AI-m of the Game

Workshop #2: Minimax

Project Ignite Instructors:
Brandon Wang, Brandon Wei

Recap

What happened last time?

AI-m of the Game Overview

Students will implement a variety of AI algorithms to beat simple 2-player classical competitive games

- Examples: Tic-Tac-Toe, Connect Four, checkers, chess, Battleships, etc.
- We'll discuss (not implement) more complex methods (eg. neural nets, Q-learning)

Workshop Structure

Workshops #1-4

- Discuss multiple game structures and basic AI techniques
- Take-away game, Tic-Tac-Toe, Sim, and Connect Four
- Walk you step-by-step

Workshops #5-10

- Choose, build, then “solve” your own game!
  - should be hard eg. checkers, Go, chess
- Do your own research
- Advisors will only give guidance and deadlines

Key Links for this Course

- Meeting Minutes: https://tinyurl.com/y5e365wn
  - workshop schedule and notes
  - links to resources for learning and reviewing
  - let me know if you can't access and edit it
- GitHub: https://github.com/project-ignite-2021-ai
  - upload/download all of the code
- CMU Canvas
  - upload/download all of the non-code file (eg. slides)

Introduction to AI Games

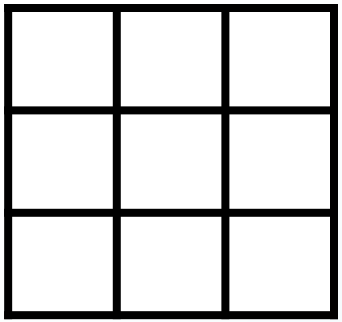
- What is AI?
- What are “games”? What games are there?
  - focus on 2-player sequential zero-sum fully-observable discrete deterministic games
- What game strategies are there?
  - During-Each-...-Wins: say ‘2’ so that opponent says ‘1’
  - Takeaway Game: working backwards, put the opponent in a losing state/position
  - Tic-Tac-Toe: ???

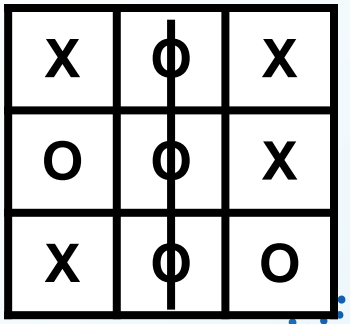
Tic-Tac-Toe

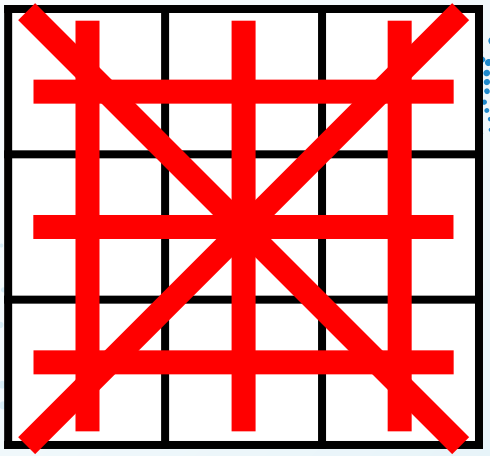
What are the basic rules and strategies?

Rules of Tic-Tac-Toe

- 3 × 3 grid of squares
- Turn player places their piece on an empty square
  - Player 1: O, Player 2: X
- The player who gets 3 of their pieces in a row, column, or diagonal wins!
- If all 9 squares of the board are filled, and no player has 3-in-a-row, it's a draw/tie

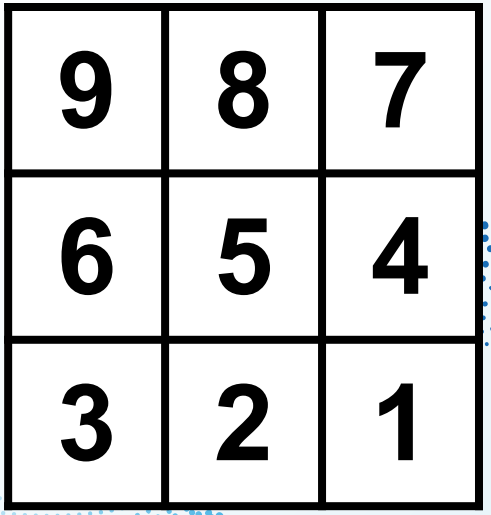






Tic-Tac-Toe Complexity

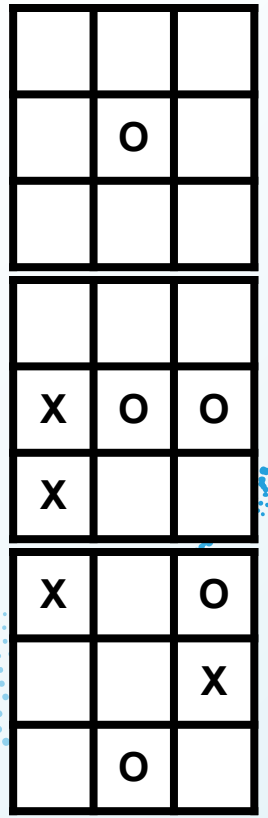
- Upper limit of games: 9! = 9*8*7*6*5*4*3*2*1 = 362,880
  - actually has 255,168 games
- Upper limit of states: 3^9 = 19,683
  - actually has 5,478 legal states
- Full game tree has 549,946 nodes
  - implies that on average, we visit each legal state over 100 times!



Typical Tic-Tac-Toe Strategy

1. start w/ piece in the center or corner
    a. these move cover the most (4/8 and 3/8) winning positions!
2. try to automatically win
3. block any of opponent's attempts to 3-in-a-row
4. "trap" your opponent by having 2 ways to make a 3-in-a-row
    a. opponent is helpless to stop it!

with perfect play, game is a ***tie***

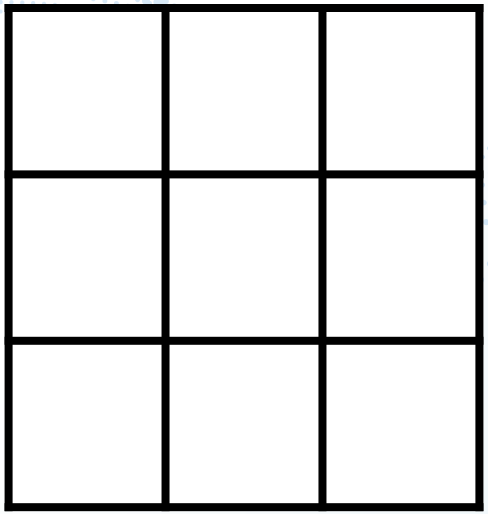


Other Tic-Tac-Toe Strategies

- Place randomly
- Place nearby your other pieces
- Place far from your other pieces
- Place randomly, but block opponent's 3-in-a-rows
- Block all of opponent's ways to get a 3-in-a-row
- Go for the fastest 3-in-a-row
- Go for "traps" (where you have multiple 3-in-a-rows after one move)

Optimal Strategy

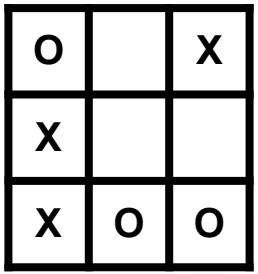
But what is the best, most perfect, unbeatable strategy to win Tic-Tac-Toe?



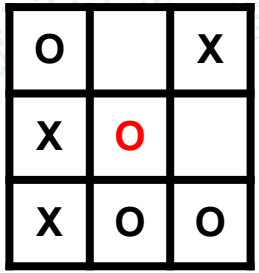
Minimax

Minimize the maximum, maximize the minimum

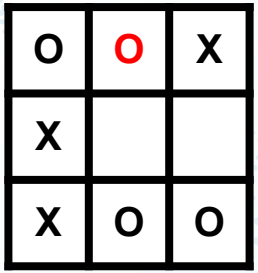
Example of a Game



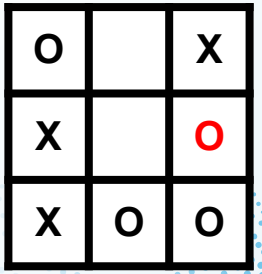
Option 1

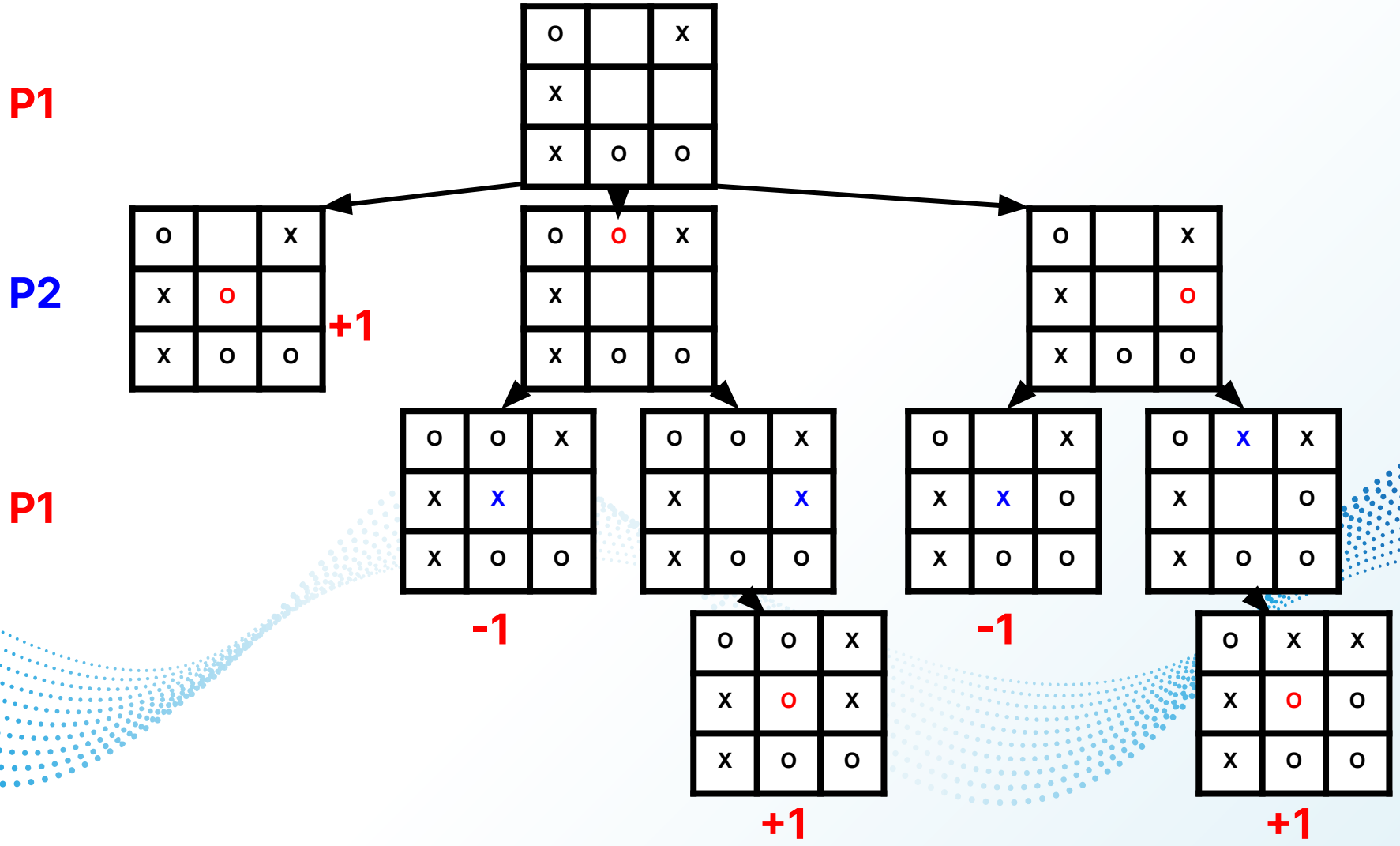


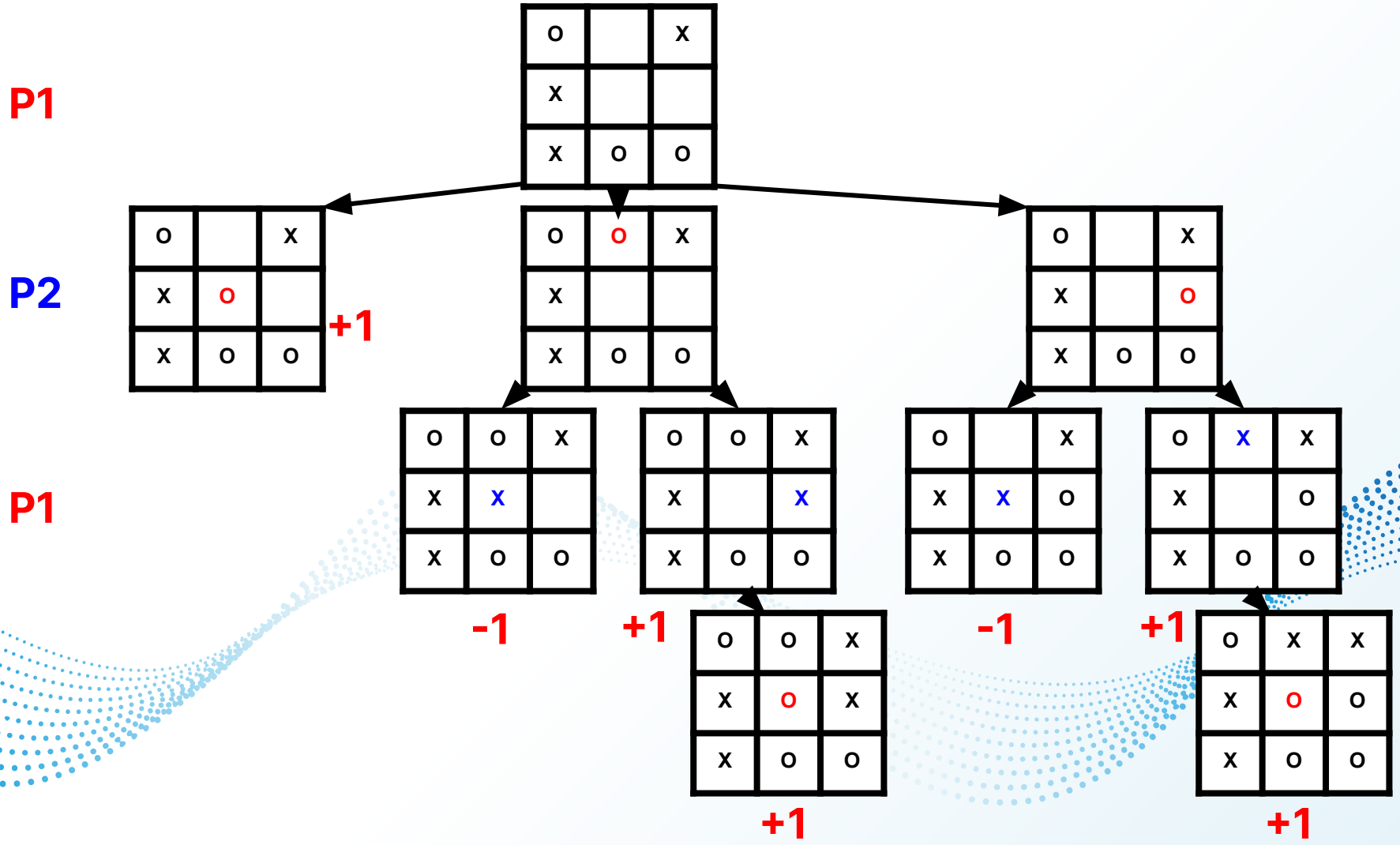
Option 2

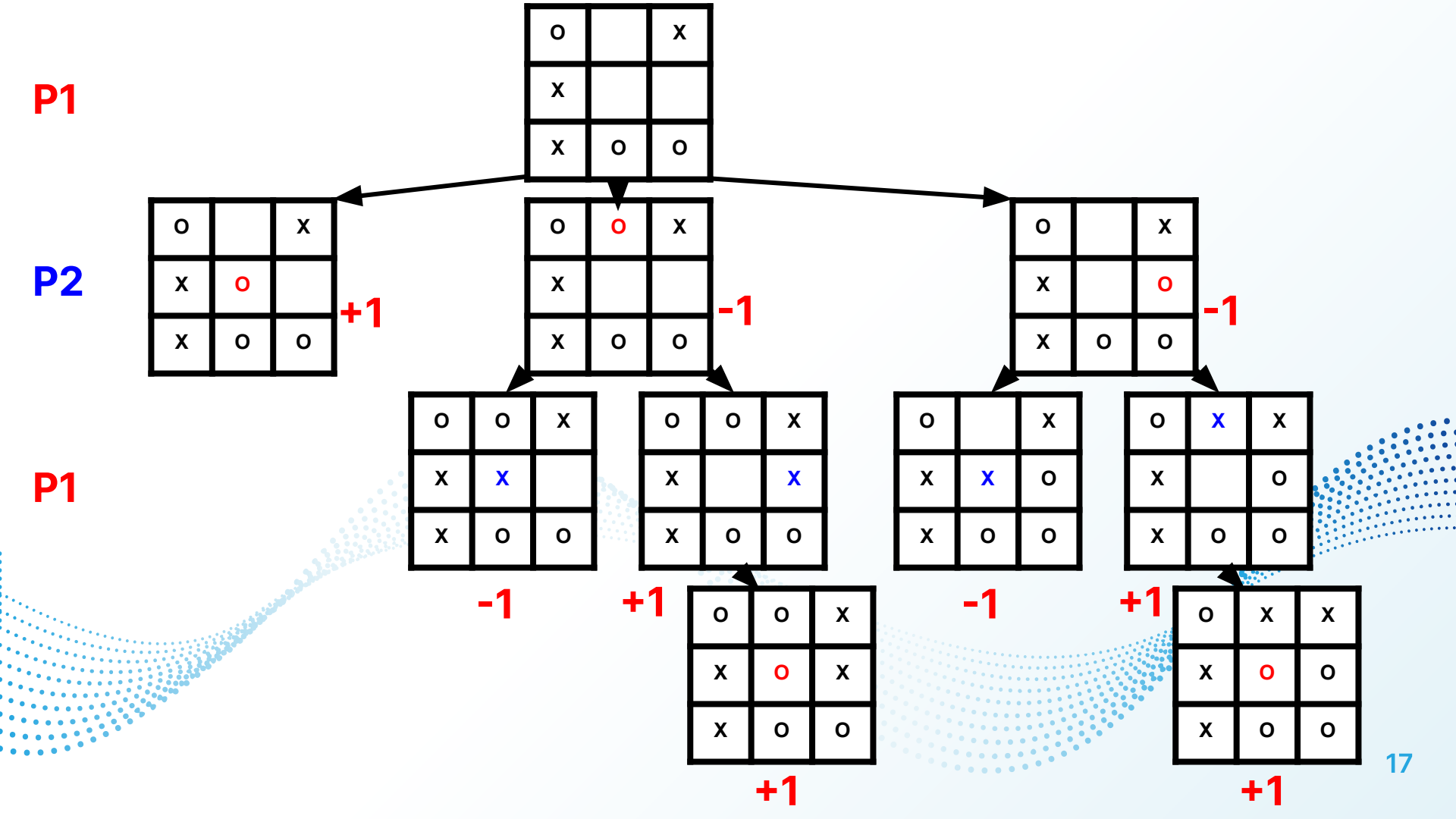


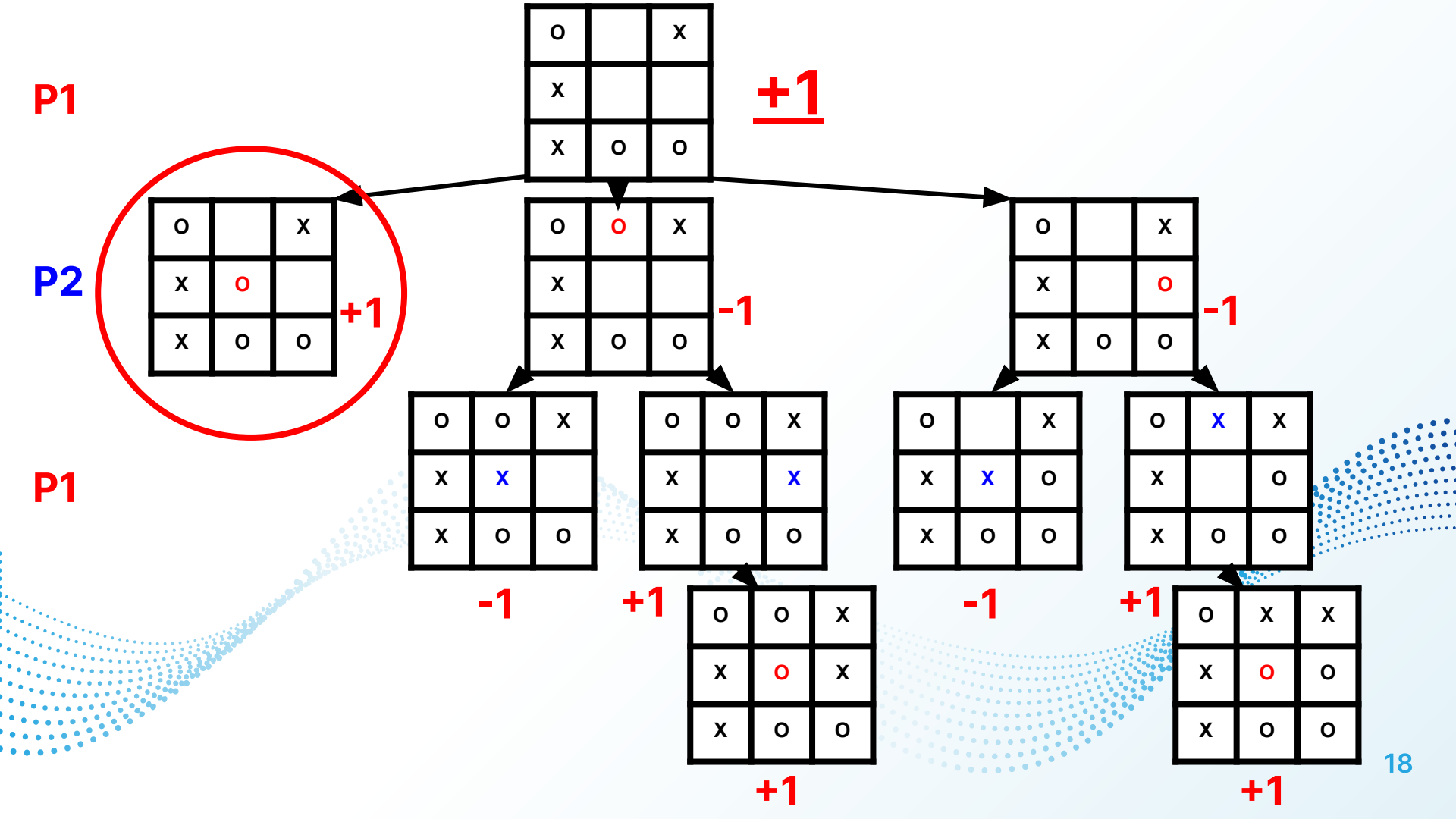
Option 3




P1
P2
P1
+1
-1
-1
+1
+1

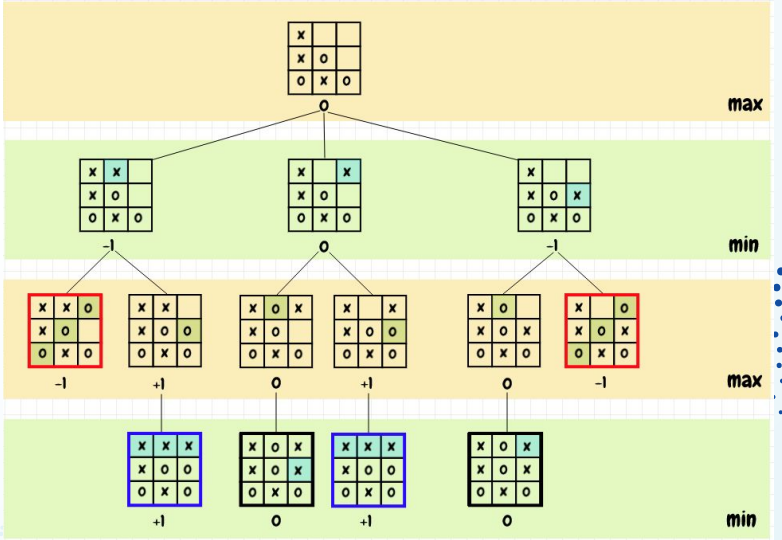

P1
P2
P1
+1
-1
+1
-1
+1
+1
+1


P1
P2
P1
+1
-1
-1
+1
-1
+1
+1
+1


P1
P2
P1
+1
-1
-1
+1
-1
+1
+1
+1

Minimax Algorithm

- you try to ***maximize*** your utility
- opponent tries to ***minimize*** your utility (ie. maximize his own)
- draw a full game tree, and evaluate every state based on how your opponent will minimize and how you'll maximize
  - for now, we only care about the ***end states***



Minimax Issues

- we're still searching through all 255,168 games and 549,946 tree nodes
  - if we find the best move early on, why do we need to search through all of the worse moves too?
  - why should we have to play through all 9 moves just to make the first move?
  - many nodes are repeated states (ie. same board), it's a waste to have to evaluate them again!

Solving these Issues

- if we find the best move early on, why do we need to search through all of the worse moves too?
  - prune the trees! (we'll talk about this right now!)
- why should we have to play through all 9 moves just to make the first move?
  - stop the minimax code at a certain depth
- many nodes are repeated states (ie. same board); it's a waste to have to evaluate them again!
  - store the states in a hashtable; we'll skip this for now

Alpha-Beta Pruning

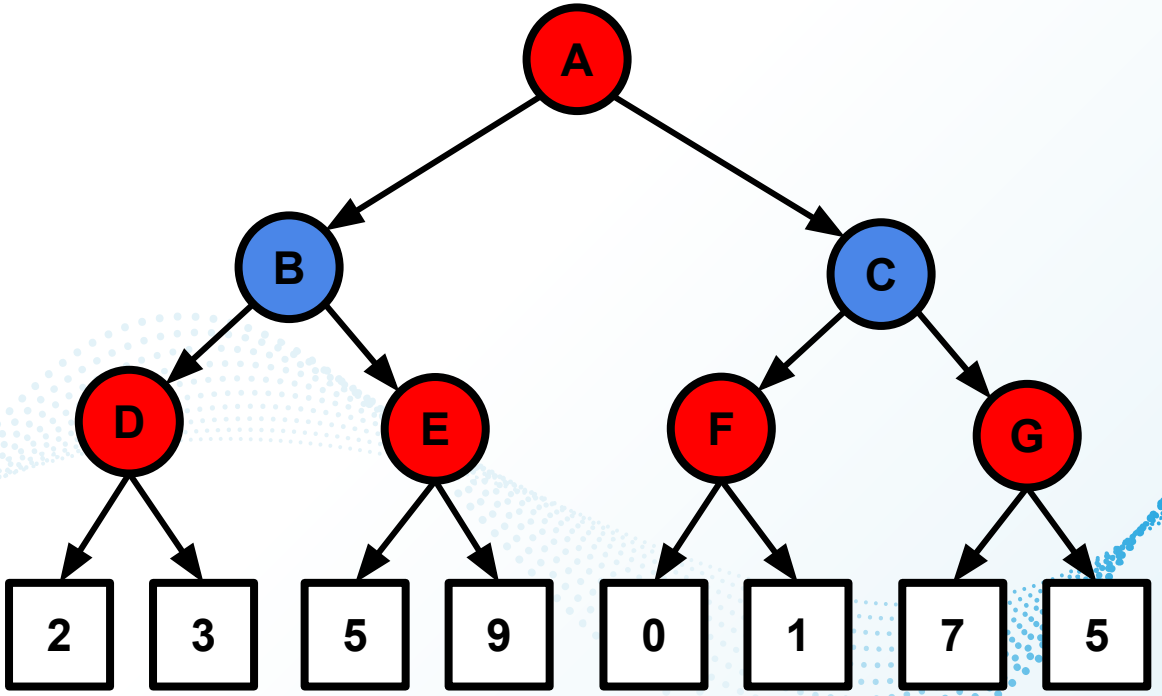
Cutting down the minimax search

Example of Game Tree

Player 1; MAX

Player 2; MIN

Player 1; MAX

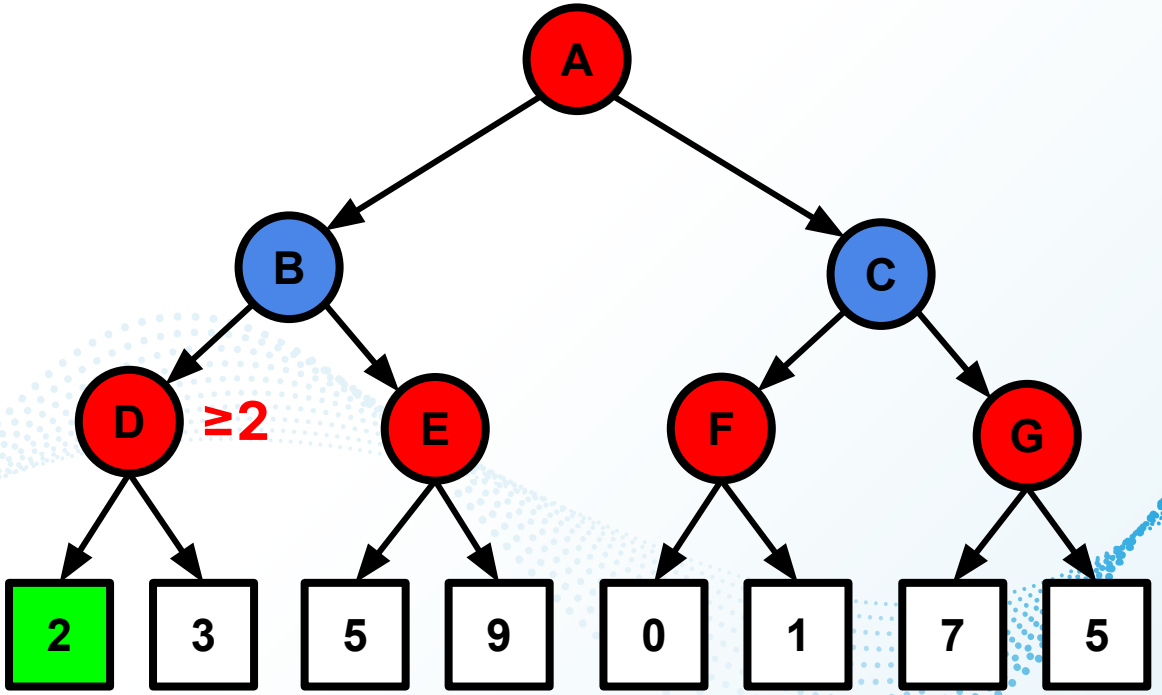


Example of Game Tree

Player 1; MAX

Player 2; MIN

Player 1; MAX

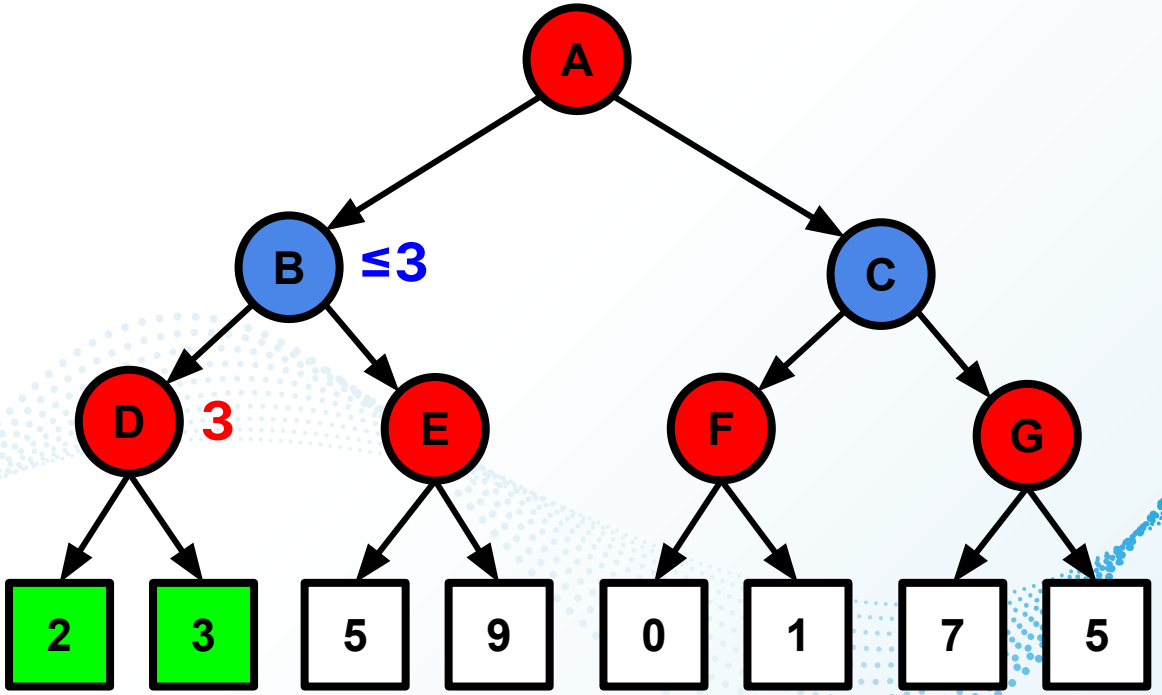


Example of Game Tree

Player 1; MAX

Player 2; MIN

Player 1; MAX

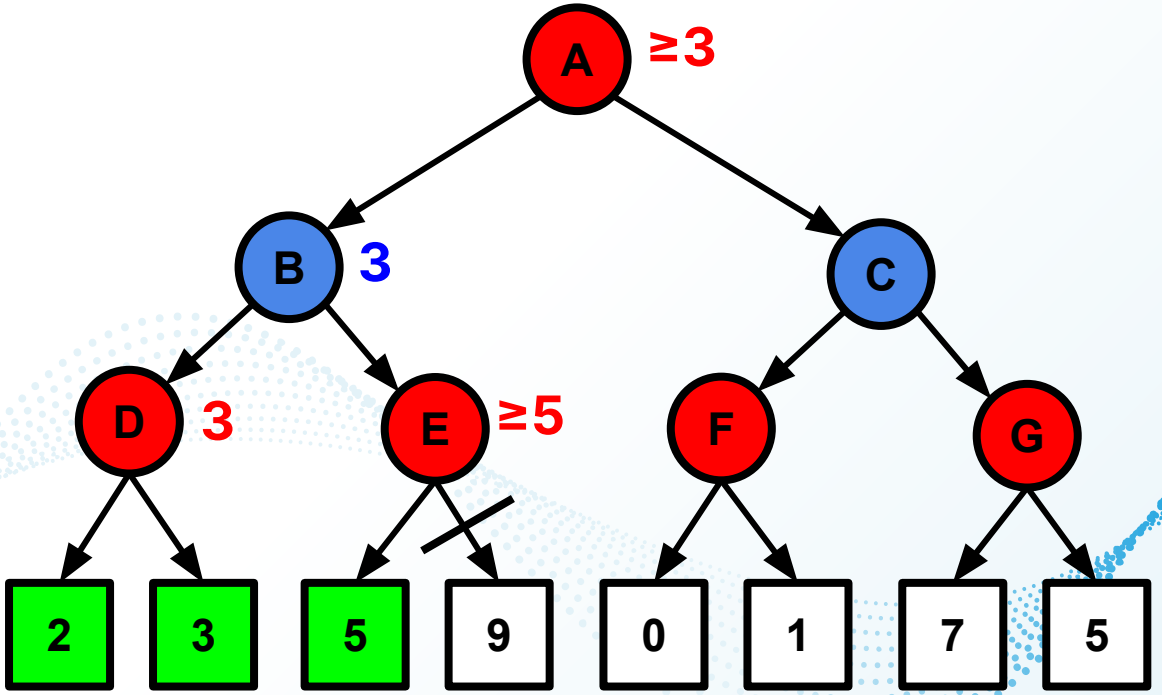


Example of Game Tree

Player 1; MAX

Player 2; MIN

Player 1; MAX

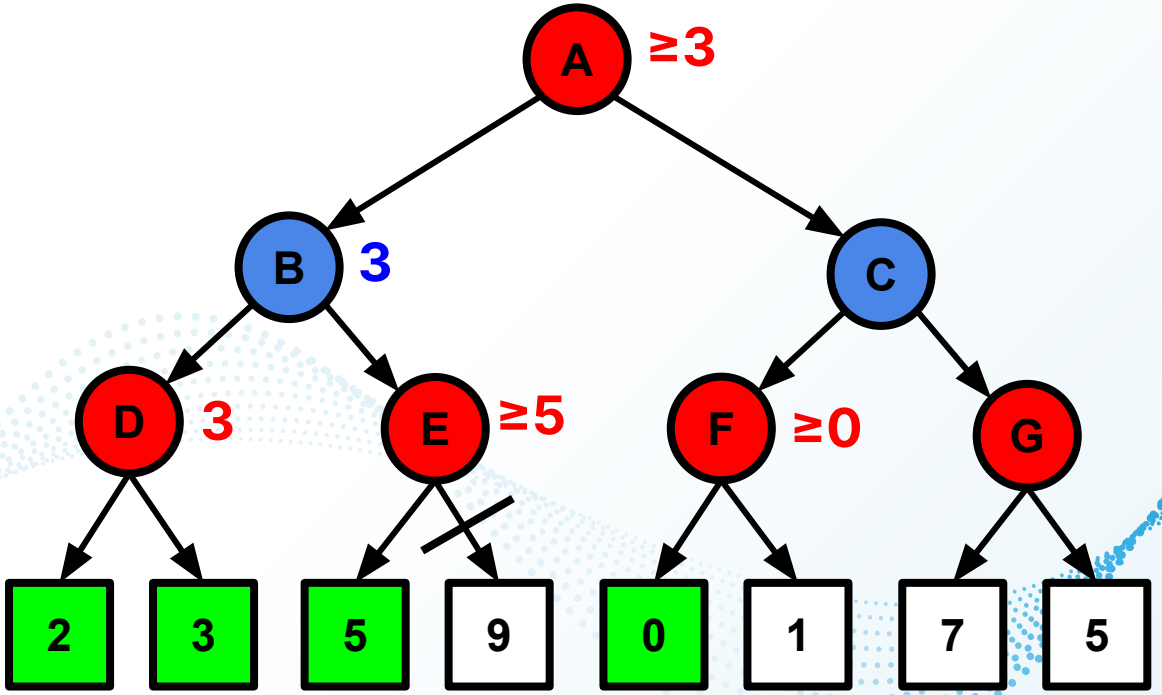


Example of Game Tree

Player 1; MAX

Player 2; MIN

Player 1; MAX

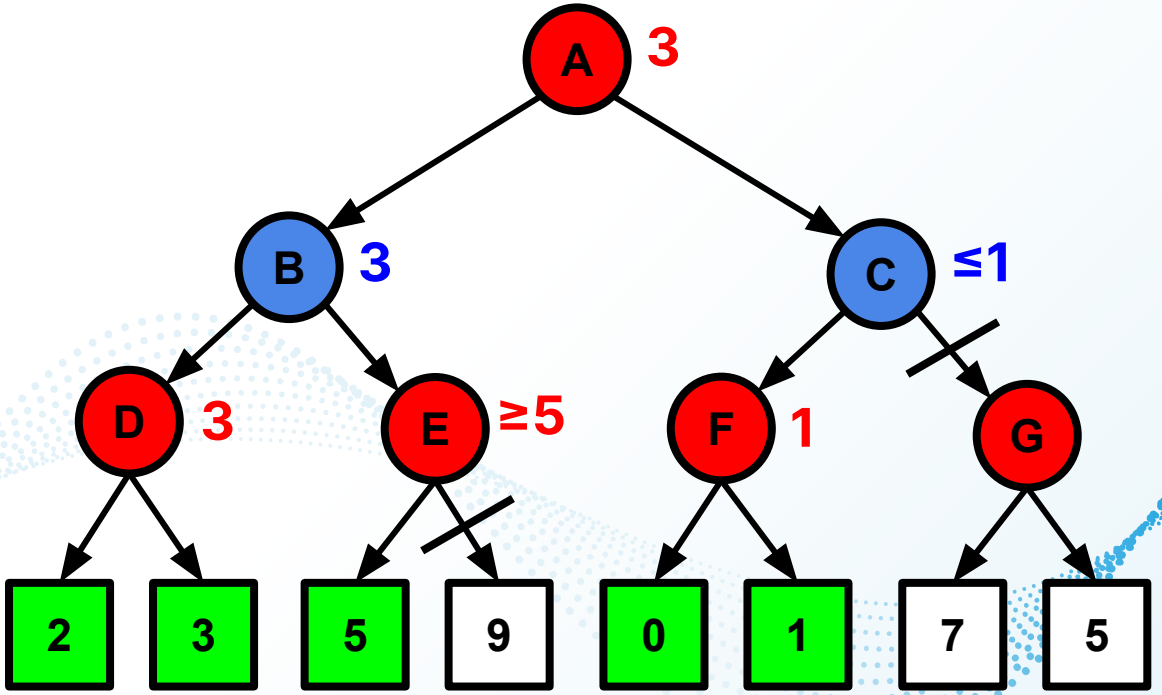


Example of Game Tree

Player 1; MAX

Player 2; MIN

Player 1; MAX

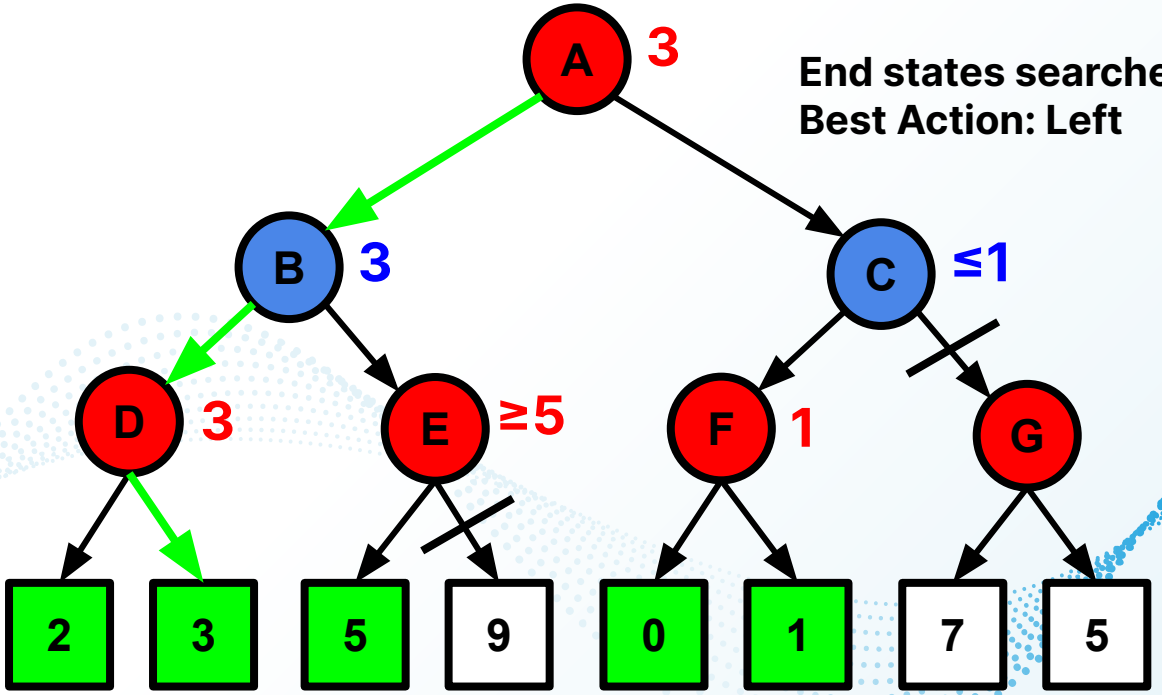


Example of Game Tree

Player 1; MAX

Player 2; MIN

Player 1; MAX

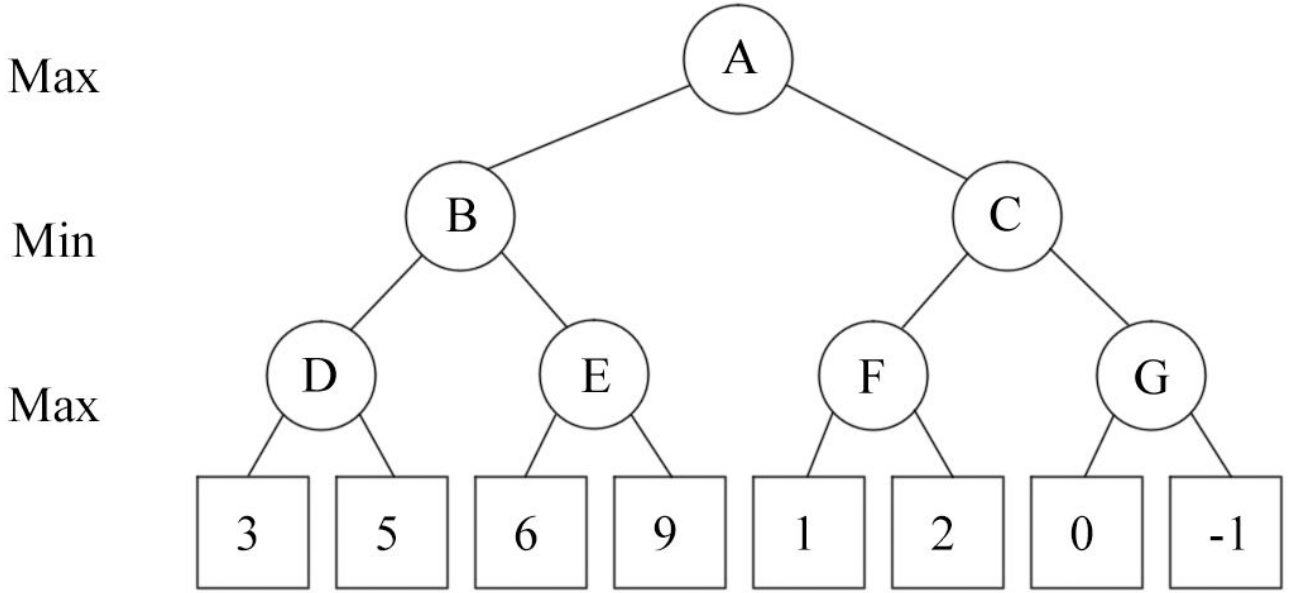


End states searched: 5/8
Best Action: Left

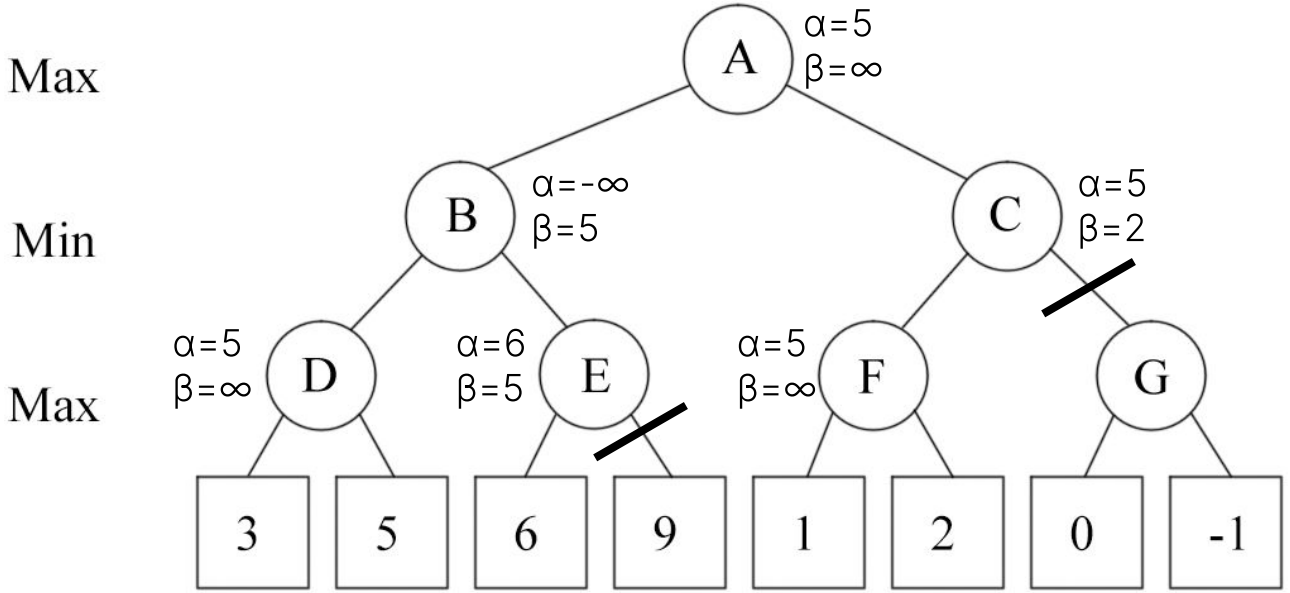
Alpha-Beta Pruning

- alpha (α) = best value for Player 1 (maximizer)
- beta (β) = best value for Player 2 (minimizer)
- alpha pruning: stop searching a MIN node if the new β value becomes ≤ the old α value
- beta pruning: stop searching a MAX node if the new α value becomes ≥ the old β value
- to see this exact alpha-beta pruning example again with α and β, go to this website

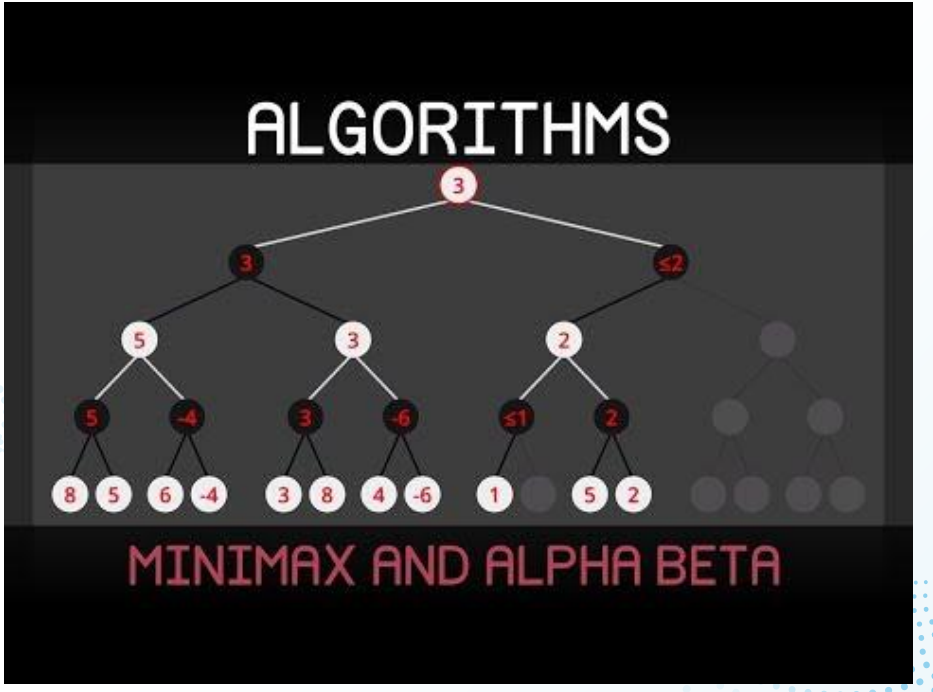
One More Example



One More Example



Minimax + Alpha-Beta Pruning



Tic-Tac-Toe in Python

Let’s use minimax and pruning on Tic-Tac-Toe!

Solve Tic-Tac-Toe

Download tictactoe.py from GitHub

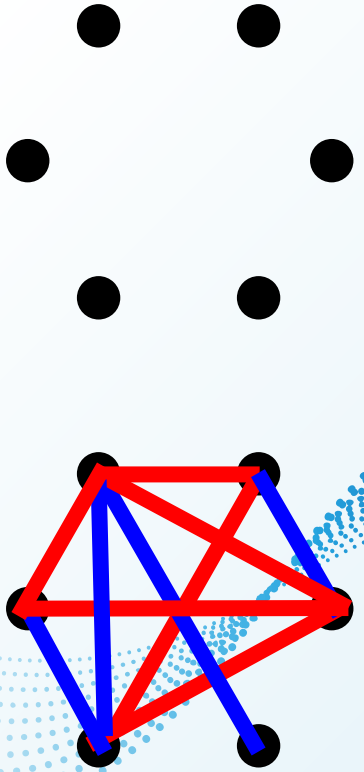
- Scan through the file to understand the structure
- Try playing it for yourself by running the code
- Complete the minimax() function
  - Use the YouTube video or the web as reference
- Complete the mmpruning() function
  - Use the YouTube video or the web as reference
- Test them with Player1Strategy and/or Player2Strategy

Sim (pencil game)

How can 6 dots be so complicated?

Rules of Sim

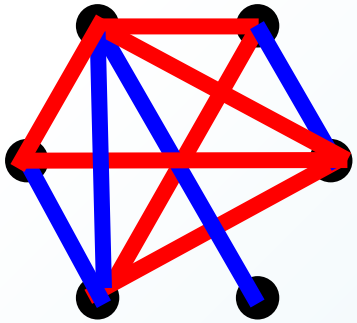
- 6 nonlinear dots (ie. a hexagon)
- Turn player draws a new line b/w any 2 points
  - Player 1: red, Player 2: blue
- The player who creates any triangle from 3 of their edges ***loses***!
- Mathematically (by Ramsey theory), there are no draws/ties



Let’s Play Sim!

Sim vs Tic-Tac-Toe

- Game ends with 3-in-a-row
- In Sim, 3-in-a-row = you lose!
- In Sim, there are no ties
- Upper limit of Sim games: 15! = 1.31E12 (over 1 trillion!)
  - more moves, higher depth
- Similar to a larger Tic-Tac-Toe game where the goal is to lose



		O	X
O	X	X	O
		O	
	O	X	

Solve Sim

Download sim.py from GitHub

- Scan through the file to understand the structure
- Try playing it for yourself by running the code
- Complete the mmpruning() function
  - Reference tictactoe.py or the rest of the sim.py
  - See negamax(), which is logically the same as the mmpruning() (but less code)
- Test them with Player1Strategy and/or Player2Strategy

To-do

Things to do by the next workshop:

- Complete tictactoe.py, or look at our sol'n on GitHub
- Complete sim.py, or look at our sol'n
- Get comfortable with minimax and alpha-beta pruning; they're the basis of all AI games!

Thanks for coming!