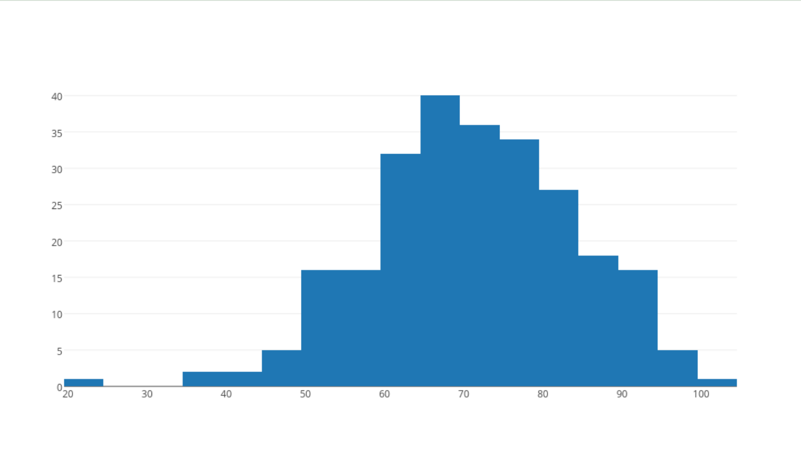
CSE 332

Data Structures and Parallelism

P3

Let's spend some time de-briefing.

What Does My Score Mean?



The "curve" was chosen such that your percentage accurately reflects what percentage of the material thus far you understand relatively deeply.

Midterm Exam (Continued.)

What Was Each Question Testing?

(1) Program analysis (understanding of asymptotics, closed forms, bounds, master theorem, data structure analysis)

(2) Dictionary vs. Set, `compareTo` of `CAFIFOQueue`, AVL rotations, understanding of why AVL trees work/lecture proofs

(3) 311 proofs, reading feedback on exercises, understanding what makes a proof good or bad, abstract discussion of definitions

(4) Mechanical running of heap algorithms, understanding more advanced manipulations of heaps

(5) Mechanical insertion into hash tables, what makes a hash function good?

(6) Understanding of how to approach recurrences, tree method

(7) Ability to implement a new Dictionary-like data structure, evaluation of various implementations, attempting to access the reverse mapping in a Dictionary, understanding of BoundedMap vs. BoundedSet vs. HashTable

(8) Ability to communicate algorithms to others, problem solving in the abstract, ability to use an old data structure in a new situation

Let's spend some time de-briefing.

What To Do Now?

A Bad Option: Throw the exam in the trash/in a drawer and never look at it again

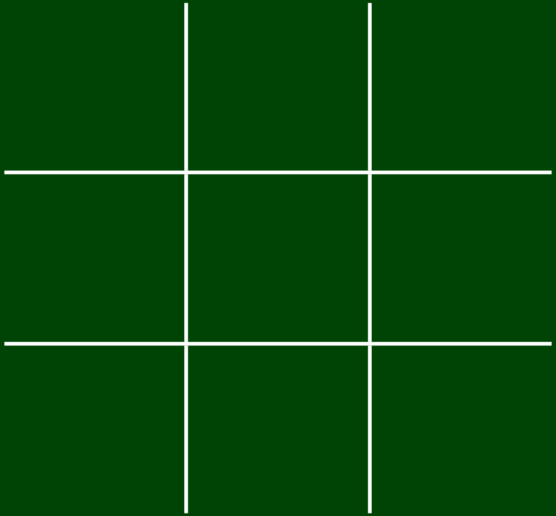
Some Better Options

- Meet with Adam to discuss the course, the exam, life, whatever. (Just please not tomorrow.)
  - `http://meeting.countablethoughts.com`
- Figure out what (if anything) went wrong:
  - insufficient/inactive/inefficient studying?
  - studied "to the test/practice exam"?
  - test anxiety?
  - ...
- Make sure to **work with** your partner for P3.

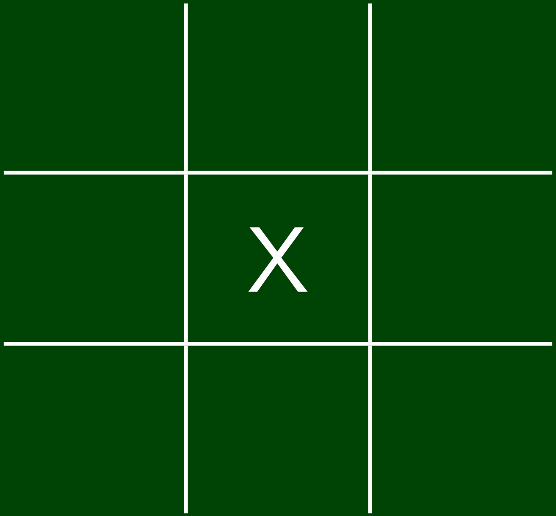
Synchronization

- P3 out tonight!
- Make your groups tomorrow!
- Decide on several weekly meeting times!

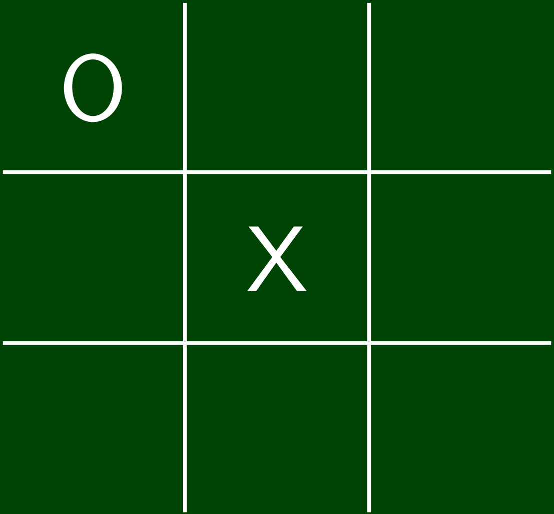
Tic-Tac-Toe

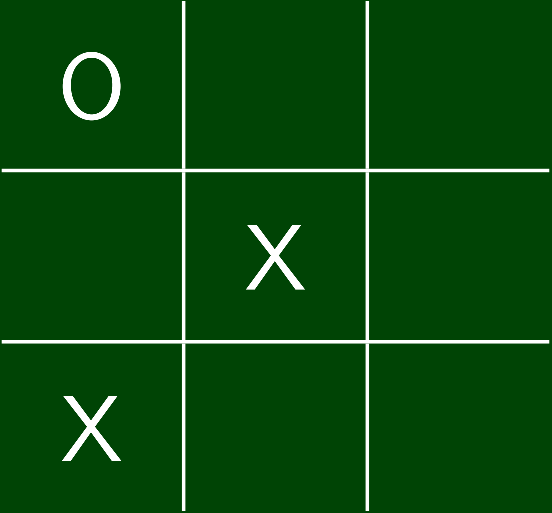


Tic-Tac-Toe



Tic-Tac-Toe




O
X
X

Tic-Tac-Toe

O		O
	X	
X		

Tic-Tac-Toe

O	X	O
	X	
X		

O	X	O
	X	
X	O	

O	X	O
	X	
X	O	

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

```
// 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!

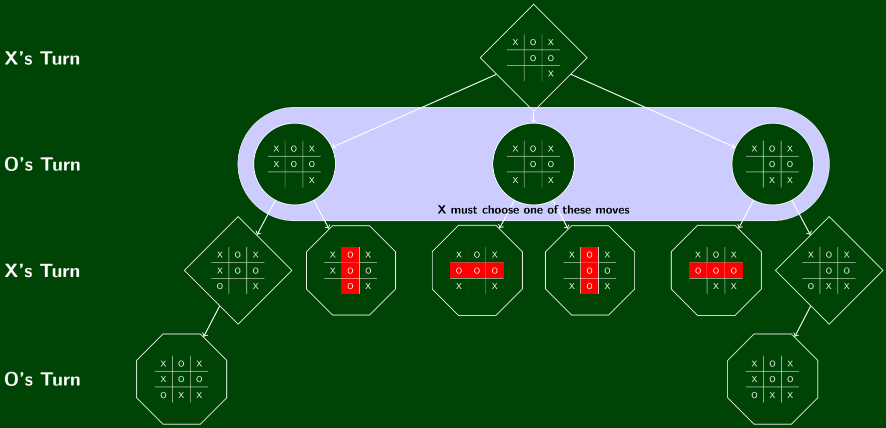
Recursion To The Rescue


```
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**!

```
// +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;
   }
```



X's Turn
O's Turn
X's Turn
O's Turn
X must choose one of these moves

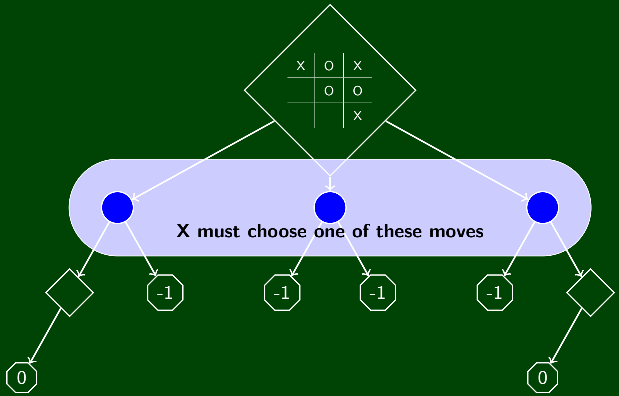
A Game of Tic-Tac-Toe: Filling in the Game Tree

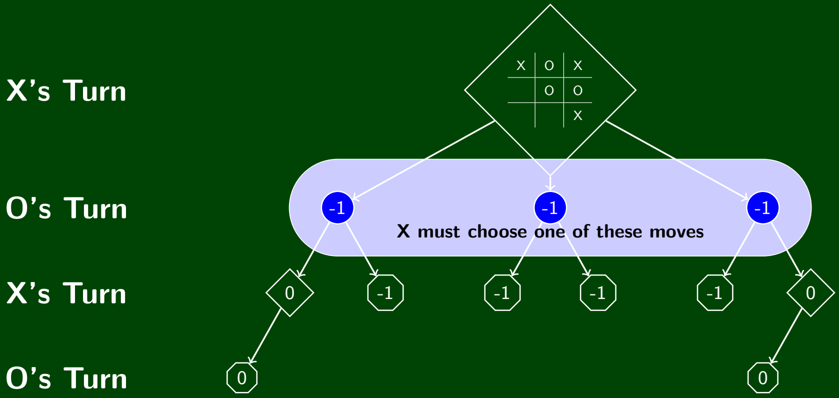
X's Turn

O's Turn

X's Turn

O's Turn



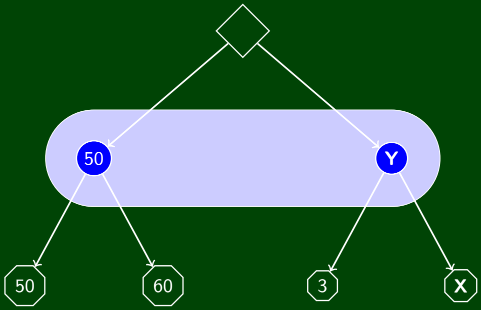

X's Turn
O's Turn
X's Turn
O's Turn
X O X
O O
X
-1
-1
-1
X must choose one of these moves
0
-1
-1
-1
-1
0
0
0

An Idea!

Max's Turn

Min's Turn

Max's Turn

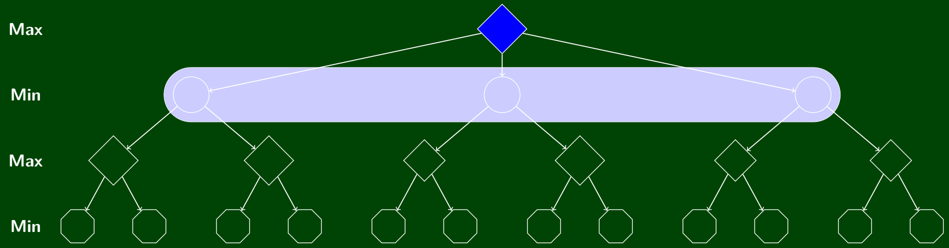


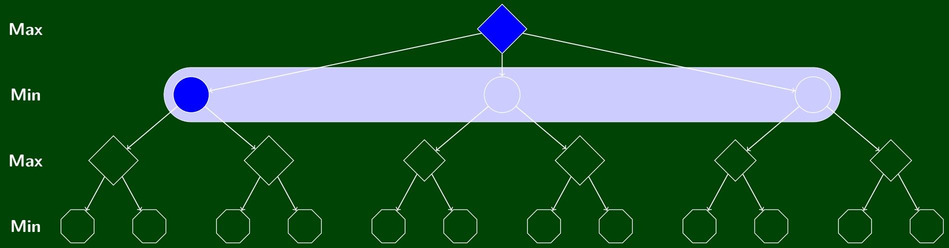
To fill in Y, **MIN** will take $\min(3,X)$. So, there are two cases:

- $4 = X > 3$. 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!

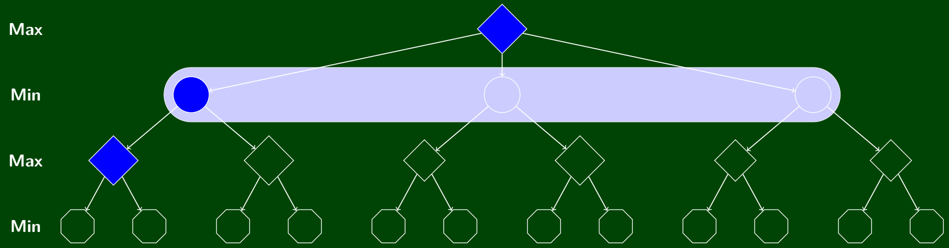
Pruning

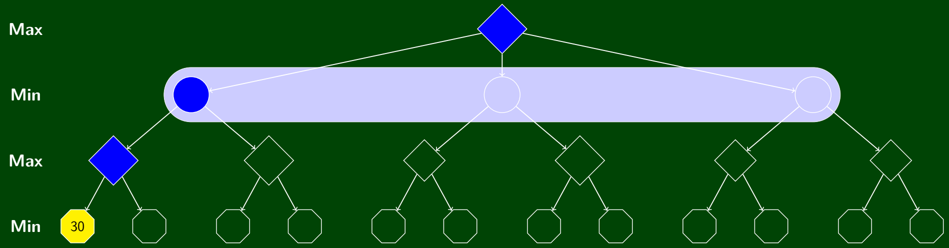




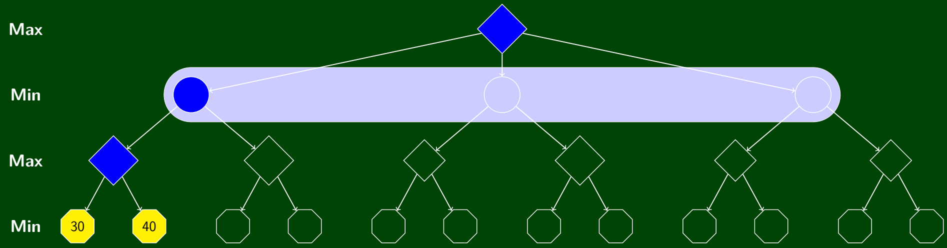
Max
Min
Max
Min

Pruning



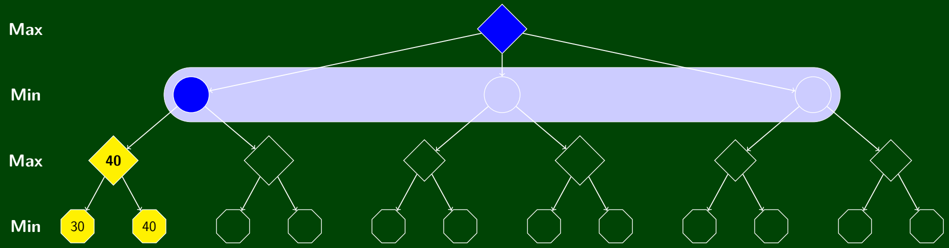


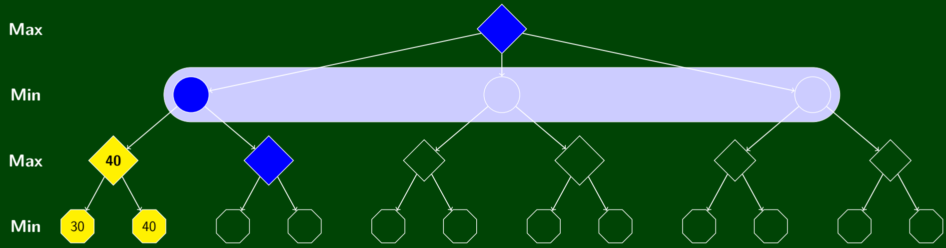
Do we check the next node?
We currently have no information. So, yes!



Do we check the next node?
We currently have no information. So, yes!

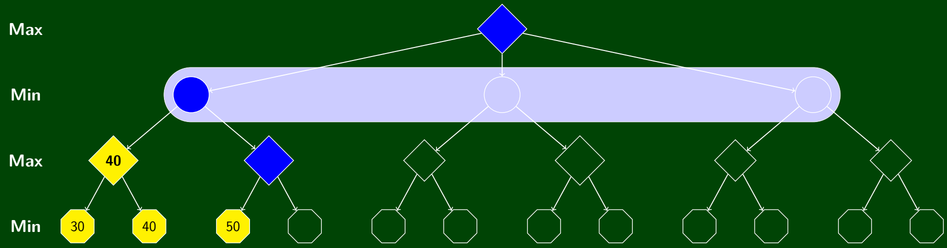
Pruning





Do we check the next node?
The current bounds are $[-\infty, 40]$. So, we **might** do better!

Pruning

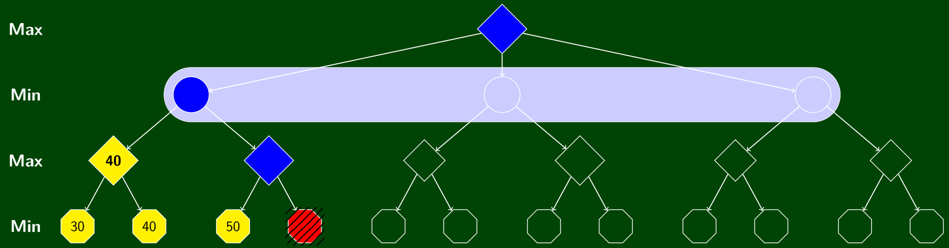


Do we check the next node?

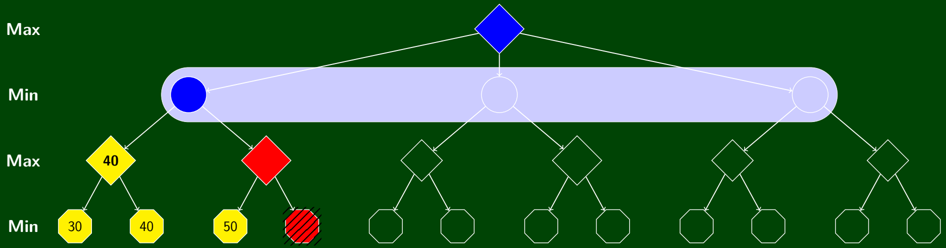
Max will choose $x \geq 50$ which is already worse than the 40.

The current bounds are $[50, 40]$. Don't bother.

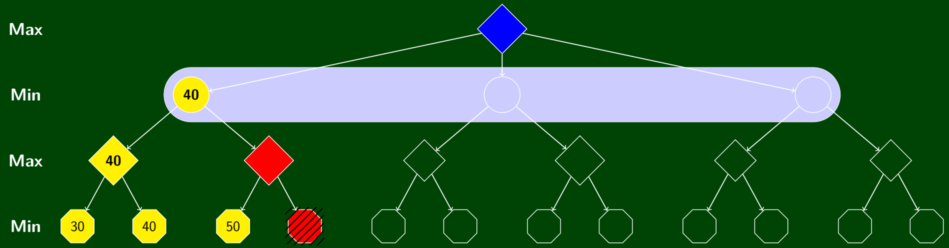
Pruning

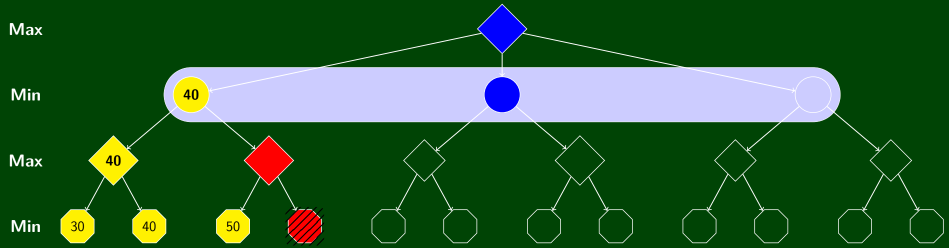


Pruning

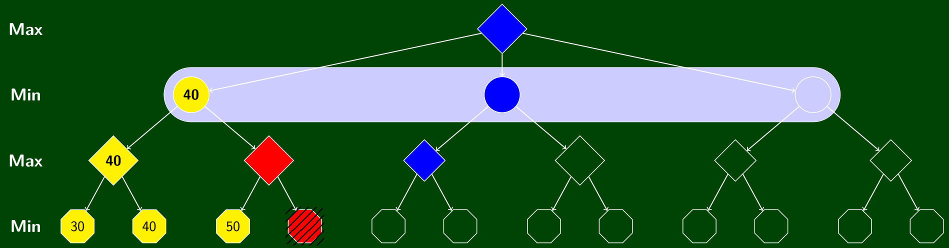


Pruning

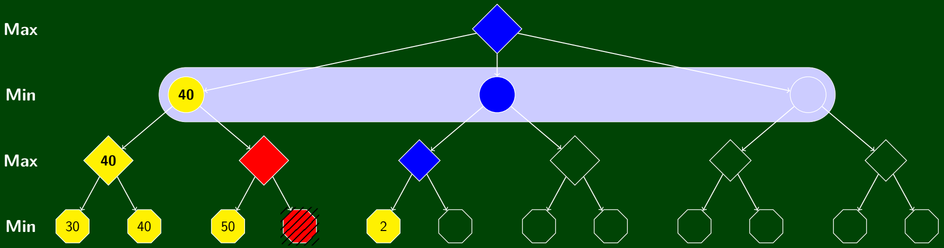



Max
Min
40
Max
40
Min
30
40
50

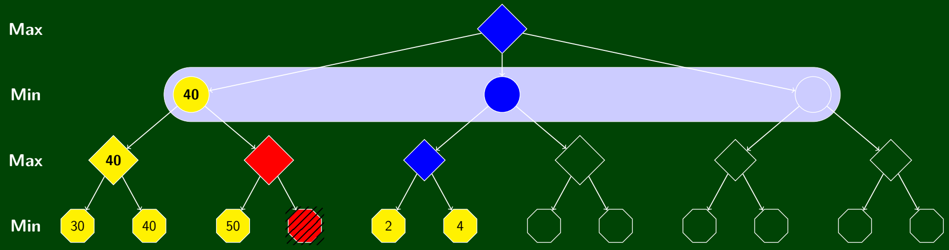
Pruning



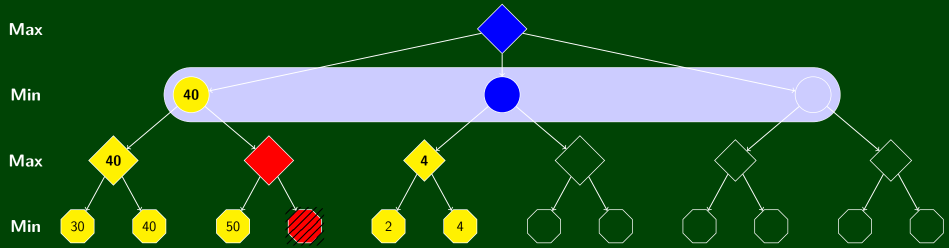
Pruning



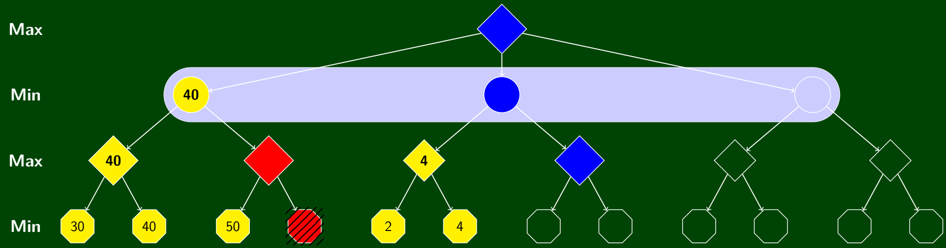
Pruning



Pruning

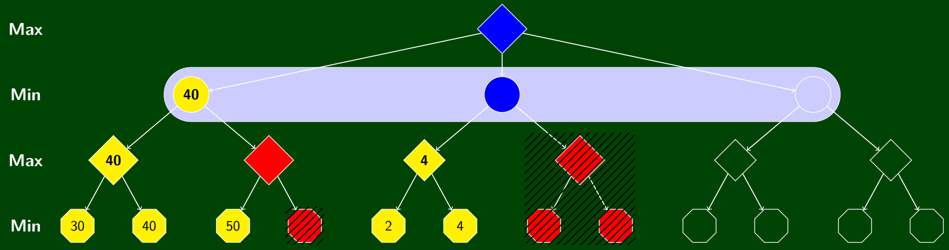


Pruning

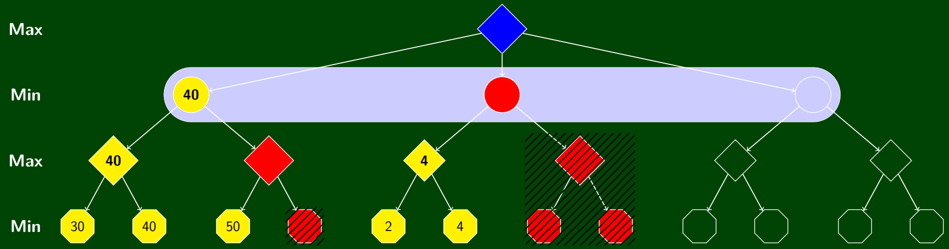


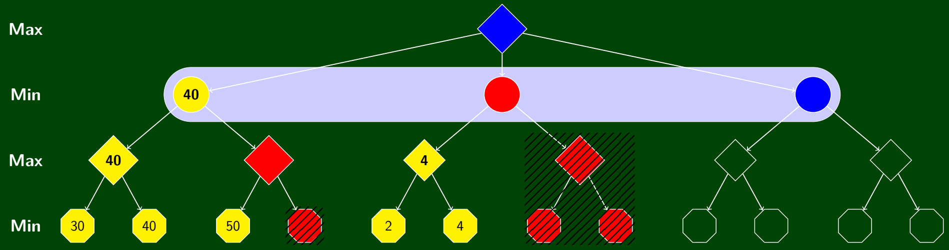
Do we check the next node?
Min will choose $x \leq 4$ which is already worse than the 40.
The current bounds are $[40, 4]$. Don't bother.

Pruning

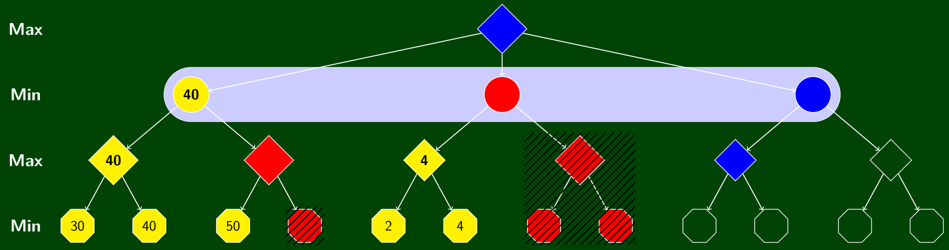


Pruning

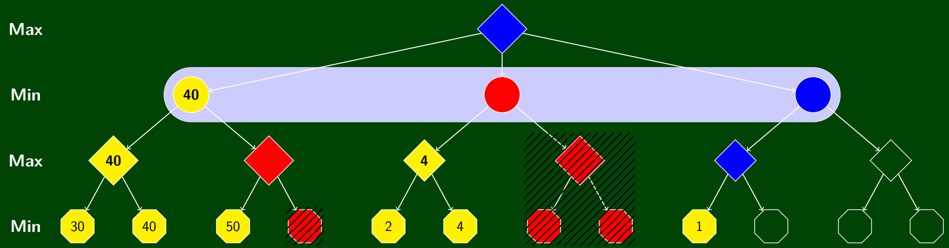



Max
Min
Max
Min
40
40
4
30
40
50
2
4

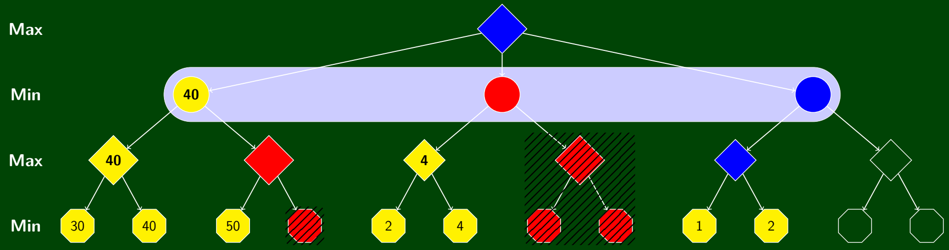
Pruning



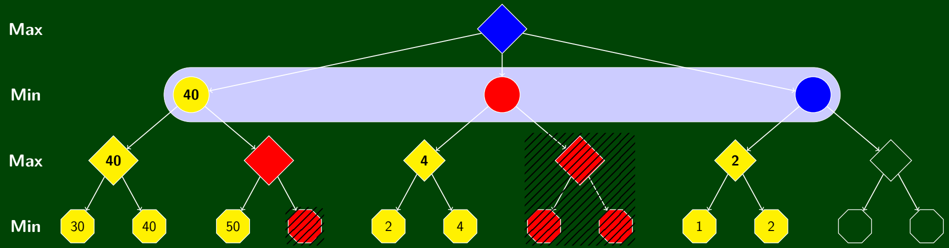
Pruning



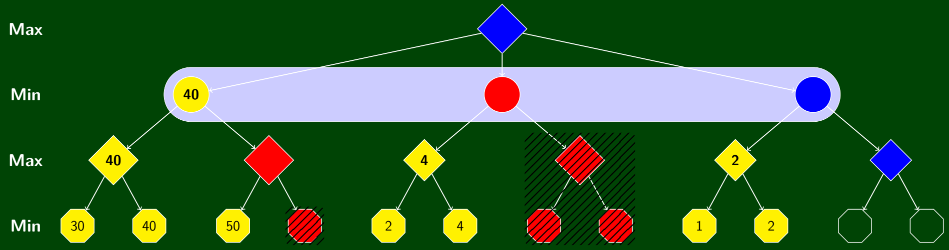
Pruning

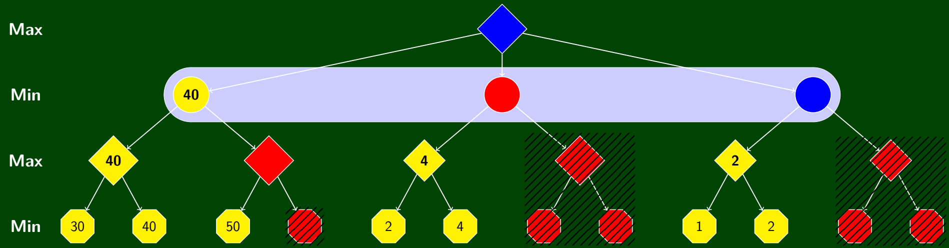


Pruning

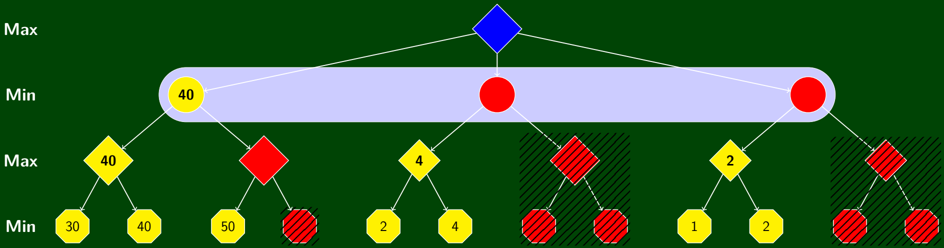


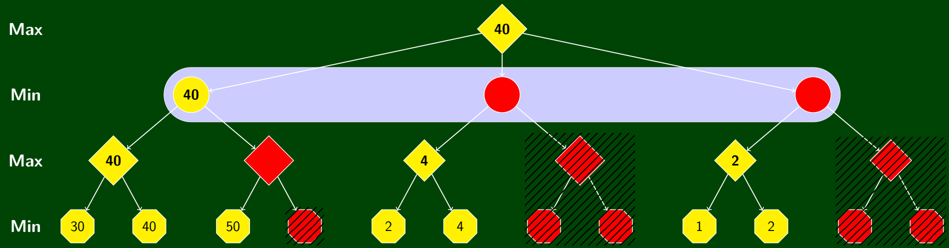
Pruning




Max
Min
40
Max
40
4
2
Min
30
40
50
2
4
1
2

Pruning





The algorithm we just ran is called **AlphaBeta**.
α is the lower bound; β is the upper bound

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.

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.

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

A demo is worth 1000 words.