext = minValue(depth + 1, newState, alpha,
beta);
}
alpha = min(alpha, utilityNext);
if(utility > beta) return utility;
return utility;
```

```
minValue(State s, int alpha, int beta, int depth) {
if(depth == maxdepth) return guessUtility of state.
Int utility = Max value;
For all possible actions a on the present state
newState s = a(oldState);
int utilityNext = maxValue(depth + 1, newState, alpha,
beta);
}
beta = max(beta, utilityNext);
if(utility < alpha) return utility;
return utility;
```

As can be seen I have used alpha, beta pruning here[4]

C. *State.java*

This is the main class of the code that holds the current game state like the present arrangement of the board. The present scores of the players, the present set of possible moves for each player, whose turn it is etc.
This has the functions discussed above like returning the map of all possible actions on a state.
The pseudo code of the same looks like :

```
Map<Coordinate, ArrayList<Action>>
getStateActions(){
For all pieces :
actions.append(all possible actions on the piece);
return actions;
}
```

This state also has the function to return a new state after application of an action on a previous state.

This class has all the complex logic of the game like, to implement multiple turns the action of the class next formed only consider the attacking actions of the attacking player. In such a situation the turn variable is not tickled and again the same player plays. As the actions of this state have already been pruned, it gives an illusion that the same player is playing his turn again while in reality the next player's turn has been skipped.

D. *Piece.java*

This class contains the information of the pieces like the coordinates of the pieces and if the piece belongs to the max player and if the piece is a king. The only functions present in this state are getters, setters and constructors.

E. *Coordinate.java*

This class only contains two variables the x coordinate and the y coordinate. I had to create this class as there is no default pair< int, int> data structure present in java.

F. *Action.java*

This class stores the actions that are applied on the states throughout the game. This has variables like oldCoordinate that contains the coordinates from where the piece is moved and newCoordinate – the coordinates to where is piece is moved after the application of the action. The scores of the players after application of the action. And if the action is a capturing action.

IV. ANALYSIS

The code is a MiniMax algorithm code with the time complexity being O(b ^ m) and space complexity is O(b * m) where b is the branching factor of the tree and m is the maximum depth of the recursion tree.

The algorithm plays better moves if the depth of the recursion tree is increased.

Challenges

One of the biggest challenges that I faced was managing consecutive turns. It took me quite a while to come to the solution to skip the other players turns.

Another major challenge was making copies of objects. As I was not aware that java always makes copies by reference I had to invest a lot of time in finding out this bug.

Acknowledgment

I am thankful to Dr. Rahul Kala for giving me this assignment.

References

While doing this project I took help from the following sources.

[1] Oxland, Kevin (2004). Gameplay and design (Illustrated ed.). Pearson Education. p. 333. ISBN 978-0-321-20467-7.

[2] https://www.siammandalay.com/blogs/puzzles/checkers-game-basic-rules-win

[3] https://www.wikihow.com/Play-Checkers

[4] https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/