Checkers

Prabal Tikeriha (IIT2018140)

V Semester B.Tech, Department of Studies in Information Technology,

Indian Institute of Information Technology, Allahabad , India

***Abstract**: This paper presents the automation of checkers game through GUI. It also gives privilege ro user to take steps using automation.*

I. **Checkers Rule**

checkers is a group of strategy board games for two players which involve diagonal moves of uniform game pieces and mandatory captures by jumping over opponent pieces.

A player may not move an opponent's piece. A move consists of moving a piece diagonally to an adjacent unoccupied square. If the adjacent square contains an opponent's piece, and the square immediately beyond it is vacant, the piece may be captured (and removed from the game) by jumping over it.

Only the dark squares of the checkerboard are used. A piece may move only diagonally into an unoccupied square. When presented, capturing is mandatory in most official rules, although some rule variations make capturing optional. In almost all variants, the player without pieces remaining, or who cannot move due to being blocked, loses the game.

MEN

Uncrowned pieces (men) move one step diagonally forwards, and capture an opponent's piece by moving two consecutive steps in the same line, jumping over the piece on the first step. Multiple enemy pieces can be captured in a single turn provided this is done by successive jumps made by a single piece; the jumps do not need to be in the same line and may "zigzag" (change diagonal direction).

KING

When a man reaches the kings row (also called crown head, the farthest

row forward), it becomes a king, and is marked by placing an additional piece on top of the first man (crowned), and acquires additional powers including the ability to move backwards and (in variants where they cannot already do so) capture backwards. Like men, a king can make successive jumps in a single turn provided that each jump captures an enemy man or king.

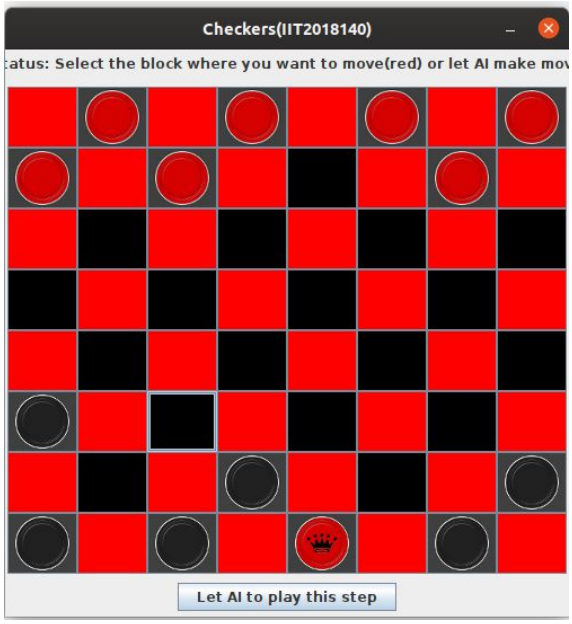

Fig 1. King is made when it reaches opponent side last row

II. **Automation method**

In this report, we are going to focus on competitive multi agents. Since in checkers both players want to make the best possible moves.

Here, we are using Adversarial Search which is likely to be used in cases where agents have conflicting goals or win lose situations.

To be more specific we are using Minmax algorithm

The Minmax algorithm tries to predict the opponent's behaviour. It predicts the opponent will take the worst action from our viewpoint. We are MAX - trying to maximise our score / move to the best state. Opponent is MIN - tries to minimise our score / move to the worst state for us.

III. Design and Implementation

GUI: It consists of 8X8 button grid which may be empty or filled with men or king of respective players.

To move a piece one should first select the piece and then select an empty cell according to the jump of one or two cell.

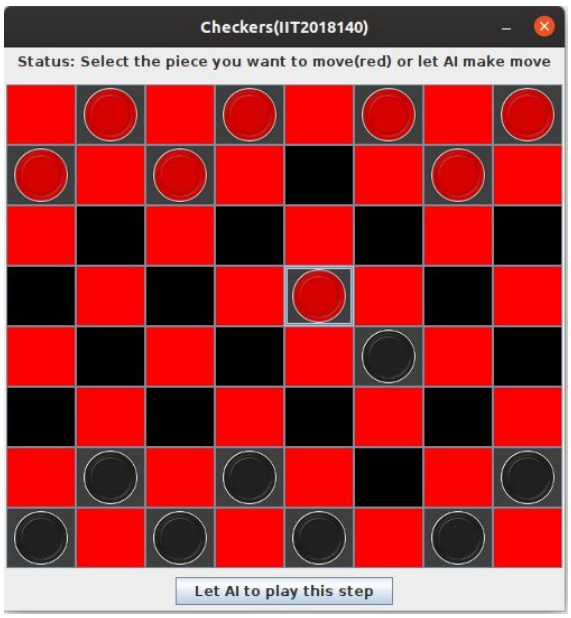

Fig 2. Snapshot of selecting piece

Two cell jump can be accomplished only when it can capture an opponent's piece.

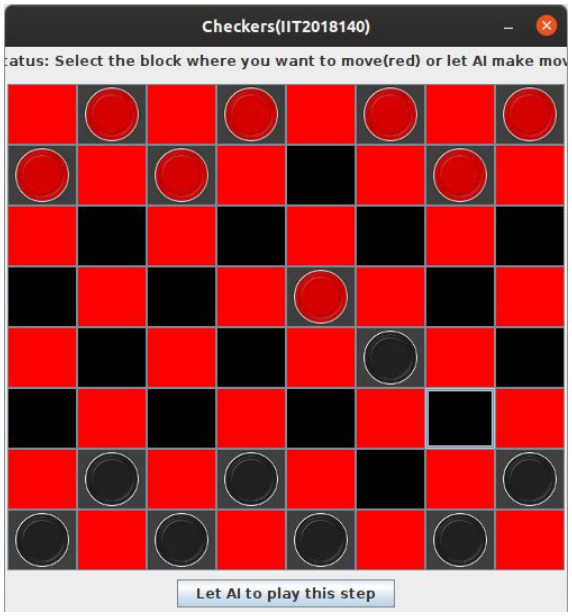

Fig 3. Snapshot of selecting empty cell to jump (it can be immediate to itself or opponent piece which is immediate to itself)

Any player can have the simultaneously play according to the AI predicted best move via Ai play button.

AI prediction:
Next move prediction by AI is done through Minmax algorithm in which turn by turn moves are calculated.
Since this can be very time consuming so we considered to use the heuristic.

Heuristics:

First most likely the next possible move is done by calculating the that player move which is done by calculating all feasible move of that situation that player can move and then opponent player's best move is considered(according to opponent) and then again repeated for further step.
Since this could be very time consuming so we have done cuttoff after some steps and considered best move.

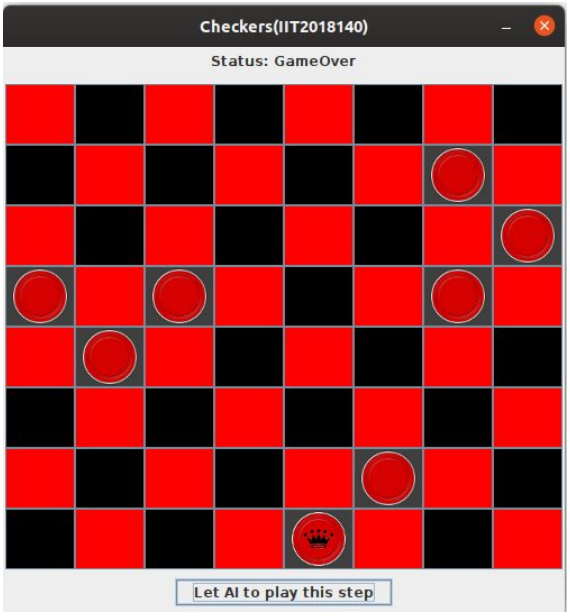

Fig 4. Snapshot when game is over.

Now, every step's best move is calculated via max(player-opponent's piece).

Pseudo code:
Minmax(state):
Next_state ← Max(state)

Return Next_state

Max(state):
If step = cutoff-step:
Return state
For all feasible next steps:
Next_best_state←Min(feasible_next_state)
Return Next_best_state

Min(state):
If step = cutoff-step:
Return state
For all feasible next steps:
Next_best_state←Max(feasible_next_state)
Return Next_best_state

IV. **References**

1. Dr. Mark Humphrys School of Computing. Dublin City University.
2. AI Course Final Project - Checkers
3. stackoverflow