AI for games: min-max search

COMP 112 2018

School of Engineering and Computer Science

Victoria University of Wellington

Menu

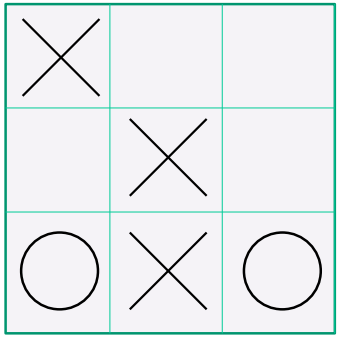
- AI and Board Games:
  min-max search for 2-person turn-taking games
- Admin

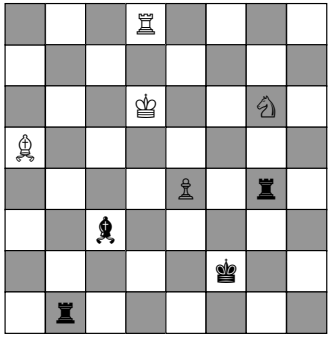
AI: Games

- Where is AI used in Games?

Min-Max search

- 2-person, turn-taking games (eg, board games)



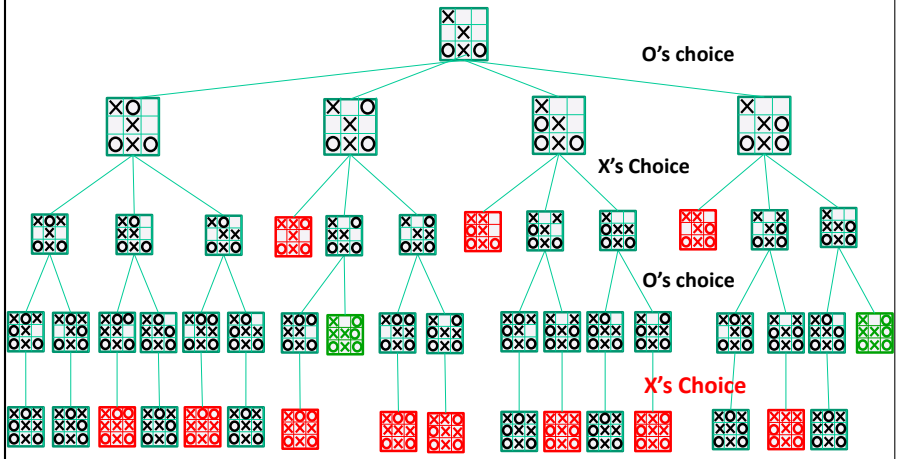


It's O's turn / white's turn
What should they do?

Approaches:

- Look-up table of good moves for each position.
- Heuristics ("rules-of-thumb") to evaluate each move.
- Look-ahead search:
  - for each possible move
    - for each possible response
      - for each possible move
        - for each possible …..
          -
          - work out if win or loss propagate it back….

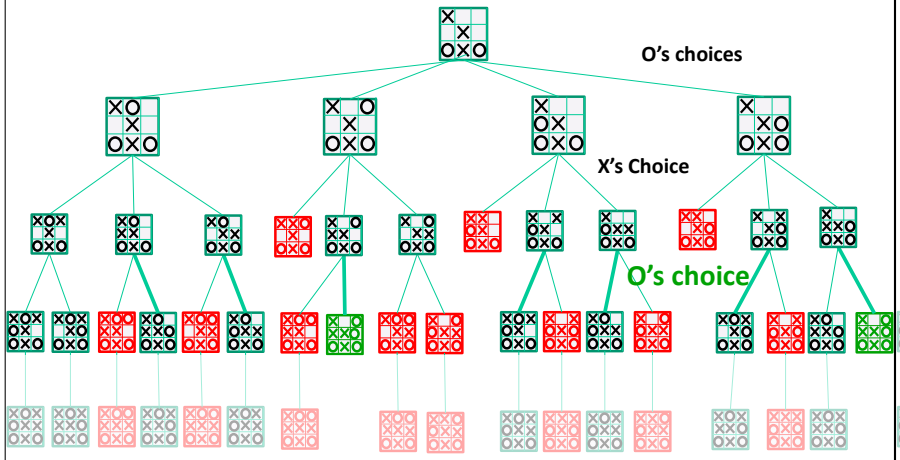
It's O's turn: what should O do?



Work backwards to determine status of earlier positions

X will try to make O lose

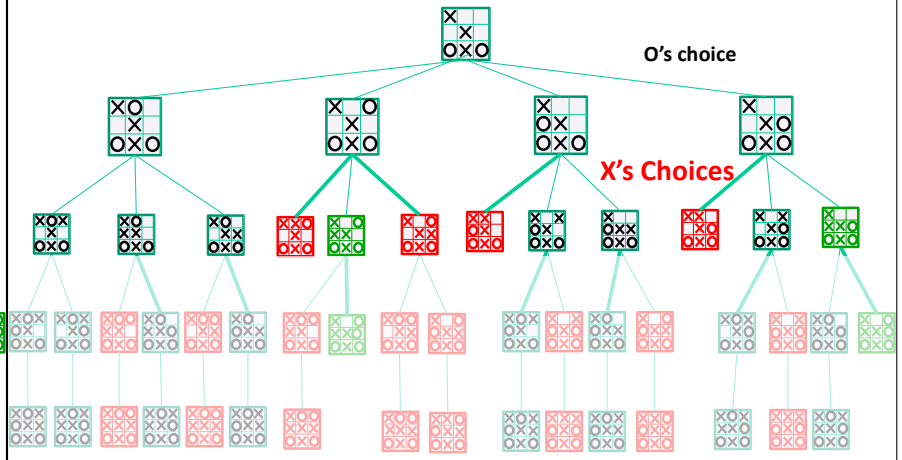
It's O's turn: what should O do?



Work backwards to determine status of earlier positions

O will try to win

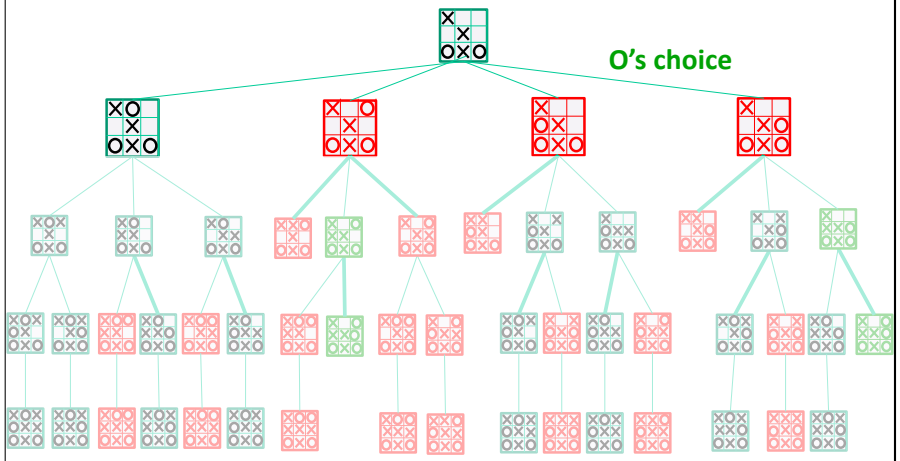
It's O's turn: what should O do?



Work backwards to determine status of earlier positions

X will try to make O lose

It's O's turn: what should O do?



Work backwards to determine status of earlier positions

O will try to win

Min-max search

- To choose move:
  - for each possible move,
    - evaluate new board state from other player's perspective
    - perform move with lowest value board (*worst for them, best for me*)
- To evaluate a board from players' perspective
  - If the game is over, return 1 for win, -1 for lose, 0 for draw
  - for each possible move of the player,
    - construct board state after move
    - value = -ve of (evaluate new state from other player's perspective)
    - keep track of maximum value (*ie, best for this player)*
  - return value

AI for O and X.

```
public void doAITurn(){
    String bestOutcome = "X";  //start pessimistic - the real player will win
    int bestRow = -1;
    for (int row=0; row<3; row++){for (int col=0; col<3; col++){
        if (board[row][col] == null ){ // it's an empty cell
            if (bestRow == -1){  bestRow = row;  bestCol = col; } //we can at least play here
            String[ ][ ] hypotheticalBoard = this.newBoard(this.board, row, col, "O");
            String value = evaluate(hypotheticalBoard, "X");
            if (value.equals("O")){        // Winning move: play here
                this.board[row][col] = "O";
                return;
            }
            else if (value.equals("draw")){     // might be best option
                bestOutcome = "draw";
                bestRow = row;   bestCol = col;
            }
        }}
    }
     if (bestRow != -1)  {  this.board[bestRow][bestCol] = "O";  } // play at best option.
}
```


AI for O and X

```
public String evaluate(String[][] brd, String player){   // player plays next
    String outcome = this.boardOutcome(brd);
    if ( outcome != null ){ return outcome; }   //the game is over
    String other = this.other(player); // the other player from this perspective
    outcome = other;                    // minimum outcome if player can't do better
    for (int r=0; r<3; r++){for (int c=0; c<3; c++){
        if (!brd[r][c].equals("")){   continue;   }
        String[][] hypotheticalBoard = this.newBoard(brd, r, c, player);
        // see what the outcome of that play would be
        String v = evaluate(hypotheticalBoard, other);
        if (v.equals(player)){ // the player could win
            return player;     // then they definitely would choose this
        }
        if (v.equals("draw")){  outcome = "draw"; } // choose this if there is no win
    }}
    return outcome;  //there was no win for the nextPlayer,
  }
```

What about 3D OandX?

- Is this approach feasible?
- What about chess or draughts (checkers)?
- How big is the search?
  - OandX, 3D 0andX, Draughts, Chess, Go?

Min-Max search for big games

- Need to search as deep as possible,
- Then use heuristics to evaluate board states:
  - eg: piece count, dominance
    $\Rightarrow$ values between -1 and 1
- Propagate the values back
  - minimising the values on the opponent's turn
  - maximising the values on the player's turn.

Min-max search.