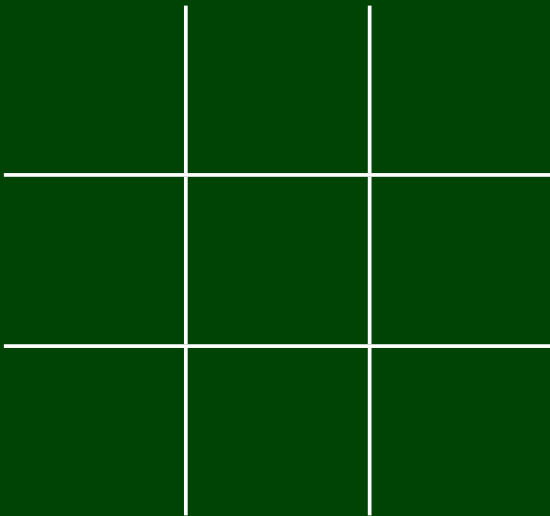
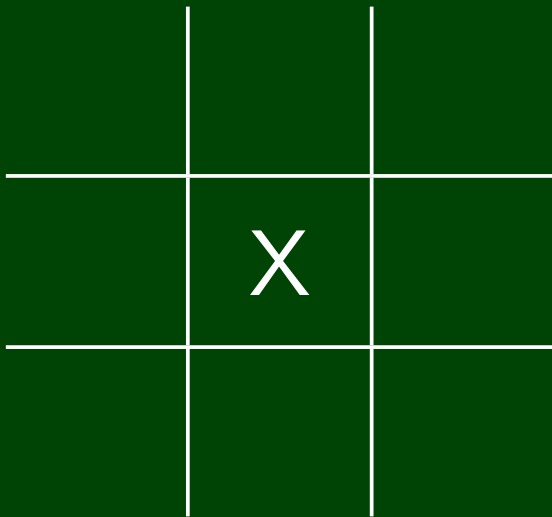


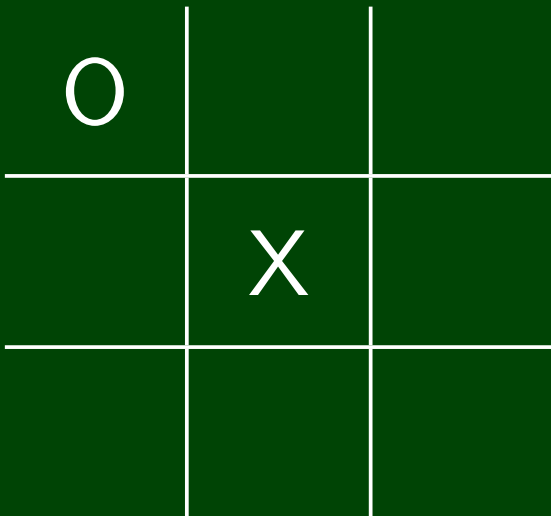
# CS 2

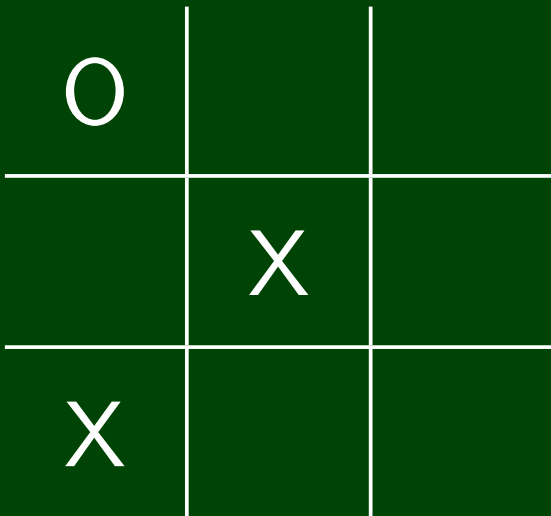
## Introduction to Programming Methods

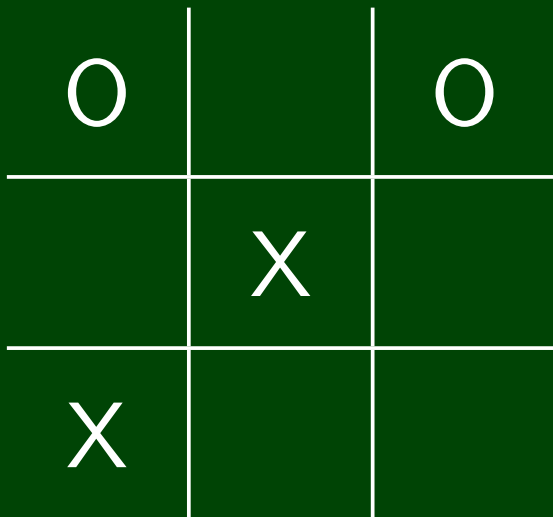
# Games

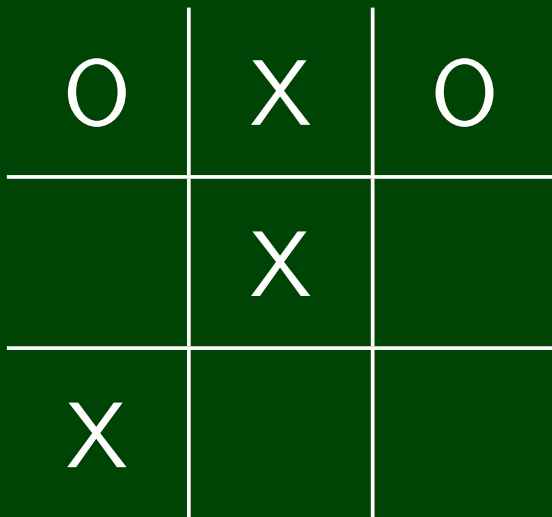




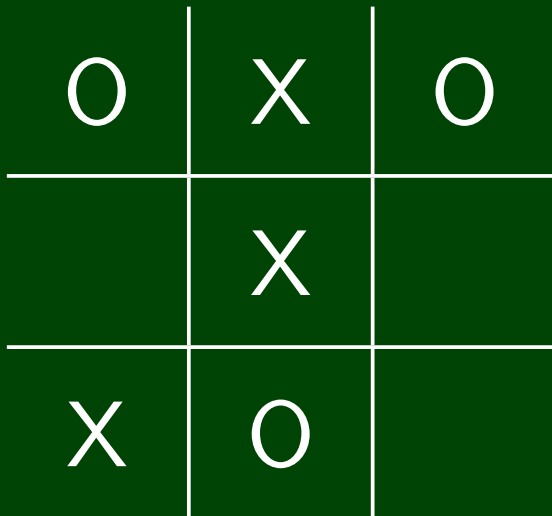


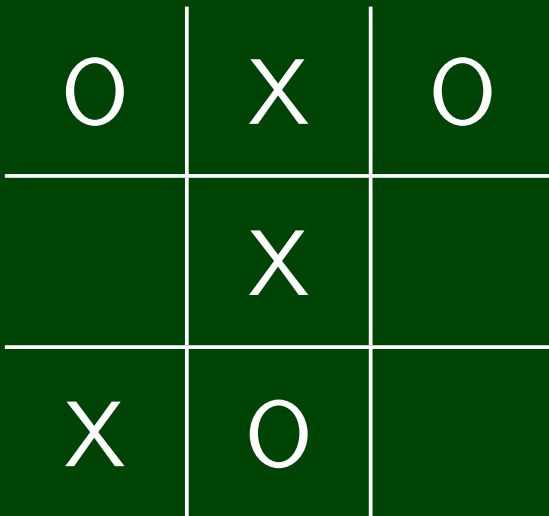












No matter what happens at this point, it's a draw.

```
1 // Let's assume I'm X
2 win(Board b) {
3     if (0 can win on the next move) {
4         block it
5     }
6     else if (the center square is open) {
7         take it
8     }
9     else if (a corner square is open) {
10        take it
11    }
12    else if (...) {
13        ...
14    }
15 }
```

```
1 // Let's assume I'm X
2 win(Board b) {
3     if (O can win on the next move) {
4         block it
5     }
6     else if (the center square is open) {
7         take it
8     }
9     else if (a corner square is open) {
10        take it
11    }
12    else if (...) {
13        ...
14    }
15 }
```

## 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!

```
1 boolean win(Board b) {
2     if (b.threeXs()) {
3         return true;
4     }
5     else {
6         for (Move m : every possible move) {
7             if (win(b.do(move))) {
8                 return true;
9             }
10        }
11        return false;
12    }
```

```
1 boolean win(Board b) {
2     if (b.threeXs()) {
3         return true;
4     }
5     else {
6         for (Move m : every possible move) {
7             if (win(b.do(move))) {
8                 return true;
9             }
10        }
11        return false;
12    }
```

## There's An Issue Here!

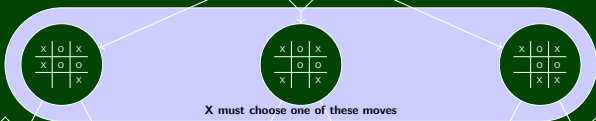
- When we make a move, it's not our turn any more.
- So the recursive call should be to **our opponent's option**
- Key Insight: Instead of guessing what the opponent is going to do, **assume she plays optimally!**

```
1 // +1 is a win; +0 is a draw; -1 is a loss
2 int eval(Board b) {
3     if (b.gameOver()) {
4         if (b.hasThree(me)) {
5             return 1;
6         }
7         else if (b.hasThree(them)) {
8             return -1;
9         }
10        else {
11            return 0;
12        }
13    }
14    else {
15        int best = -1;
16        for (Move m : every possible move) {
17            best = max(best, -eval(b.apply(move)));
18        }
19        return best;
20    }
```

X's Turn

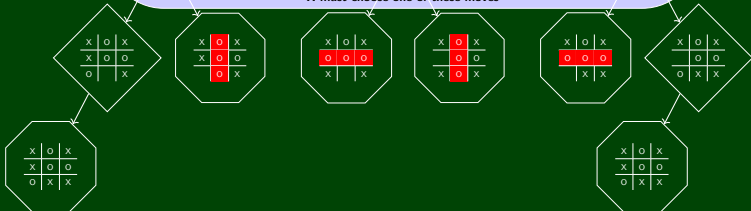


O's Turn



X must choose one of these moves

X's Turn



O's Turn

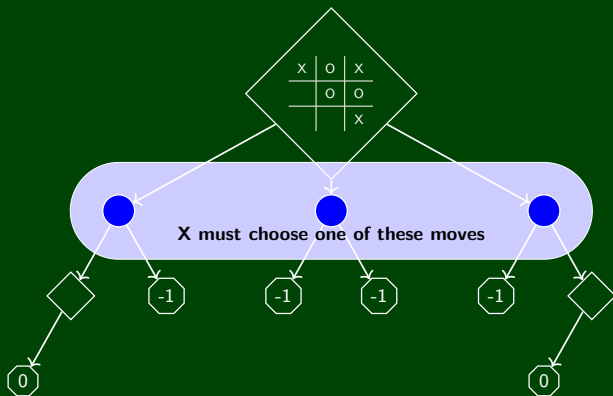


X's Turn

O's Turn

X's Turn

O's Turn

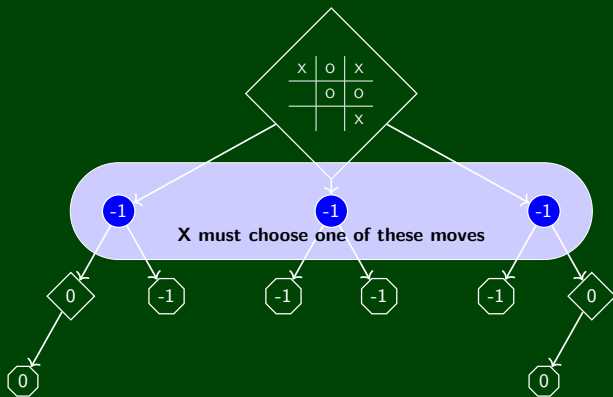


X's Turn

O's Turn

X's Turn

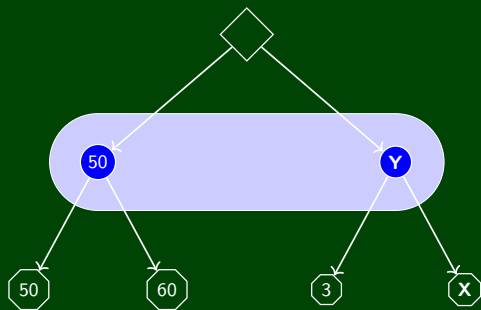
O's Turn



Max's Turn

Min's Turn

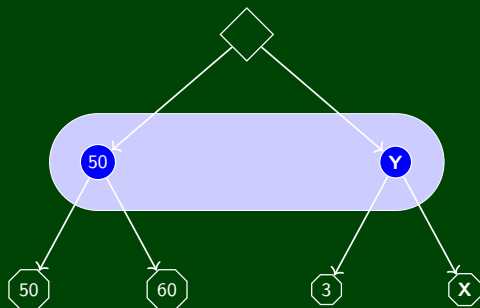
Max's Turn



Max's Turn

Min's Turn

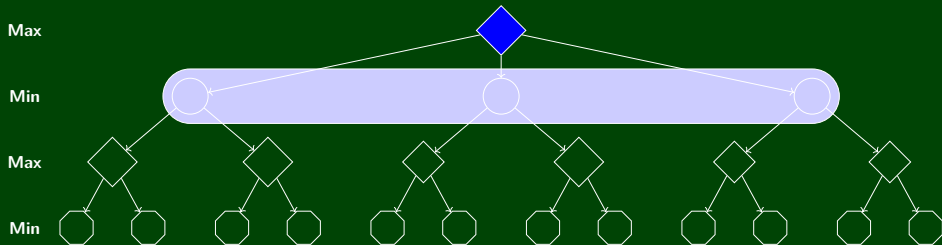
Max's Turn

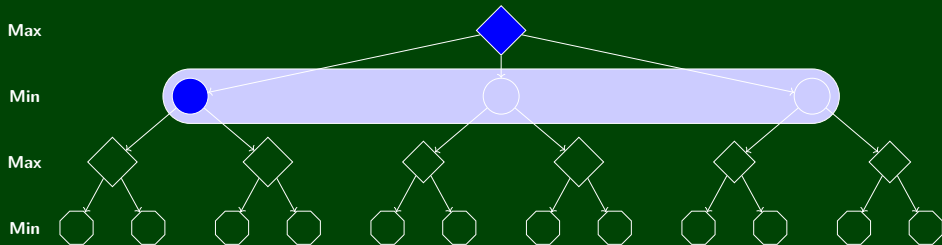


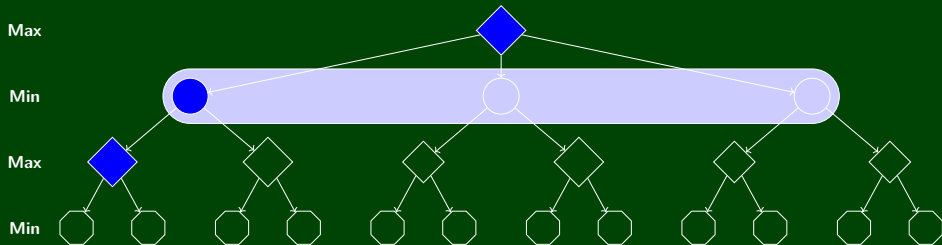
To fill in  $Y$ , **MIN** will take  $\min(3, X)$ . So, there are two cases:

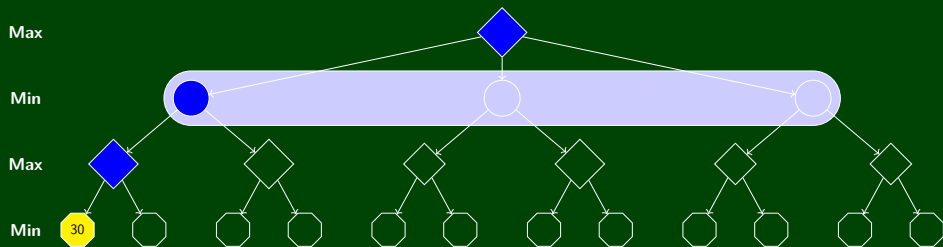
- $4 = X > 3$ . Then,  $Y = \min(3, 4) = 3$ . So, the box is 50.
- $2 = X < 3$ . Then,  $Y = \min(3, 2) = 2$ . So, the box is 50.

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

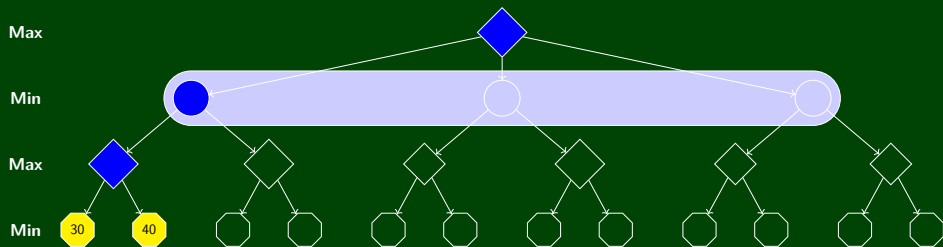




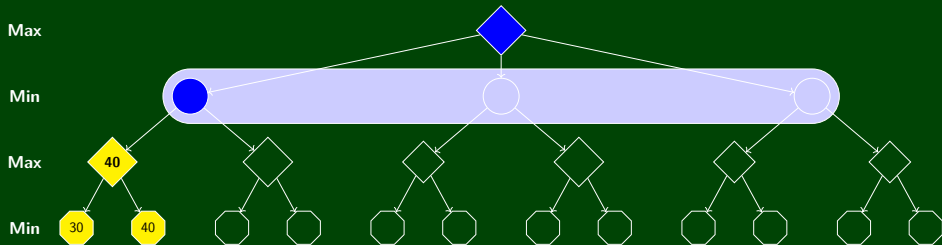


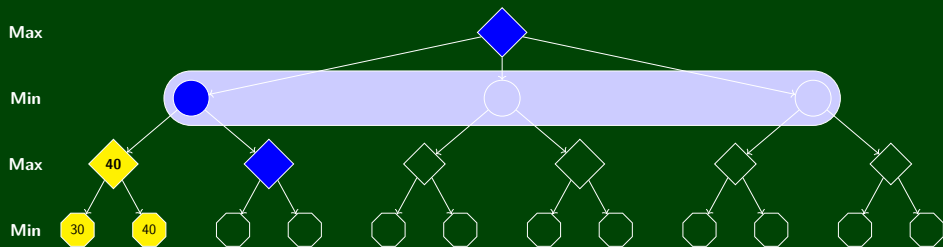






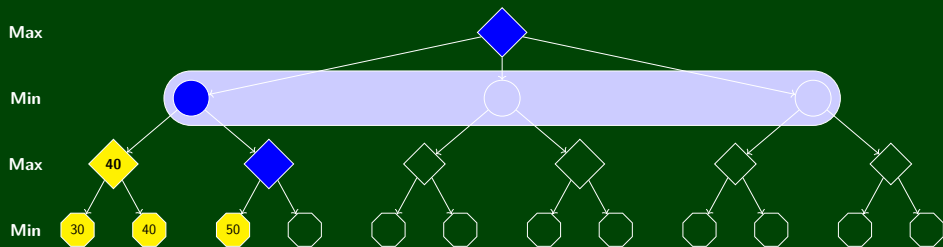
Do we check the next node?  
We currently have no information. So, yes!





Do we check the next node?

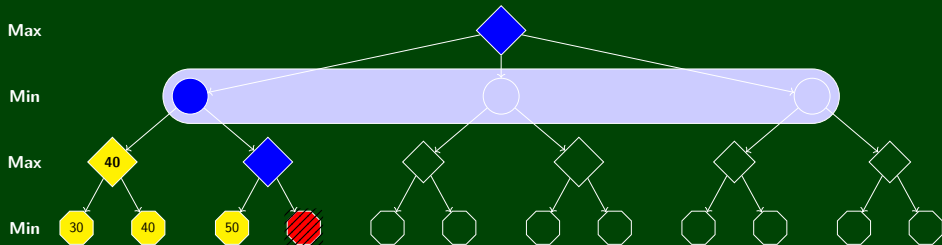
The current bounds are  $[-\infty, 40]$ . So, we **might** do better!

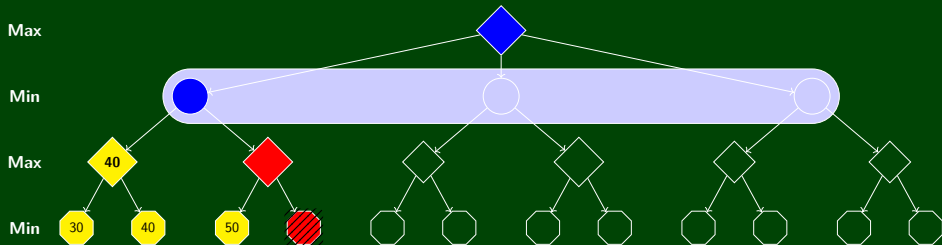


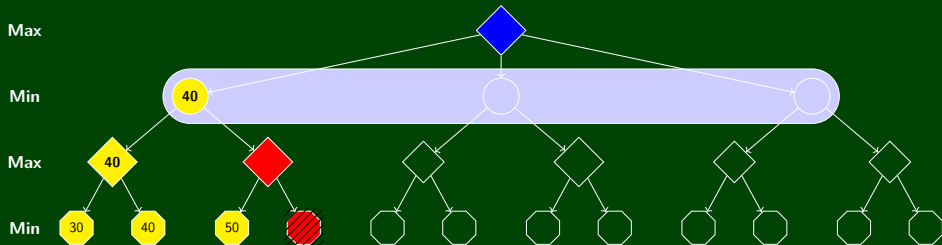
Do we check the next node?

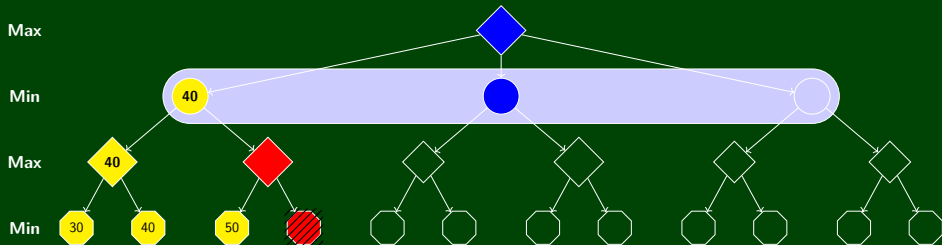
Max will choose  $x \geq 50$  which is already worse than the 40.

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

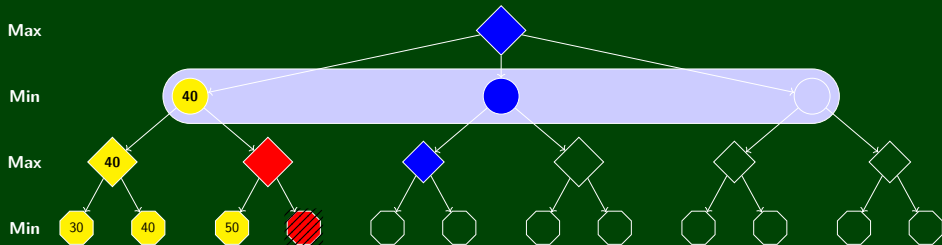


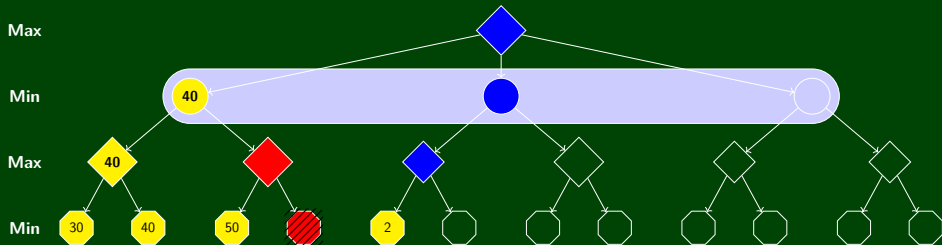


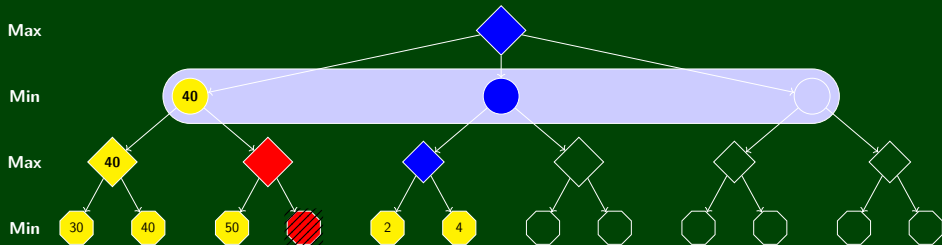


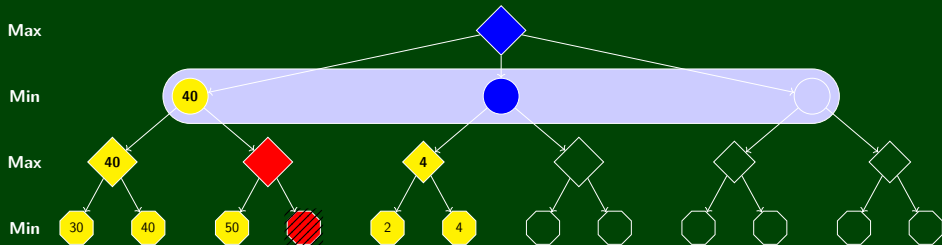


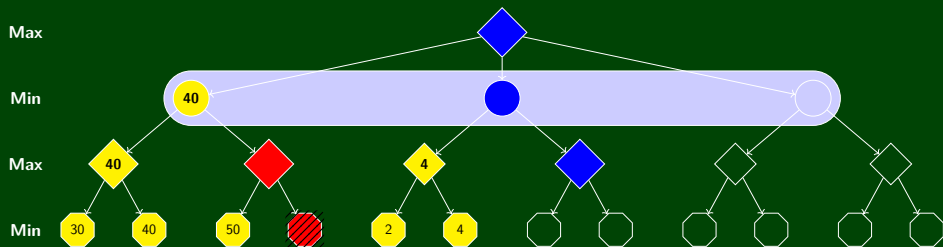








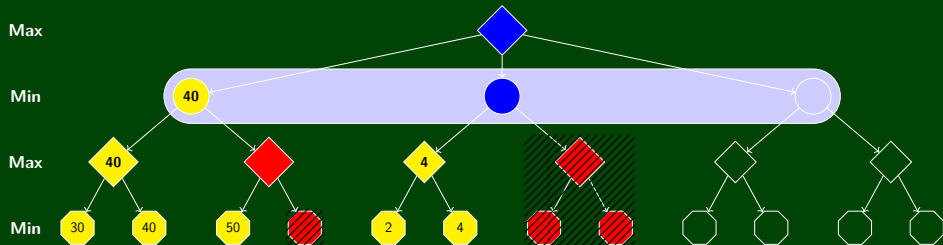




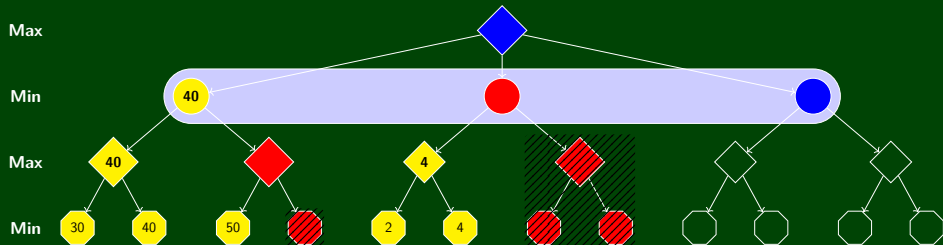
Do we check the next node?

Min will choose  $x \leq 4$  which is already worse than the 40.

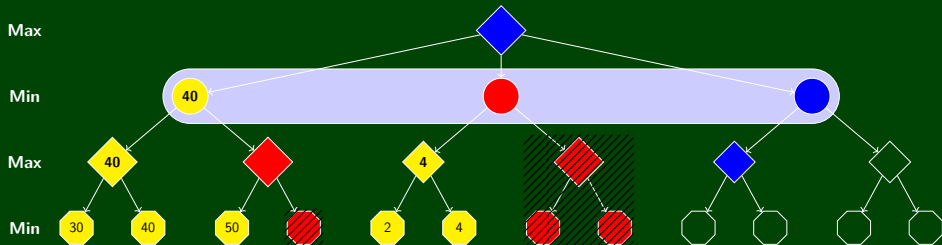
The current bounds are  $[40, 4]$ . Don't bother.

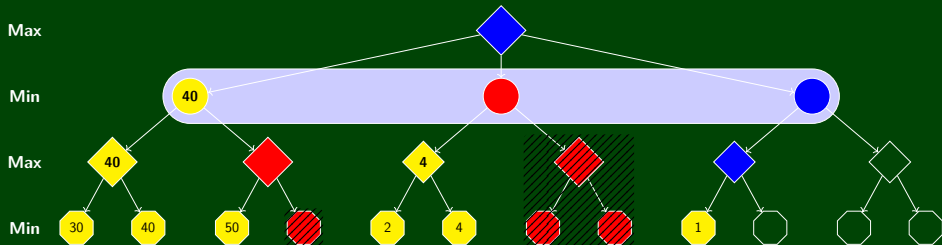


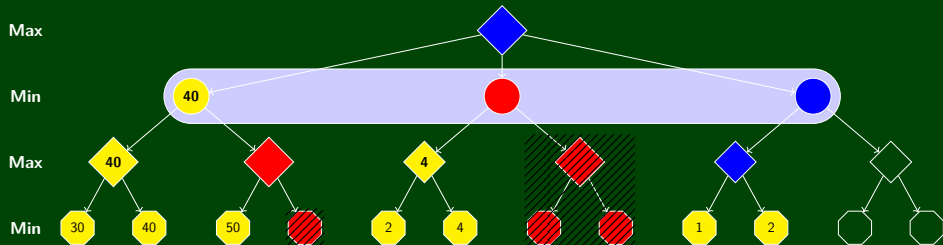


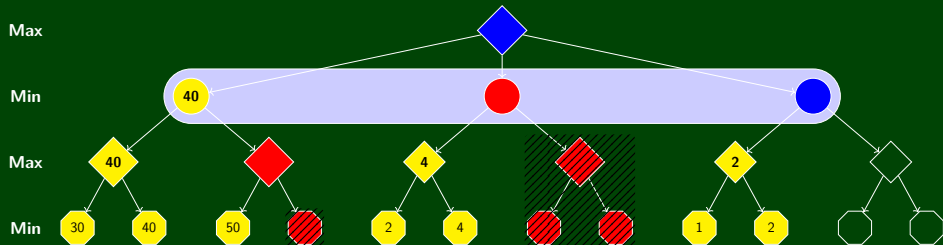


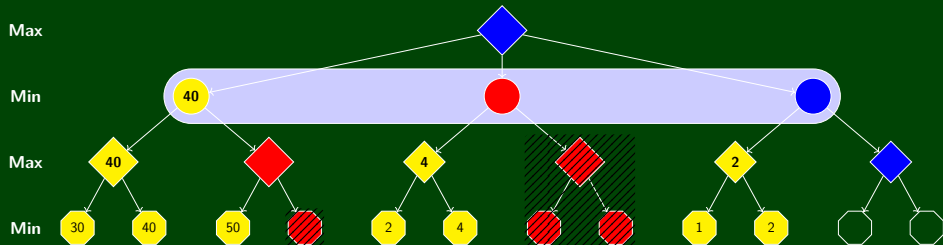


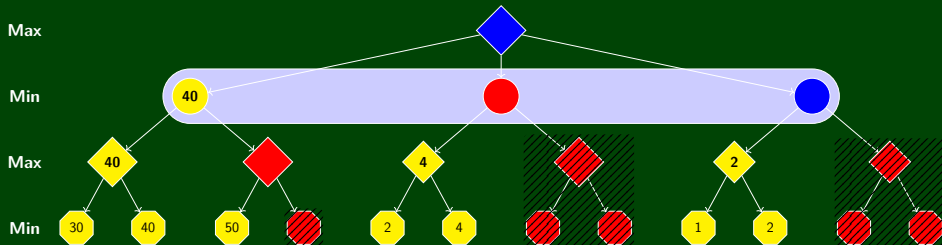


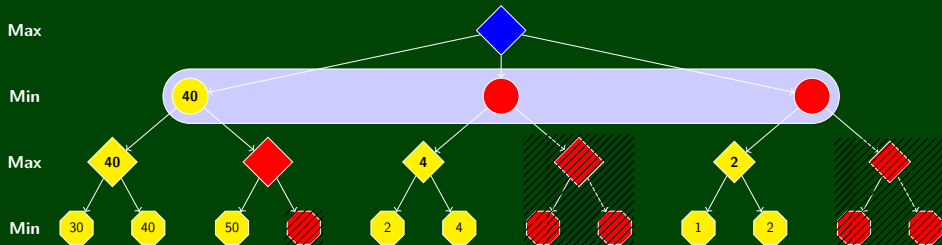


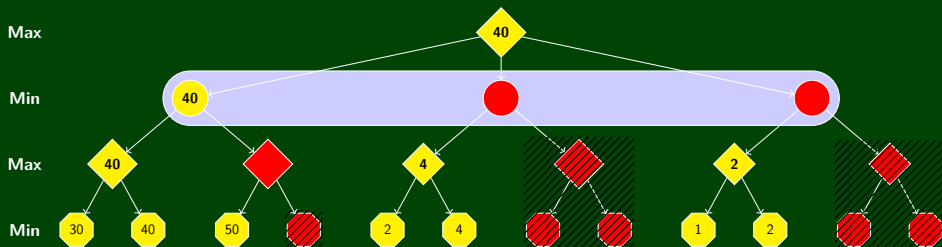












The algorithm we just ran is called **AlphaBeta**.  
 $\alpha$  is the lower bound;  $\beta$  is the upper bound



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

10

The average othello game lasts about

58 Moves

If we wanted to evaluate the whole game, we would be evaluating  $10^{58}$  **leaves**. If we were able to evaluate **1 trillion** leaves a second, we would need  $10^{46}$  seconds.

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