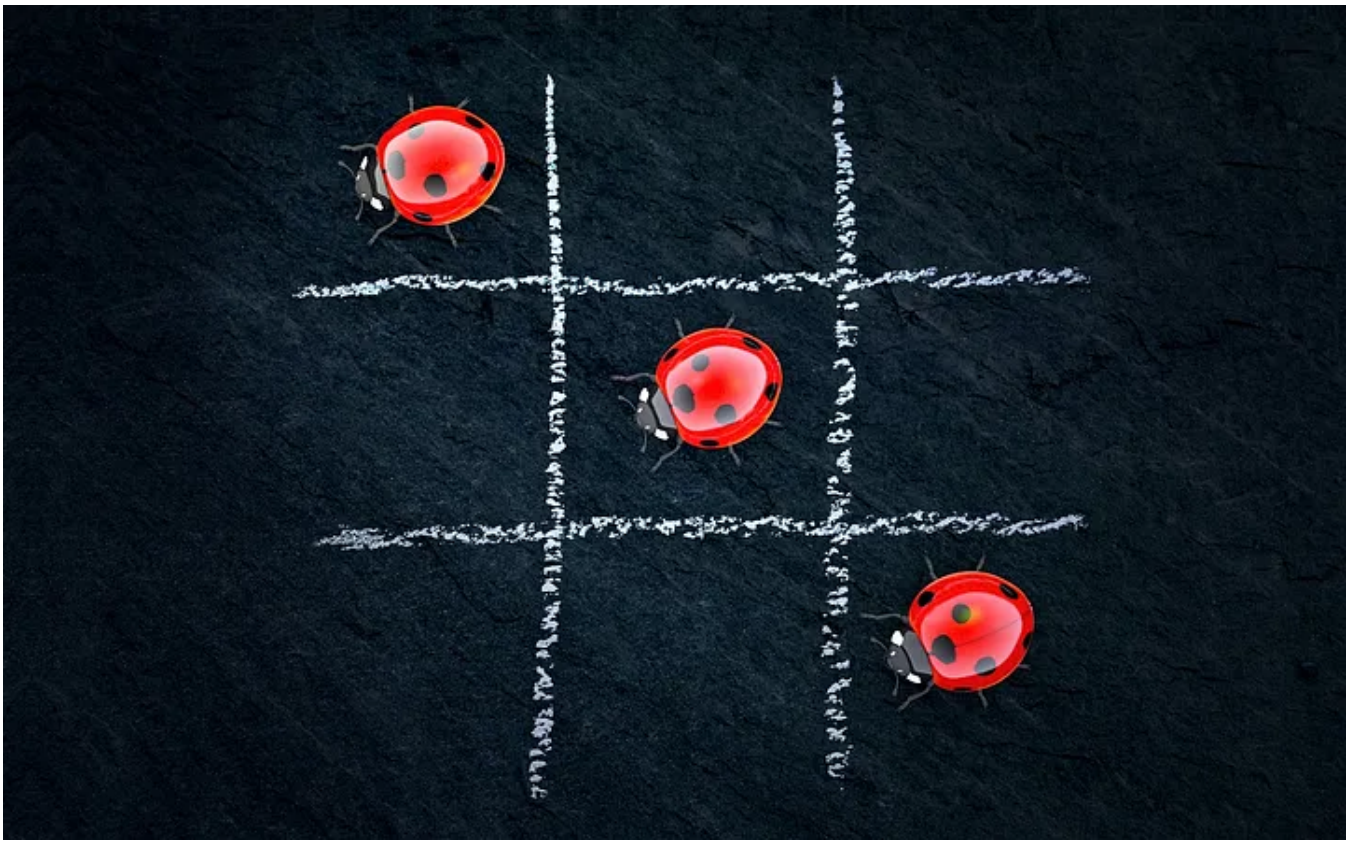
Mastering Tic-Tac-Toe with Minimax Algorithm in Python

Yiğit PIRILDAK · Follow
Published in Level Up Coding
6 min read · May 13, 2020

Listen Share



I've been dabbling in Game Theory recently and had to re-visit the Minimax algorithm. The algorithm was already familiar to me but I never took the time to actually implement it. It seemed like a good opportunity to implement it for a simple game such as Tic-Tac-Toe and explain its basics. Full source is available here. Without further ado, let's get into it!

Minimax Algorithm

Minimax is a type of **adversarial search** algorithm for generating and exploring game trees n is equivalent s end up being zero.

Adversarial search differs from conventional searching algorithms by adding **opponents** into the mix. Minimax algorithm keeps playing the turns of both player and the opponent optimally to figure out the best possible move.

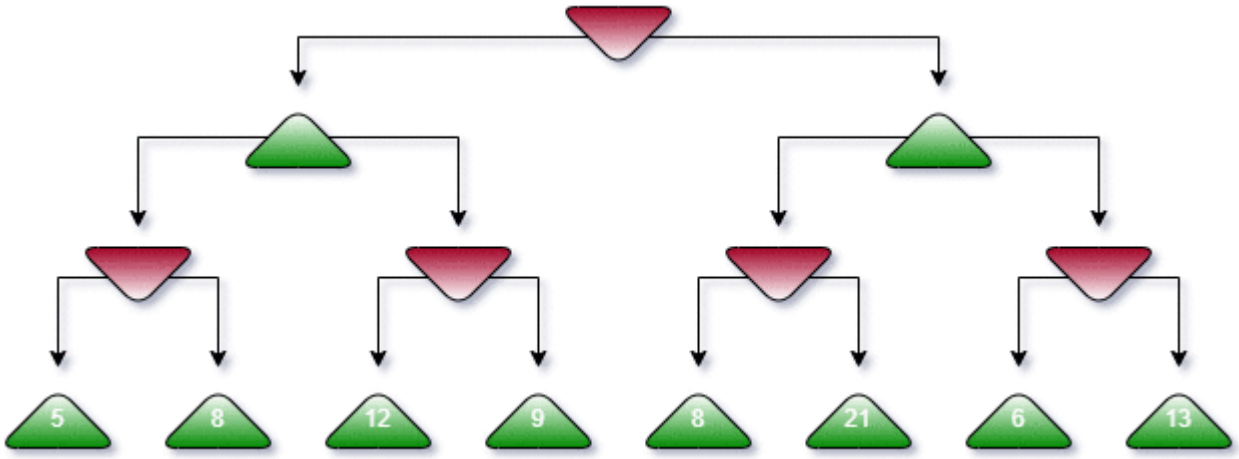
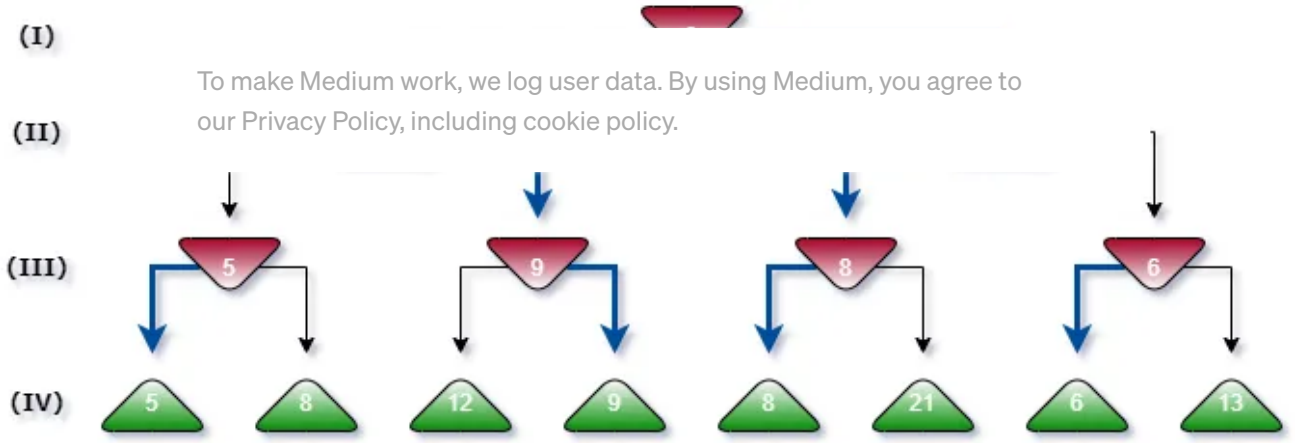


Figure-1 : Minimax algorithm in action.

Let's consider a simple game where there are only 2 possible moves in each state, such as **Figure-1**. Nodes of the tree represents **game states**, and edges represent **moves**. Top node is the **minimizer**, which means it's your opponents turn. Naturally, it will pick the least favorable position **for you.**
In order for minimizer to figure out the least favorable move given the current game state, all possible states that can be reached from there must be generated. Luckily we are only looking 3 moves ahead, so our tree is rather manageable.

Let's take a look at each level of the tree:

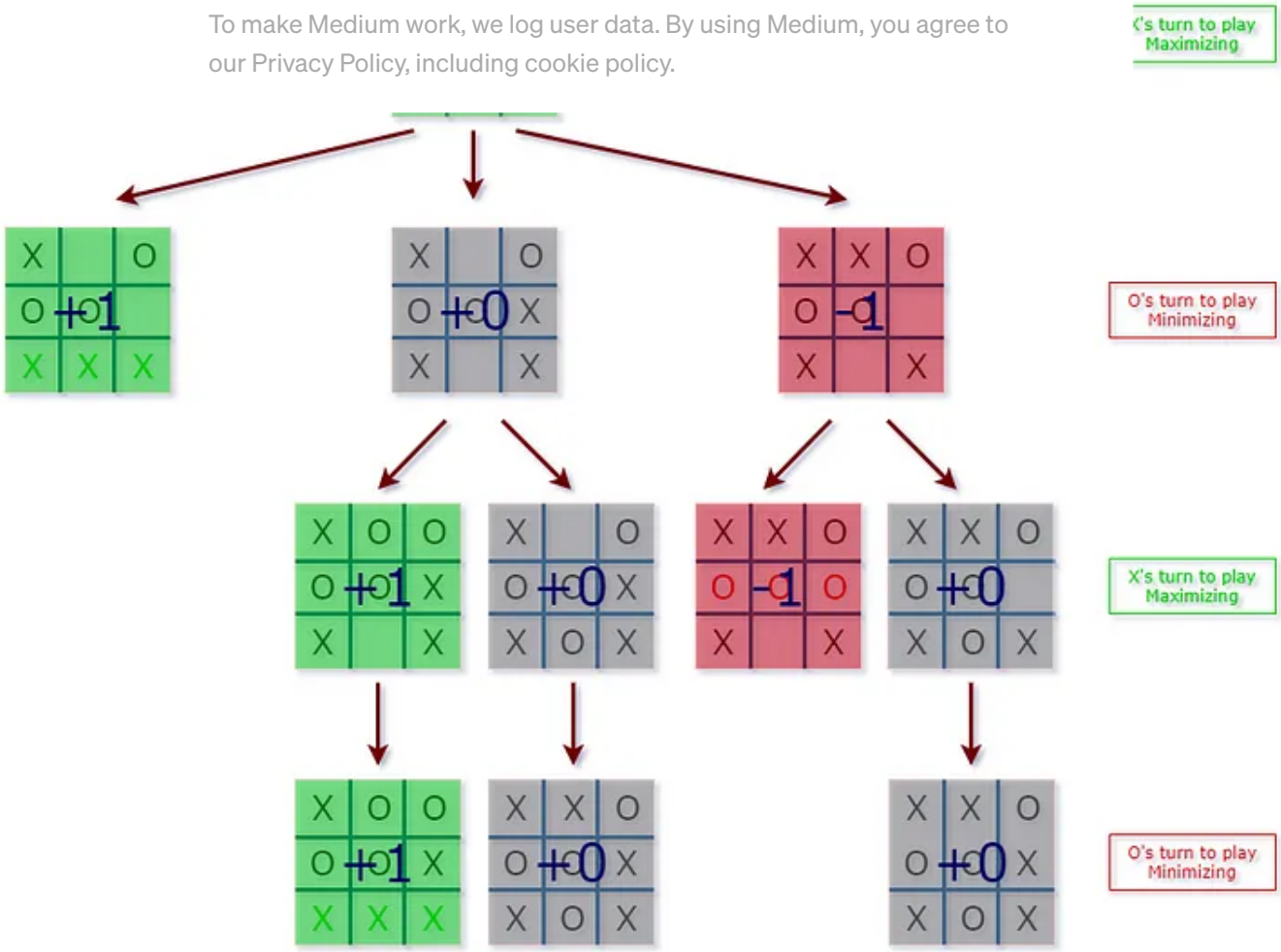


- **(IV)** These are game's terminal states. Each of these are evaluated and assigned a score based on how favorable they are.
- **(III)** Last move before the game ends, and it's minimizer's turn. The least favorable state for the maximizer is chosen.
- **(II)** Now maximizer knows how minimizer will play the next turn, out of all possible states that can be reached, the most optimal one is chosen.
- **(I)** Now that we're back to the top, it's decision time. By looking three moves ahead, minimizer knows exactly which path to take. Naturally, the path that had a score of **8** is chosen.

As you can see, since leaf nodes are terminal states, once minimax algorithm plays out each turn that leads to them, everything is decided. Assuming you are the one playing against that AI, even if you're playing perfectly there's no way for you to get a score higher than 8 at this point.

Minimax with Tic-Tac-Toe

Tic-Tac-Toe is a rather simply game with only **255,168** possible games that can be played. This number is trivial for today's computers, that's why Tic-Tac-Toe is considered to be a **solved game** which means the outcome can be predicted given **any state**. The game is so simple that we can generate the entire game tree without trouble, whereas in chess there are already **69,352,859,712,417** possible games that could've been played after just 10 moves.



Part of a Tic-Tac-Toe game tree solved with minimax

I've used the **p5** library for python to create a simple Tic-Tac-Toe board. Board state is represented in a two-dimensional, 3x3 matrix. After each move, board state is evaluated to check if a terminal state is reached.

Implementing minimax for Tic-Tac-Toe is simple. We will recursively generate the game tree l eaching a terminal st or draw.

Then based on these terminal states, for each explored turn either maximizer or minimizer will pick the most appropriate move. Doing so will propagate these outcomes all the way up to the root of the tree. The algorithm looks somewhat like this:

```
procedure minimax(maximizing:Boolean, board:Board)
    if board.is_terminal()
        if board.state() is draw
            return 0
        else
            return board.won() ? 1 : -1
        end if
    end if

    scores ← []
    for move in board.possible_moves()
        board.make_move(move)
        scores.add(minimax(!maximizing, board))
        board.revert_move()
    end for

    if maximizing
        return max(scores)
    else
        return min(scores)
    end if
end procedure
```

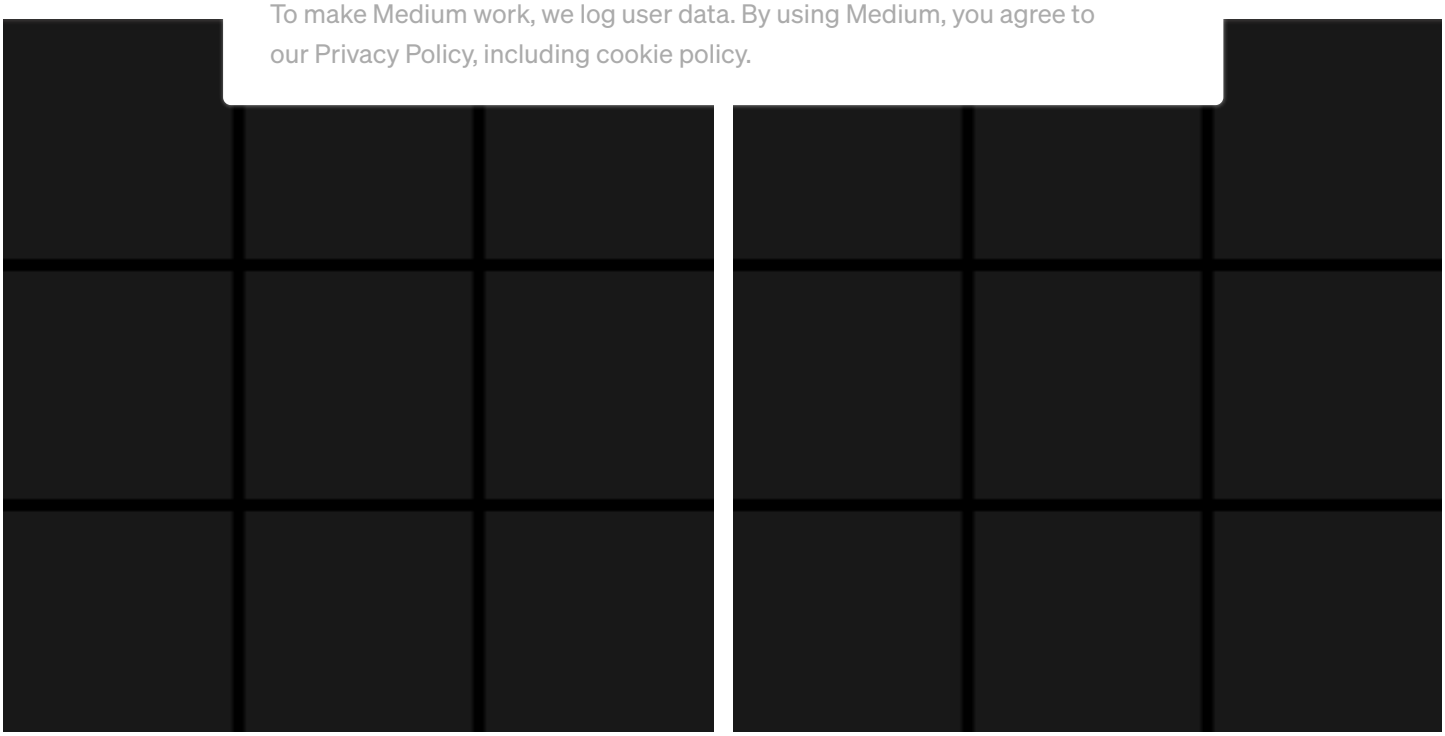
Implementation in python can be found below. First, we check whether the current state is terminal and return a score if it is. If it's not a terminal state, for all possible moves, we make the move, switch players, call minimax with the new board state, add its return value to a list of scores and undo the move.

After exhausting all possible moves, we will either return the maximum or minimum of our scores list, depending on whether we're maximizing or minimizing.

```python
def make_best_move():
    be
    be
    fo
        ticTacBoard.make_move(move)
        score = minimax(False, aiPlayer, ticTacBoard)
        ticTacBoard.undo()
        if (score > bestScore):
            bestScore = score
            bestMove = move
    ticTacBoard.make_move(bestMove)

def minimax(isMaxTurn, maximizerMark, board):
    state = board.get_state()
    if (state is State.DRAW):
        return 0
    elif (state is State.OVER):
        return 1 if board.get_winner() is maximizerMark else -1

    scores = []
    for move in board.get_possible_moves():
        board.make_move(move)
        scores.append(minimax(not isMaxTurn, maximizerMark, board))
        board.undo()

    return max(scores) if isMaxTurn else min(scores)
```


minimax.py hosted with ❤ by **GitHub** **view raw**

Since it's a solved game, we already know that if both sides play perfectly, there's no chance of winning in Tic-Tac-Toe. So putting AI against AI would only result in a series of draws.



X : Human, O : AI

Improving Minimax Algorithm

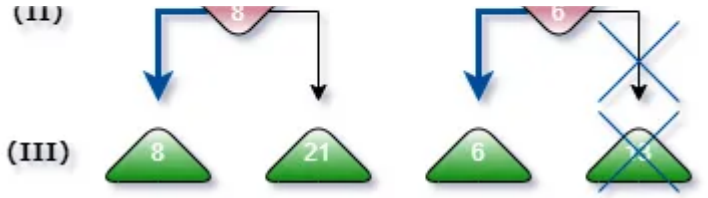
For simple games generating the entire tree works great. If you think about a game as complex as chess, unfortunately that's hardly feasible. There are several ways the algorithm can be applied to more complex games.

Depth-Limited Minimax

It is possible assign a maximum depth limit to the algorithm in order to explore only a couple moves ahead. You will need an algorithm to evaluate non-terminal game states in order to make reasonable predictions.

Alpha-beta Pruning

This is an algorithm that eliminates parts of the tree so that they are not explored. Based on the scores you've already obtained, it's possible to decide it cannot be improved by exploring a certain part of the tree. Consider the following part of the tree:



Alpha-beta pruning

Assuming the algorithm explores from left to right, once we reach right side of level (II) and realize there's a score of six in the left child, there's no point in exploring the right side of it. The reason is that Maximizer decides which state in level (II) is chosen and there's already a score of 8 that it can pick. Seeing that right side of level (II) is **guaranteed to have a score of 6 or less**, a value of 8 becomes the clear winner here.

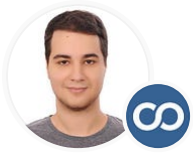
In complicated games with a lot of possible moves in each turn, dropping huge parts of the tree improves the algorithm significantly and may even allow you to go deeper.

IBM's DeepBlue chess computer that beat Garry Kasparov in 1997 used alpha-beta search, which is basically minimax with alpha-beta pruning. The algorithm is said to look 6 to 20 moves ahead, depending on the situation.

Wrap-up

Minimax is an old algorithm, but it still has many uses. It is a great introduction to game trees due to its simple nature. In games where exploring every possible state is unfeasible, finding a good algorithm for evaluating non-terminal states is the key. I hope the article can help you implement your own version and aid you in gaining a basic understanding of it!

Artificial Intelligence | Python | Software Development | Machine Learning | Programming



Follow

Written by Yiğit PIRILDAK

93 Followers · Writer for Level Up Coding

A curious Software Engineer who is interested in Embedded Systems and ML. Wastes time by playing video games, watching TV Shows and reading fantasy novels.

More from Yiğit PIRILDAK and Level Up Coding

Yiğit PIRILDAK

An Overview of OCPP (Open Charge Point Protocol)

If you’re in the EV Charging game, chances are you’ve probably heard about the OCPP. Designed by the Open Charge Alliance, OCPP is a very...

8 min read · Mar 4, 2020

69 3

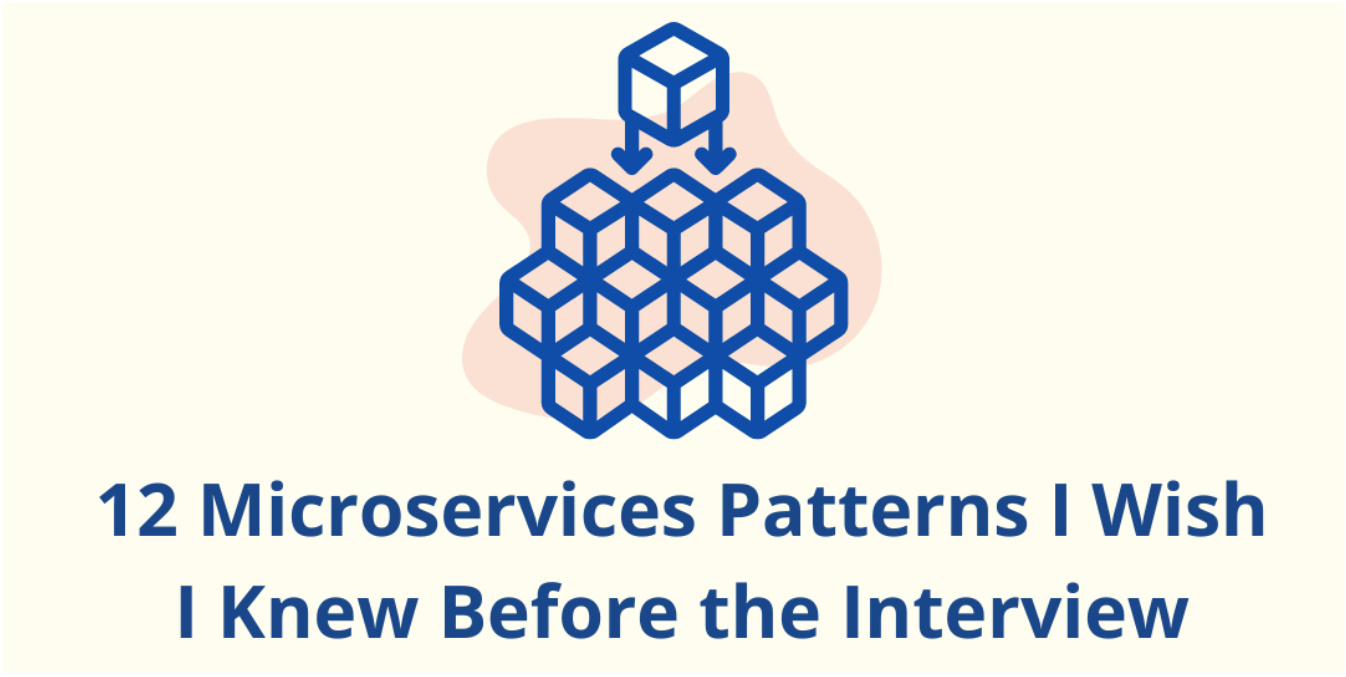
 Dr. Ashish Bamania in Level Up Coding

Google Rejected Max Howell(Creator Of Homebrew) For Getting This Interview Question Wrong. Can You?

Can you solve this Google interview question?

4 min read · Oct 3

6.2K 68



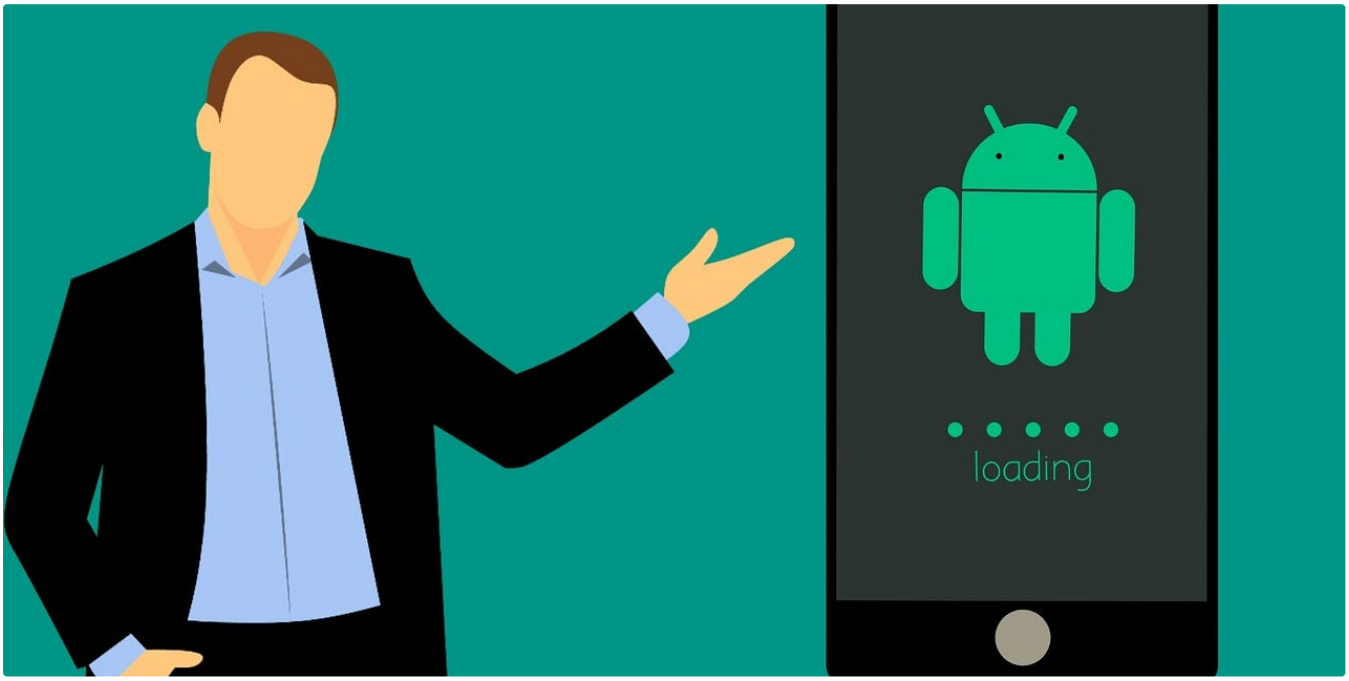
 Arslan Ahmad in Level Up Coding

12 Microservices Patterns I Wish I Knew Before the System Design Interview

Mastering th esign Patterns

13 min read · May 16

4.6K 21



Yiğit PIRILDAK

Android OTA Update Mechanism

OTA (Over The Air) updates, while not specific to Android, is a popular way of updating remote devices over the internet. Update packages...

7 min read · Mar 7, 2020

11 1

See all from Yiğit PIRILDAK

See all from Level Up Coding

Recommended from Medium

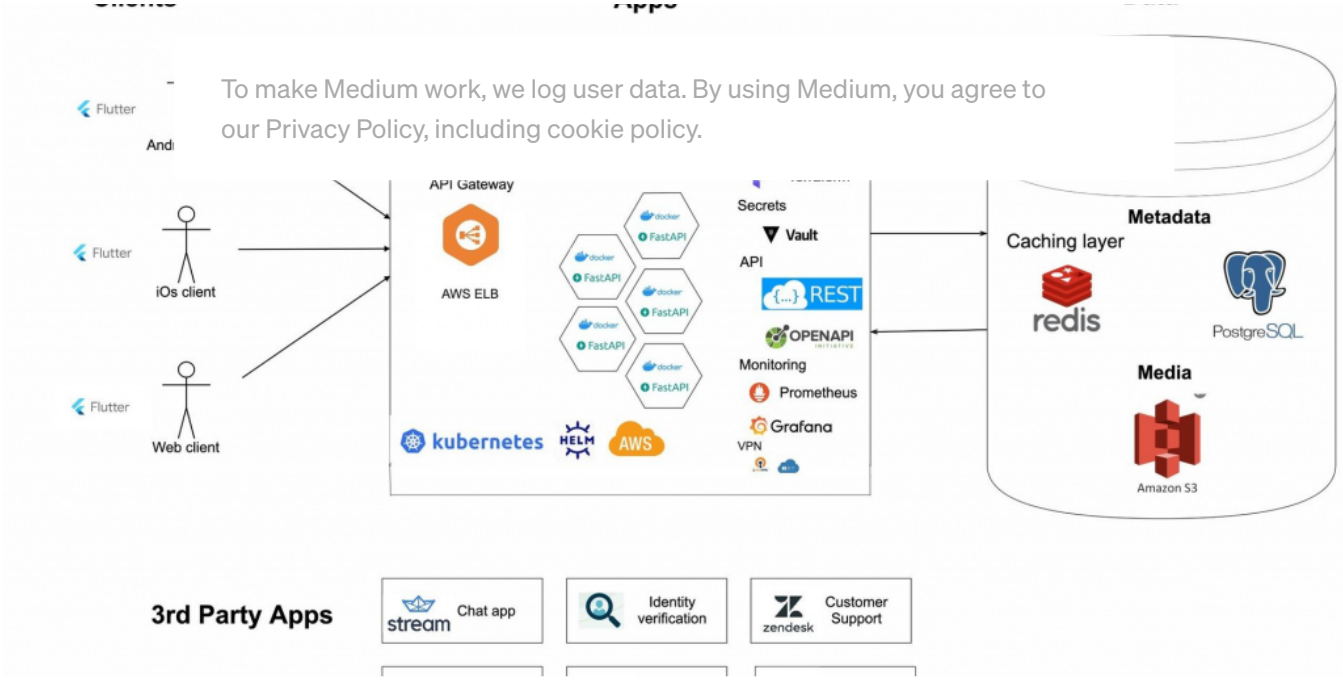
Arslan Ahmad in Level Up Coding

Don’t Just LeetCode; Follow the Coding Patterns Instead

What if you don’t like to practice 100s of coding questions before the interview?

5 min read · Sep 15, 2022

5.6K 30



Dmitry Kruglov in Better Programming

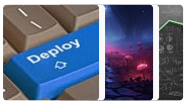
The Architecture of a Modern Startup

Hype wave, pragmatic evidence vs the need to move fast

16 min read · Nov 7, 2022

6.4K 63

Lists

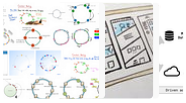


Predictive Modeling w/ Python

20 stories · 527 saves

Coding & Development

11 stories · 233 saves

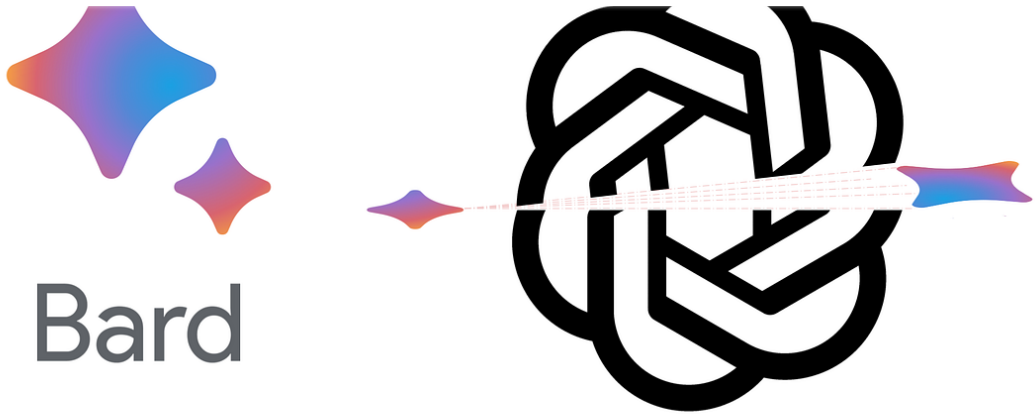


General Coding Knowledge

20 stories · 482 saves

ChatGPT

21 stories · 219 saves



AL Anany

The ChatGPT Hype Is Over—Now Watch How Google Will Kill ChatGPT.

It never happens instantly. The business game is longer than you know.

· 6 min read · Sep 1

16.7K 492

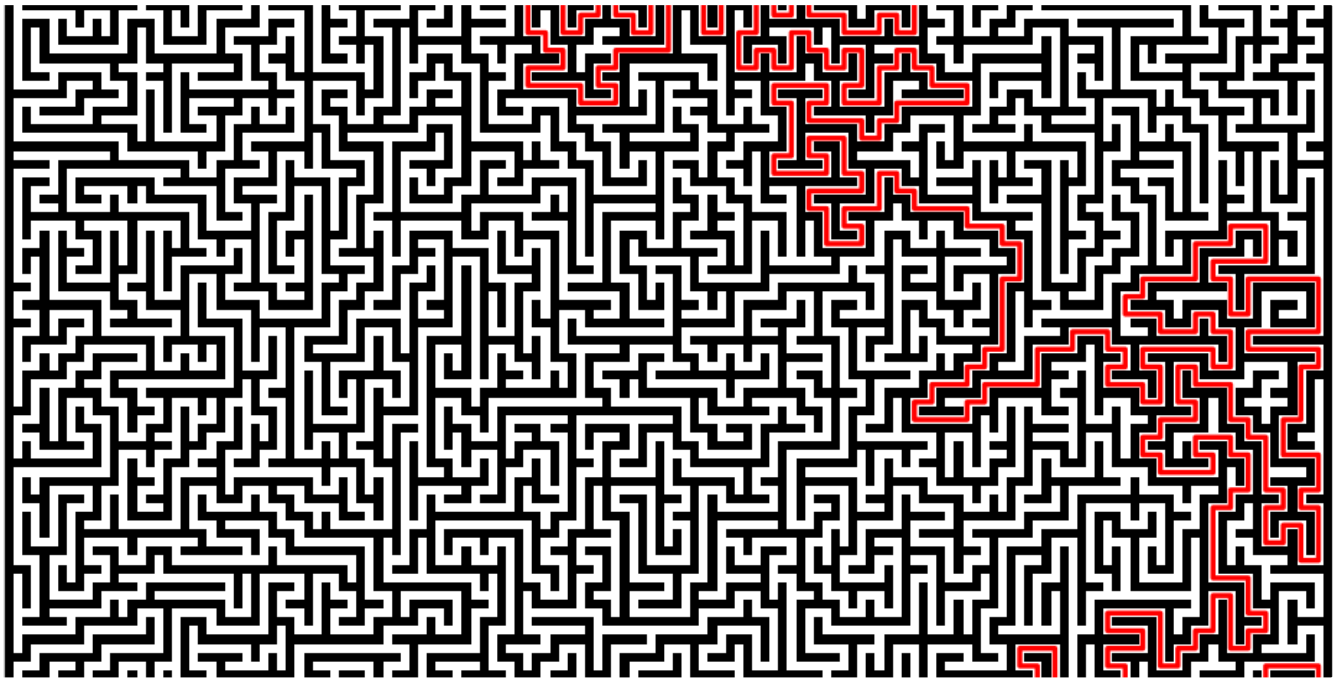
Lucia

The Rise of AI That Can Code

A Game-Cha

2 min read · C

40 1

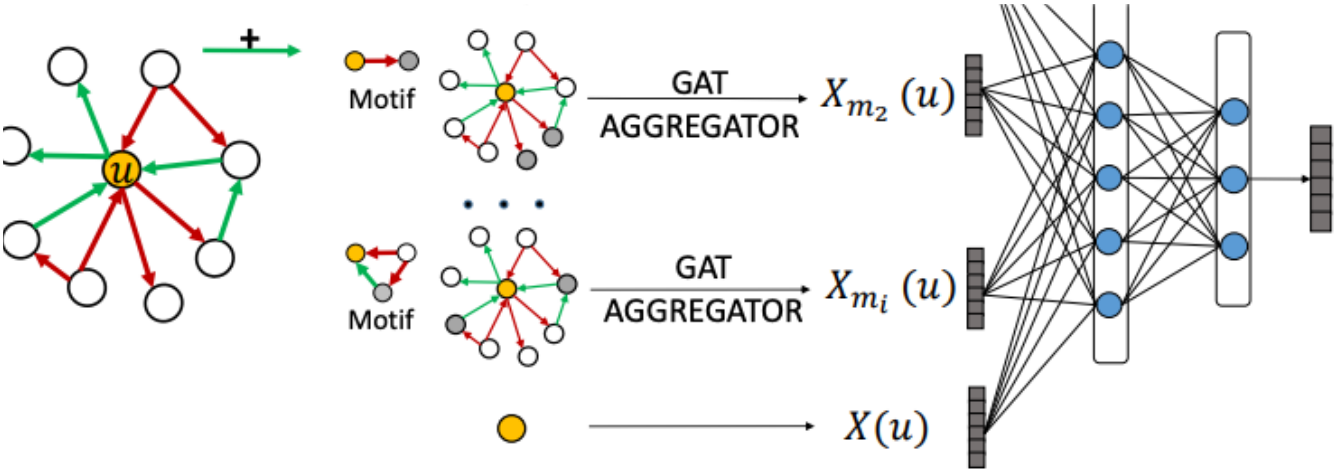


Michael Gold

Python's Path Through Mazes: A Journey of Creation and Solution

Introduction

11 min read · Aug 4



Daniel Ko

Graph Neural Networks: From CNNs to GATs

A high level understanding of the progression to SiGATs.

4 min read · Jun 6

1