Alpha-Beta Pruning

Background Concepts

**Minimax** algorithm performs depth first exploration of the game tree. It computes minimax decision from the current state. It uses a simple recursive computation of the minimax values of each successor state. The recursion proceeds all the way down to the leaves and then the minimax values are backed up through the tree. The time complexity of minimax algorithm is b^m where b represents the number of actions for each state and m denotes the depth of the tree.

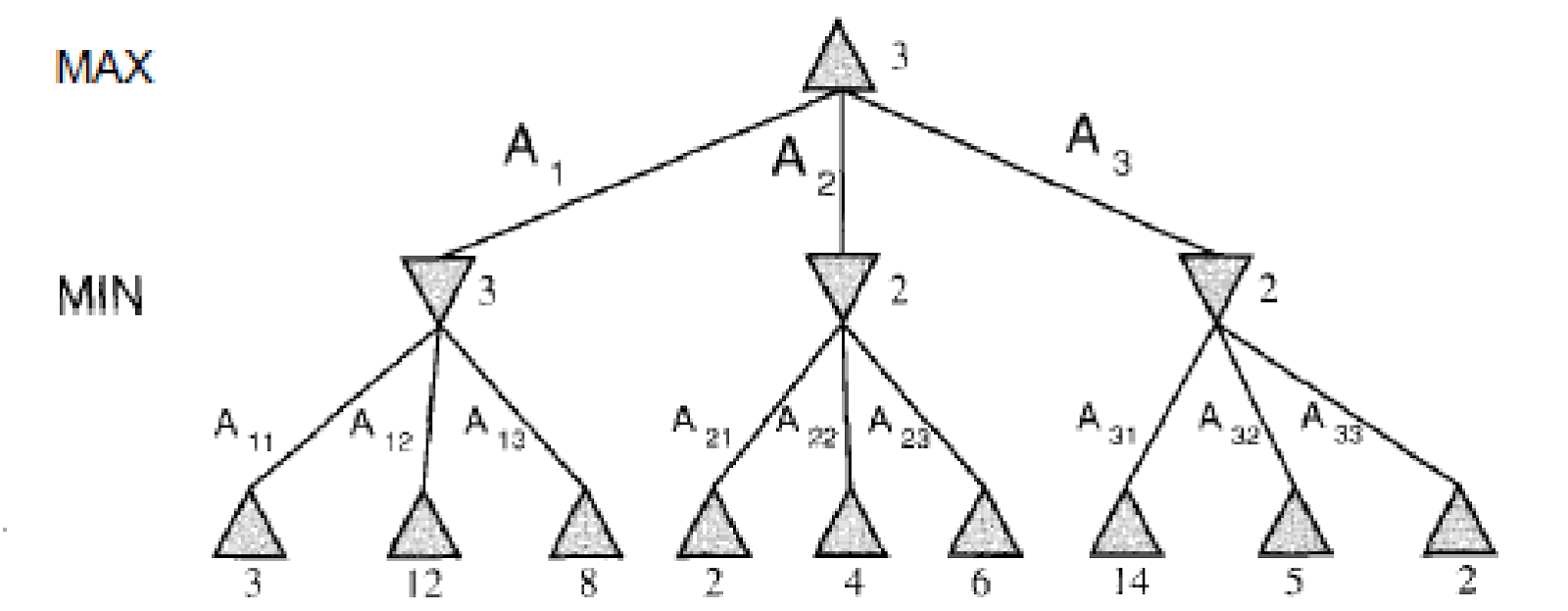


Figure 1.1

Figure 1.1 traces the tree for two players. It recurses down to the left bottom three nodes and MIN chooses minimum value 3 from the leaves 3,12,8 and backwards up this value to left most node. Similar process will be applied for middle node (2) and right most node (2). Finally, the maximum value of 3 is taken from 3, 2, 2 for the root node. Time-complexity for this scenario is 3^2 = 9.

***Alpha-Beta*** pruning technique reduces the number of comparisons of minimax algorithm. It prunes away the branches which cannot influence the final result. Following are the 2 parameters which describe the bounds for backed-up values of the tree.

**α = the highest-value discovered so far at any choice point along the path for MAX.**

**β= the lowest-value discovered so far at any choice point along the path for MIN.**

Note that ***Alpha-beta*** search updates **α** and **β** as it goes along and prunes the remaining branches at a node (i.e., terminates depth first call) as soon as the value of current node is known to be worse than the current alpha or beta for **MAX** or **MIN**, respectively. Figure 1.2 depicts the scenario after applying *alpha-beta* pruning strategy in the problem simulated in Figure 1.1. Time-complexity for this scenario is **7**.

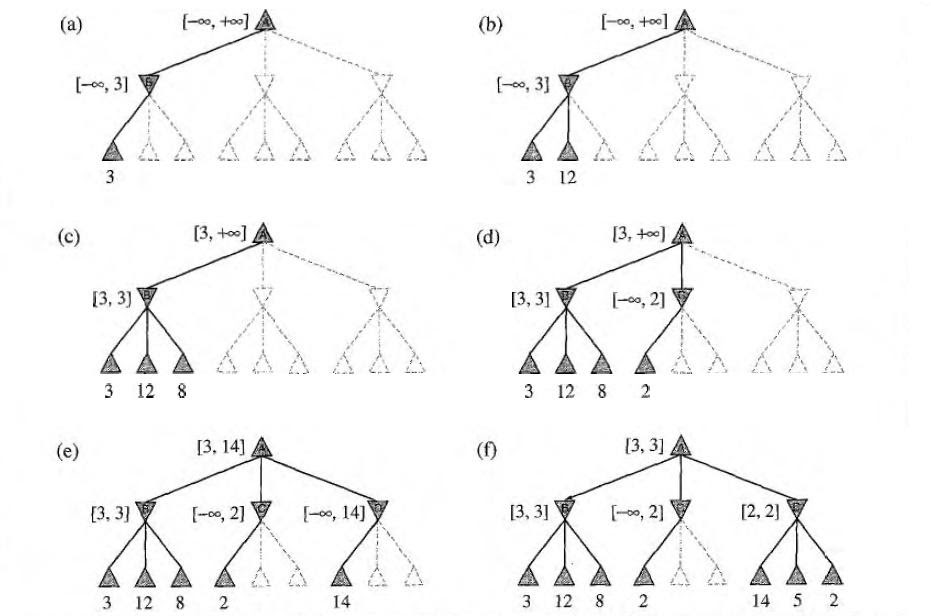


Figure 1.2

Problem

Assume Arko and Riyad are two friends. Riyad is engaged in collecting funds for the upcoming fresher’s orientation party. Map the above mentioned minimax algorithm to solve the problem. Assume each of them gets fair chance to optimize the amount. For example, if Arko gets 2 chances to minimize the amount then Riyad also gets the same number of chances in order to maximize the amount. Arko or Riyad has to select 1 note from 3 notes (given) at a time. Sample input and output is provided below.

Sample Input

|  |  |
| --- | --- |
| 1  3  1 20 | #Number of turns for Riyad [Assume both of them get equal chance, this should determine the depth of the tree]  #Number of notes from which the choice has to be made at certain time (Branches per each node)  #Minimum and Maximum value for the range of notes. (Value of leaf nodes) |

Sample Output

|  |  |
| --- | --- |
| Depth:2  Branch:3  Terminal States (Leaf Nodes): 9  Maximum amount: 3  Comparisons: 9  Comparisons: 7 | =>2\*1  =>Given!  =>3^2  =>Maximum amount collected by Riyad  =>Before Alpha-beta Pruning  =>After Alpha-beta Pruning |

Reading Materials

Artificial Intelligence, A modern approach [Norvig, Russel], 2nd/3rd Edition

CODE:

[Skip to content](https://github.com/mehadihn/BRACUCSE422/blob/master/Alpha%20Beta%20Minmax/minimax.py#start-of-content)

* Why GitHub?
* [Team](https://github.com/team)
* [Enterprise](https://github.com/enterprise)
* Explore
* [Marketplace](https://github.com/marketplace)
* Pricing

Top of Form



Bottom of Form

[Sign in](https://github.com/login?return_to=%2Fmehadihn%2FBRACUCSE422%3Ffbclid%3DIwAR2o_Xc4OQSbSD5e1iVHWKkbK-4HMCXXbxk0oIyYD4A4lZRCJsIv0ECcopE)[Sign up](https://github.com/join?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F%3Cuser-name%3E%2F%3Crepo-name%3E&source=header-repo&source_repo=mehadihn%2FBRACUCSE422)

# [mehadihn](https://github.com/mehadihn)/[BRACUCSE422](https://github.com/mehadihn/BRACUCSE422)

* [Watch](https://github.com/login?return_to=%2Fmehadihn%2FBRACUCSE422)[**1**](https://github.com/mehadihn/BRACUCSE422/watchers)
* [Star](https://github.com/login?return_to=%2Fmehadihn%2FBRACUCSE422)[**0**](https://github.com/mehadihn/BRACUCSE422/stargazers)
* [Fork](https://github.com/login?return_to=%2Fmehadihn%2FBRACUCSE422)[**0**](https://github.com/mehadihn/BRACUCSE422/network/members)
* [**Code**](https://github.com/mehadihn/BRACUCSE422)
* [Issues](https://github.com/mehadihn/BRACUCSE422/issues)
* [Pull requests](https://github.com/mehadihn/BRACUCSE422/pulls)
* [Actions](https://github.com/mehadihn/BRACUCSE422/actions)
* [Projects](https://github.com/mehadihn/BRACUCSE422/projects)
* [Security](https://github.com/mehadihn/BRACUCSE422/security)
* [Insights](https://github.com/mehadihn/BRACUCSE422/pulse)

### GitHub is where the world builds software

Millions of developers and companies build, ship, and maintain their software on GitHub — the largest and most advanced development platform in the world.

[Sign up for free](https://github.com/join?source=prompt-blob-show&source_repo=mehadihn%2FBRACUCSE422)Dismiss

 master

## [BRACUCSE422](https://github.com/mehadihn/BRACUCSE422)/[Alpha Beta Minmax](https://github.com/mehadihn/BRACUCSE422/tree/master/Alpha%20Beta%20Minmax)/****minimax.py**** / Jump to

[Go to file](https://github.com/mehadihn/BRACUCSE422/find/master)

[@mehadihn](https://github.com/mehadihn)

[mehadihn](https://github.com/mehadihn) [alpha](https://github.com/mehadihn/BRACUCSE422/commit/6901eecf48719d72bc15cae6b7179f32d31832df)

Latest commit [6901eec](https://github.com/mehadihn/BRACUCSE422/commit/6901eecf48719d72bc15cae6b7179f32d31832df) on Dec 5, 2019[**History**](https://github.com/mehadihn/BRACUCSE422/commits/master/Alpha%20Beta%20Minmax/minimax.py)

**1** contributor

311 lines (252 sloc)  7.28 KB

[Raw](https://github.com/mehadihn/BRACUCSE422/raw/master/Alpha%20Beta%20Minmax/minimax.py)[Blame](https://github.com/mehadihn/BRACUCSE422/blame/master/Alpha%20Beta%20Minmax/minimax.py)

|  |  |
| --- | --- |
|  | #!/usr/bin/env python3 |
|  | from math import inf as infinity |
|  | from random import choice |
|  | import platform |
|  | import time |
|  | from os import system |
|  |  |
|  | """ |
|  | An implementation of Minimax AI Algorithm in Tic Tac Toe, |
|  | using Python. |
|  | This software is available under GPL license. |
|  | Author: Clederson Cruz |
|  | Year: 2017 |
|  | License: GNU GENERAL PUBLIC LICENSE (GPL) |
|  | """ |
|  |  |
|  | HUMAN = -1 |
|  | COMP = +1 |
|  | board = [ |
|  | [0, 0, 0], |
|  | [0, 0, 0], |
|  | [0, 0, 0], |
|  | ] |
|  |  |
|  |  |
|  | def evaluate(state): |
|  | """ |
|  | Function to heuristic evaluation of state. |
|  | :param state: the state of the current board |
|  | :return: +1 if the computer wins; -1 if the human wins; 0 draw |
|  | """ |
|  | if wins(state, COMP): |
|  | score = +1 |
|  | elif wins(state, HUMAN): |
|  | score = -1 |
|  | else: |
|  | score = 0 |
|  |  |
|  | return score |
|  |  |
|  |  |
|  | def wins(state, player): |
|  | """ |
|  | This function tests if a specific player wins. Possibilities: |
|  | \* Three rows [X X X] or [O O O] |
|  | \* Three cols [X X X] or [O O O] |
|  | \* Two diagonals [X X X] or [O O O] |
|  | :param state: the state of the current board |
|  | :param player: a human or a computer |
|  | :return: True if the player wins |
|  | """ |
|  | win\_state = [ |
|  | [state[0][0], state[0][1], state[0][2]], |
|  | [state[1][0], state[1][1], state[1][2]], |
|  | [state[2][0], state[2][1], state[2][2]], |
|  | [state[0][0], state[1][0], state[2][0]], |
|  | [state[0][1], state[1][1], state[2][1]], |
|  | [state[0][2], state[1][2], state[2][2]], |
|  | [state[0][0], state[1][1], state[2][2]], |
|  | [state[2][0], state[1][1], state[0][2]], |
|  | ] |
|  | if [player, player, player] in win\_state: |
|  | return True |
|  | else: |
|  | return False |
|  |  |
|  |  |
|  | def game\_over(state): |
|  | """ |
|  | This function test if the human or computer wins |
|  | :param state: the state of the current board |
|  | :return: True if the human or computer wins |
|  | """ |
|  | return wins(state, HUMAN) or wins(state, COMP) |
|  |  |
|  |  |
|  | def empty\_cells(state): |
|  | """ |
|  | Each empty cell will be added into cells' list |
|  | :param state: the state of the current board |
|  | :return: a list of empty cells |
|  | """ |
|  | cells = [] |
|  |  |
|  | for x, row in enumerate(state): |
|  | for y, cell in enumerate(row): |
|  | if cell == 0: |
|  | cells.append([x, y]) |
|  |  |
|  | return cells |
|  |  |
|  |  |
|  | def valid\_move(x, y): |
|  | """ |
|  | A move is valid if the chosen cell is empty |
|  | :param x: X coordinate |
|  | :param y: Y coordinate |
|  | :return: True if the board[x][y] is empty |
|  | """ |
|  | if [x, y] in empty\_cells(board): |
|  | return True |
|  | else: |
|  | return False |
|  |  |
|  |  |
|  | def set\_move(x, y, player): |
|  | """ |
|  | Set the move on board, if the coordinates are valid |
|  | :param x: X coordinate |
|  | :param y: Y coordinate |
|  | :param player: the current player |
|  | """ |
|  | if valid\_move(x, y): |
|  | board[x][y] = player |
|  | return True |
|  | else: |
|  | return False |
|  |  |
|  |  |
|  | def minimax(state, depth, player): |
|  | """ |
|  | AI function that choice the best move |
|  | :param state: current state of the board |
|  | :param depth: node index in the tree (0 <= depth <= 9), |
|  | but never nine in this case (see iaturn() function) |
|  | :param player: an human or a computer |
|  | :return: a list with [the best row, best col, best score] |
|  | """ |
|  | if player == COMP: |
|  | best = [-1, -1, -infinity] |
|  | else: |
|  | best = [-1, -1, +infinity] |
|  |  |
|  | if depth == 0 or game\_over(state): |
|  | score = evaluate(state) |
|  | return [-1, -1, score] |
|  |  |
|  | for cell in empty\_cells(state): |
|  | x, y = cell[0], cell[1] |
|  | state[x][y] = player |
|  | score = minimax(state, depth - 1, -player) |
|  | state[x][y] = 0 |
|  | score[0], score[1] = x, y |
|  |  |
|  | if player == COMP: |
|  | if score[2] > best[2]: |
|  | best = score # max value |
|  | else: |
|  | if score[2] < best[2]: |
|  | best = score # min value |
|  |  |
|  | return best |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | def render(state, c\_choice, h\_choice): |
|  | """ |
|  | Print the board on console |
|  | :param state: current state of the board |
|  | """ |
|  |  |
|  | chars = { |
|  | -1: h\_choice, |
|  | +1: c\_choice, |
|  | 0: ' ' |
|  | } |
|  | str\_line = '---------------' |
|  |  |
|  | print('\n' + str\_line) |
|  | for row in state: |
|  | for cell in row: |
|  | symbol = chars[cell] |
|  | print(f'| {symbol} |', end='') |
|  | print('\n' + str\_line) |
|  |  |
|  |  |
|  | def ai\_turn(c\_choice, h\_choice): |
|  | """ |
|  | It calls the minimax function if the depth < 9, |
|  | else it choices a random coordinate. |
|  | :param c\_choice: computer's choice X or O |
|  | :param h\_choice: human's choice X or O |
|  | :return: |
|  | """ |
|  | depth = len(empty\_cells(board)) |
|  | if depth == 0 or game\_over(board): |
|  | return |
|  |  |
|  |  |
|  | print(f'Computer turn [{c\_choice}]') |
|  | render(board, c\_choice, h\_choice) |
|  |  |
|  | if depth == 9: |
|  | x = choice([0, 1, 2]) |
|  | y = choice([0, 1, 2]) |
|  | else: |
|  | move = minimax(board, depth, COMP) |
|  | x, y = move[0], move[1] |
|  |  |
|  | set\_move(x, y, COMP) |
|  | time.sleep(1) |
|  |  |
|  |  |
|  | def human\_turn(c\_choice, h\_choice): |
|  | """ |
|  | The Human plays choosing a valid move. |
|  | :param c\_choice: computer's choice X or O |
|  | :param h\_choice: human's choice X or O |
|  | :return: |
|  | """ |
|  | depth = len(empty\_cells(board)) |
|  | if depth == 0 or game\_over(board): |
|  | return |
|  |  |
|  | # Dictionary of valid moves |
|  | move = -1 |
|  | moves = { |
|  | 1: [0, 0], 2: [0, 1], 3: [0, 2], |
|  | 4: [1, 0], 5: [1, 1], 6: [1, 2], |
|  | 7: [2, 0], 8: [2, 1], 9: [2, 2], |
|  | } |
|  |  |
|  |  |
|  | print(f'Human turn [{h\_choice}]') |
|  | render(board, c\_choice, h\_choice) |
|  |  |
|  | while move < 1 or move > 9: |
|  | try: |
|  | move = int(input('Use numpad (1..9): ')) |
|  | coord = moves[move] |
|  | can\_move = set\_move(coord[0], coord[1], HUMAN) |
|  |  |
|  | if not can\_move: |
|  | print('Bad move') |
|  | move = -1 |
|  | except (EOFError, KeyboardInterrupt): |
|  | print('Bye') |
|  | exit() |
|  | except (KeyError, ValueError): |
|  | print('Bad choice') |
|  |  |
|  |  |
|  | def main(): |
|  | """ |
|  | Main function that calls all functions |
|  | """ |
|  |  |
|  | h\_choice = '' # X or O |
|  | c\_choice = '' # X or O |
|  | first = '' # if human is the first |
|  |  |
|  | # Human chooses X or O to play |
|  | while h\_choice != 'O' and h\_choice != 'X': |
|  | try: |
|  | print('') |
|  | h\_choice = input('Choose X or O\nChosen: ').upper() |
|  | except (EOFError, KeyboardInterrupt): |
|  | print('Bye') |
|  | exit() |
|  | except (KeyError, ValueError): |
|  | print('Bad choice') |
|  |  |
|  | # Setting computer's choice |
|  | if h\_choice == 'X': |
|  | c\_choice = 'O' |
|  | else: |
|  | c\_choice = 'X' |
|  |  |
|  | # Human may starts first |
|  |  |
|  | while first != 'Y' and first != 'N': |
|  | try: |
|  | first = input('First to start?[y/n]: ').upper() |
|  | except (EOFError, KeyboardInterrupt): |
|  | print('Bye') |
|  | exit() |
|  | except (KeyError, ValueError): |
|  | print('Bad choice') |
|  |  |
|  | # Main loop of this game |
|  | while len(empty\_cells(board)) > 0 and not game\_over(board): |
|  | if first == 'N': |
|  | ai\_turn(c\_choice, h\_choice) |
|  | first = '' |
|  |  |
|  | human\_turn(c\_choice, h\_choice) |
|  | ai\_turn(c\_choice, h\_choice) |
|  |  |
|  | # Game over message |
|  | if wins(board, HUMAN): |
|  |  |
|  | print(f'Human turn [{h\_choice}]') |
|  | render(board, c\_choice, h\_choice) |
|  | print('YOU WIN!') |
|  | elif wins(board, COMP): |
|  |  |
|  | print(f'Computer turn [{c\_choice}]') |
|  | render(board, c\_choice, h\_choice) |
|  | print('YOU LOSE!') |
|  | else: |
|  |  |
|  | render(board, c\_choice, h\_choice) |
|  | print('DRAW!') |
|  |  |
|  | exit() |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | main() |