Engineering and Applied Science Programs for Professionals
Whiting School of Engineering
Johns Hopkins University
685.621 Algorithms for Data Science
Game Theory

This document provides a rollup of game theory methods covered in the game theory module.

Contents

1 Introduction to Two-Player Games

There are many theoretical concepts in game theory. The history of game theory has roots dating back to the 1700s, however one of the first books was publishing in 1944 by John von Neumann and Oskar Morgenstern titled Theory of Games and Economic Behavior. In 1934 R.A. Fisher published Randomisation and an Old Enigma of Card Play. Since these publishings, several articles and books have been written with a variety of mathematical, engineering and scientific concepts to solving a variety of games. In this document theory and analytical solutions for two player games is discussed in support of the course lecture slides.

2 Basic Concepts

The basic concepts for the purpose of this document are to provide an initial insight into the definitions of agents, environments, game search strategies and search methods.

When defining agents it is important to have a clear understanding of environments. In the context of tic-tac-toe the environment is the board with the pieces placed on it. In this context the agent is the either the physical view of the board on the monitor or the representation of the board in code. There are various forms of agent having access to the board, for the case of writing code for a two play game of tic-tac-toe it is easiest to use an array to represent the board. This allows an easy view of the board using a set of assigned values in the array for board representation. When discussing the board state of tic-tac-toe it is always necessary to determine the type of search strategy of how to determine the board state. In the case of tic-tac-toe the board environment is fully observable giving the agent (the computer program) access to the complete state of the board at any point in time. The nice thing about this type of development is that the environment is static allowing the board to remain unchanged while the AI (developed code) deliberates on the next best move to take.

3 The Game of Tic-Tac-Toe

In the game of Tic-Tac-Toe there are set know rules to help play the game when two-players sit at a table to play on a piece of paper, https://en.wikipedia.org/wiki/Tic-tac-toe. After examining these rules how can a two-player game be implemented in code where an individual plays against an AI. When developing the AI the developer must have a clear understanding of the board and its states are represented. In the following board the locations are represented from 0 to 8 in order to determine where specific pieces lie on the tic-tac-toe board.

0	1	2
3	4	5
6	7	8

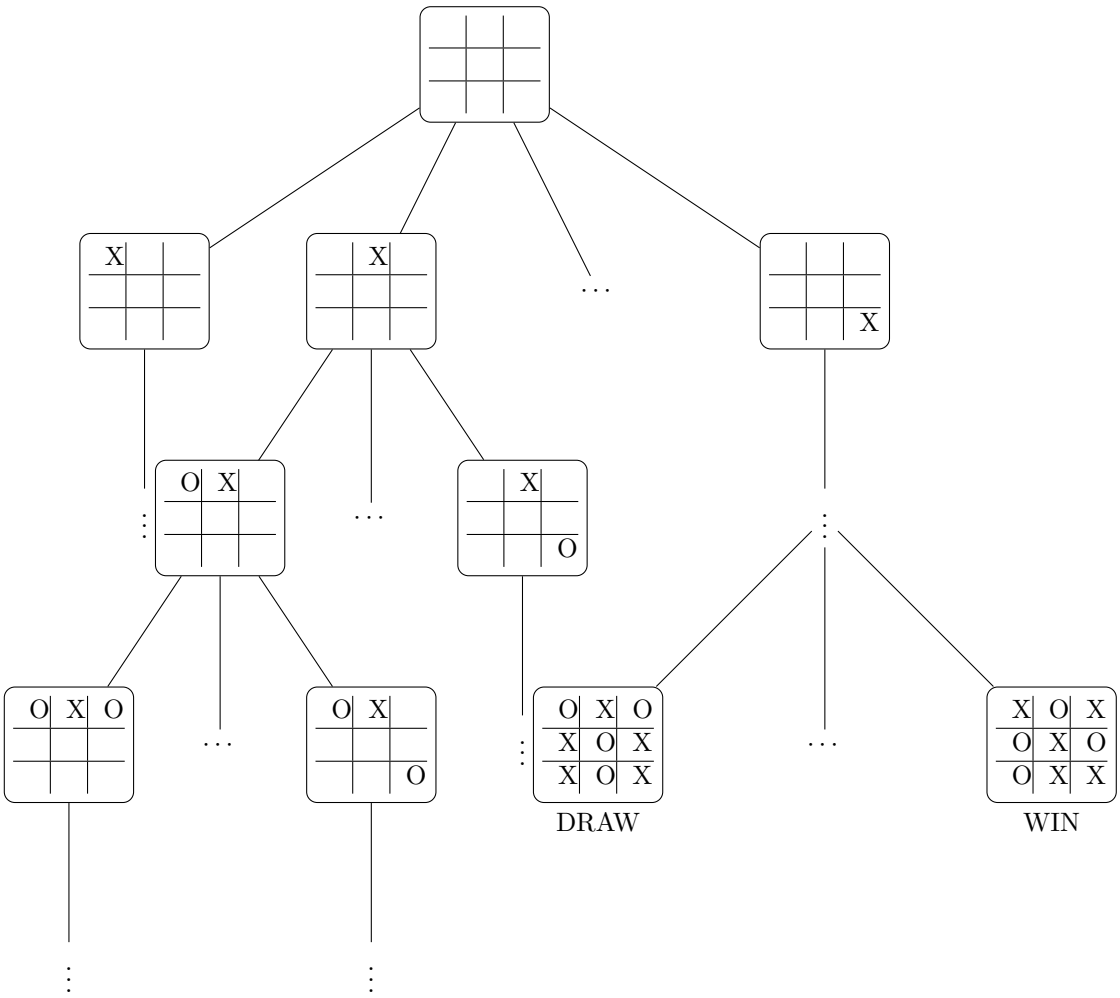
Let's examine an initial state where where there are two X's along the diagonal represented as locations board-location(2) and board-location(4) with an O located at position board-location(0) as follows:

O		X
	X	

In this case the actions to take are to determine a set of actions based on the current state of the board. A check is to be implemented to determine if the next move for the AI pieces produces a WIN move, if so take it. In the event a WIN is not possible for the AI the state of the board must be examined to determine if the opponent will produce a WIN move on the opponents next move, If this is the case the AI must be developed with sufficient knowledge of the board to block the opponents move. In this example the state of the board represented as S(board-location()) = block. This will lead to a board-location(6) for the AI to block the opponent from winning as represented below:

O		X
	X	
O		

The decision to make specific moves is determined by a cost function when examining the state of the board. This is done in various ways. In the case of a search algorithms it is best to have the entire space explored and a large cost returned when a path to a WIN state is identified. This can be seen in the following diagram where a WIN sate is achieved at the right more leaf node of the search tree.



In this example the state of the board is represented using a set of nodes with connecting edges from top to bottom as the game board is populated. This is best achieved when the each level of the tree is explored using a recursive call where the next state of the board is passed as the opponents pieces are added to the board creating new nodes in the search tree. The search strategy must be expanded using a specific order, e.g., from top to bottom and left to right. The strategies must consider completeness of finding a solution, the time complexity allowed by a pre-defined set of rule, the space complexity in the number of nodes explored with allowable memory, and the optimality of determining a low cost solution.

4 Tree Search Algorithms

The two methods described in this section are optimal in searching the entire search space. These methods only differ in that the alpha beta pruning method reduces the need to search additional nodes assuming the leaf nodes and the board states return adequate values. The minimax decision algorithm searches each state of the game board and every possible move combination. The alpha-beta decision is an improvement of the minimax decision algorithm that reduces the search space by terminating the search at a specific node when at least one board state proves to be worse that a previously examined board state.

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game

    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
     return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

```
function ALPHA-BETA-DECISION(state) returns an action
   return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
function MAX-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for MAX along the path to state
           β, the value of the best alternative for MIN along the path to state

   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do
      v ← MAX(v, MIN-VALUE(s, α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
   return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
   same as MAX-VALUE but with roles of α, β reversed
```

5 Making a Smart AI

The two methods described in the search algorithm strategy provide a unique perspective of developing an intelligent code base that will never loose in the game of tic-tac-toe. This of course requires that the board state be represented correctly, the change in the boar state be developed in a manner that allows the entire search space to be explored and a set of metrics to determine what is the best move to avoid loosing.

References

[1] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronal L., and Stein, Clifford, *Introduction to Algorithms*, 3rd Edition, MIT Press, 2009

[2] Russell, S. J. and Norvig, P., *Artificial Intelligence: A Modern Approach*, 3rd Edition, Prentice Hall, 2009

[3] Winston, P., *Artificial Intelligence*, 3rd Edition, Pearson, 1992