Checkers

Human vs AI using Minimax

Game Rules

01

Board Size

Our board is of 8X8 Size, i.e 64 squares.

02

Pieces Movement

Each player takes their turn by moving a piece . Pieces are always moved DIAGONALLY and in the Forward Direction only.

03

King Pieces

The last row is called the King row. If a person manages to get a piece to the king row then the said piece becomes a king and that piece can move forward and backward.

04

Win Condition

You win the game when the opponent has no more pieces or can't move. If neither can move then its a draw.

ALGORITHM USED BY AI

01

MINIMAX ALGORITHM

It is a recursive or backtracking Algorithm used in decision-making and game theory.

MINIMAX

1. There are two players involved in a game, called MIN and MAX. The player MAX tries to get the highest possible score and MIN tries to get the lowest possible score, i.e., MIN and MAX try to act opposite of each other.
2. Our AI plays as the beige pieces and the player plays as the black pieces.
3. The score is evaluated as The number of beige pieces – the number of black pieces.
4. The algorithm aims to maximise the number of beige pieces and predict the players moves by assuming that the player wants to minimize the number of beige pieces.
5. Mini-Max algorithm uses recursion/backtracking to search through the game-tree.

02

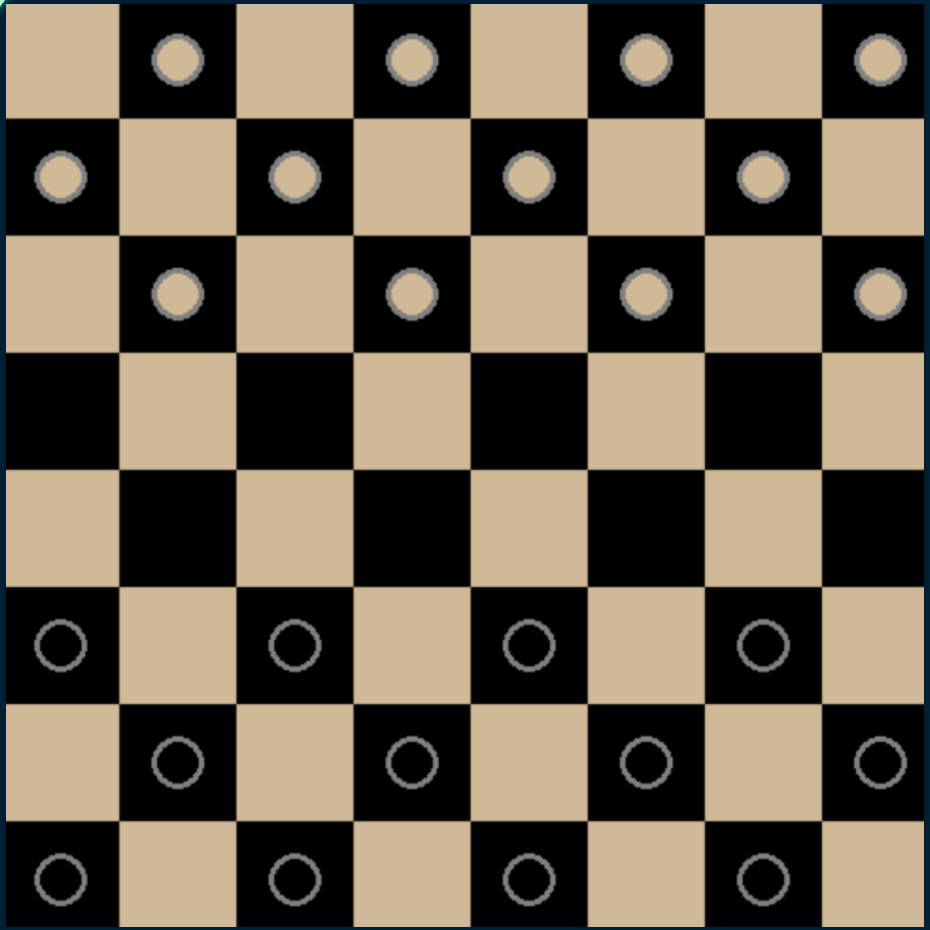
MINIMAX WITH ALPHA BETA PRUNING

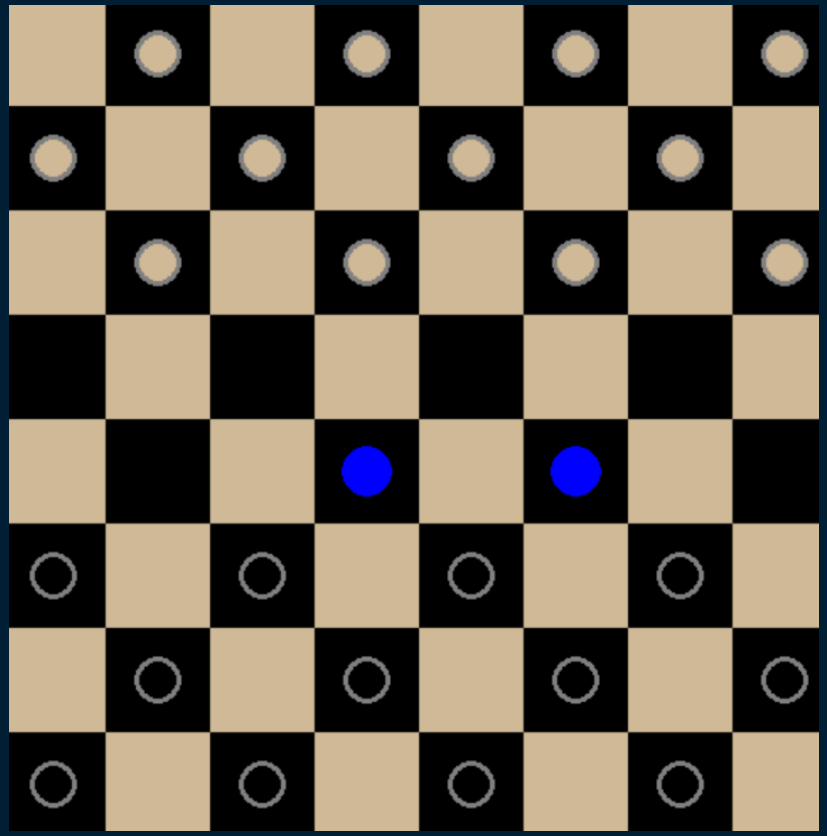
Alpha-Beta is a pruning method used in conjunction with a minimax search, and it is best suited for two-player, zero-sum games.

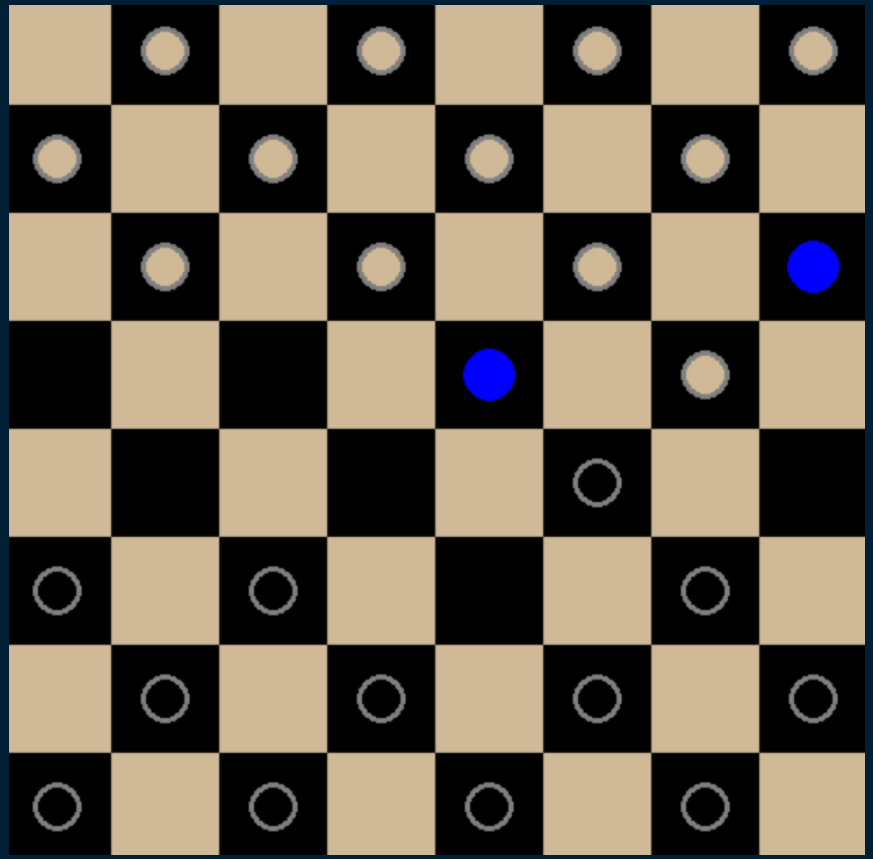
MINIMAX WITH ALPHA BETA PRUNING

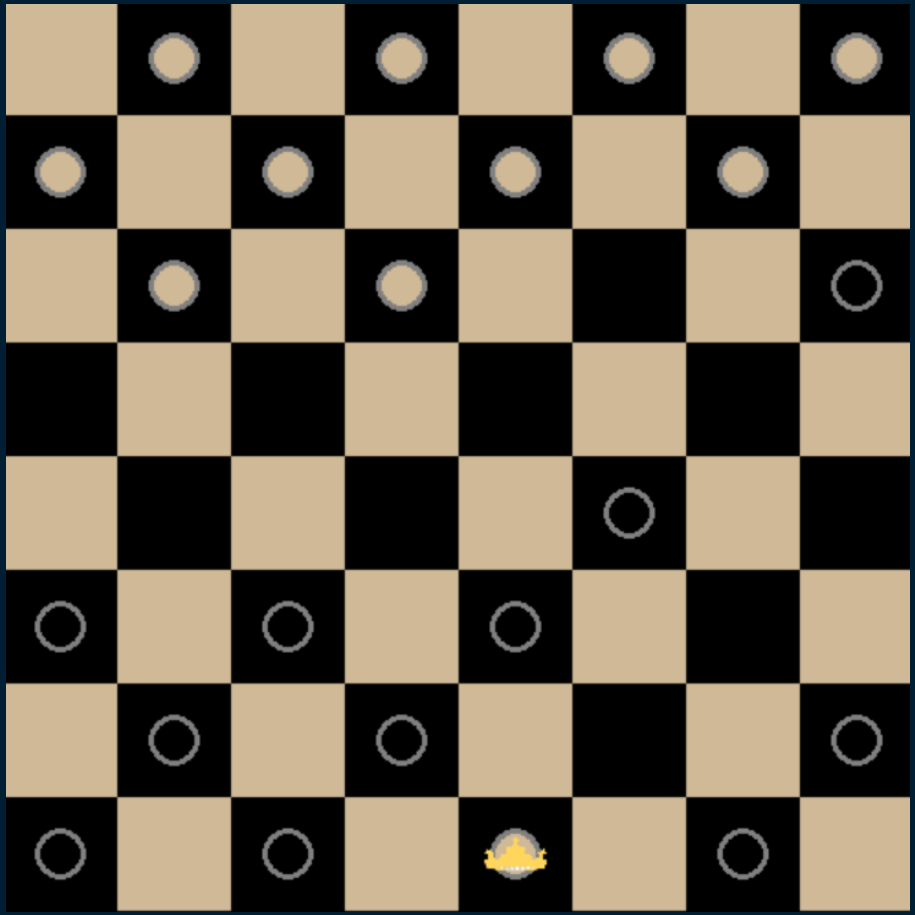
1. Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
2. We have applied Alpha-beta pruning to a standard minimax algorithm, it returns the same move as the standard one, but it removes (prunes) all the nodes that are possibly not affecting the final decision.
3. As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called as Alpha-Beta Algorithm

DEMO









Minimax and alpha-beta pruning Code

gamesurface.py | gamelogic.py | minmax.py | main.py | settings.py

Checkers_AI_Proj > minmax.py > simulate_move

```python
# alpha = max_player
# beta = game

#position = the board object. based on the current position of the board, give the best possible board after this
#depth of minimax algo: gets decremented by 1 everytime we call this (its a recursive call)

#Takes in a board/pos, depth of how far to evaluate, max_player to see if maximize or minimize, game
def minimax(position, depth, max_player, game, alpha = float('-inf'), beta = float('inf')):
    #max_player is boolean if maxing or min-ing
    #game = game object if game
    if depth == 0 or position.winner() != None: #If we're furthest down in the branch or if someone has won
        return position.evaluate(), position ##gives best (max/min) number and the position it corresponds to


    if max_player: #Maximizing beige
        best = float('-inf') #Initial min value
        best_move = None #No best move decided
        for move in get_all_moves(position, beige, game): #Moves related to beige
            evaluation = minimax(move, depth-1, False, game,alpha,beta)[0] #explore next depth
            best = max(best, evaluation) #Pick the max between best and eval(last depth)

            if best >= beta:
                return best, best_move
            if best > alpha:
                alpha = best
            if best == evaluation:
                best_move = move


        return best, best_move
```

0 0 | Ln 75, Col 27 | Spaces: 4 | UTF-8 | CRLF | Python | 3.10.5 64-bit | Go Live

ENG 10:06 06-11-2022

gamesurface.py | gamelogic.py | minmax.py | main.py | settings.py

Checkers_AI_Proj > minmax.py > simulate_move

```python
        return best, best_move

    else: # Minimizing player
        best = float('inf')
        best_move = None
        for move in get_all_moves(position, black, game):
            evaluation = minimax(move, depth-1, True, game,alpha,beta)[0]
            best = min(best, evaluation)
            if best == evaluation:
                best_move = move
            if best <= alpha:
                return best, best_move
            if best < beta:
                beta = best


        return best, best_move


# ALL the moves of all the pieces
def get_all_moves(board, color, game):
    moves = []

    for piece in board.get_all_pieces(color):
        valid_moves = board.get_valid_moves(piece)
        # (row, col) : [pieces]
        for move, skip in valid_moves.items():
            temp_board = deepcopy(board)
            temp_piece = temp_board.get_piece(piece.row, piece.col)
            new_board = simulate_move(temp_piece, move, temp_board, game, skip)
```

0 ⚠ 0 | Ln 75, Col 27 | Spaces: 4 | UTF-8 | CRLF | Python | 3.10.5 64-bit | Go Live

ENG 10:07 06-11-2022

gamesurface.py | gamelogic.py | minmax.py | main.py | settings.py

Checkers_AI_Proj > minmax.py > simulate_move

```python

# ALL the moves of all the pieces
def get_all_moves(board, color, game):
    moves = []

    for piece in board.get_all_pieces(color):
        valid_moves = board.get_valid_moves(piece)
        # (row, col) : [pieces]
        for move, skip in valid_moves.items():
            temp_board = deepcopy(board)
            temp_piece = temp_board.get_piece(piece.row,
            new_board = simulate_move(temp_piece, move, temp_board, game, skip)
            moves.append(new_board)

    return moves

def simulate_move(piece, move, board, game, skip):
    board.move(piece, move[0], move[1])
    if skip:
        board.remove(skip)

    return board
```

(variable) temp_board: Any

Ln 75, Col 27 | Spaces: 4 | UTF-8 | CRLF | Python | 3.10.5 64-bit | Go Live

ENG 10:07 06-11-2022