CSE 332

Data Abstractions

P3

Synchronization

- P3 out today!
- Make your groups today!
- Decide on several weekly meeting times!
- The exercises due today will help you with P3!

Tic-Tac-Toe

O	X	O
	X	
X	O	

No matter what happens at this point, it's a draw.

Solving Tic-Tac-Toe


```
// Let's assume I'm X
win(Board b) {
   if (O can win on the next move) {
      block it
   }
   else if (the center square is open) {
      take it
   }
   else if (a corner square is open) {
      take it
   }
   else if (...) {
      ...
   }
}
```

Do We Really Want To Do This?

- Difficult to code
- Different for every game
- How do we even know we're right?
- **Way** too much thinking–that's what computers are for!

Recursion To The Rescue


```
boolean win(Board b) {
   if (b.threeXs()) {
      return true;
   }
   else {
      for (Move m : every possible move) {
         if (win(b.do(move))) {
            return true;
         }
      }
      return false;
   }
```

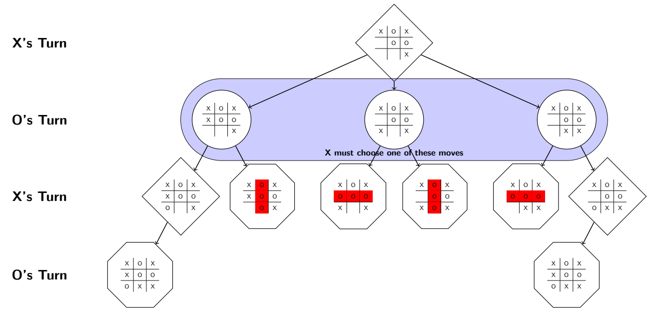
There's An Issue Here!

- When we make a move, it's not our turn any more.
- So the recursive call should be to **our opponent's option**
- Key Insight: Instead of guessing what the opponent is going to do, **assume she plays optimally**!

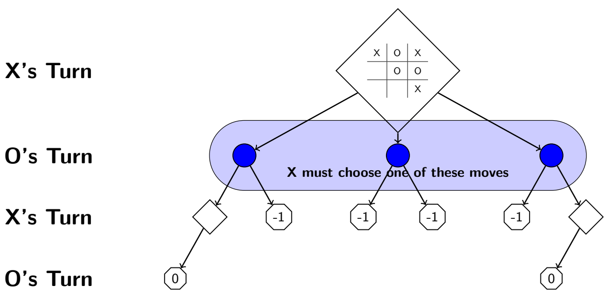
win:lose::me:you


```
// +1 is a win; +0 is a draw; -1 is a loss
int eval(Board b) {
   if (b.gameOver()) {
      if (b.hasThree(me)) {
         return 1;
      }
      else if (b.hasThree(them)) {
         return -1;
      }
      else {
         return 0;
      }
   }
   else {
      int best = -1;
      for (Move m : every possible move) {
         best = max(best, -eval(b.apply(move)));
      }
      return best;
   }
```

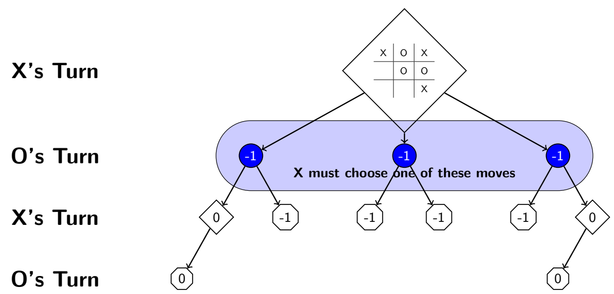
A Game of Tic-Tac-Toe



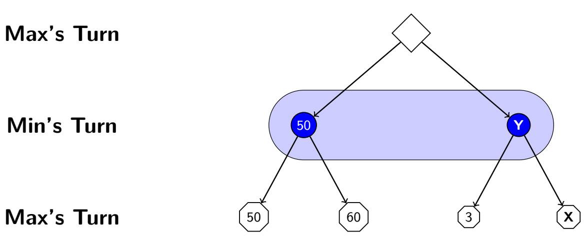
A Game of Tic-Tac-Toe: Filling in the Game Tree



A Game of Tic-Tac-Toe: Filling in the Game Tree



An Idea!

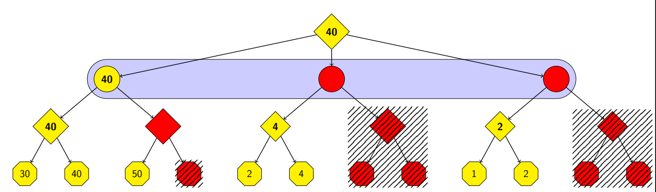


To fill in Y, **MIN** will take $\min(3, X)$. So, there are two cases:

- $4 = X > 3$. Then, $Y = \min(3,4) = 3$. So, the box is 50.
- $2 = X < 3$. Then, $Y = \min(3,2) = 2$. So, the box is 50.

The values of X and Y don't matter! Don't calculate them!

Pruning



Do we check the next node?
We currently have no information. So, yes!
Do we check the next node?
The current bounds are $[-\infty, 40]$. So, we **might** do better!
Do we check the next node?
Max will choose $x \geq 50$ which is already worse than the 40.
The current bounds are $[50, 40]$. Don't bother.
Do we check the next node?
Min will choose $x \leq 4$ which is already worse than the 40.
The current bounds are $[40, 4]$. Don't bother.

The algorithm we just ran is called **AlphaBeta**.

Parallel Searching

P3 combines **graph algorithms** (more on this later) with **parallelism**.

You will implement four algorithms:

- Minimax (the first one we discussed)
- Parallel Minimax
- Alpha-Beta Pruning (the second one we discussed)
- Jamboree (a parallel alpha-beta)

Each of these four algorithms has their own wrinkles. Each builds on the last.

Game Trees & Ply

A **branching factor** is how many times a node splits at each level. In chess, for a random position, the average branching factor is:

35

The average chess game lasts about

40 Moves

If we wanted to evaluate the whole game, we would be evaluating 35^{40} **leaves**. If we were able to evaluate **1 trillion** leaves a second, we would need 10^{48} seconds.

End Game

In addition to writing these bots, you'll get to watch them play.

A demo is worth 1000 words.