

# ADVERSARIAL SEARCH

ECS170 Spring 2018  
Josh McCoy, @deftjams

# Multi-agent Search

- Game Theory
- Competition
- Zero Sum
  - Better described as constant payout
- Perfect Information

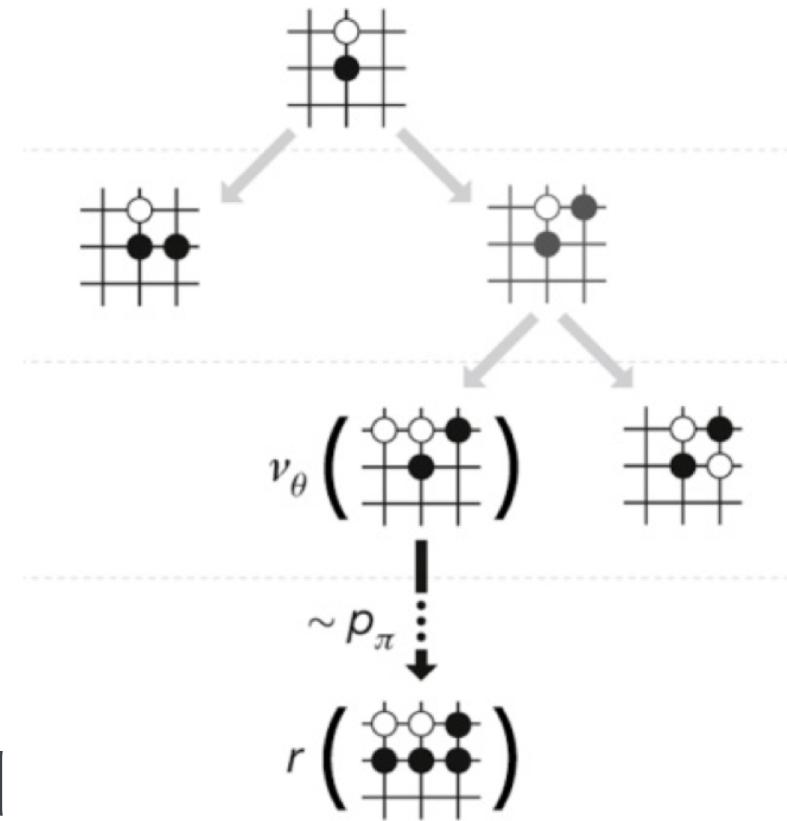
**Prisoner's dilemma payoff matrix**

	<b>B</b>	<b>B stays silent</b>	<b>B betrays</b>
<b>A</b>			
<b>A stays silent</b>	-1	0	0
<b>A betrays</b>	0	-3	-2

# Game Tree

Adversarial search  
builds a search tree  
just like previous  
search algorithms.

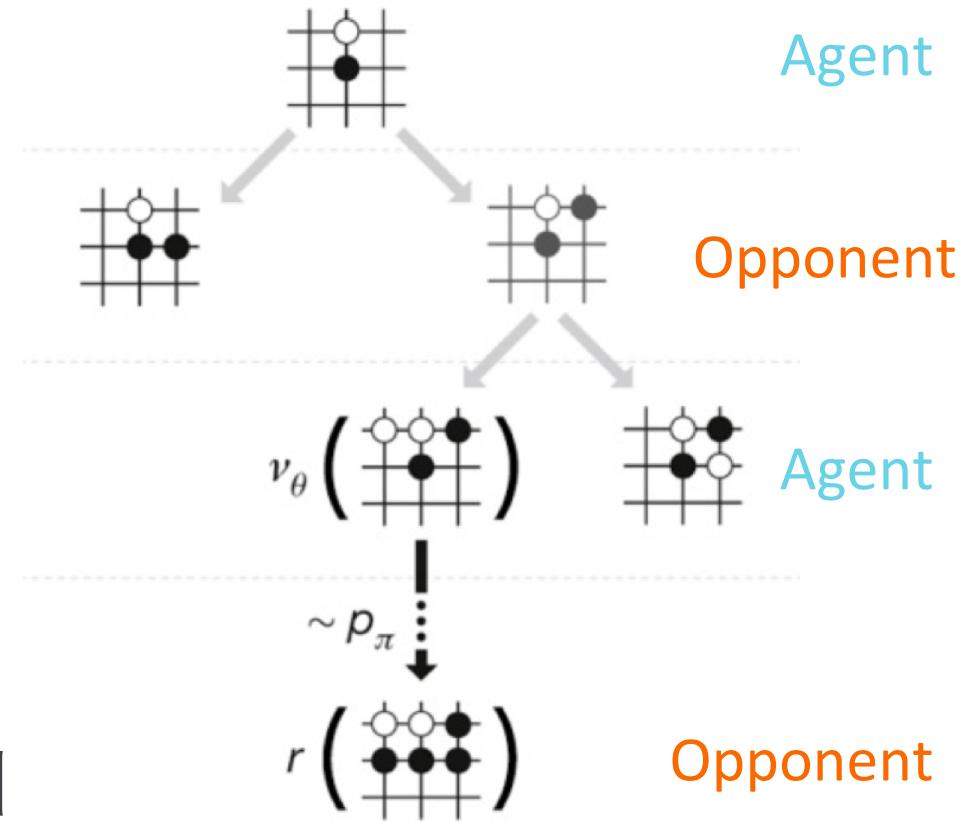
Twist: control is  
passed between  
players at each level  
of depth.



# Game Tree

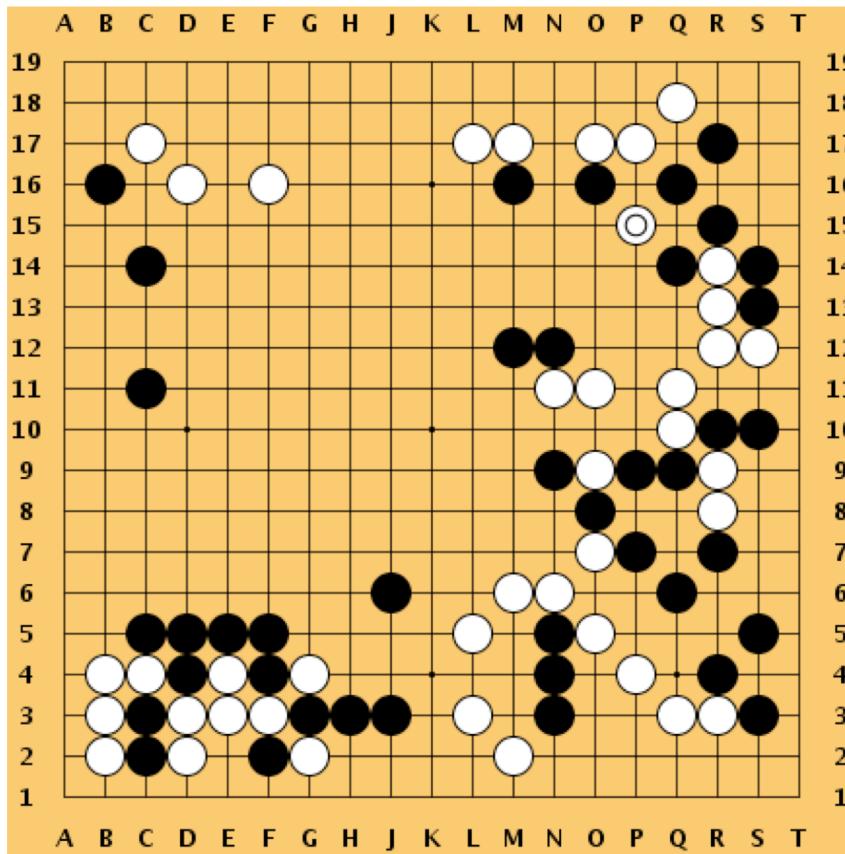
Adversarial search builds a search tree just like previous search algorithms.

Twist: control is passed between players at each level of depth.



# Evaluation Functions

f



= V

[https://commons.wikimedia.org/wiki/File:Go\\_game\\_Kobayashi-Kato.png](https://commons.wikimedia.org/wiki/File:Go_game_Kobayashi-Kato.png)

# Human Evaluation (WWLSD?)



<https://www.theatlantic.com/technology/archive/2016/03/the-invisible-opponent/475611/>

# Adversarial Search

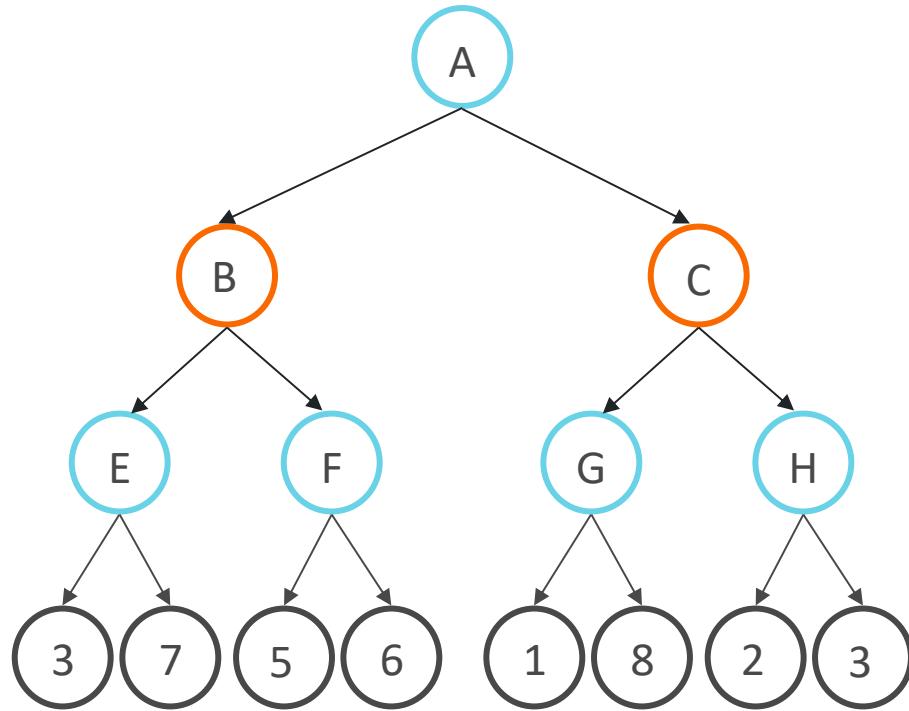
- Initial State
- Player(state): which player has agency
- Actions(state): available moves
- Result(state, action): next game state
- Terminal(state): true if state is a leaf.
- Utility(state, player): score of state for player
- Min(...) - least of args
- Max(...) - most of args

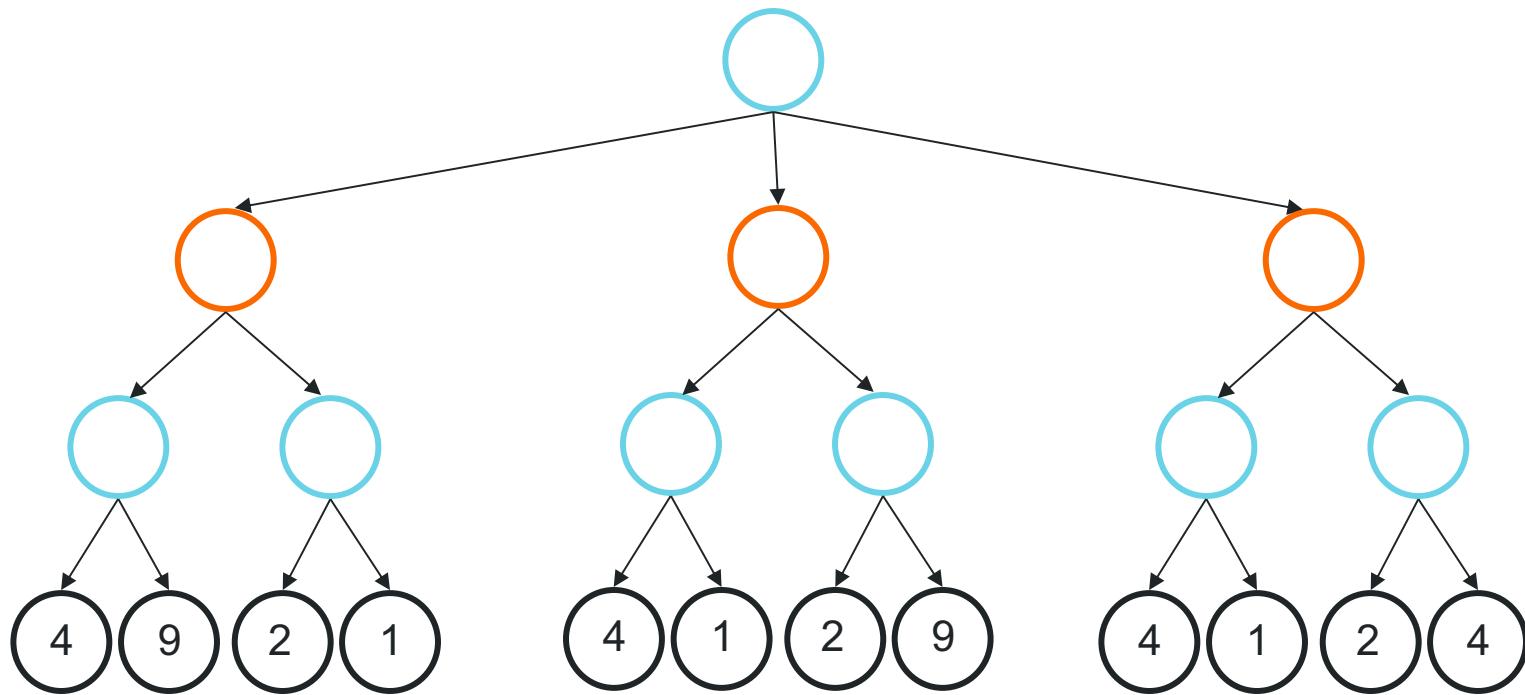
# A Tale of Min and Max

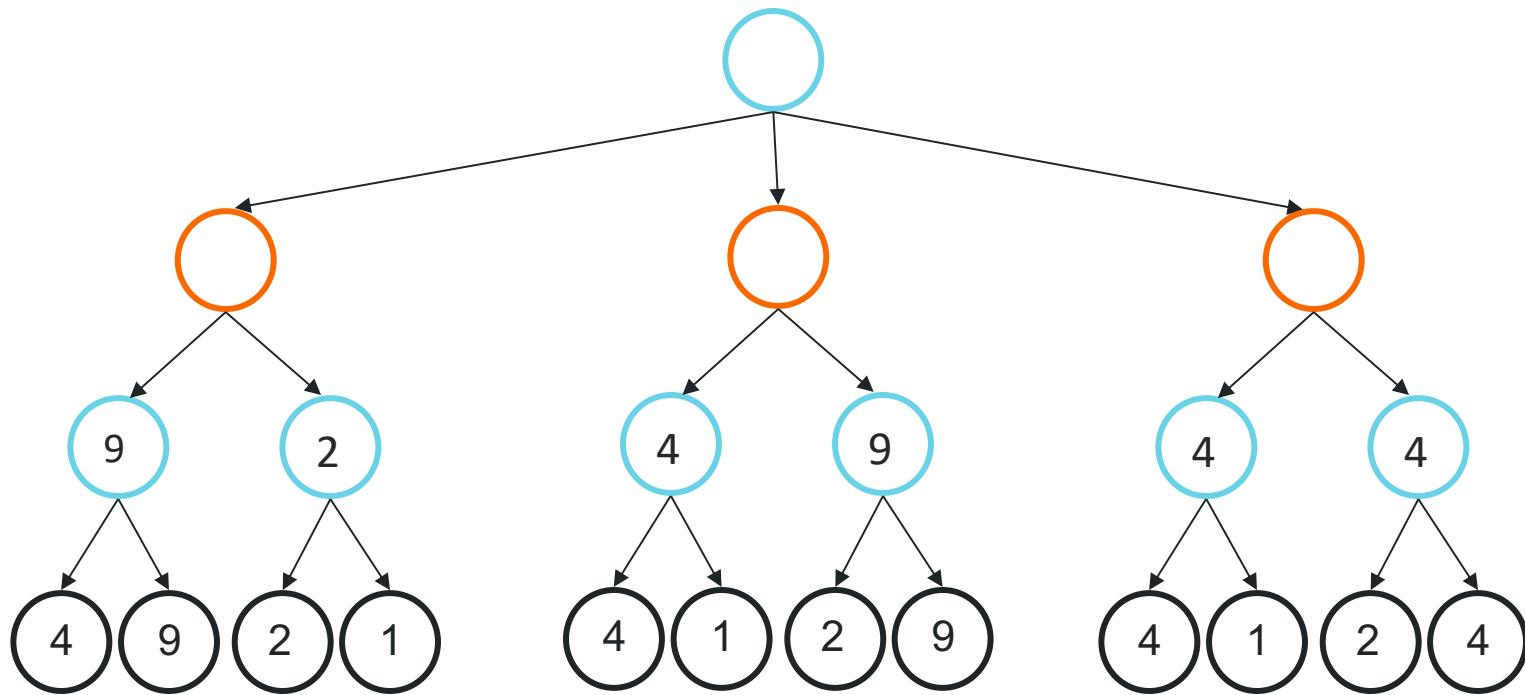
- Player wants to maximize their score.
- Opponent wants to minimize the player's score.
- Alternating rounds of decisions.
  - max -> min -> max -> min -> ...

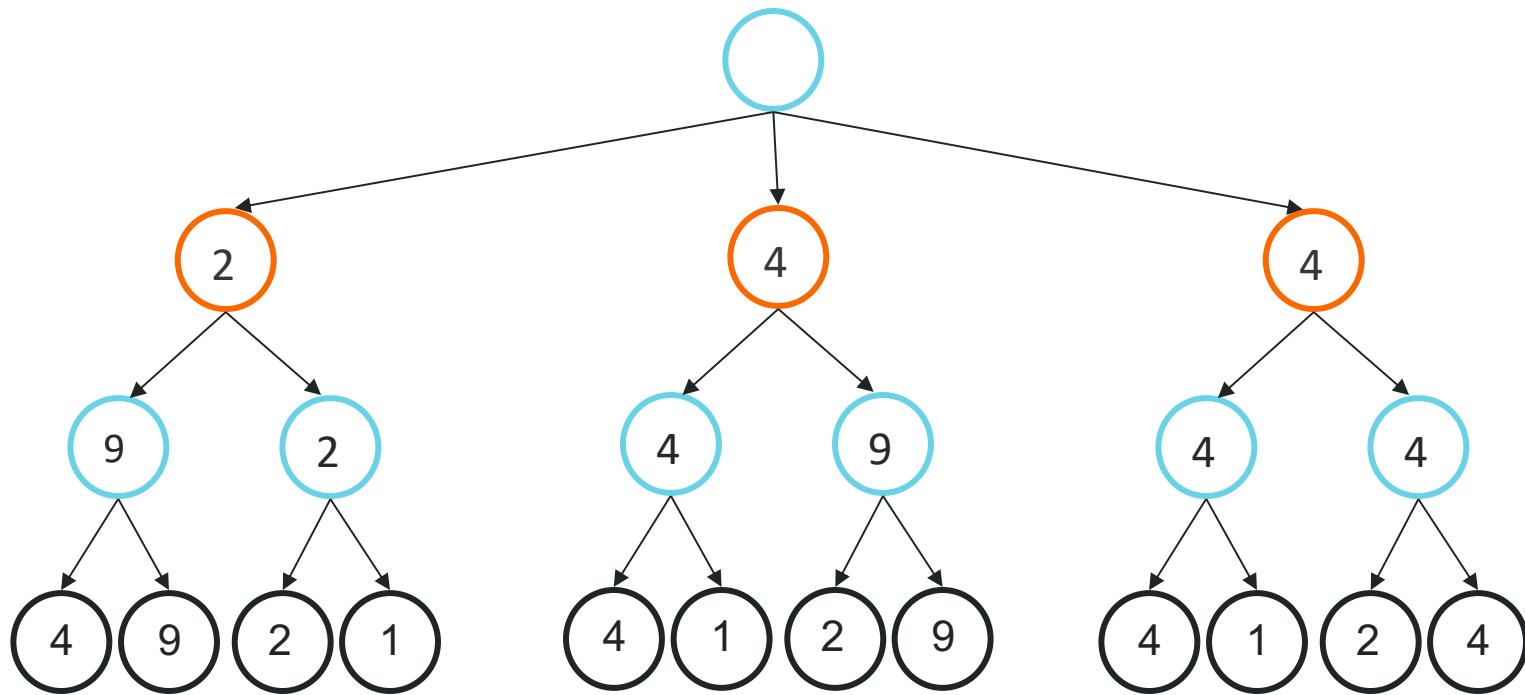
# Minimax

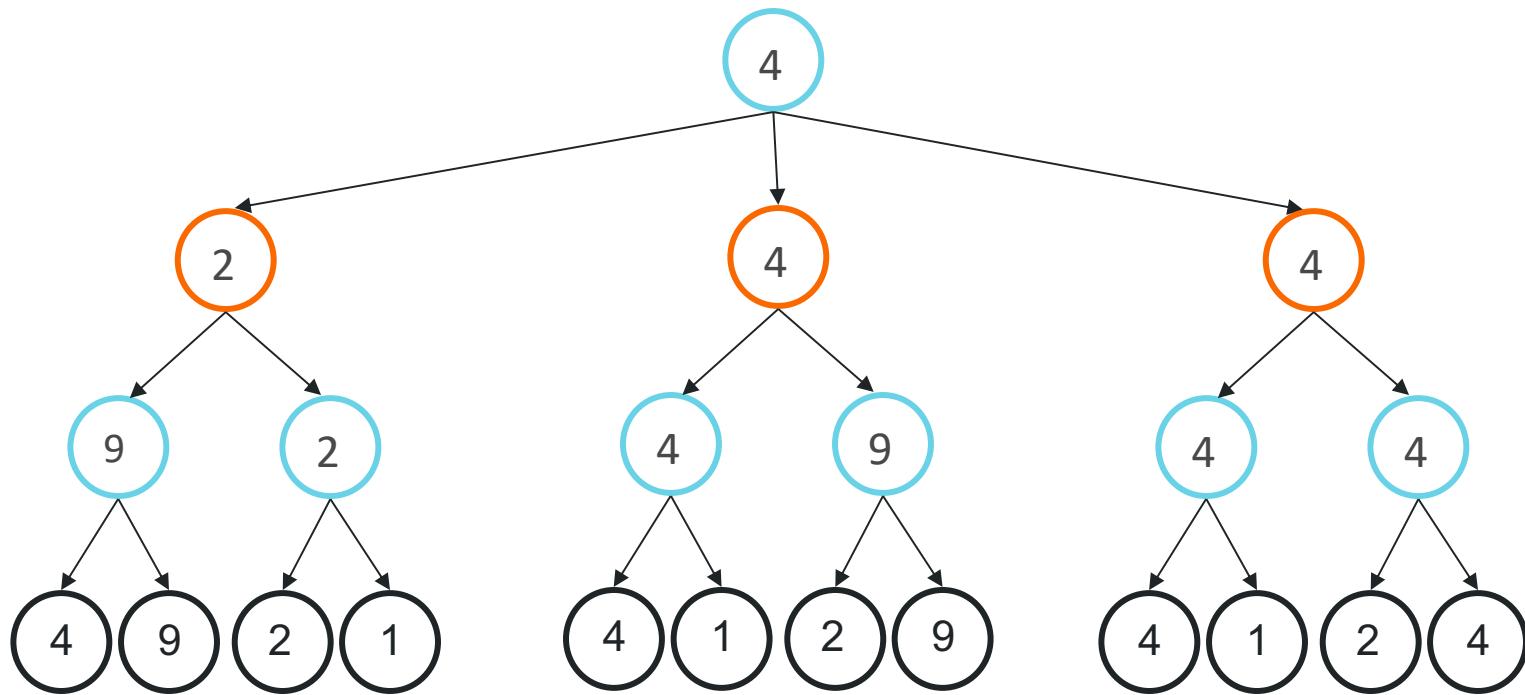
```
minimax(s=initial_state, p=starting_player):
    if terminal(s) then return utility(s,p)
    if max=player(s) then
        value = -infinity
        for each a in actions(s)
            value = max(value, minimax(result(s,a), min))
        return value
    if min=player(s) then
        value = +infinity
        for each a in actions(s)
            value = min(value, minimax(result(s,a), max))
        return value
```



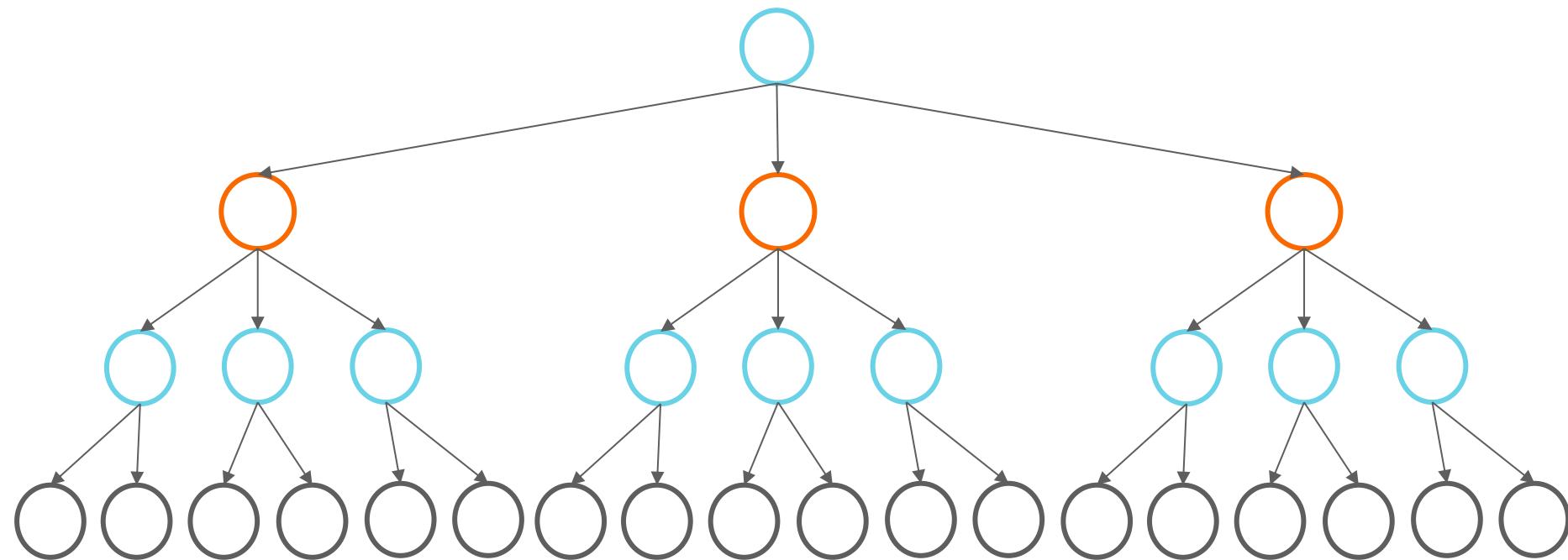








# Together!



# Too Many Paths

Time complexity:

$O(moves^{depth})$  or  $O(b^m)$

Space complexity:

$O(moves * depth)$  or  $O(bm)$

Depth limited minimax?

# Alpha-Beta ( $\alpha$ - $\beta$ ) Pruning

```
alphabeta(state, α, β):
```

```
    if terminal(state) then return utility(state, max)
```

```
    else if max=player(state)
```

```
        v = -∞
```

```
        For each a in actions(state)
```

```
            v = max(α, alphabeta(result(state, a), α, β))
```

```
            if v >= β return v
```

```
            α = max(α, v)
```

```
        return v
```

```
    else min=player(state)
```

```
        v = ∞
```

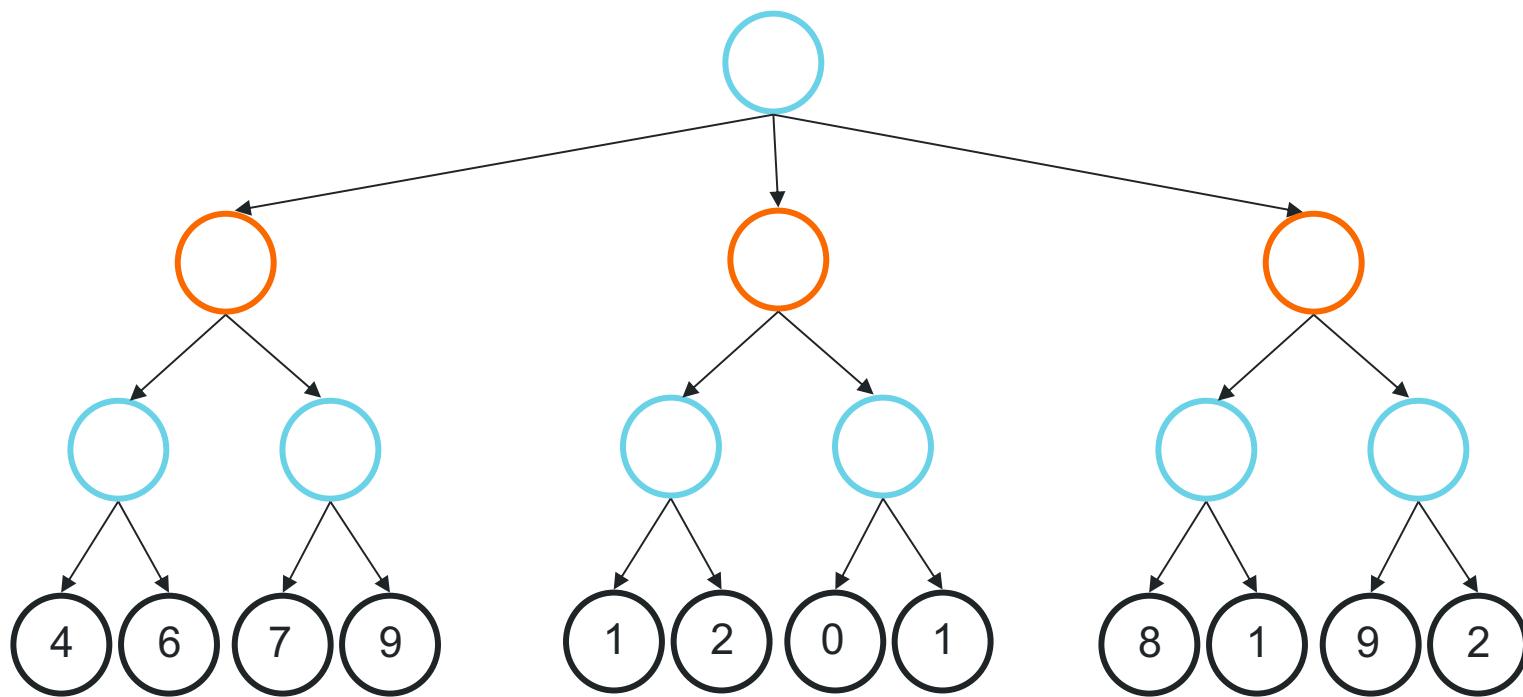
```
        for each a in actions(state)
```

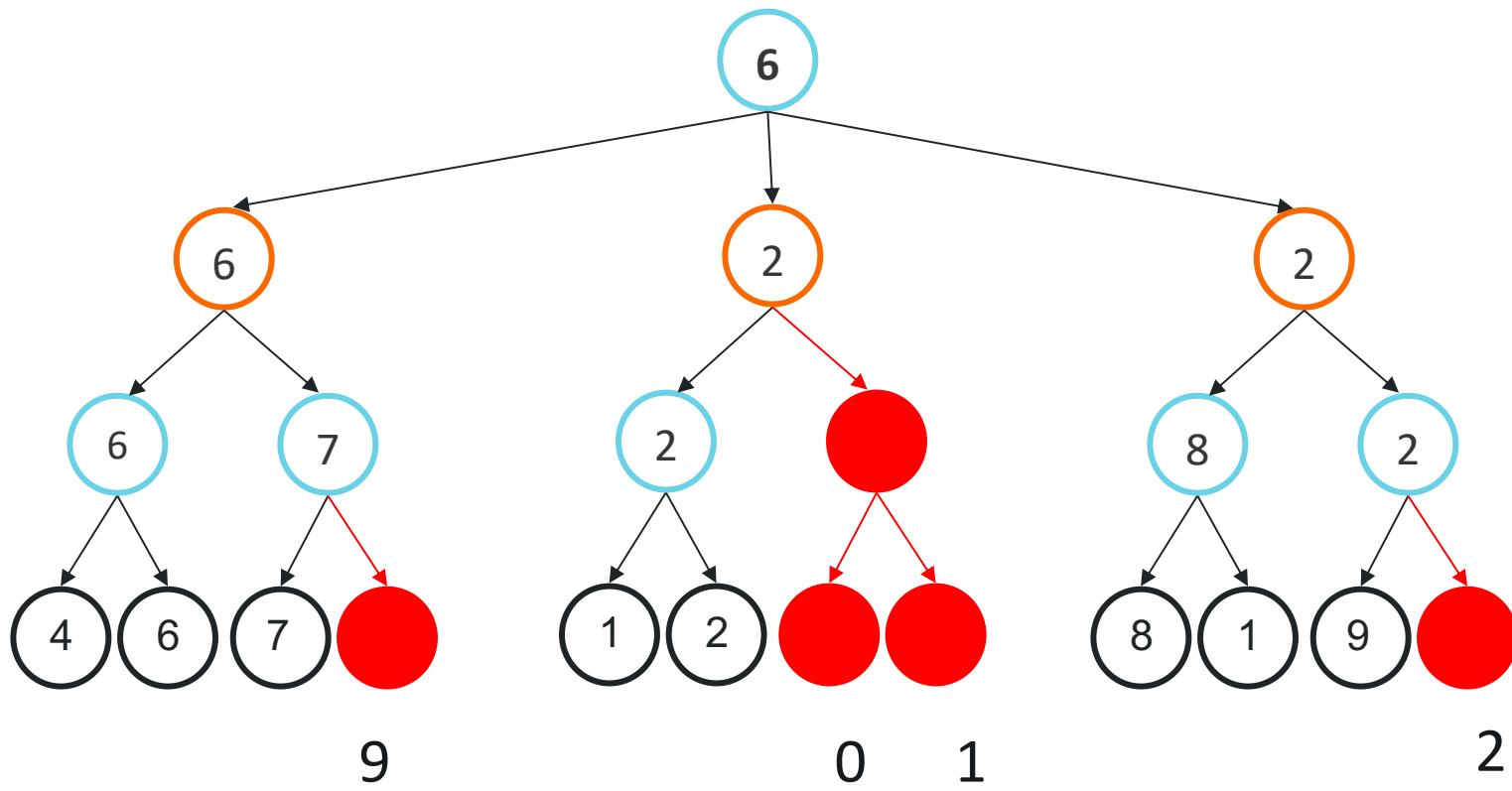
```
            v = min(β, alphabeta(result(state, a), α, β))
```

```
            if v <= α return v
```

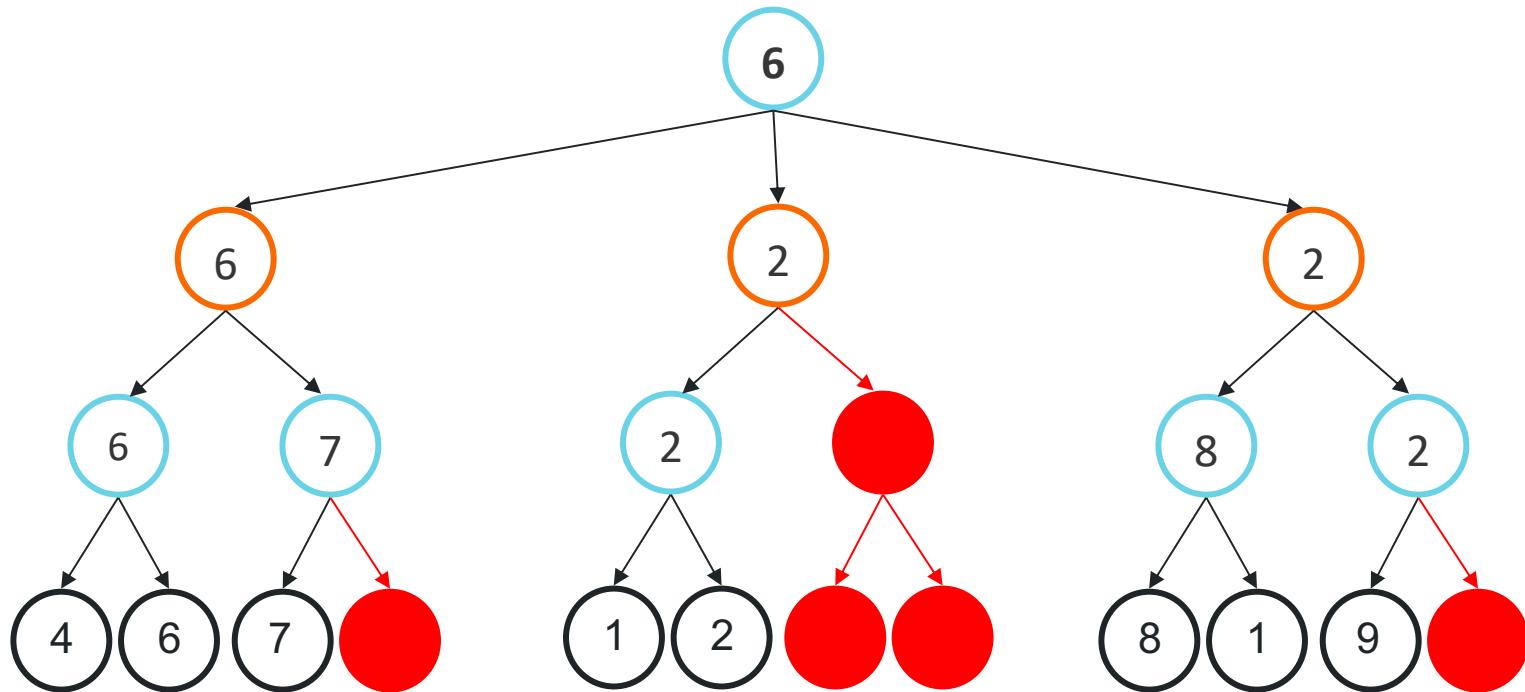
```
            β = min(β, v)
```

```
        return v
```





Time complexity:  
 $O(moves^{depth/2})$  or  $O(b^{m/2})$



# Too Many Paths

Space complexity:  
 $O(\text{moves} * \text{depth})$  or  $O(bm)$