

Game AI: Project 1

Simple-strategies for turn-based games

Mariia Rybalka Elchin Valiyev Abbas Khan Maxim Radomskyi Maxim Maltsev

Colloquium, 2016

1 Simple strategies for tic tac toe

- Probabilistic strategy
- Heuristic strategy

2 Connect 4

- Random Play
- Statistical Approach

Outline:

1 Simple strategies for tic tac toe

- Probabilistic strategy
- Heuristic strategy

2 Connect 4

- Random Play
- Statistical Approach

Probabilistic strategy

Main idea

Approach: Find out, which positions *usually* (over the huge number of games) contribute to the victory the most and choose one of them as the next move in the game.

Victory is Mine!



Probabilistic strategy

Pseudo code

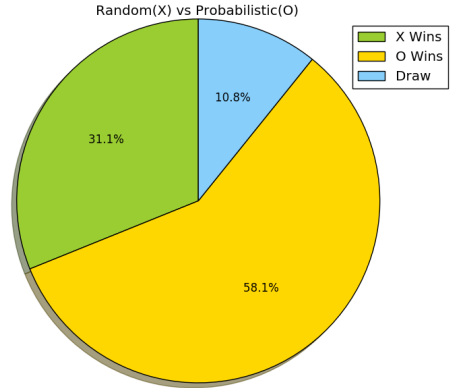
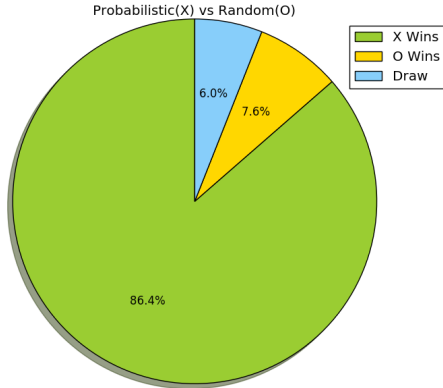
Learn probabilities

1. keep data structure with counters for each possible move;
2. for 10000 (or any large number) times:
 3. play_game;
 4. update counters of winner's moves;
5. normalize all counters in data structure → obtain moves' probabilities;
6. write probabilities to file.

Play using probabilistic approach

1. read probabilities from file;
2. while move is still possible:
 - ...
3. next move = possible move which has maximal probability;
 - ...

Performance



Outline:

1 Simple strategies for tic tac toe

- Probabilistic strategy
- Heuristic strategy

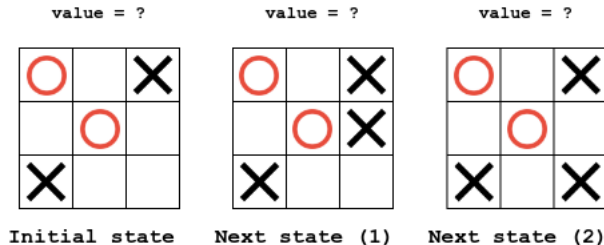
2 Connect 4

- Random Play
- Statistical Approach

Evaluating the quality of a potential move

How to pick next move from possible ones?

We need to see difference!

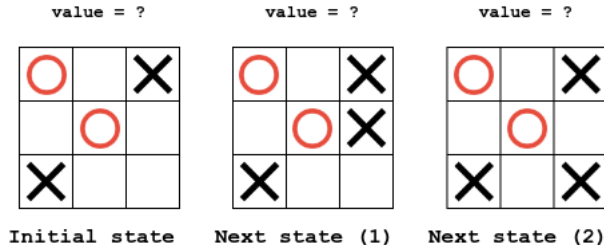


Heuristic / Evaluation function

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)

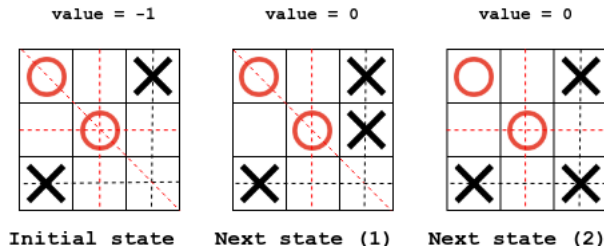


Heuristic / Evaluation function

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)

$\text{Eval}(\mathbf{n}, \mathbf{p}) = (\text{number of lines where } \mathbf{p} \text{ can win}) - (\text{number of lines where } -\mathbf{p} \text{ can win})$



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

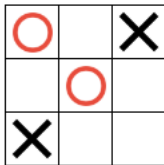
Heuristic / Evaluation function (cont.)

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

$$\text{Eval}(n) = 3 * X2 + X1 \quad (3 * O2 + O1)$$

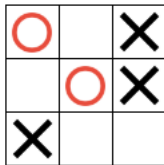
- X2 is the number of lines with 2 Xs and a blank
- X1 is the number of lines with 1 X and 2 blanks
- O2 is the number of lines with 2 Os and a blank
- O1 is the number of lines with 1 O and 2 blanks

value = ?



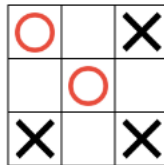
Initial state

value = ?



Next state (1)

value = ?



Next state (2)

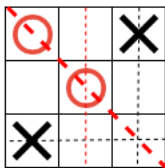
Heuristic / Evaluation function (cont.)

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

$$\text{Eval}(n) = 3 * X2 + X1 \quad (3 * O2 + O1)$$

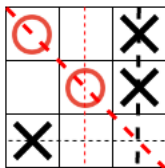
- X2 is the number of lines with 2 Xs and a blank
- X1 is the number of lines with 1 X and 2 blanks
- O2 is the number of lines with 2 Os and a blank
- O1 is the number of lines with 1 O and 2 blanks

value = -1



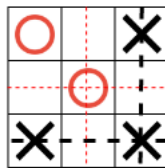
Initial state

value = 1



Next state (1)

value = 6

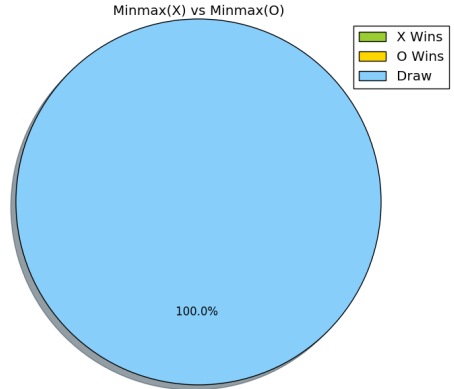
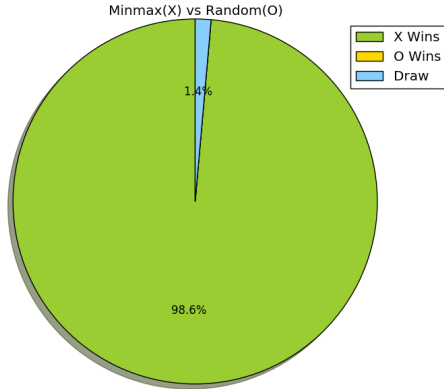


Next state (2)

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:

1 Simple strategies for tic tac toe

- Probabilistic strategy
- Heuristic strategy

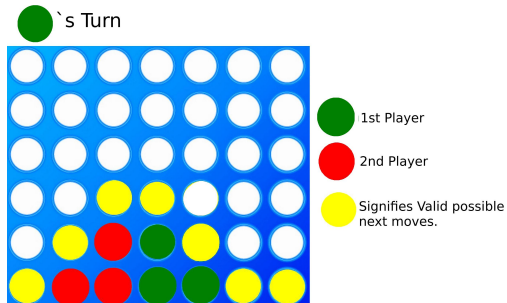
2 Connect 4

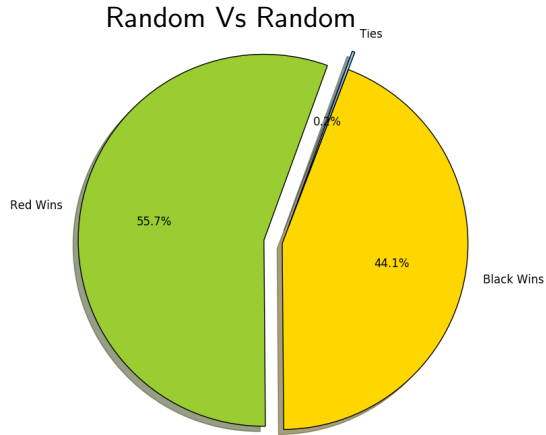
- Random Play
- Statistical Approach

Evaluating the results

How to pick next move from possible/valid ones?
We choose randomly!

```
while NotFound:
    move=Generate a random move
    if move isValid
        return move
```





- No strategy is followed , moves are picked completely radomly

Outline:

- 1 Simple strategies for tic tac toe
 - Probabilistic strategy
 - Heuristic strategy
- 2 Connect 4
 - Random Play
 - Statistical Approach

Learning from random play

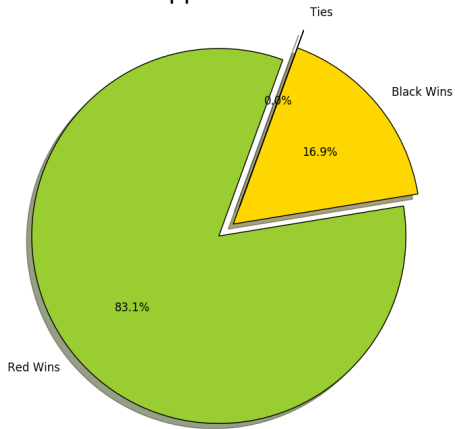
Using a lookup table

- We store all the bins used by a winner over a million games in a lookup matrix
- From all the valid moves in a given state we pick the one which was used the most in lookup matrix

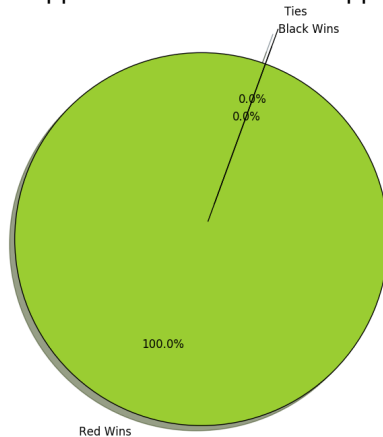


601	577	570	533	523	558	579
1155	1076	1081	1066	1054	1075	1110
2138	2118	2225	2372	2148	2201	2250
2753	2987	3175	3235	3209	3044	2758
3500	3784	4167	4417	4144	3880	3582
4303	4493	4971	5789	5075	4582	4320

Statistics Approach VS Random



Statistics Approach VS Statistical Approach



Do we need Summary ?

That's All Folks!

Thank you for attention!