Game AI: Project 1

Simple-strategies for turn-based games

Mariia Rybalka Elchin Valiyev Abbas Khan Maxim Radomskyi Maxim Maltsev

Colloqium, 2016

Outline

Outline:

Probabilistic strategy slide 1

Optional Subtitle

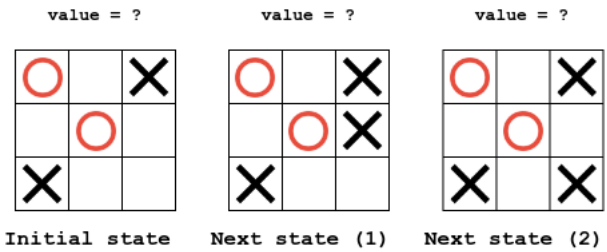
content ...

Performance

Outline:

Evaluating the quality of a potential move

How to pick next move from possible ones?
We need to see difference!

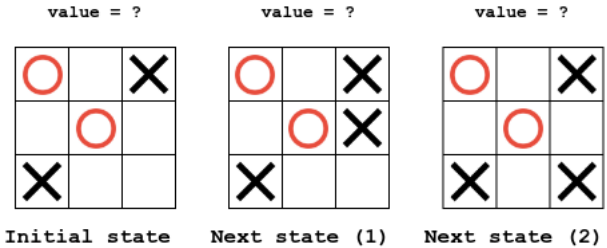


Heuristic / Evaluation function

Provides an estimate of the utility of a game state that is not a terminal state

Simple evaluation function for Tic-Tac-Toe (from slides)

Eval(n, p) = (number of lines where **p** can win) − (number of lines where **-p** can win)

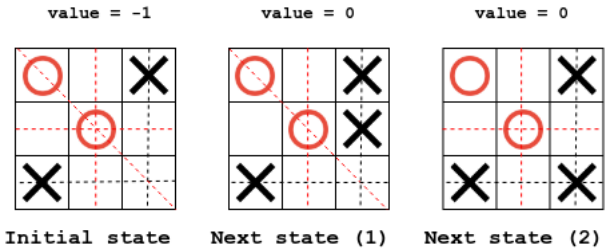


Heuristic / Evaluation function

Provides an estimate of the utility of a game state that is not a terminal state

Simple evaluation function for Tic-Tac-Toe (from slides)

Eval(n, p) = (number of lines where **p** can win) − (number of lines where **-p** can win)



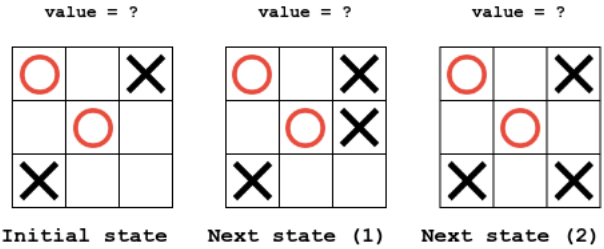
Obviously, current function is not a good one! We can do better!

Heuristic / Evaluation function (cont.)

A Better Evaluation Function (Russell & Norvig, Artificial Intelligence)

Eval(n) = 3 * X2 + X1 (3 * O2 + O1)

- X2 is the number of lines with 2 Xs and a blank
- X1 is the number of lines with 1 X and 2 blanks
- O2 is the number of lines with 2 Os and a blank
- O1 is the number of lines with 1 O and 2 blanks

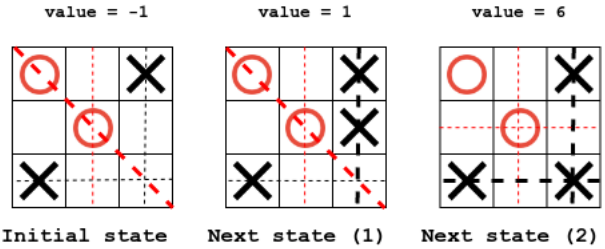


Heuristic / Evaluation function (cont.)

A Better Evaluation Function (Russell & Norvig, Artificial Intelligence)

Eval(n) = 3 * X2 + X1 (3 * O2 + O1)

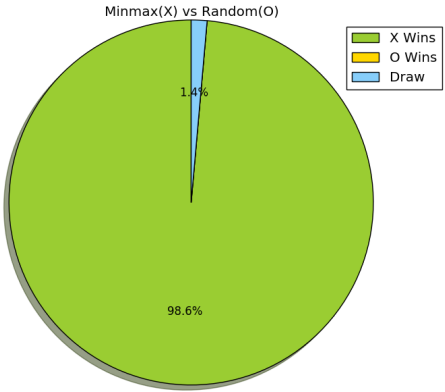
- X2 is the number of lines with 2 Xs and a blank
- X1 is the number of lines with 1 X and 2 blanks
- O2 is the number of lines with 2 Os and a blank
- O1 is the number of lines with 1 O and 2 blanks

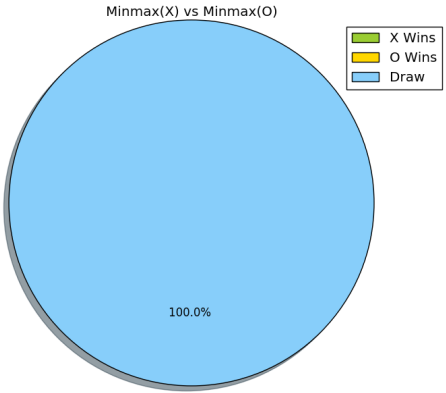


Minmax algorithm

```
def minmax(board, player, max_depth, current_depth):
        # Check if we're done recursing
        if board.game_is_over() or current_depth == max_depth:
                return board.evaluate(player), None

        best_move = None
        if board.current_player() == player:
                best_score = -INFINITY
        else:
                best_score = INFINITY

        # Go through each move
        for move in board.get_moves():
                new_board = board.makeove(move)

                # Recurse
                current_score, current_move = minmax(new_board, player, max_depth, current_depth + 1)

                # Update the best score
                if board.current_player() == player:
                        if current_score > best_score:
                                best_score = current_score
                                best_move = move
                        else:
                                if current_score < best_score:
                                        best_score = current_score
                                        best_move = move

        # Return the score and the best move
        return best_score, best_move
```

Performance





Outline:

Connect 4 Slide 1

content ...

Connect 4 Slide 2

content...

Do we need Summary ?

That's All Folks!

Thank you for attention!