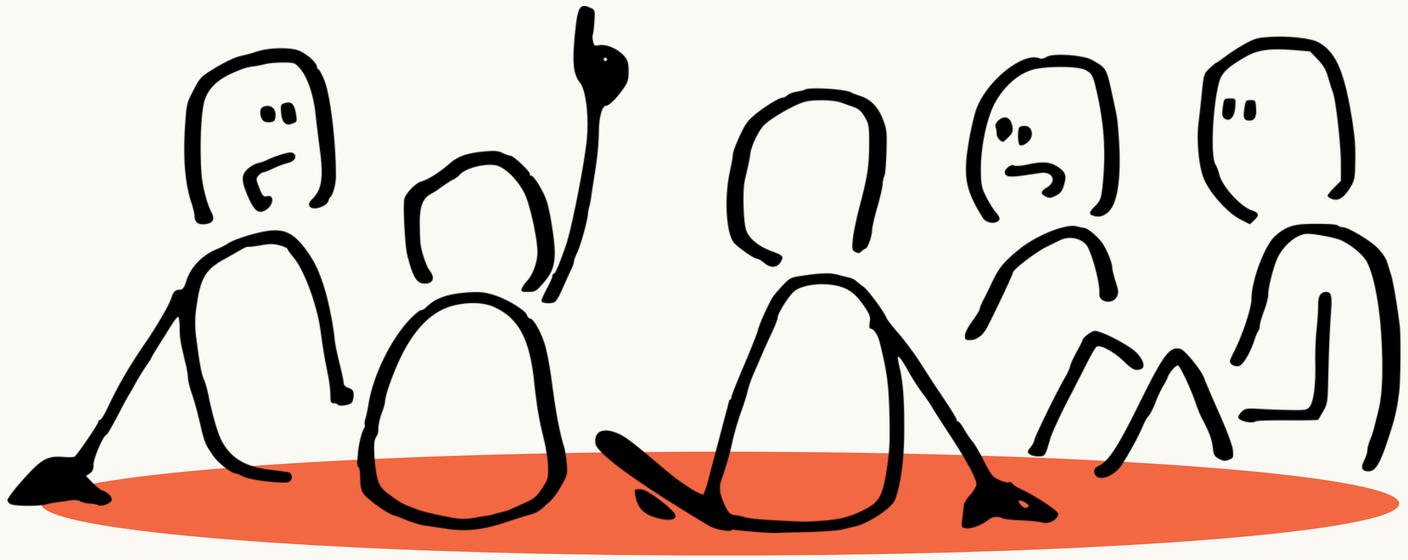
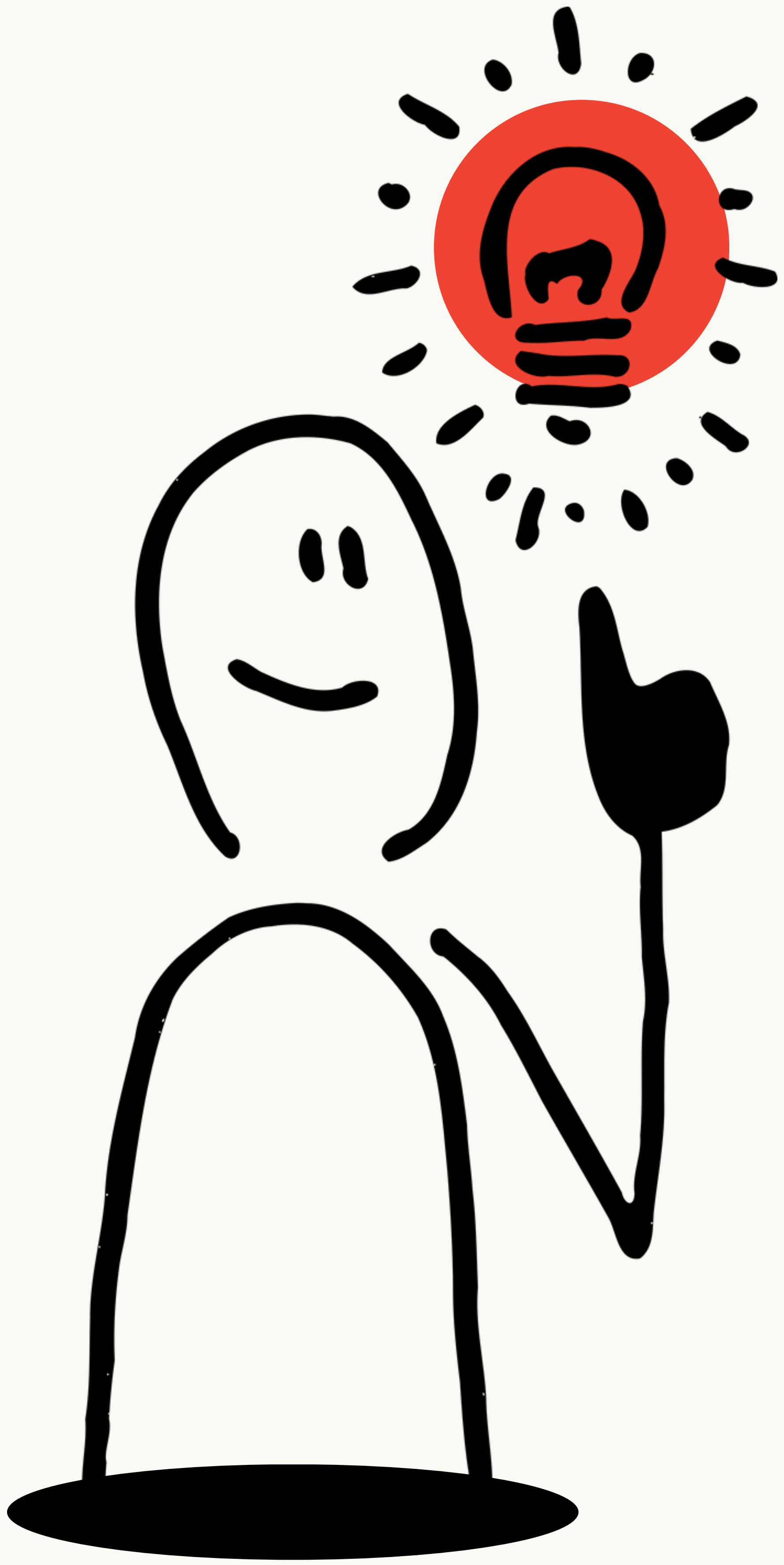
Algorithm and Data Structure

IMPLEMENTATION OF CLASSICAL AI ALGORITHMS IN A CONSOLE-*BASED CONNECT 4 GAME*



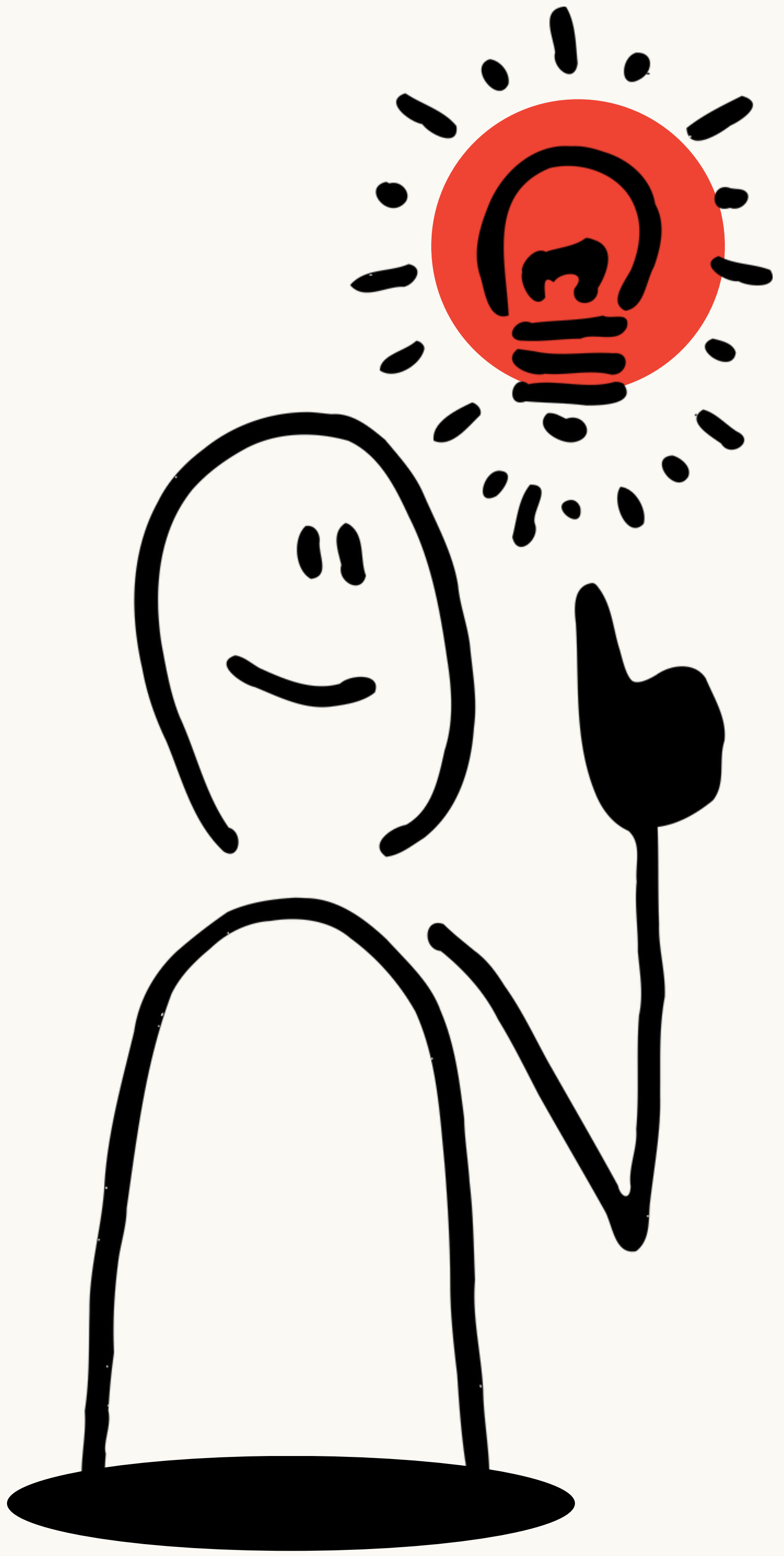
GROUP 4

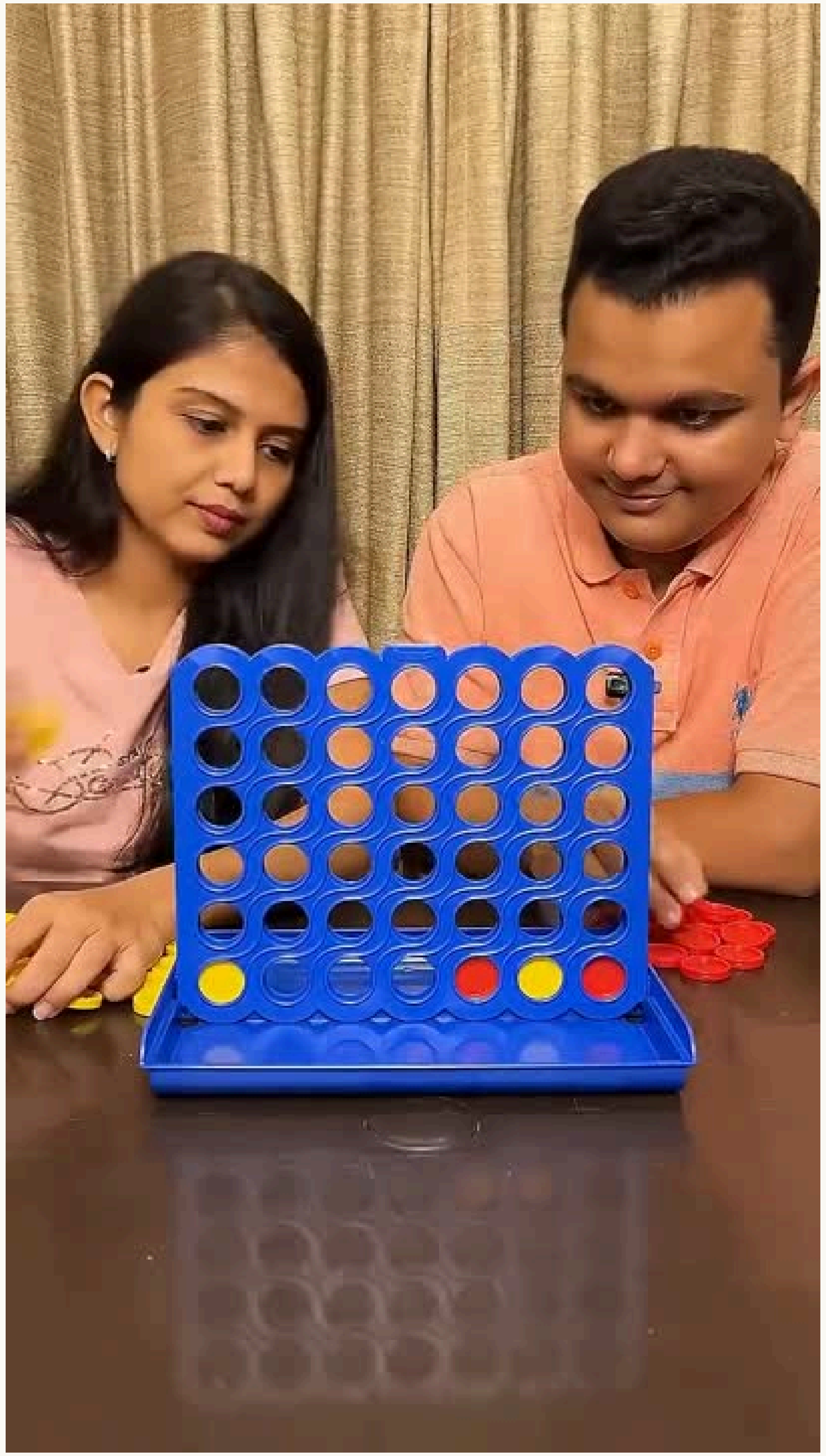
Agha Aryo Utomo / 5026241003
Maria Stephanie F. K / 5026241052
Dyandra R-Noor B / 5026241051

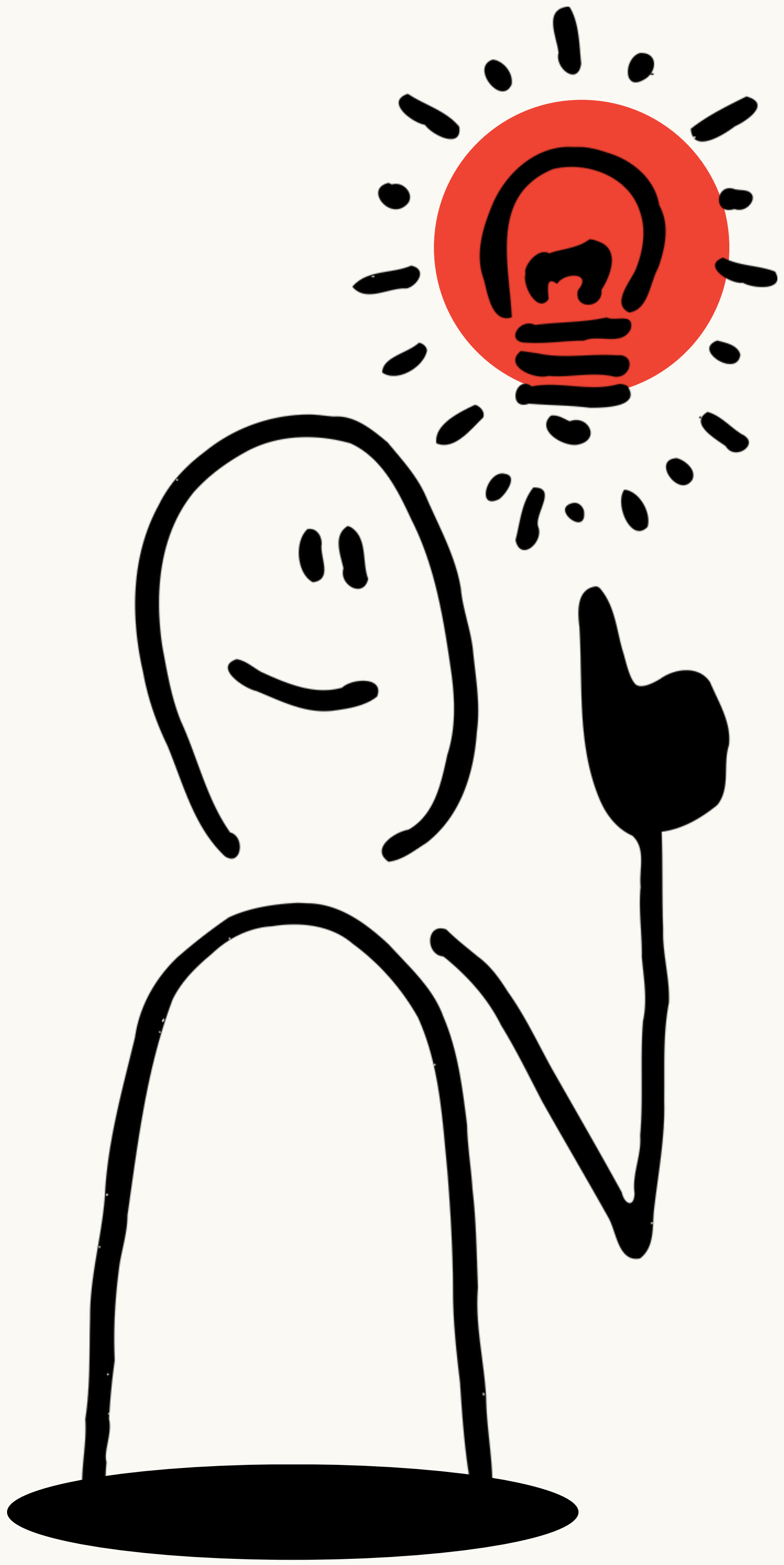


What is Connect 4?

Connect 4 is a two-player strategy game where opponents drop colored discs into a vertical grid, and the first player to get four of their own discs in a row wins







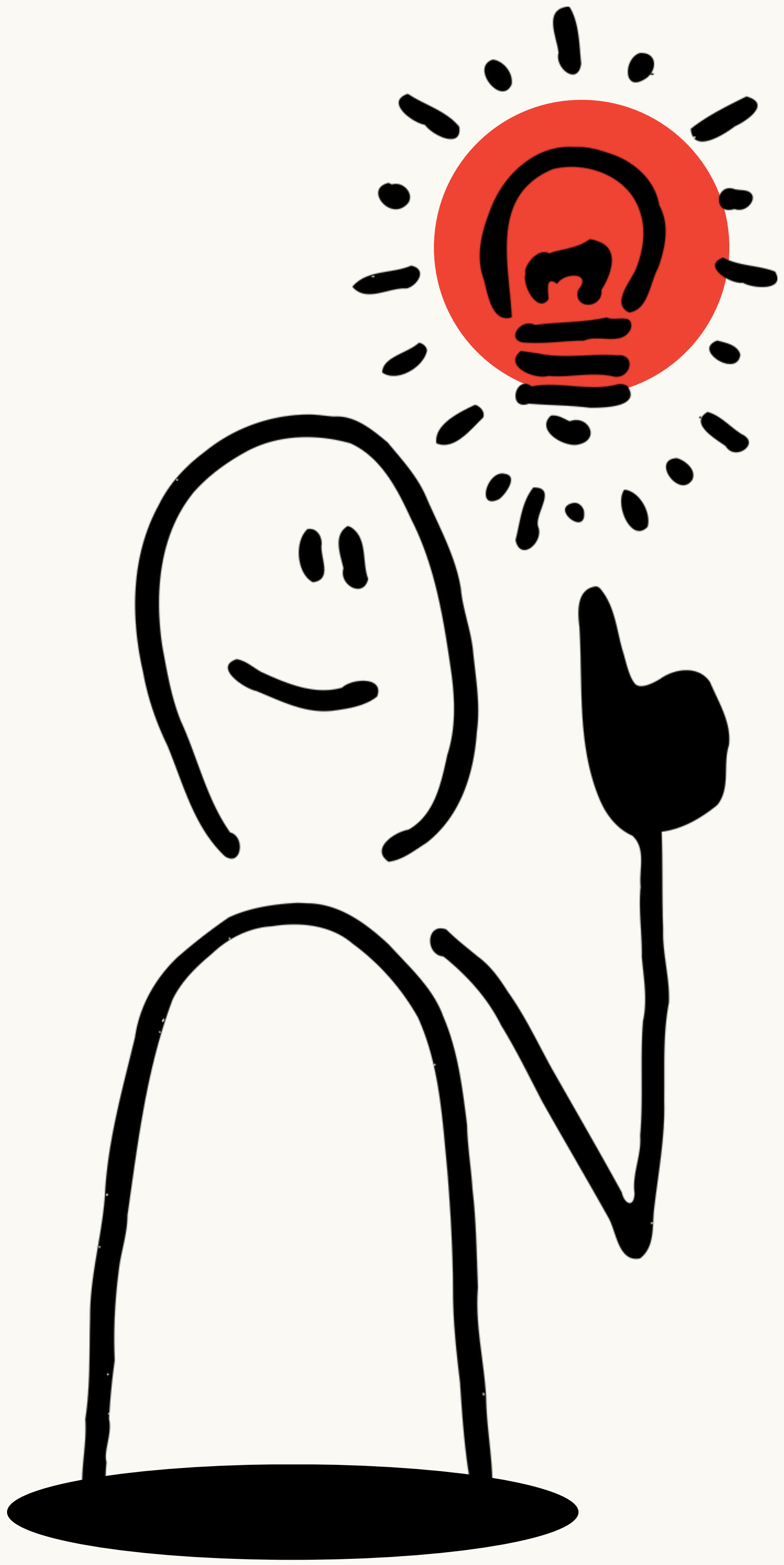
Background

Connect 4 require analyzing board states, detecting winning conditions, and predicting optimal moves

Artificial Intelligence (AI) enables the game to provide a challenging opponent by simulating future gameplay scenarios

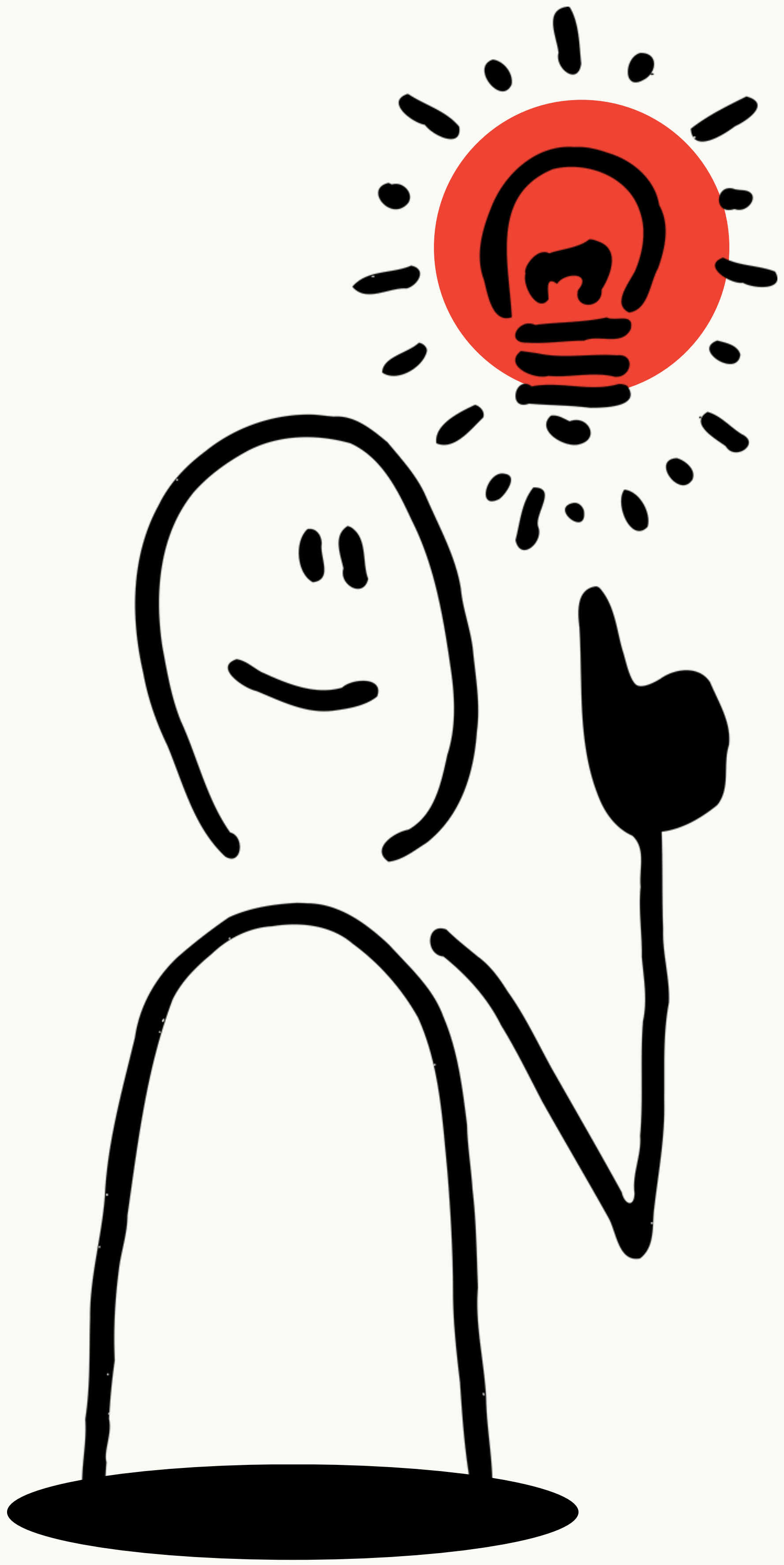
Connect 4 is ideal because its mechanics can be implemented using algorithms such as :

- **Searching** for valid moves
- **Sorting** potential moves based on heuristic scores
- **Graph-based** evaluation for detecting connected tokens
- **Tree-based** decision making through Minimax AI



Problem Statement

How to design and implement an intelligent Connect 4 game opponent that can make optimal decisions in real time?

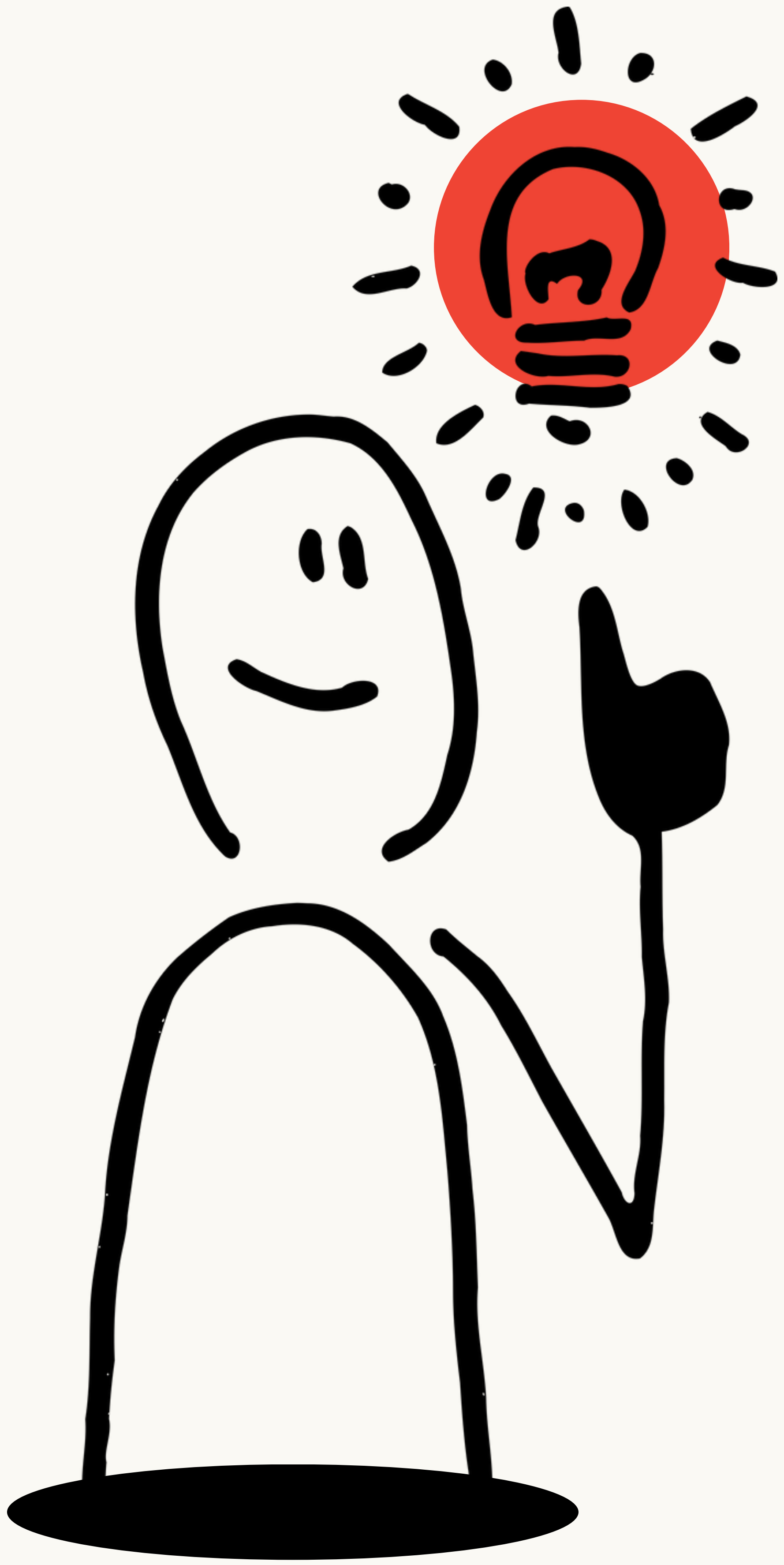


Objectives

1. Build a fully functioning console-based Connect 4 game in Java
2. Implement basic algorithms from class:

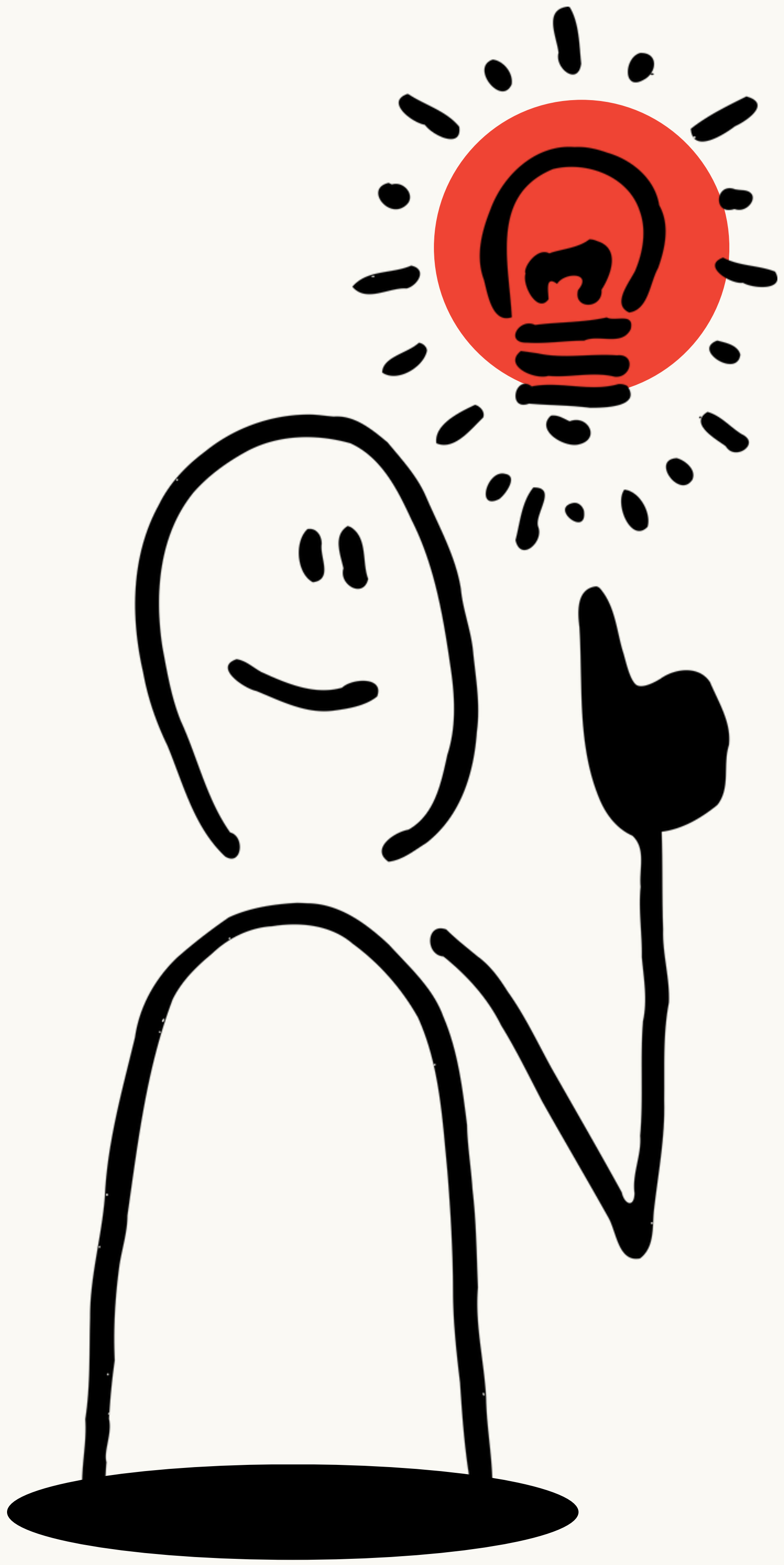
- Linear Search
- Merge Sort
- BFS (Graph)
- Tree Traversal + Minimax

3. Create an AI opponent that is smarter than static bots
4. Demonstrate functionality through in-exam code demo



System Overview

1. Player inputs move
2. Game validates with Searching
3. AI evaluates board
4. AI uses Sorting + Minimax
5. BFS checks for win



Algorithms Used in The Connect 4

Searching

Why use search algorithms?

- In a physical game of connect 4, the pieces will **drop down to the lowest possible position due to gravity**
- But since there is no gravity or physics in play here we will have to search for the lowest available position

Why linear search?

We will be using linear search to achieve this since we will only need a simple and straightforward algorithm to search each collumn

Example (AI or Player choose column 3):

Row	Token
5	“X”
4	“O”
3	“X”
2	**-**
1	-
0	-

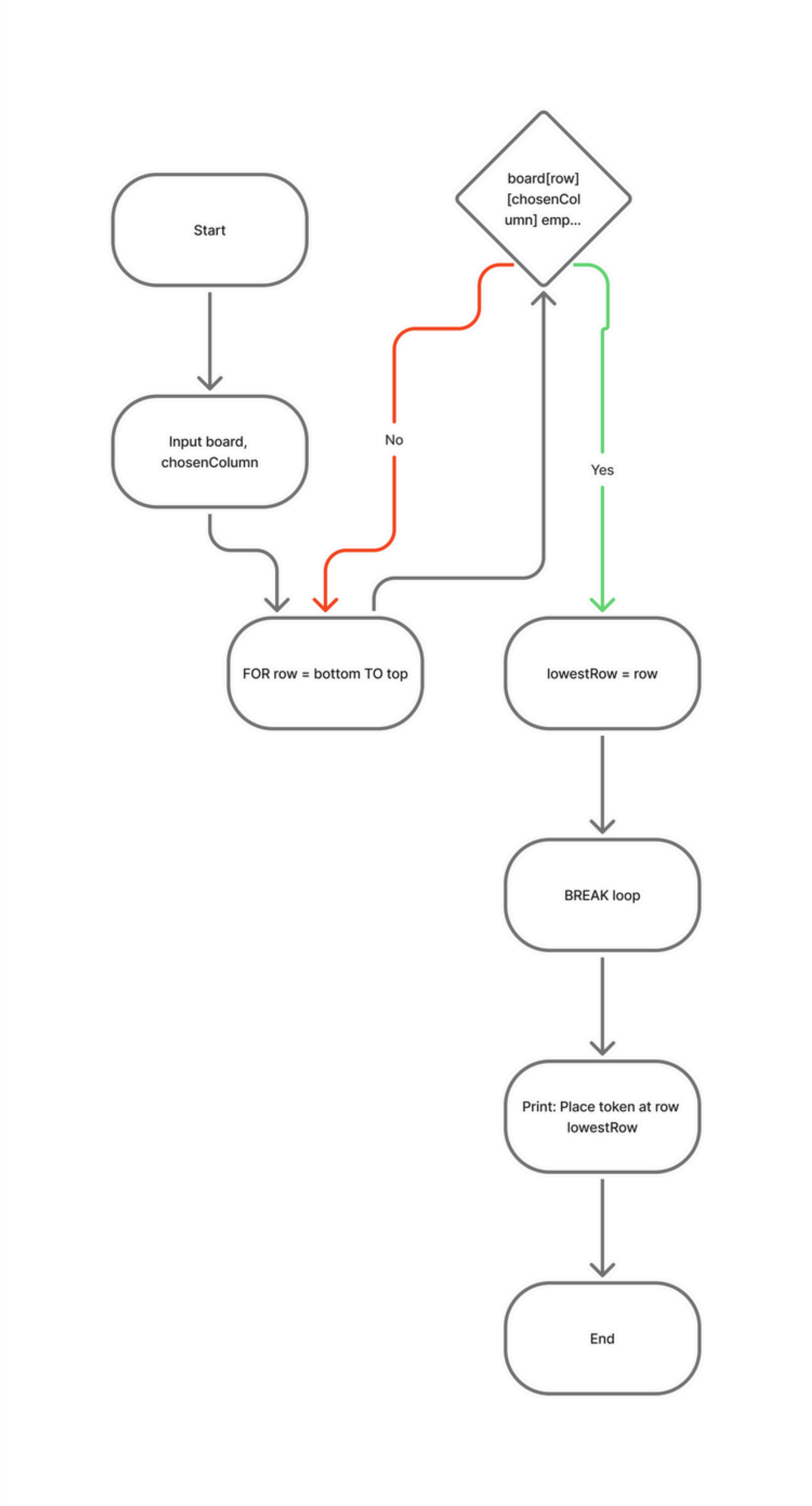
How Linear Search Works:

- Check row 5 → filled (X)
- Check row 4 → filled (O)
- Check row 3 → filled (X)
- Check row 2 → empty → drop token here

Column 3 = [" ", " ", " ", "X", "O", "X"] → fill the index 2

Searching Flowchart

https://intip.in/linearsearchflow



Tree

Why do we use trees?

A game of connect 4 will naturally branch into many possible futures.

Each new move

↓

Creates a new board state

↓

Which creates more possible moves

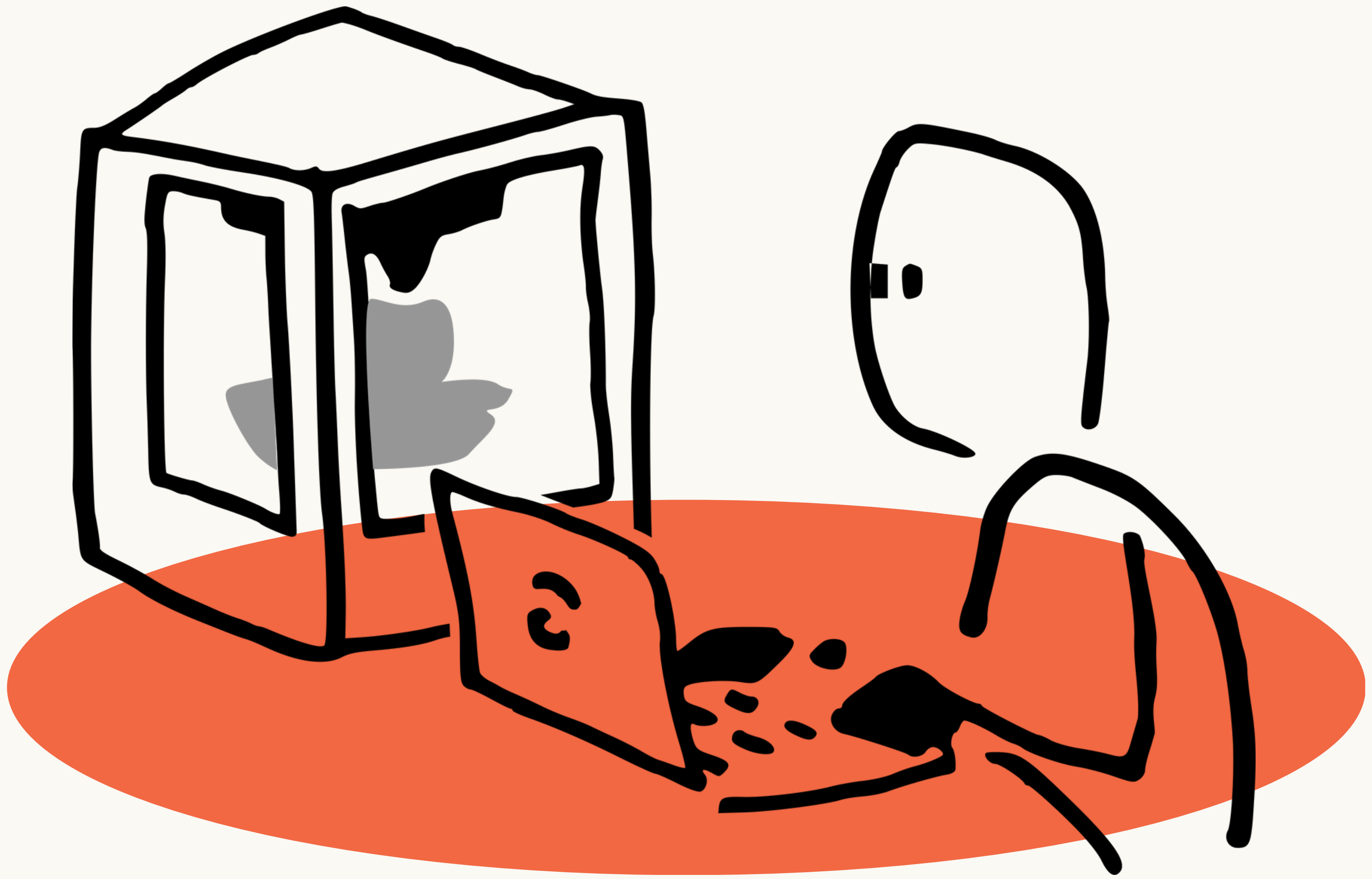
↓

Which creates even more board states

That is why we need a tree to help visualize future moves so the AI can decide what to do

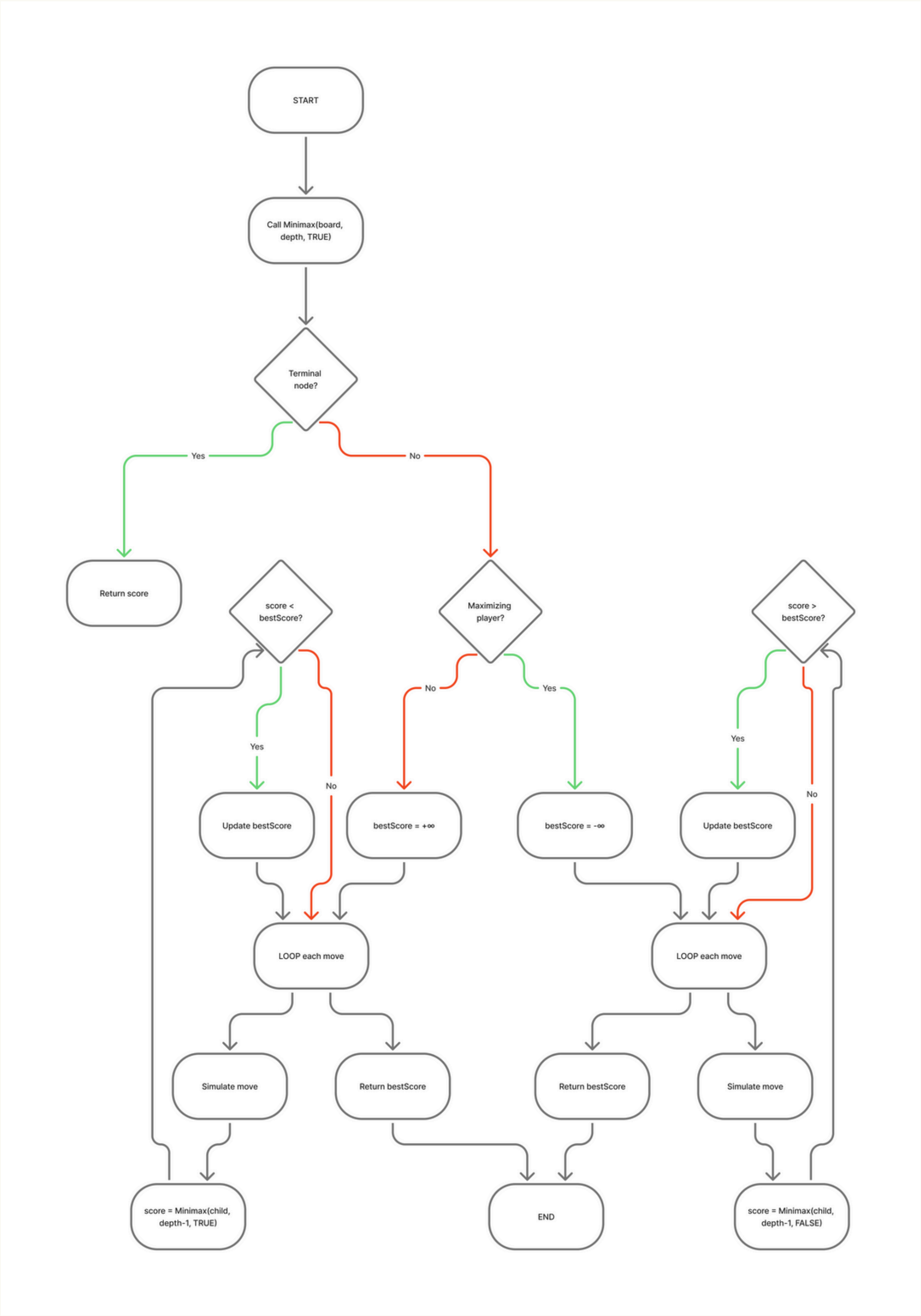
How do we use trees?

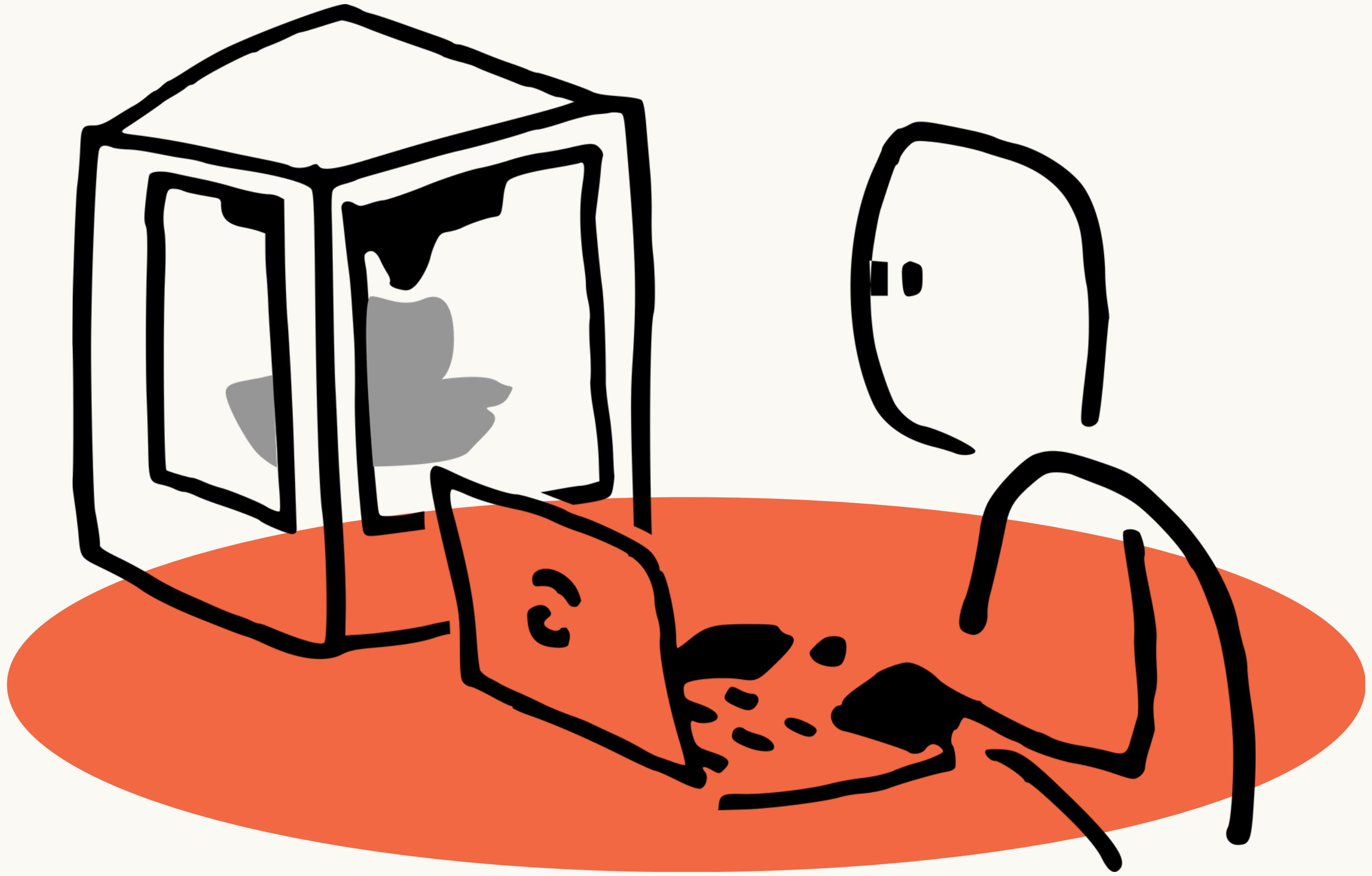
We will use post-order traversal to look at child nodes on the tree, which are possible future moves, and evaluate them using minimax to determine the most optimal moves to make



Tree

https://intip.in/minmaxflow





Sorting

Why Do We Sort Moves?

- Minimax will simulate many possible future boards.
- But not all moves are equally good, some are much more promising (moves that create a 4-in-a-row).
- AI first computes board strength scores for all possible moves, then sorts them from best to worst.

Example analogy:
Instead of checking random moves, the AI says:
"I will check the best moves first, because those are most likely to lead to a winning path."

Why Merge Sort

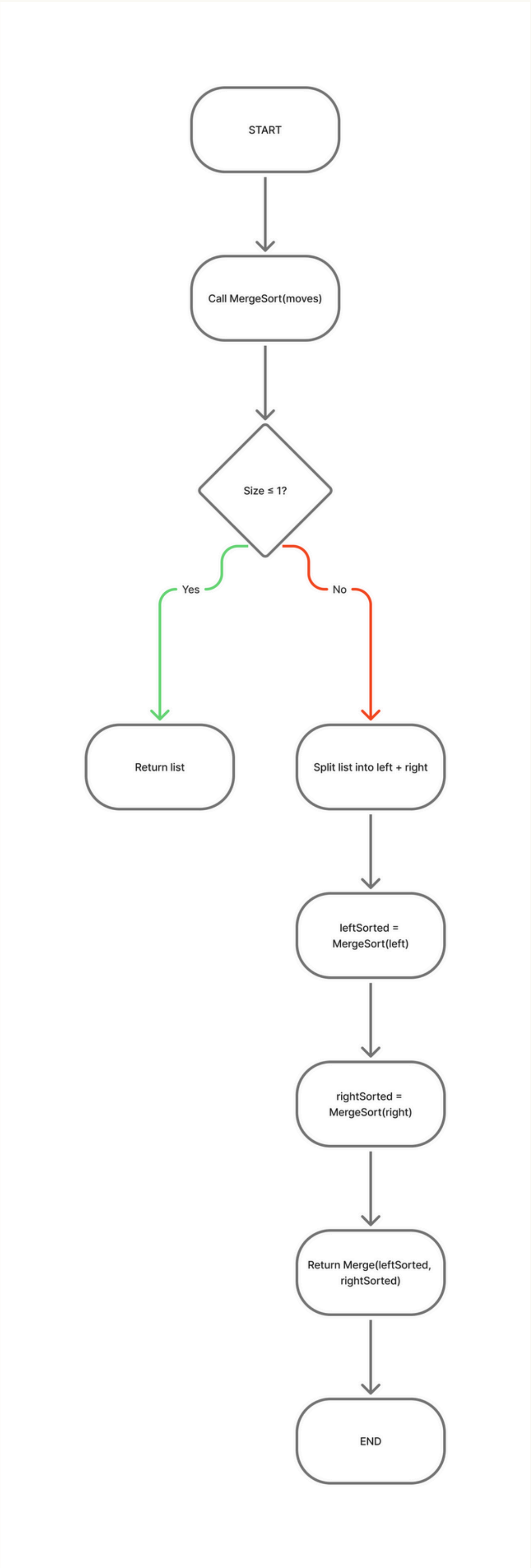
Merge Sort is ideal here because:

- Stable
- Always O(n log n)
- Handles small or large move lists efficiently
- Fits well with recursive minimax (same mental model: divide & conquer)

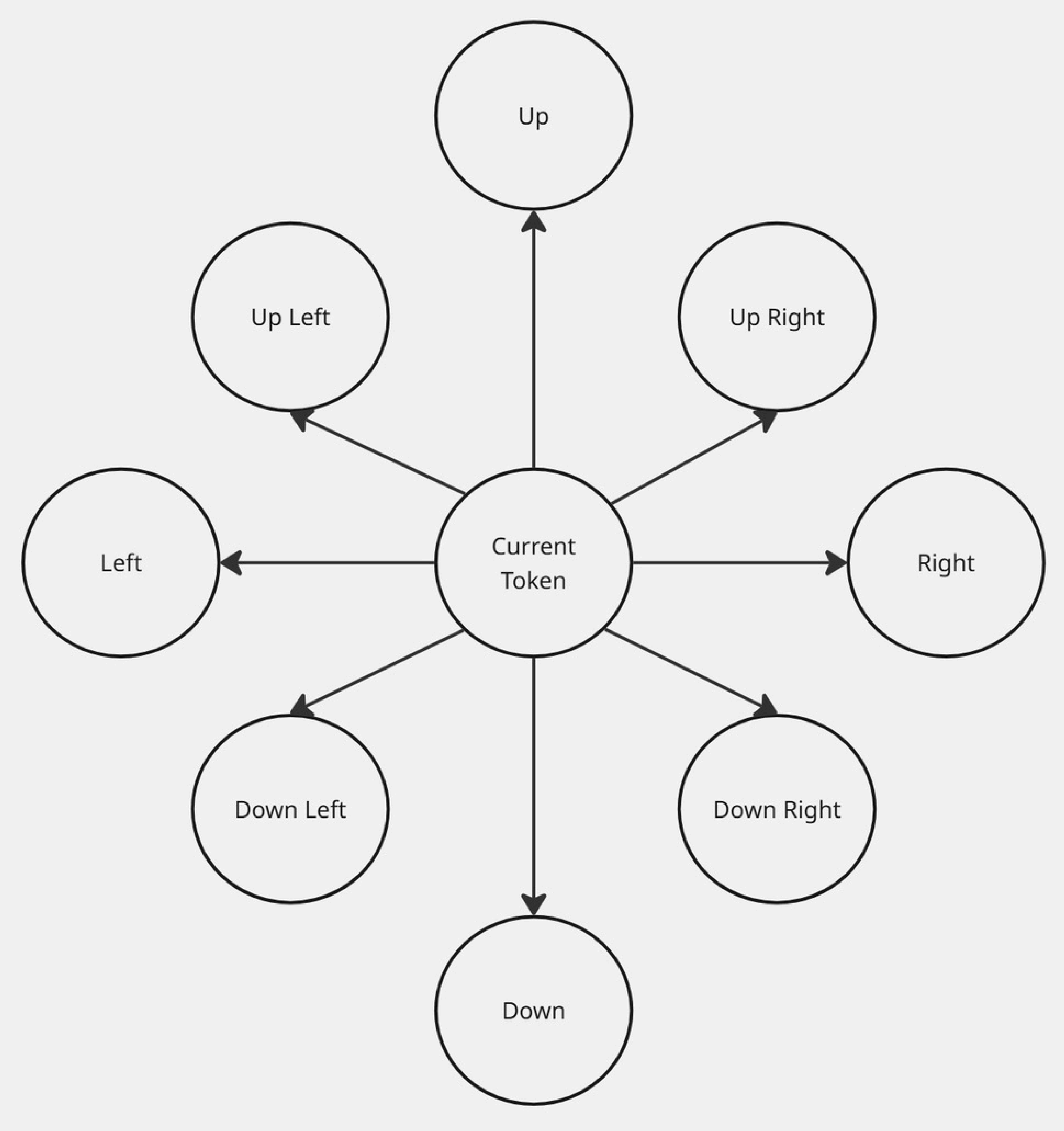
Even though Connect 4 has at most 7 columns, using Merge Sort demonstrates algorithmic understanding & OOP usage.

Sorting

https://intip.in/mergesortflow



BFS Graph



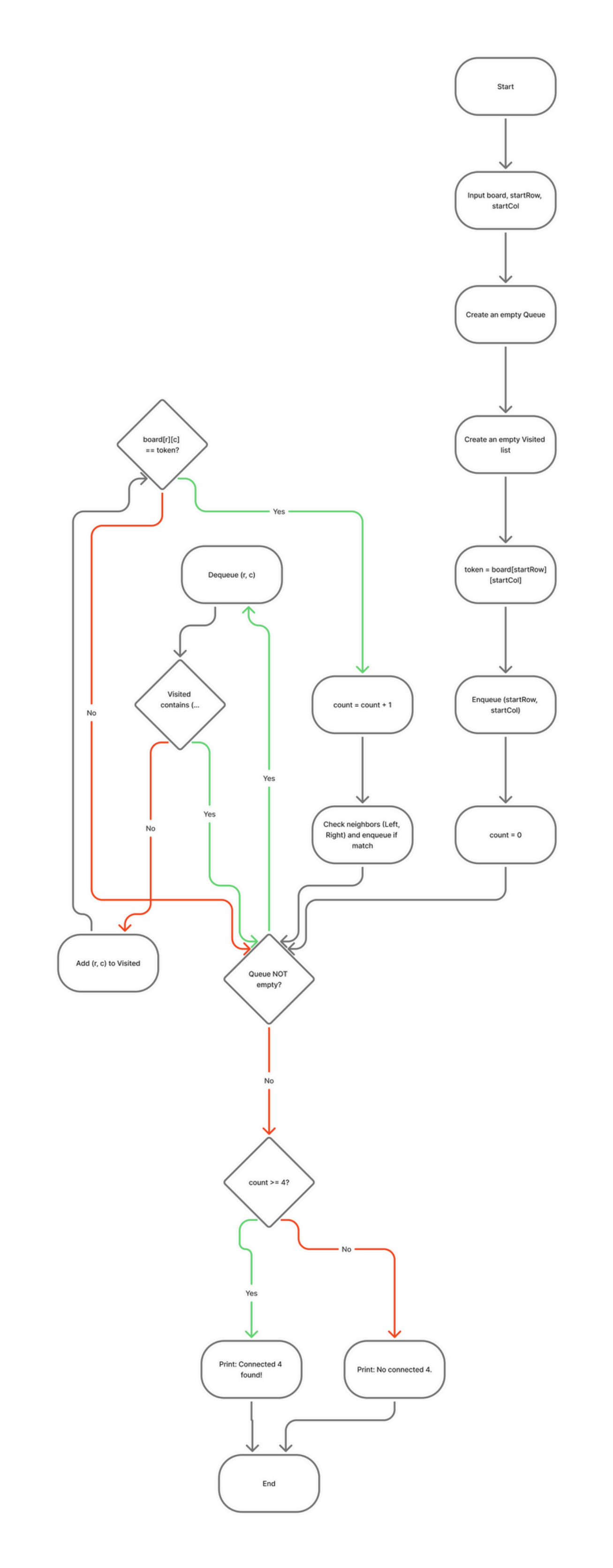
Checking token connectivity horizontally, vertically, and diagonally

Scanning 8 direction node

Example, your piece is placed at column 3 (from linear search)

BFS will check:
Start → col 3 (●)
Then its neighbors → col 2 and 4
Then their neighbors → col 1 and 5
(And so on until the connect 4 founded or all nodes explored)

BFS Flowchart



https://intip.in/bfsFlow

Algorithm Summary

1. Generate all valids move

(Linear Search finds lowest free row)

2. Score each move

(BFS evaluates board connectivity: chains, threats, wins)

3. Sort potential moves

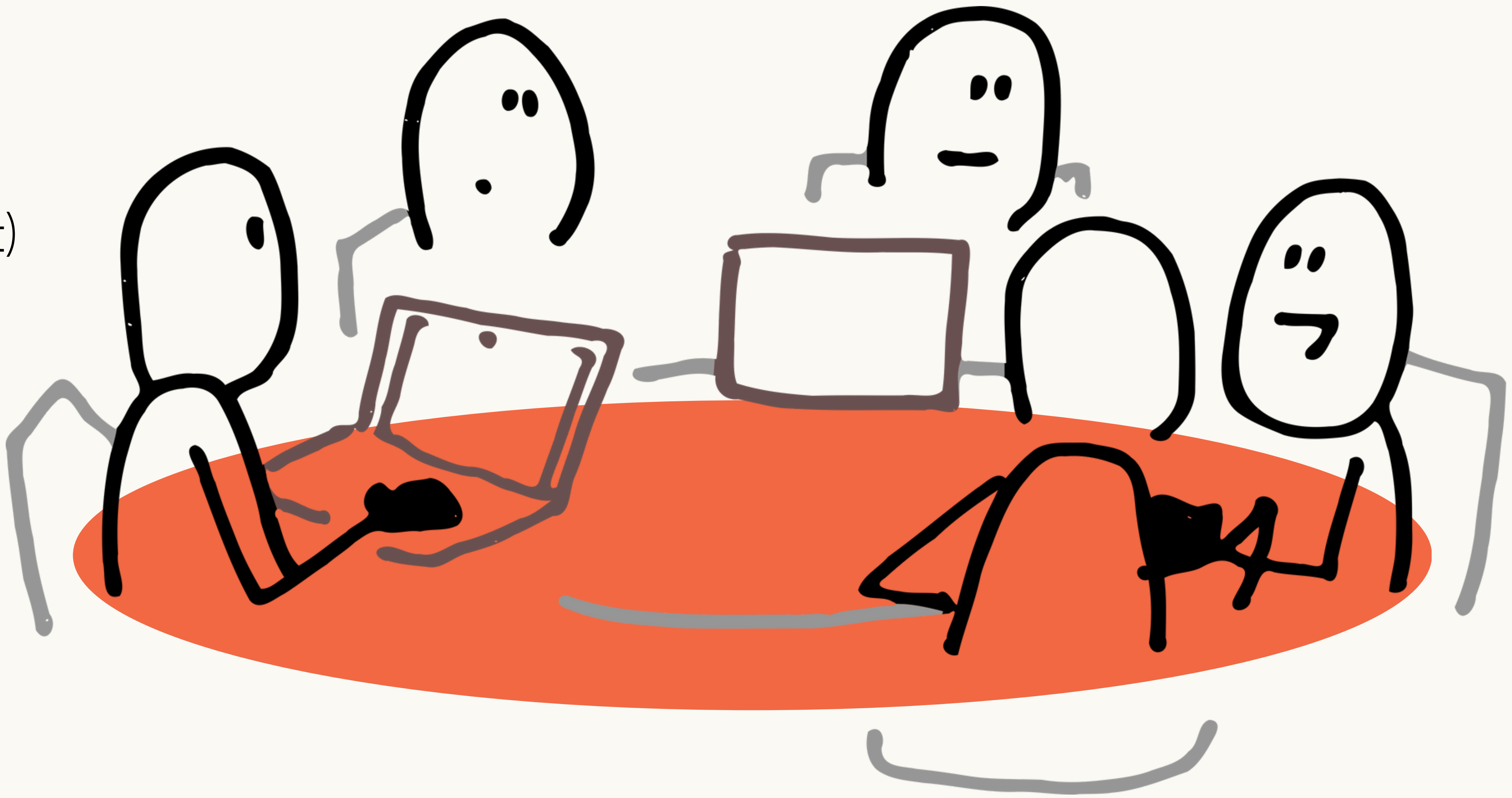
(Merge Sort ranks moves from strongest to weakest)

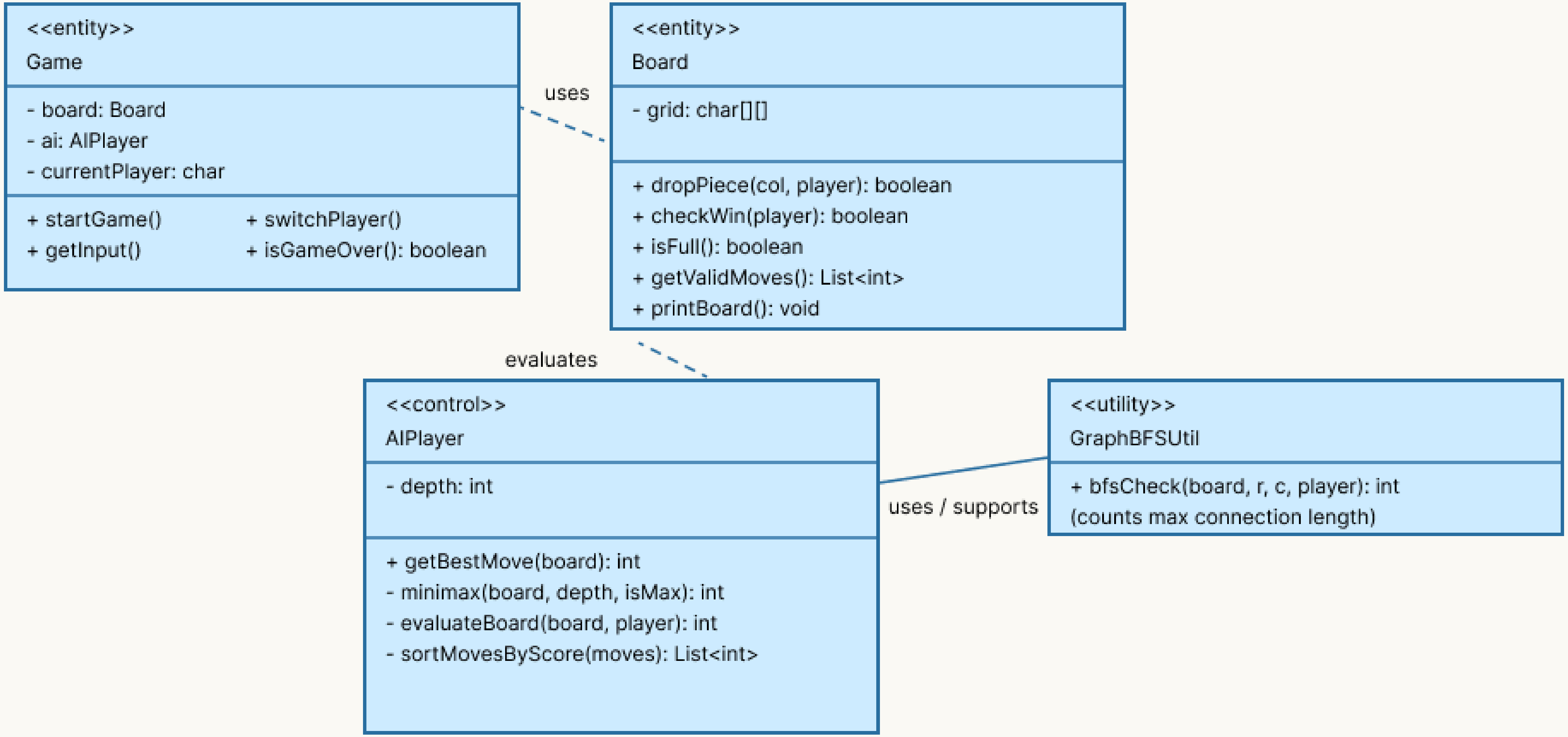
4. Build game tree

(Each move becomes a node; future moves become children)

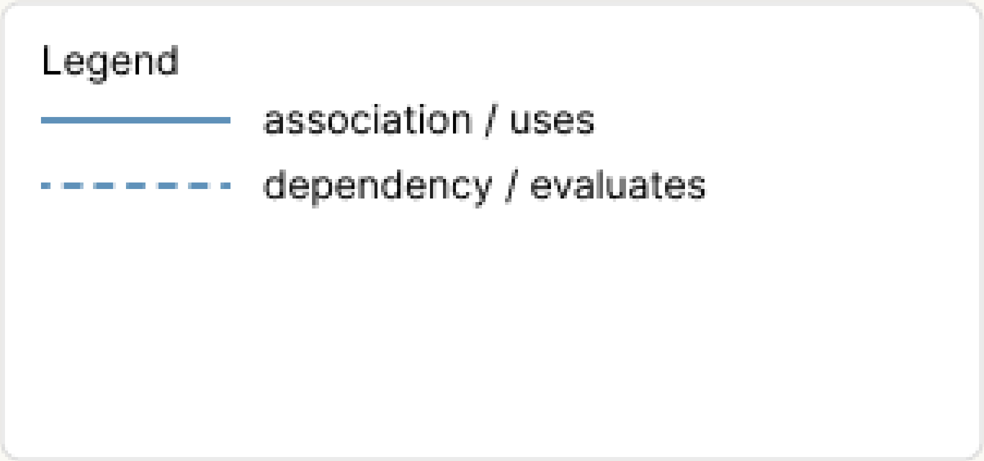
5. Run Minimax

(Post-order traversal evaluates outcomes and chooses optimal move)





UML OOP Diagram



Thank you