Term Project
Report

Tic-Tac-Toe Game

Submitted in fulfillment of
the requirements of the term project of

Artificial Intelligence Course

Submitted by
Anwar Mohamed - 2491

Department of Computer and Communications Engineering
Faculty of Engineering - Alexandria University
Alexandria, Egypt

Winter Semester 2016

Contents

Chapter 1

Problem Definition

You are required to implement a tic-tac-toe game using alpha-beta pruning where a user plays against the computer.

You are not expected to create a graphical user interface; The game could have a command line interface, but the user must have a way to place an X (or an O) where ever it is possible to do so. But if you think GUI will make it easier for you to test the game, then feel free to use it.

Tic-tac-toe is a game with a score of zero, which means that if both players play optimally, no one should win.

If you were able to beat the computer (even if it was rare), then this means that your implementation of minimax alpha-beta is incorrect. Every game you play against the computer should end either with a tie or with the computer winning.

Chapter 2

Introduction

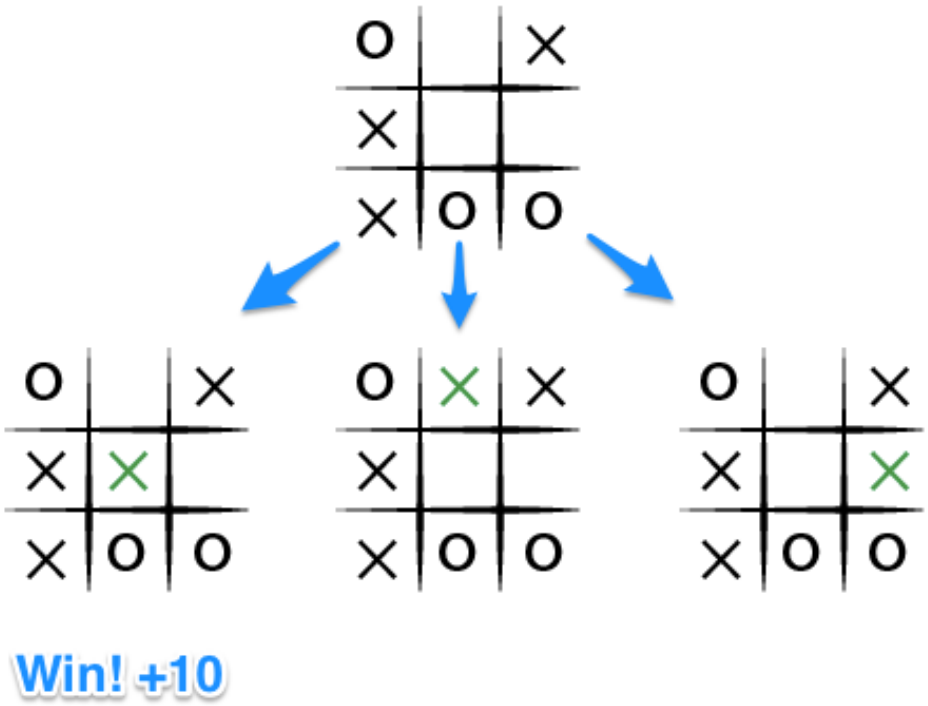
2.1 Perfect Game of Tic Tac Toe

To begin, let's start by defining what it means to *play a perfect game of tic tac toe*:

If I play perfectly, every time I play I will either win the game, or I will draw the game. Furthermore if I play against another perfect player, I will always draw the game. How might we describe these situations quantitatively? Let's assign a score to the "*end game conditions*:"

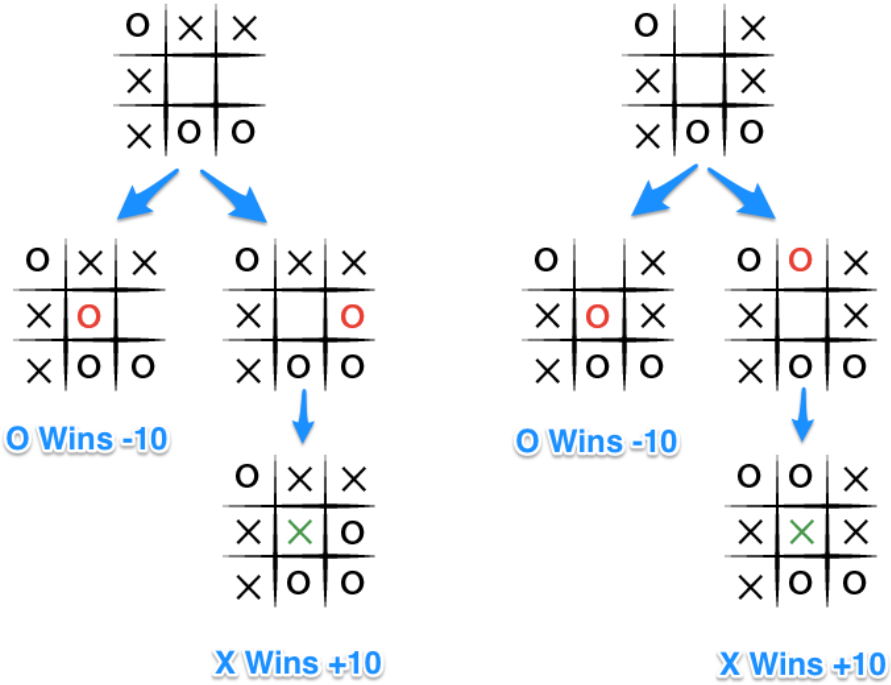
- I win, hurray! I get 10 points!
- I lose, shit. I lose 10 points (because the other player gets 10 points).
- I draw, whatever. I get zero points, nobody gets any points.

To apply this, let's take an example from near the end of a game, where it is my turn. I am X. My goal here, obviously, is to maximize my end game score.



If the top of this image represents the state of the game I see when it is my turn, then I have some choices to make, there are three places I can play, one of which clearly results in me wining and earning the 10 points. If I don't make that move, O could very easily win. And I don't want O to win, so my goal here, as the first player, should be to pick the maximum scoring move.

What do we know about O? Well we should assume that O is also playing to win this game, but relative to us, the first player, O wants obviously wants to chose the move that results in the worst score for us, it wants to pick a move that would minimize our ultimate score. Let's look at things from O's perspective, starting with the two other game states from above in which we don't immediately win:



The choice is clear, O would pick any of the moves that result in a score of -10.

2.1.1 Minimax Algorithm

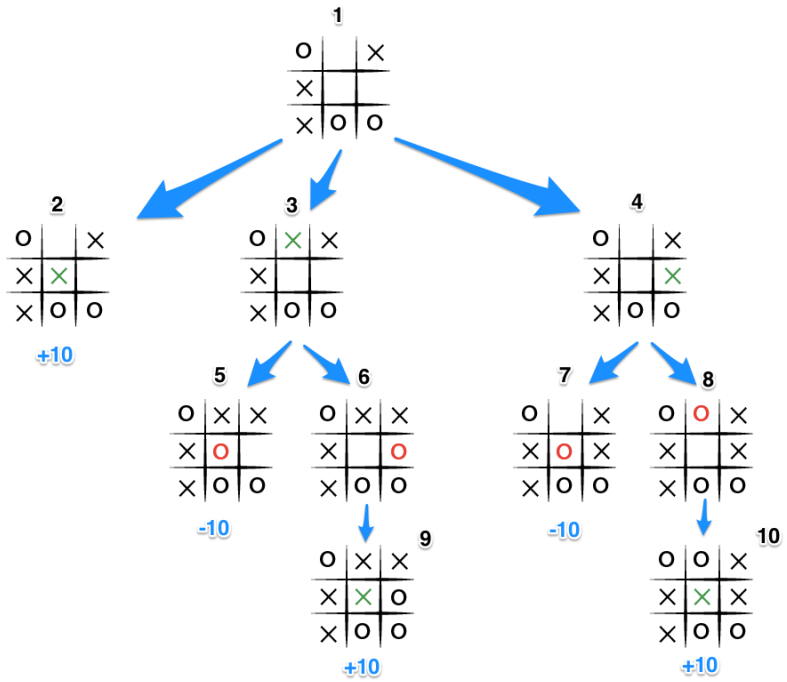
The key to the Minimax algorithm is a back and forth between the two players, where the player whose "turn it is" desires to pick the move with the maximum score. In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score. And the scores for the opposing players moves are again determined by the turn-taking player trying to maximize its score and so on all the way down the move tree to an end state.

A description for the algorithm, assuming X is the "turn taking player," would look something like:

- If the game is over, return the score from X's perspective.
- Otherwise get a list of new game states for every possible move.
- Create a scores list.
- For each of these states add the minimax result of that state to the scores list.
- If it's X's turn, return the maximum score from the scores list.
- If it's O's turn, return the minimum score from the scores list.

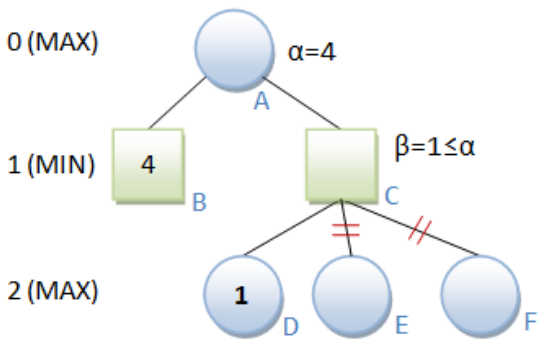
```
def minimax(depth, player)
  if gameover || depth == 0
    return calculated_score
  end
  children = all legal moves for player
  if player is AI (maximizing player)
    best_score = -infinity
    for each child
      score = minimax(depth - 1, opponent)
      if (score > best_score)
        best_score = score
      end
      return best_score
    end
  else #player is minimizing player
    best_score = +infinity
    for each child
      score = minimax(depth - 1, player)
      if (score < best_score)
        best_score = score
      end
      return best_score
    end
  end
end

#then you would call it like
minimax(2, computer)
```

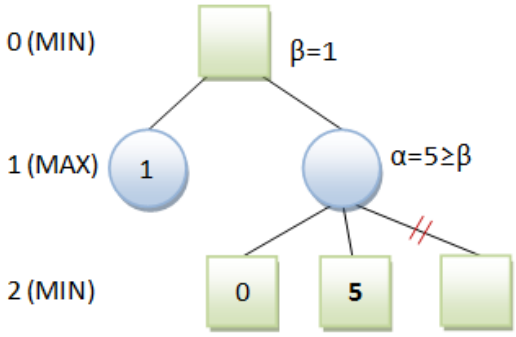


2.1.2 Alpha-Beta Pruning

Theres a different algorithm that makes Minimax better, called Alpha-beta pruning. This eliminates paths that are worse than paths that have already been evaluated. Therefore, you have to store a few more values: alpha, which holds the maxmimum score for the maximum path, and beta, which holds the minimum score for the minimum path. You throw away a path when its alpha value is >= the beta value.



Alpha Cut-off



Beta Cut-off

```
def minimax(depth, player, alpha, beta)
  if gameover || depth == 0
    return calculated_score
  end
  children = all legal moves for player
  if player is AI (maximizing player)
    for each child
      score = minimax(depth - 1, opponent, alpha, beta)
      if (score > alpha)      alpha = score  end
      if alpha >= beta    break          end
      return alpha
    end
  else #player is minimizing player
    best_score = +infinity
    for each child
      score = minimax(depth - 1, player, alpha, beta)
      if (score < beta)   beta = score       end
      if alpha >= beta    break          end
      return beta
    end
  end
end

#then you would call it like
minimax(2, computer, -inf, +inf)
```

Chapter 3

Work Done

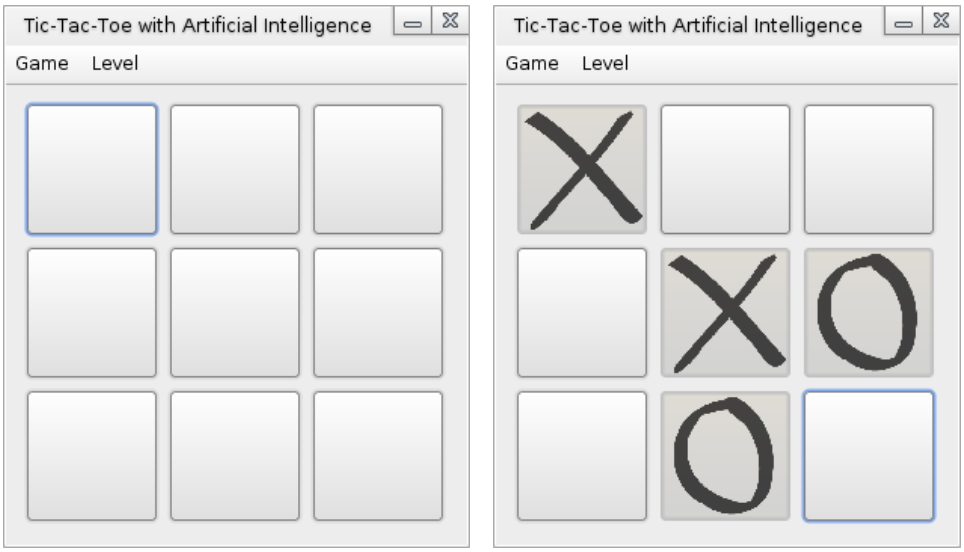
3.1 Java Implementaion

I have implemented a simple tic-tac-toe game in Java that utilizes alpha-beta pruning so that the computer can play against computer users. The structure of the game is as following:

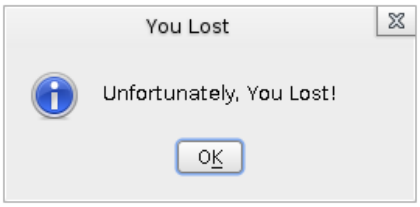
- TicTacToe, the driver class for the game.
- GameFrame, the graphical user interface class.
- GameController, the logic implementaion class.

I have added the ability to change the level of difficulty of the game, so there are **Easy**, **Medium** and **Hard** levels.

3.2 Game Testing




Tic-Tac-Toe with Artificial Intelligence
Game
Level


You Lost
Unfortunately, You Lost!
OK


Tic-Tac-Toe with Artificial Intelligence
Game
Level
Easy
Medium
Hard


Tic-Tac-Toe with Artificial Intelligence
Game
Level
It's a Draw
Lol, It's a Draw!
OK

References

[1] Tic Tac Toe: Understanding The Minimax Algorithm, http://neverstopbuilding.com/minimax

[2] Solving Tic-Tac-Toe, Part II: A Better Way, http://catarak.github.io/blog/2015/01/07/solving-tic-tac-toe/