Artificial Intelligence

Lecture 05: Monte Carlo Tree Search

Lecture Summary

- Monte Carlo Tree Search (MCTS)

Monte Carlo

Monte Carlo = Random Sampling

Monte Carlo Approximation

Monte Carlo Simulation

Monte Carlo Rollout (playout)

https://en.wikipedia.org/wiki/Monte_Carlo_Casino#/media/File:Monaco_-_panoramio_(138).jpg

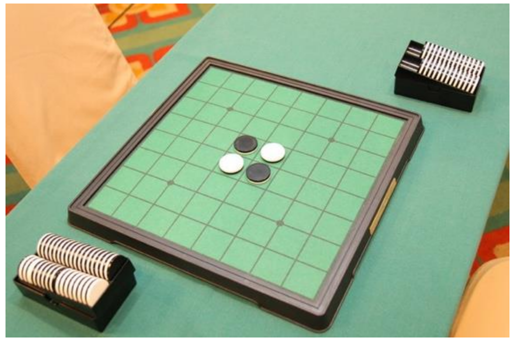
The State of Play

	X	
	X	O

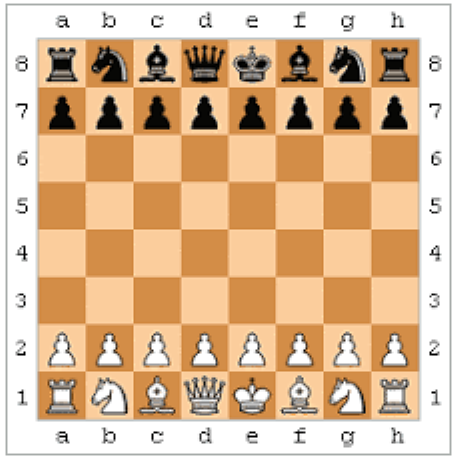
Tic-Tac-Toe
10^{4} positions
PERFECT!

Oware
10^{11} positions
PERFECT!

Checkers
10^{20} positions
PERFECT!



Othello
10^{28} positions
Superhuman (1997)



Chess
10^{45} positions
Superhuman (1996)

Go
10^{172} positions
Superhuman (2016)

Starcraft II
??? positions
Superhuman (2019)

Table Tennis
??? positions
??? (2022)

Why is Go so hard?

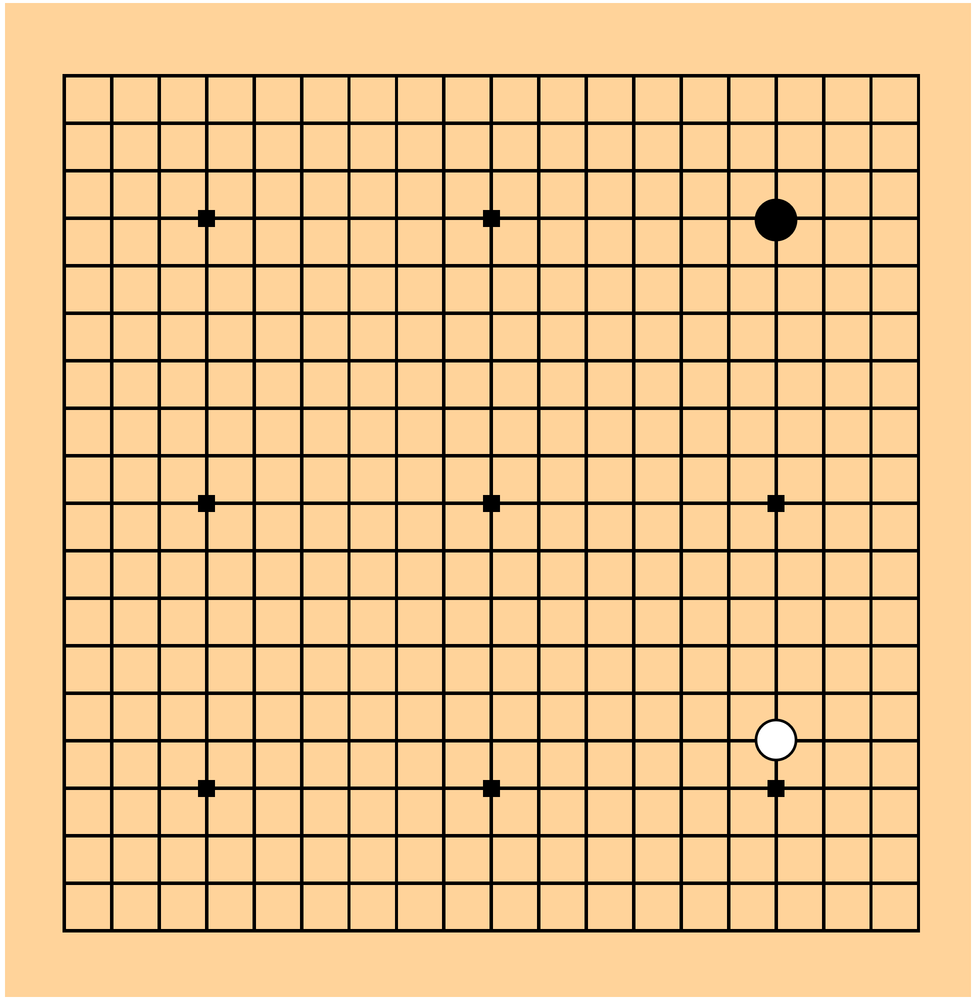
English: Go

Japanese: 囲碁 (いご, I go)

Chinese: 围棋/圍棋 (*wéiqí*)

- The encircling strategy game

How does it work?



19 x 19 board

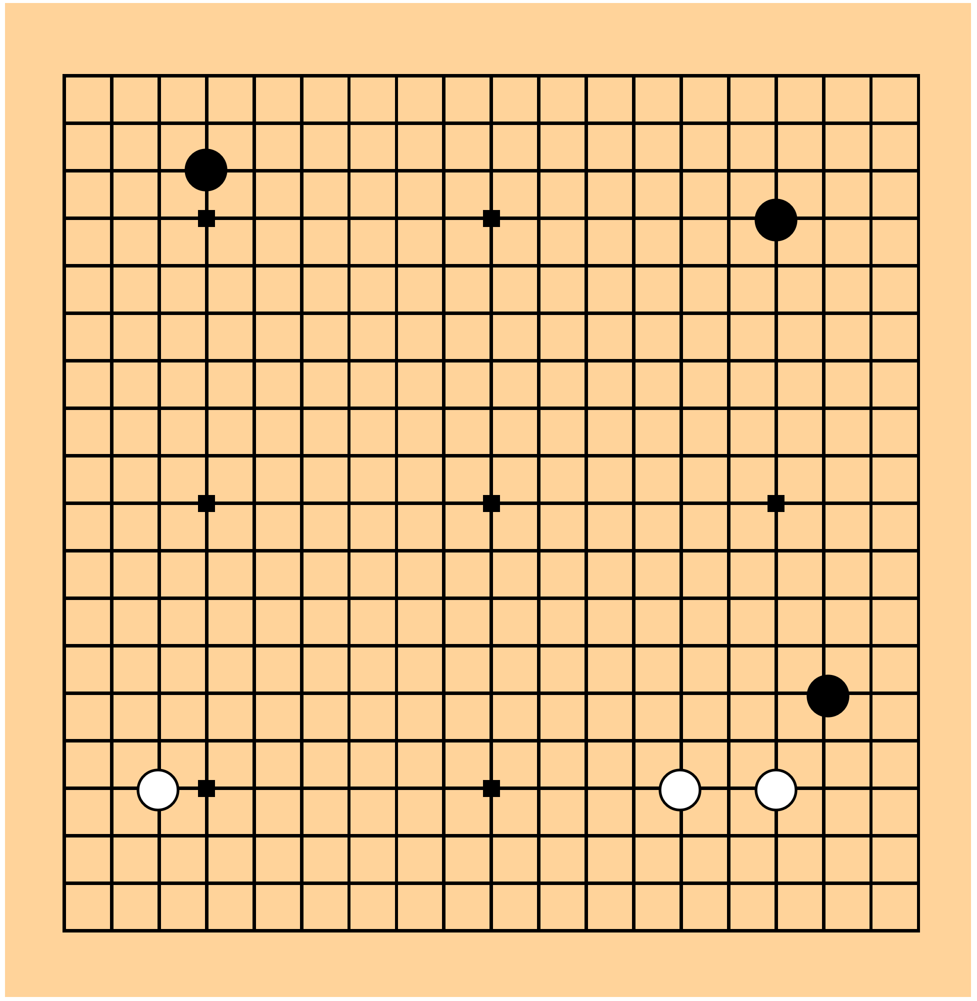
Two players

- White & Black (black moves first)

Players take turns placing stones on intersections

OK, first observations… why is this not CS friendly

Why is Go so hard?



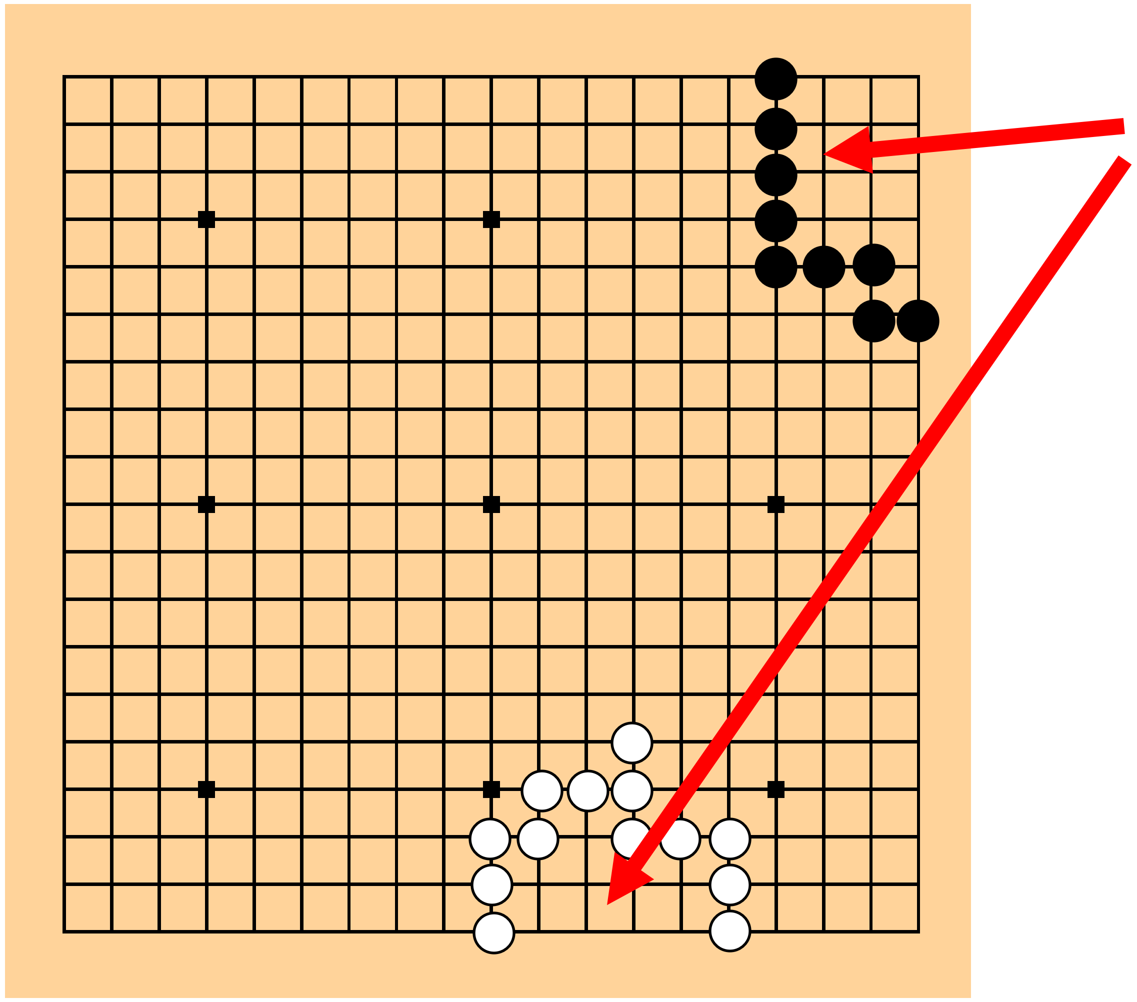
We have 361 opening moves (well really 45 not counting reflections)

We have… 360 counter moves (even counting reflections)

Six moves in… 2 billion… well 250 million not counting reflections

We haven't even started

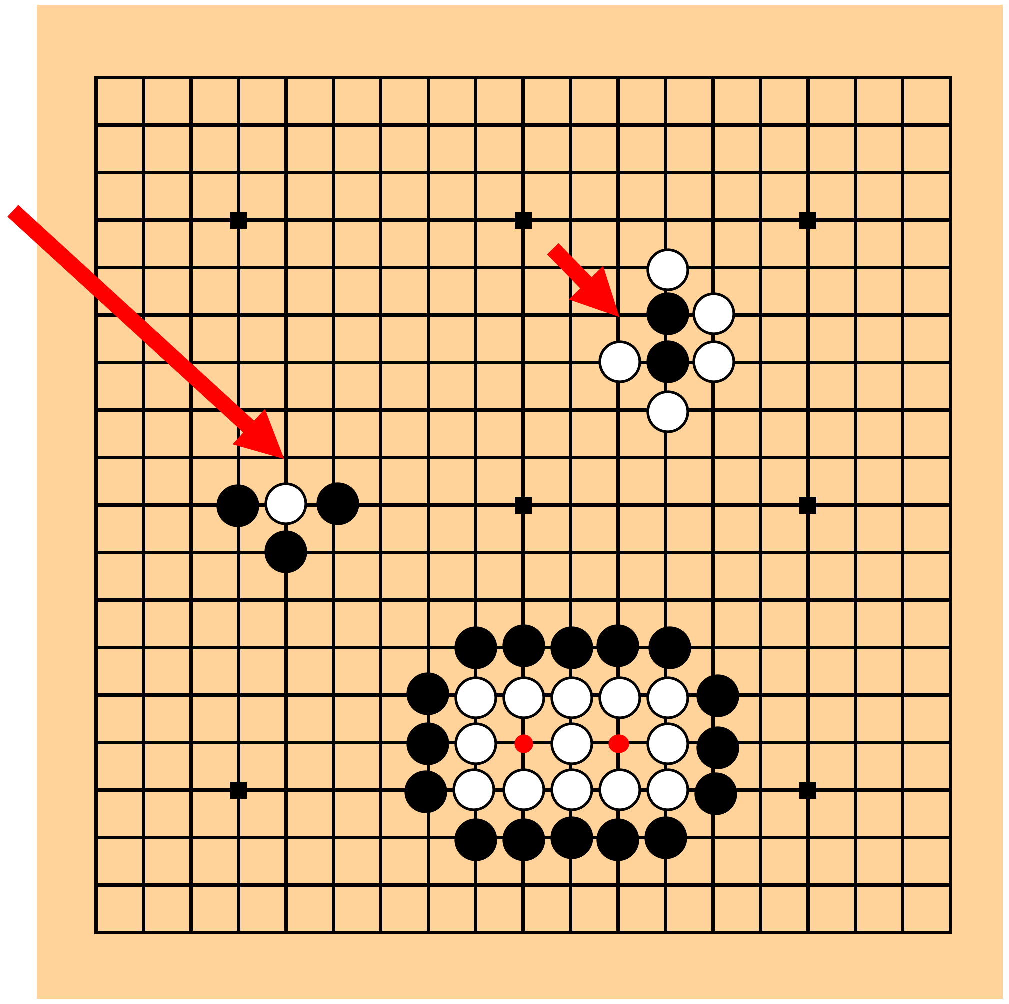
Some more rules…



The goal is to 'capture territory'

- The final score is the number of empty spaces you 'surround' plus any stones you capture

Some more rules…



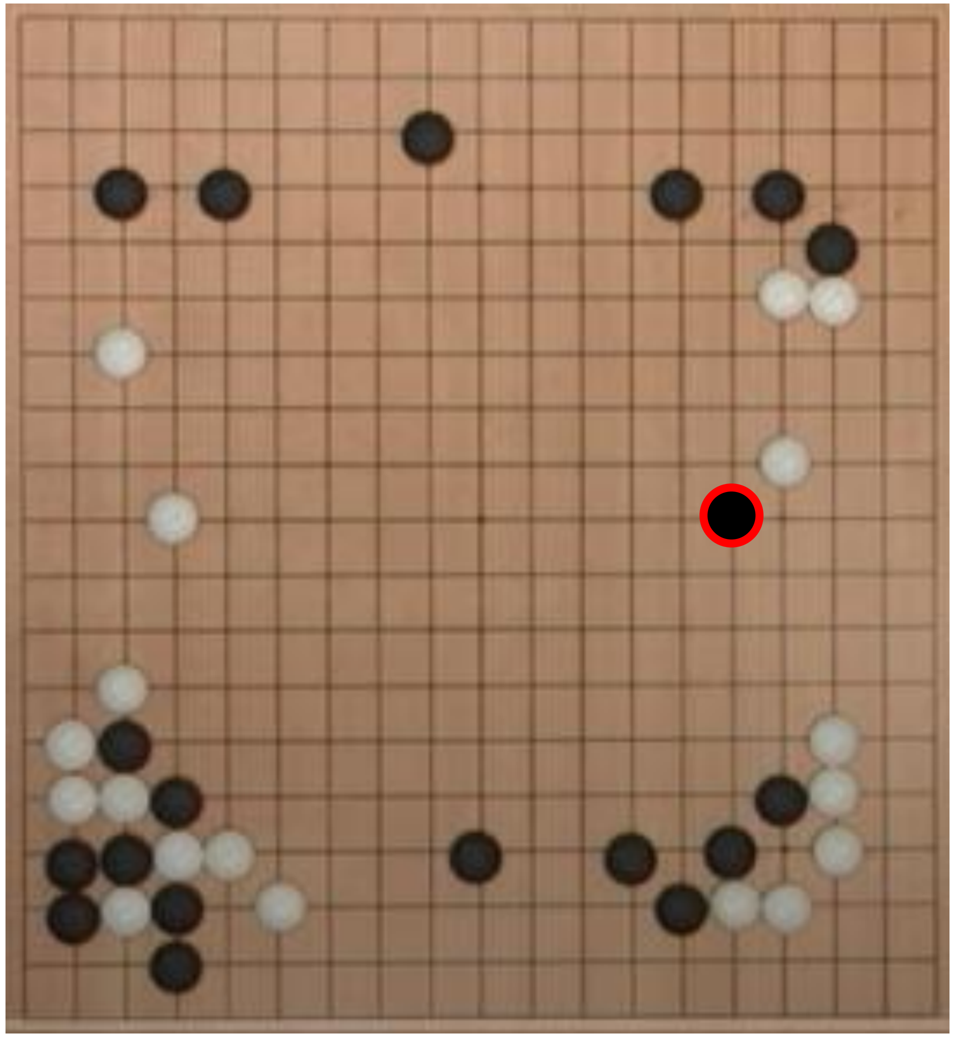
A stone is captured if it has no liberties (i.e., free spots adjacent either up, down, left or right)

Adjacent stones of the same colour, share liberties (and are captured together if surrounded)

Either top group could be captured by placing a single stone

The bottom white group cannot be captured. Placing a black stone in either liberty (red dots) would be suicide and you can't place two stones simultaneously

Go is complex



How do you evaluate?

Is this a good move?

https://www.youtube.com/watch?v=l-GsfyVCBu0

AlphaGo vs Lee Sedol

Go to 1:17-ish

Why is Go so hard?

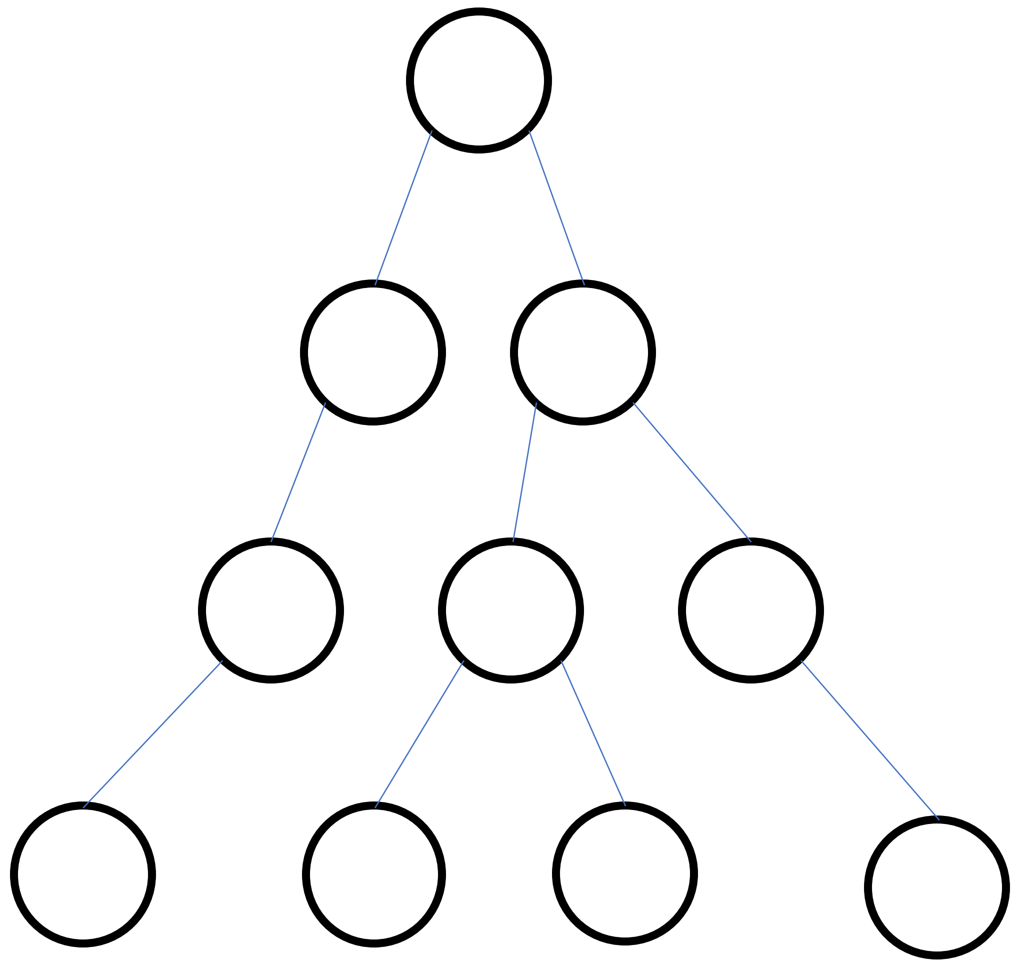
- A big search isn't going to work (too much branching)
- We can't easily evaluate a board state (i.e., no alpha-beta pruning)

What can we do?

Monte Carlo Tree Search

The 'Algorithm'

- Select
- Expand
- Simulate
- Back Up/Propagate

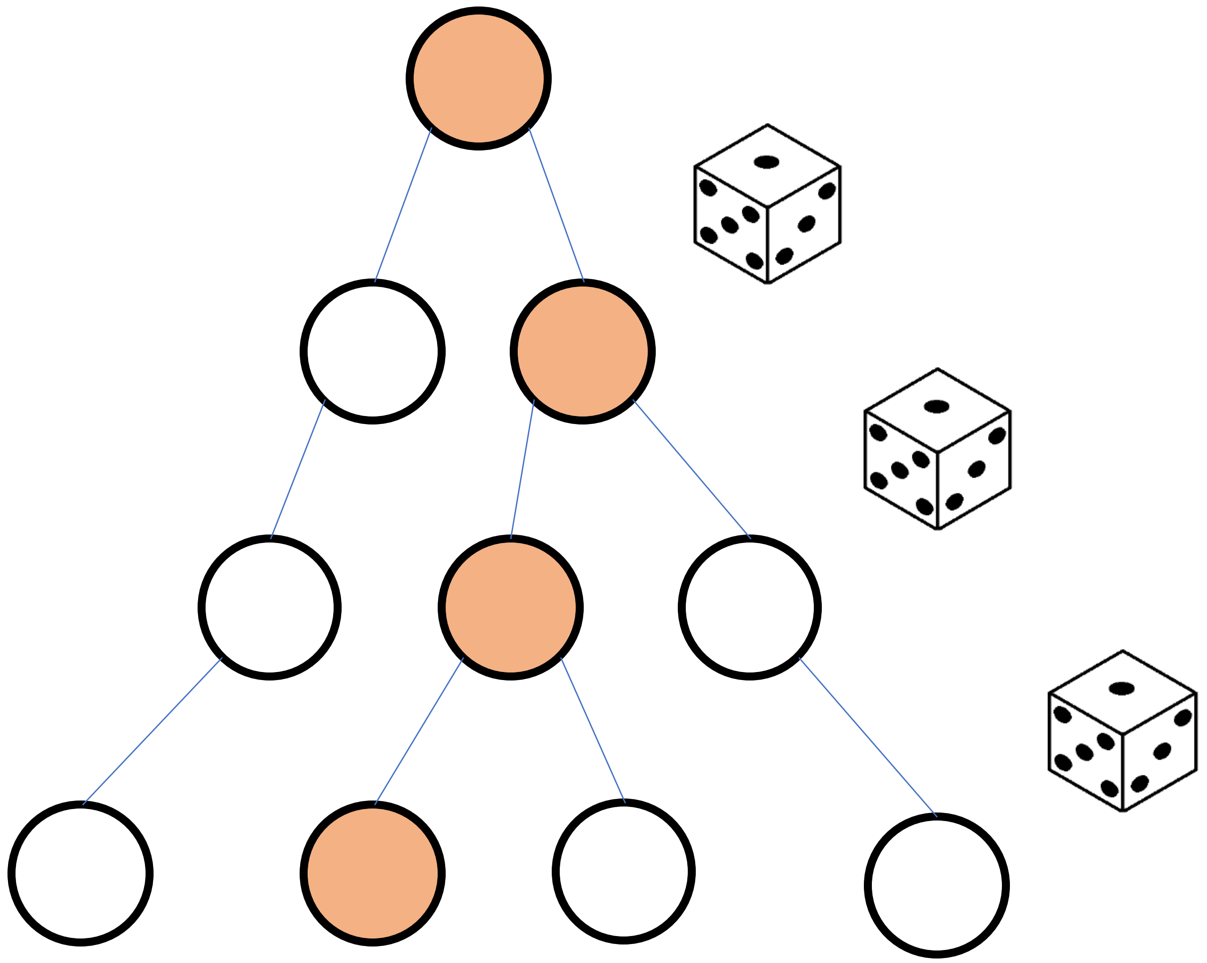


Monte Carlo Tree Search

What if we pick a random move, go down the tree and continue until we get to the 'end'.

We will get some idea of the quality of the 'first move' (either good or bad or some score).

We can then take that score and propagate it back up.



This is Monte Carlo Rollout

Monte Carlo Tree Search

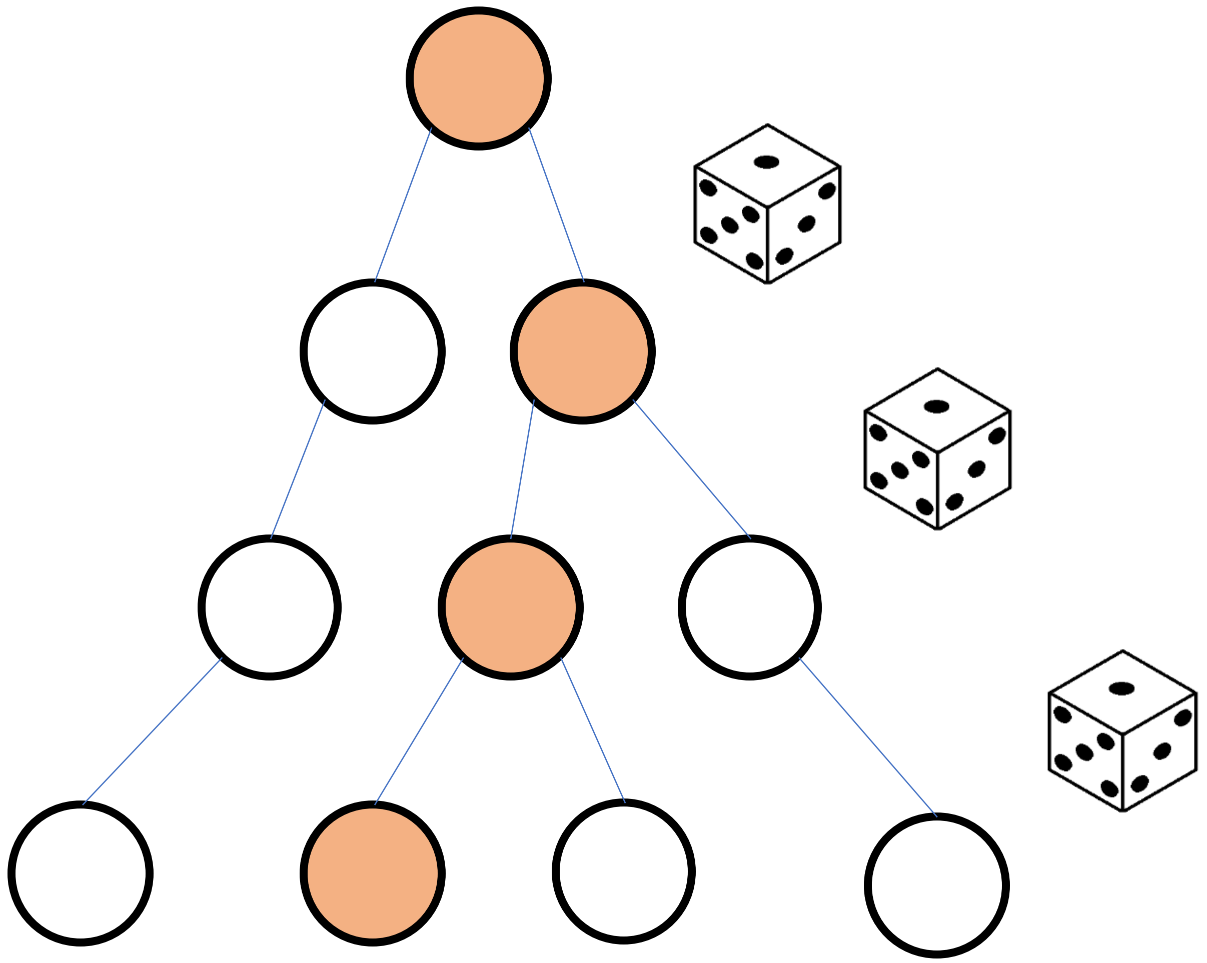
We continue this process until we run out of time and then just 'pick the winner' (led to better outcomes more often).

Pros:

- It is a bit like sampling

Cons:

- Random isn't always the best



Tree Policy

What if instead of purely ‘random’, we make some pseudo-random rules to make searching more efficient

Tree policy generates a formula for determining which of many nodes ‘should be explored’

$$\text{argmax UCT}(v', v) = \overset{\text{exploitation}}{\frac{Q(v')}{N(v')}} + \overset{\text{exploration}}{c\sqrt{\frac{\ln N(v)}{N(v')}}}$$

v’ is a child of node **v**

N(v): number of times a node has been visited

Q(v): Total reward of all playouts that passed through node v.

c: Magic number (usually $2^{0.5}$) that balances exploration verses exploitation

UCT = **U**pper **C**onfidence Bound applied to **t**rees (a solution from multi-armed bandit problems)
https://www.computer-go.info/resources/bandit.html

Tree Policy: Intuitions

exploitation

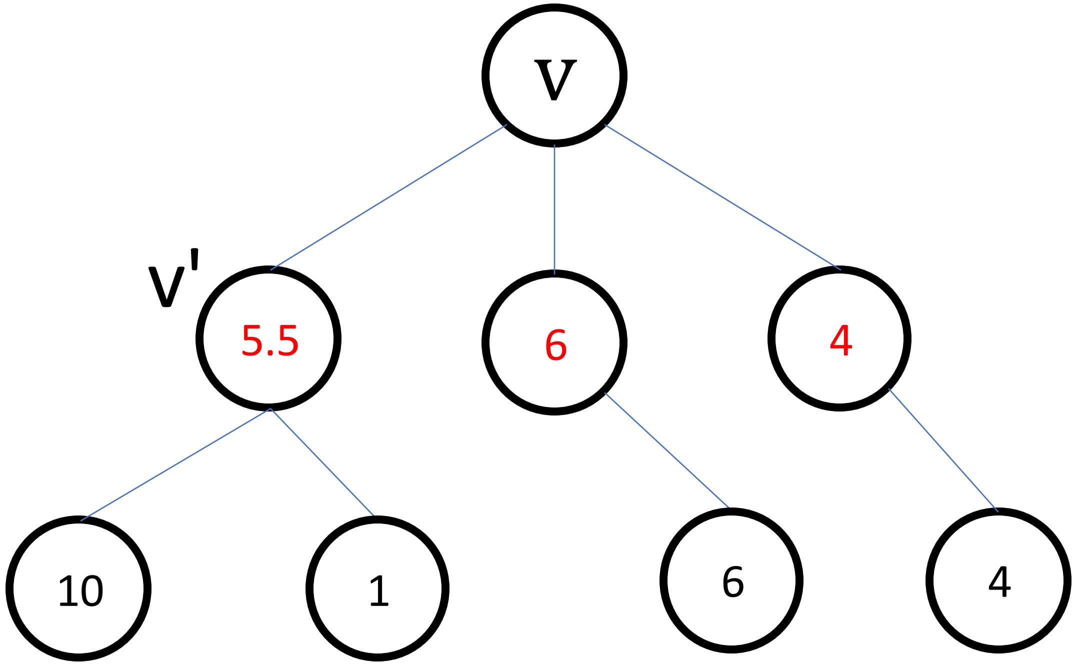
$$\text{UCT} = \frac{Q(v')}{N(v')} + c\sqrt{\frac{\ln N(v)}{N(v')}}$$

N(v): number of times a node has been visited

Q(v): Total reward of all playouts that passed through node v.

c: Magic number (usually $2^{0.5}$) that balances exploration verses exploitation

Total reward divided by number of times visited (which is kind of the average score of ‘going that way’)



Tree Policy: Intuitions

$$UCT = \frac{Q(v')}{N(v')} + \overset{\text{exploration}}{c\sqrt{\frac{\ln N(v)}{N(v')}}}$$

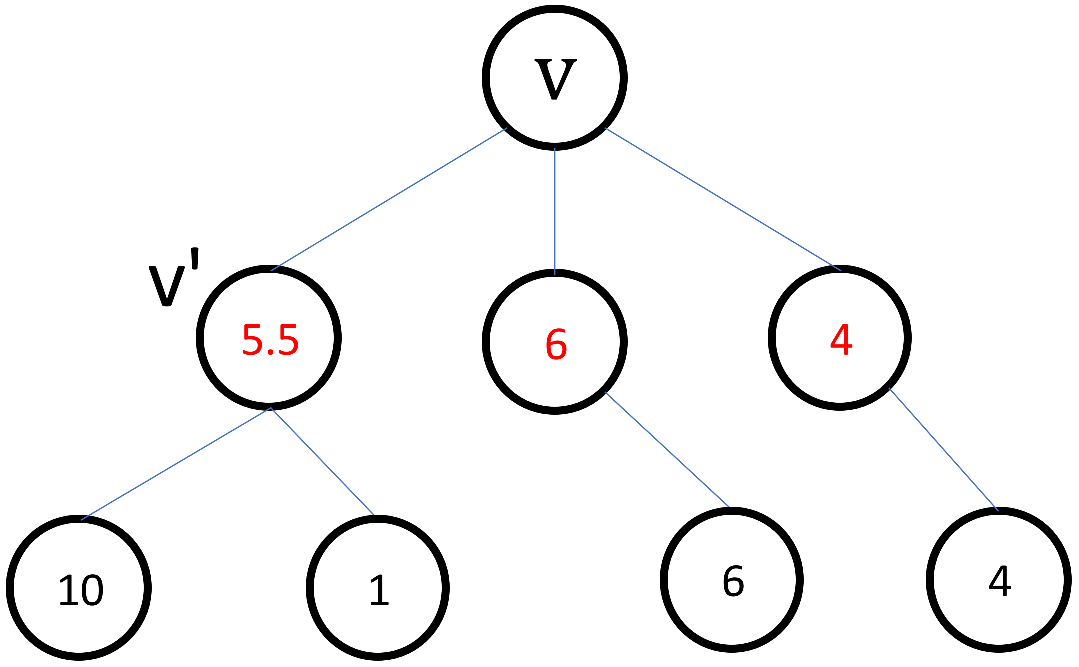
N(v): number of times a node has been visited

Q(v): Total reward of all playouts that passed through node v.

c: Magic number (usually $2^{0.5}$) that balances exploration verses exploitation

Number of visits of the parent divided by visits to the child

… children with less visits are more likely to be searched



Tree Policy: Intuitions

What is the logic?

- If I have a good score by trying a move… I want to keep exploring that move because it looks like a ‘good idea’
- If I have not explored a move at all… I might be missing out on a great move!

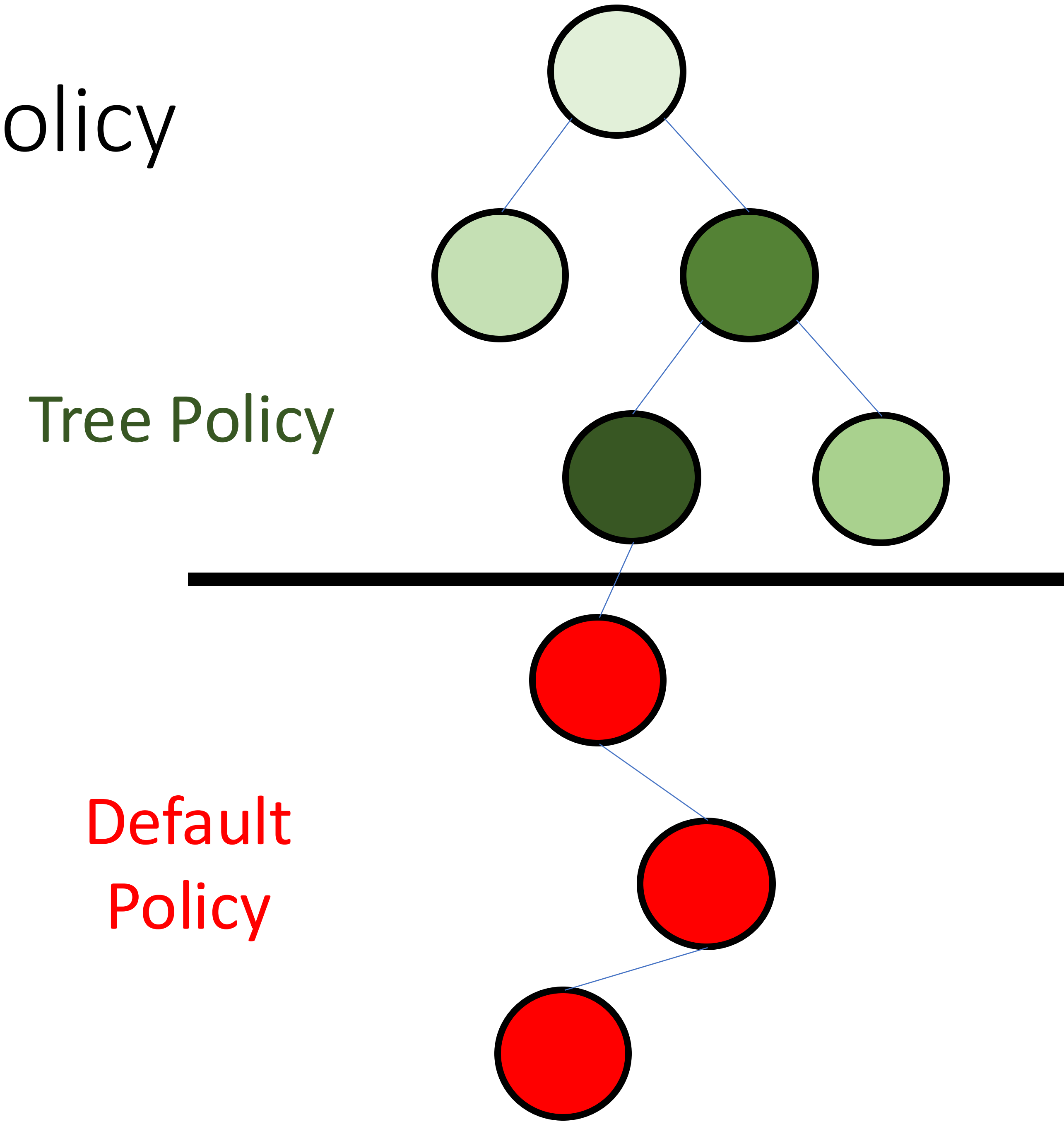
The equation minimises:

“expected regret”

Tree Policy vs Default Policy

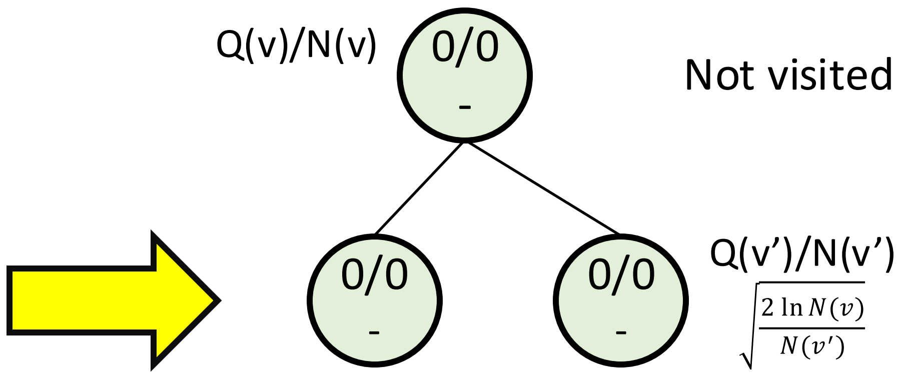
If there are children... Expand the other children.

Otherwise (i.e., first time down) random all the way down.



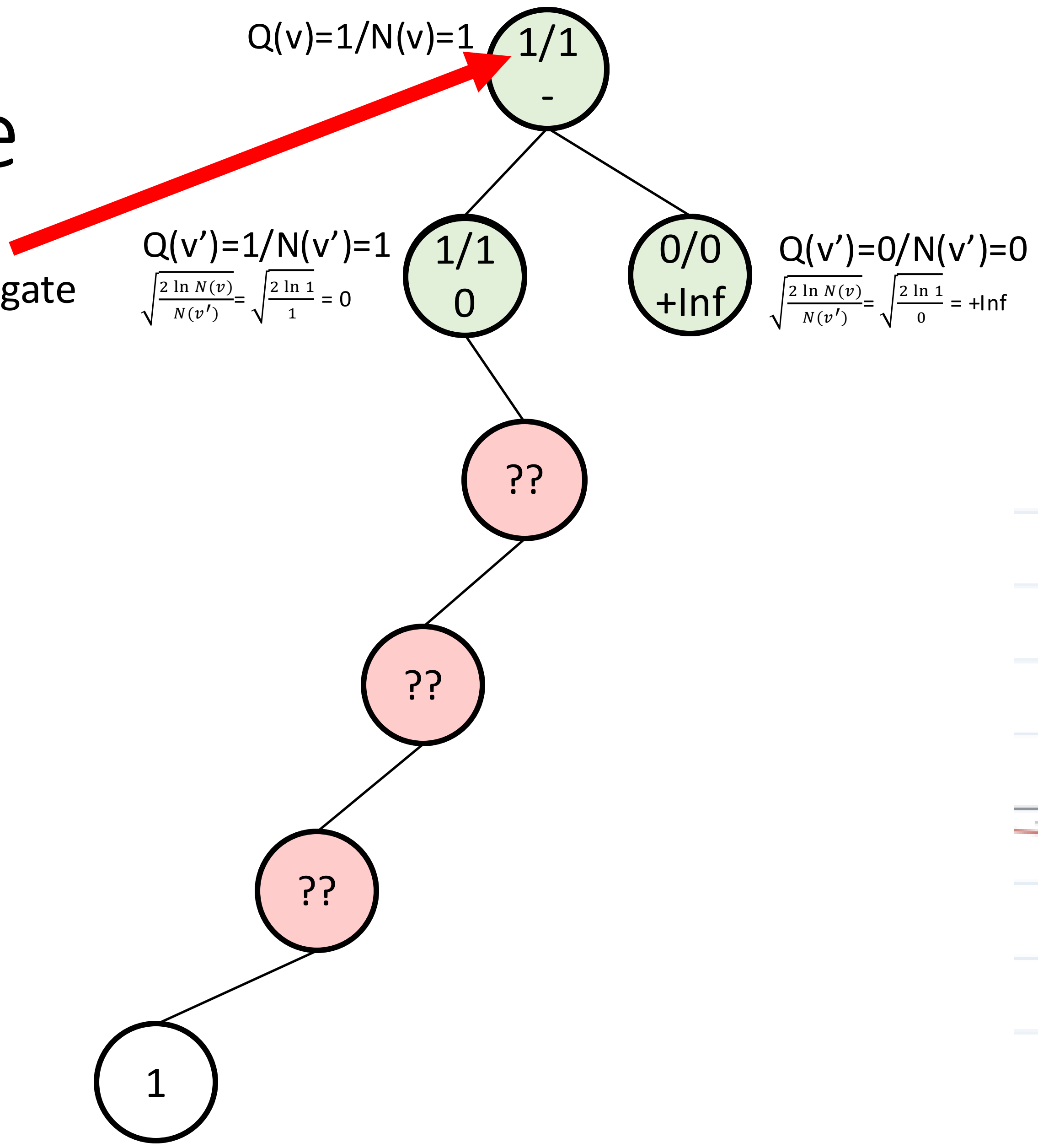
Example

Pick a random child

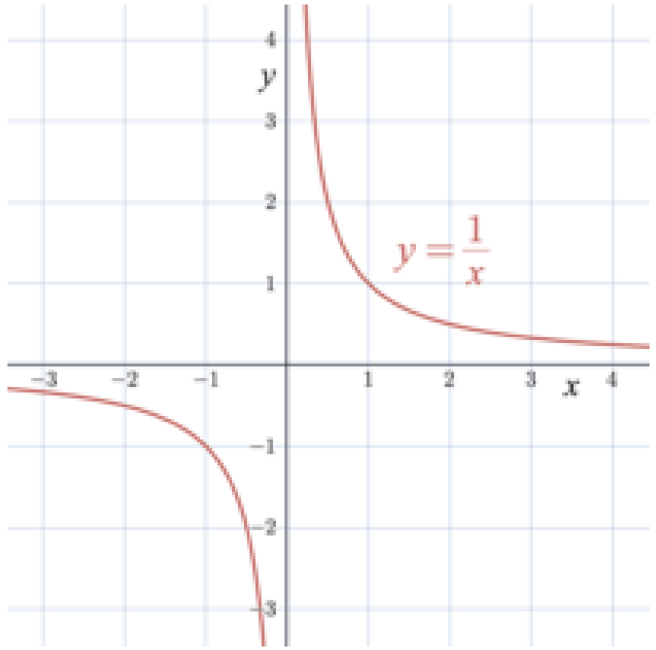


Simulate

Example



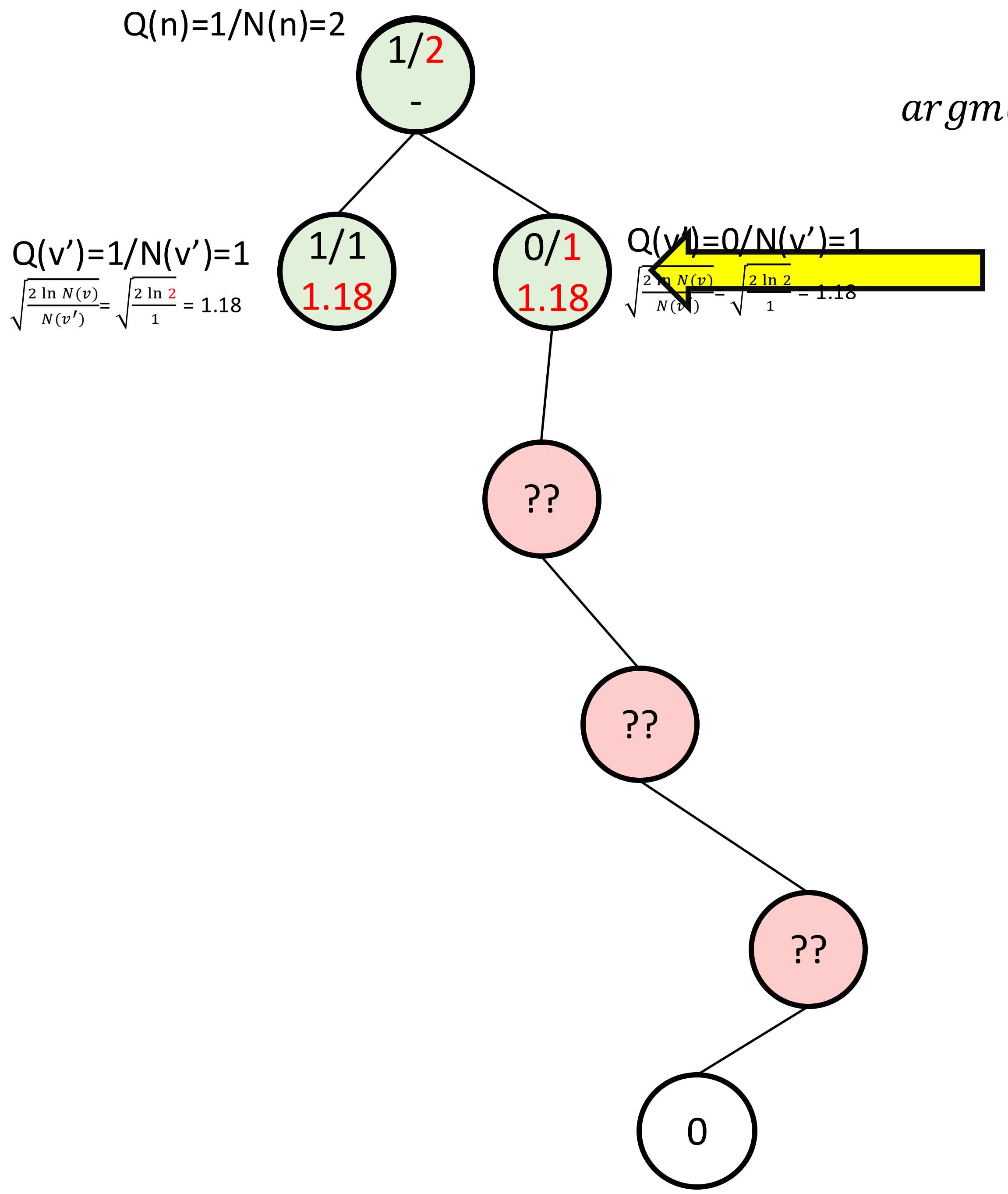
Back Propagate
Q(v)=1/N(v)=1
1/1
-
Q(v')=1/N(v')=1
$\sqrt{\frac{2\ln N(v)}{N(v')}} = \sqrt{\frac{2\ln 1}{1}} = 0$
1/1
0
0/0
+Inf
Q(v')=0/N(v')=0
$\sqrt{\frac{2\ln N(v)}{N(v')}} = \sqrt{\frac{2\ln 1}{0}} = +\text{Inf}$
??
??
??
1



Divide by zero???

Example

$$\operatorname{argmax} \frac{Q(v')}{N(v')} + \sqrt{\frac{2 \ln N(v)}{\boldsymbol{N(v')}}}$$


Q(n)=1/N(n)=2
1/2
-
Q(v')=1/N(v')=1
$\sqrt{\frac{2 \ln N(v)}{N(v')}} = \sqrt{\frac{2 \ln 2}{1}} = 1.18$
1/1
1.18
0/1
1.18
Q(v')=0/N(v')=1
$\sqrt{\frac{2 \ln N(v)}{N(v')}} = \sqrt{\frac{2 \ln 2}{1}} = 1.18$
??
??
??
0

Pick the ‘better’ child

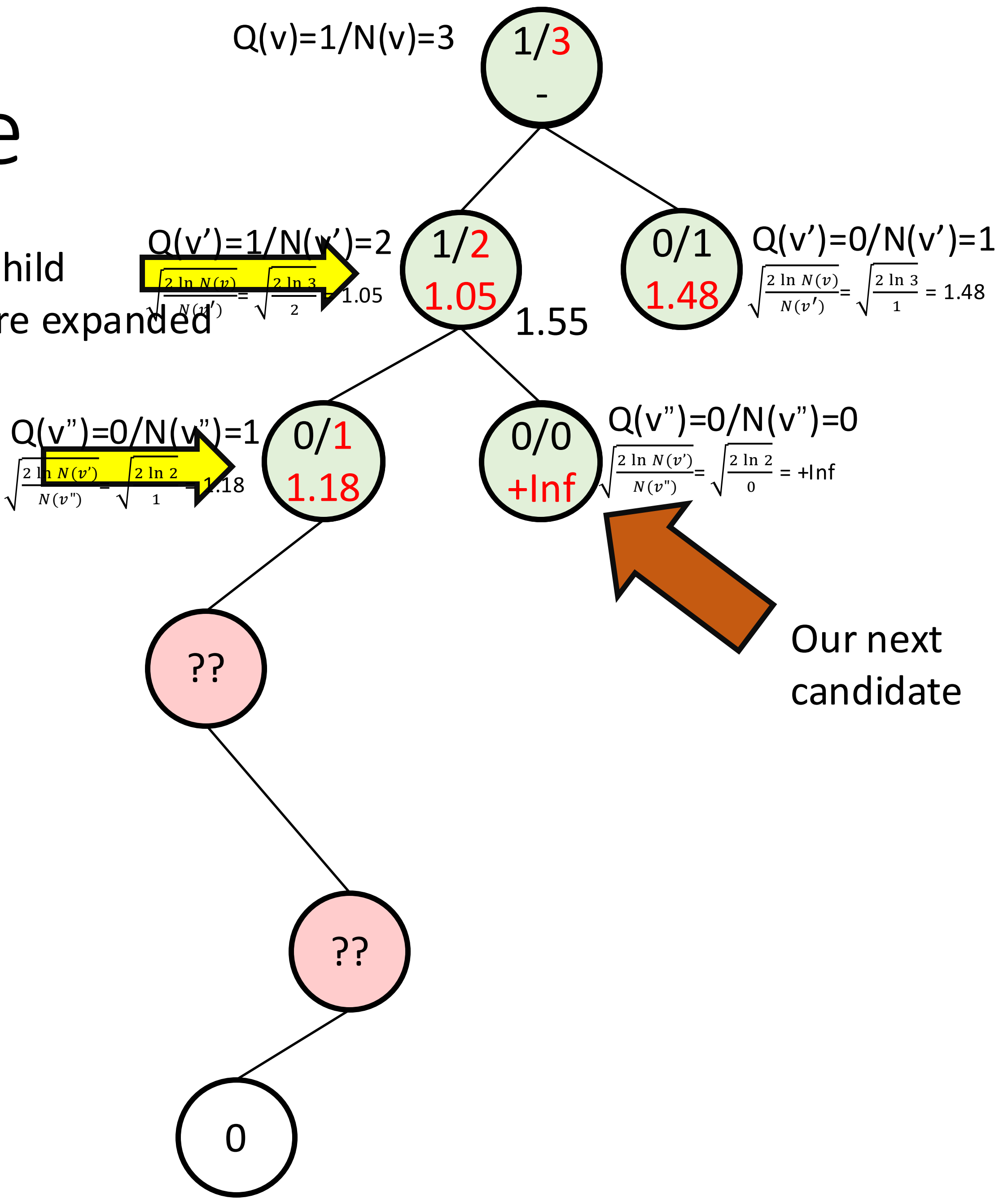
Example

$$\operatorname{argmax} \frac{Q(v')}{N(v')} + \sqrt{\frac{2\ln N(v)}{\boldsymbol{N(v')}}}$$

Pick the ‘better’ child

Visited children are expanded

Pick a random child

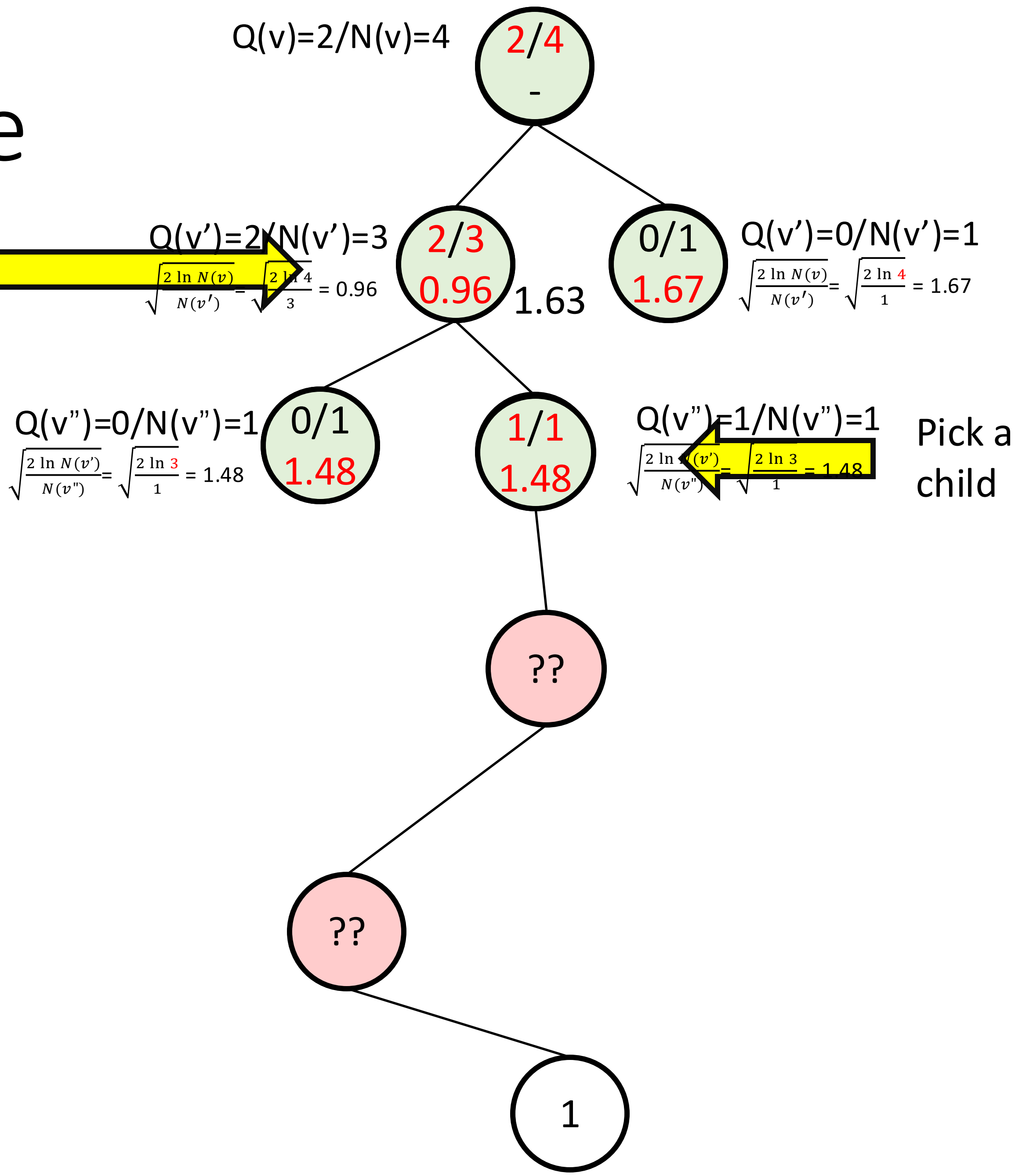


Q(v)=1/N(v)=3
1/3
-
Q(v’)=1/N(v’)=2
$\sqrt{\frac{2\ln N(v)}{N(v')}} = \sqrt{\frac{2\ln 3}{2}} = 1.05$
1/2
1.05
1.55
0/1
1.48
Q(v’)=0/N(v’)=1
$\sqrt{\frac{2\ln N(v)}{N(v')}} = \sqrt{\frac{2\ln 3}{1}} = 1.48$
Q(v”)=0/N(v”)=1
$\sqrt{\frac{2\ln N(v')}{N(v'')}} = \sqrt{\frac{2\ln 2}{1}} = 1.18$
0/1
1.18
0/0
+Inf
Q(v”)=0/N(v”)=0
$\sqrt{\frac{2\ln N(v')}{N(v'')}} = \sqrt{\frac{2\ln 2}{0}} = +\mathrm{Inf}$
Our next candidate
??
??
0

Example

Pick the 'better' child
Visited children are expanded

$$\mathrm{argmax}\ \frac{Q(v')}{N(v')} + \sqrt{\frac{2\ln N(v)}{\boldsymbol{N(v')}}}$$

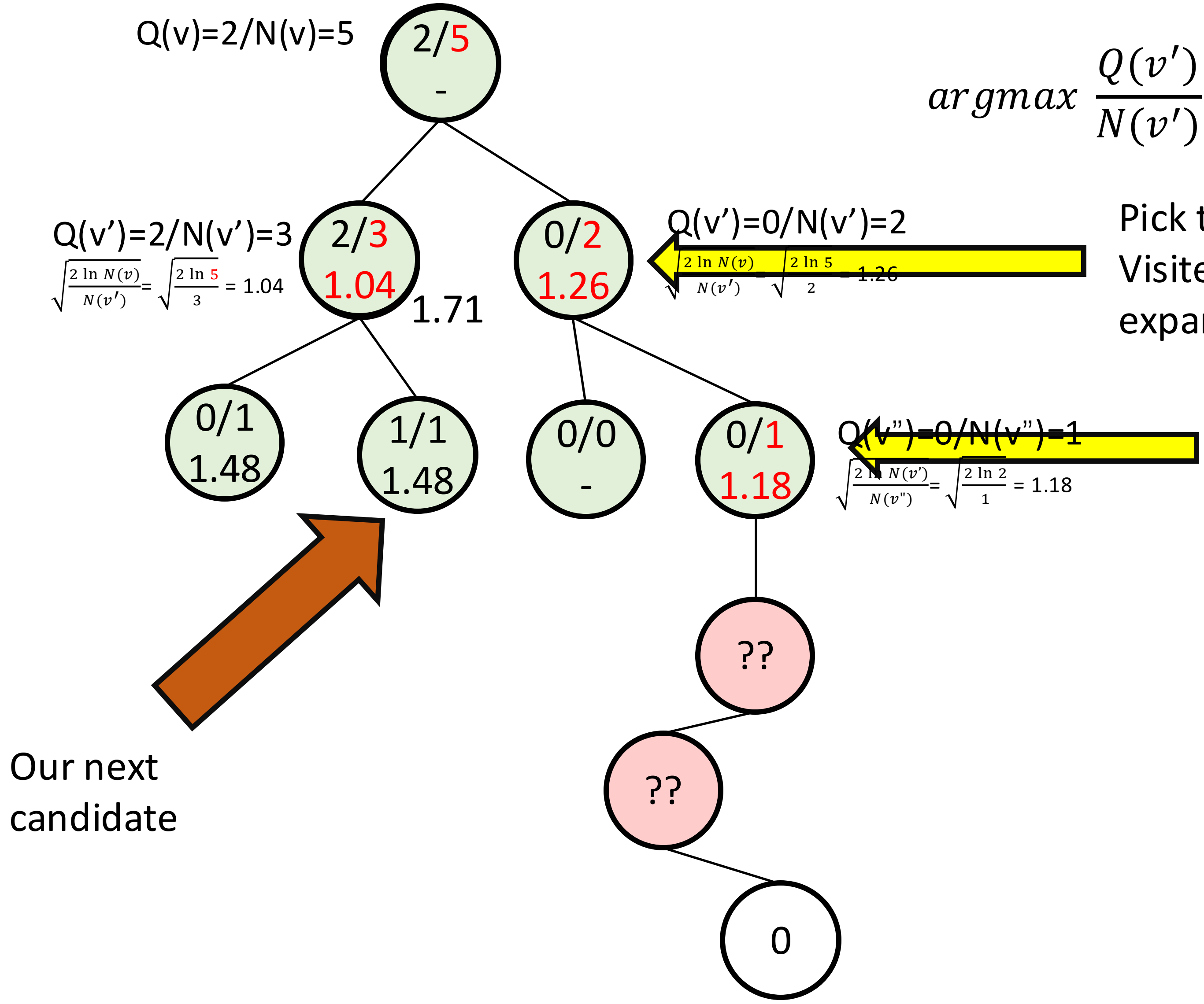


Q(v)=2/N(v)=4
2/4
-
Q(v')=2/N(v')=3
$\sqrt{\frac{2\ln N(v)}{N(v')}} = \sqrt{\frac{2\ln 4}{3}}$ = 0.96
2/3
0.96
1.63
0/1
1.67
Q(v')=0/N(v')=1
$\sqrt{\frac{2\ln N(v)}{N(v')}} = \sqrt{\frac{2\ln 4}{1}}$ = 1.67
Q(v")=0/N(v")=1
$\sqrt{\frac{2\ln N(v')}{N(v")}} = \sqrt{\frac{2\ln 3}{1}}$ = 1.48
0/1
1.48
1/1
1.48
Q(v")=1/N(v")=1
$\sqrt{\frac{2\ln N(v')}{N(v")}} = \sqrt{\frac{2\ln 3}{1}}$ = 1.48
??
??
1

Pick an unexplored child

Example

$$\operatorname{argmax} \frac{Q(v')}{N(v')} + \sqrt{\frac{2\ln N(v)}{\boldsymbol{N(v')}}}$$


Q(v)=2/N(v)=5
2/5
-
Q(v')=2/N(v')=3
$\sqrt{\frac{2\ln N(v)}{N(v')}} = \sqrt{\frac{2\ln 5}{3}} = 1.04$
2/3
1.04
1.71
0/2
1.26
Q(v')=0/N(v')=2
$\sqrt{\frac{2\ln N(v)}{N(v')}} = \sqrt{\frac{2\ln 5}{2}} = 1.26$
0/1
1.48
1/1
1.48
0/0
-
0/1
1.18
Q(v")=0/N(v")=1
$\sqrt{\frac{2\ln N(v')}{N(v")}} = \sqrt{\frac{2\ln 2}{1}} = 1.18$
??
??
0
Our next candidate

Pick the 'better' child
Visited children are expanded

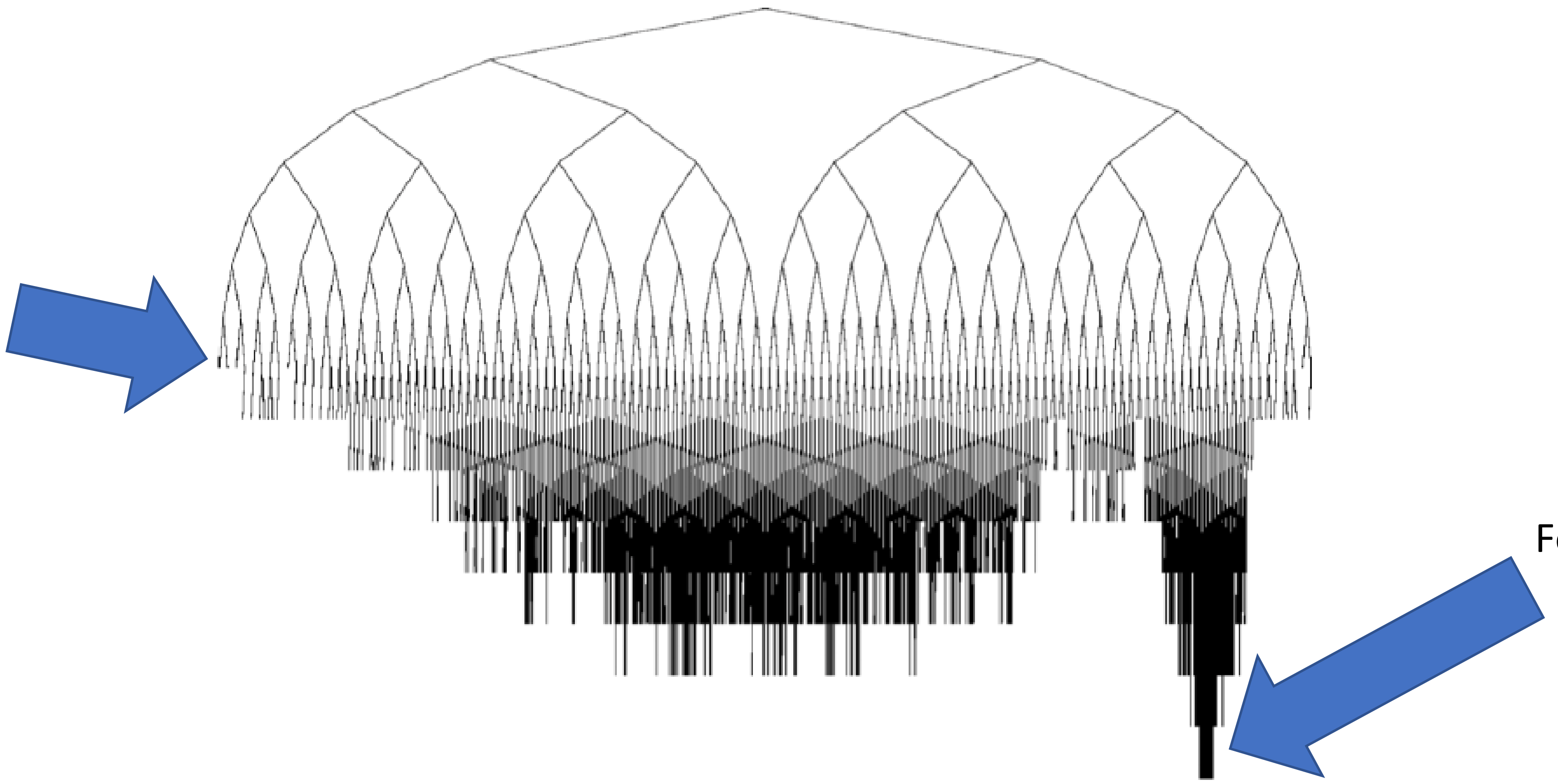
Pick a random child

$$\sqrt{\frac{2\ln 2}{1}}=1.18$$

$$\sqrt{\frac{2\ln 5}{2}}=1.26$$

$$\sqrt{\frac{2\ln 5}{3}}=1.04$$

What it ends up looking like



How does this work in Go?

#1: We are slowly expanding a small tree.

#2: We pick ‘the best leaf’

#3: We expand only that leaf

#4: We play to the end of the game from that leaf (we don’t expand our tree, we just simulate)

#5: We mark off all the nodes in our tree with the result of the ‘random game’

Exploitation:

- The tree policy formula indicates that a particular leaf is a ‘good one to follow’ (i.e., leads to many wins)

Exploration:

- The tree policy formula indicates that a leaf is ‘unexplored’ (there could be wins here, we don’t know yet)

Monte Carlo Search vs Minimax/α-β pruning

Similarities

We evaluate games with a ‘win-loss’.

We propagate stuff up the tree to determine the value of moves.

We pick the ‘best’ utility.

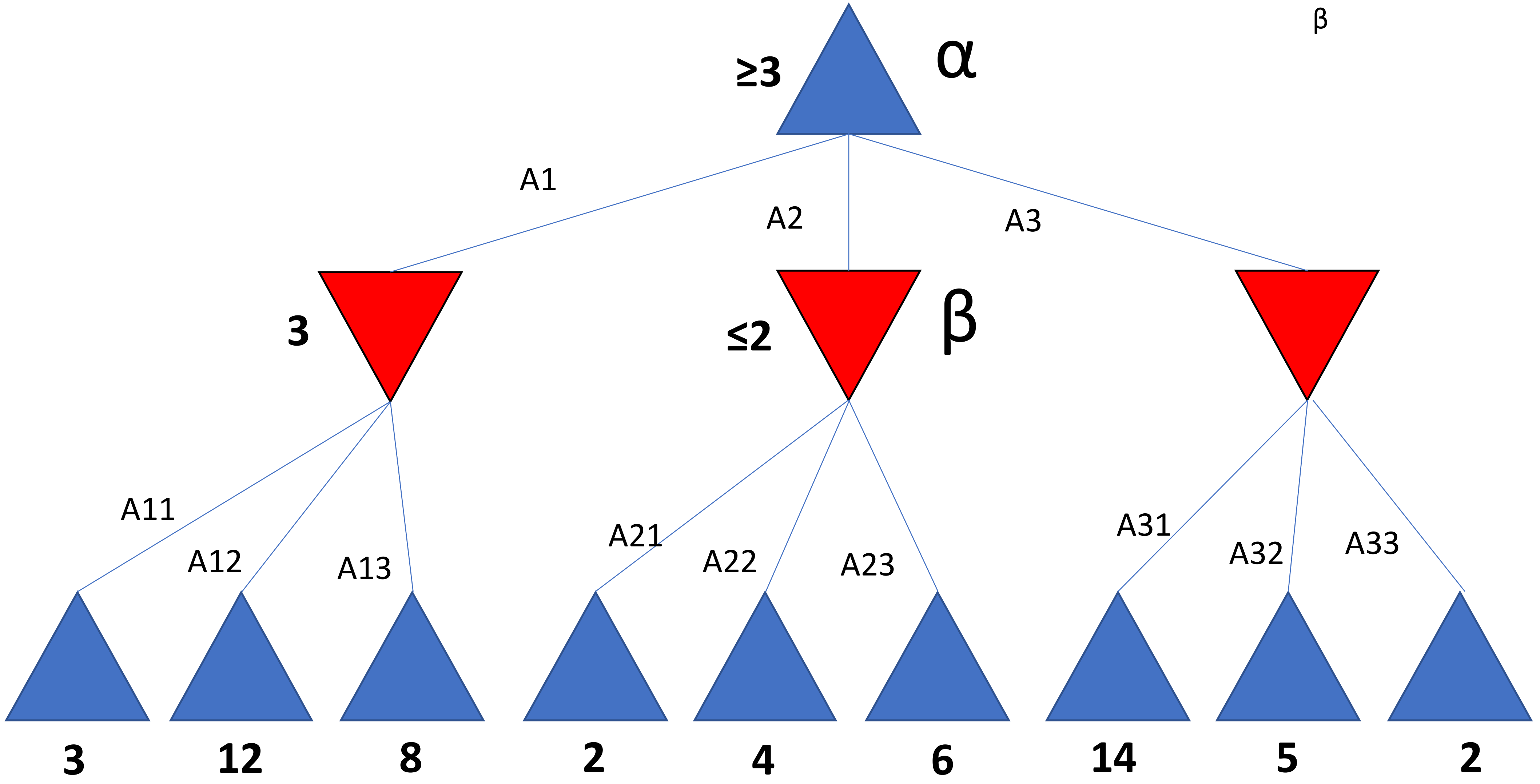
Differences

We do <u>not</u> explore the whole tree

Monte Carlo can be described as an ‘intelligent search’. We are choosing where to look and what to ignore (it can obviously be wrong)

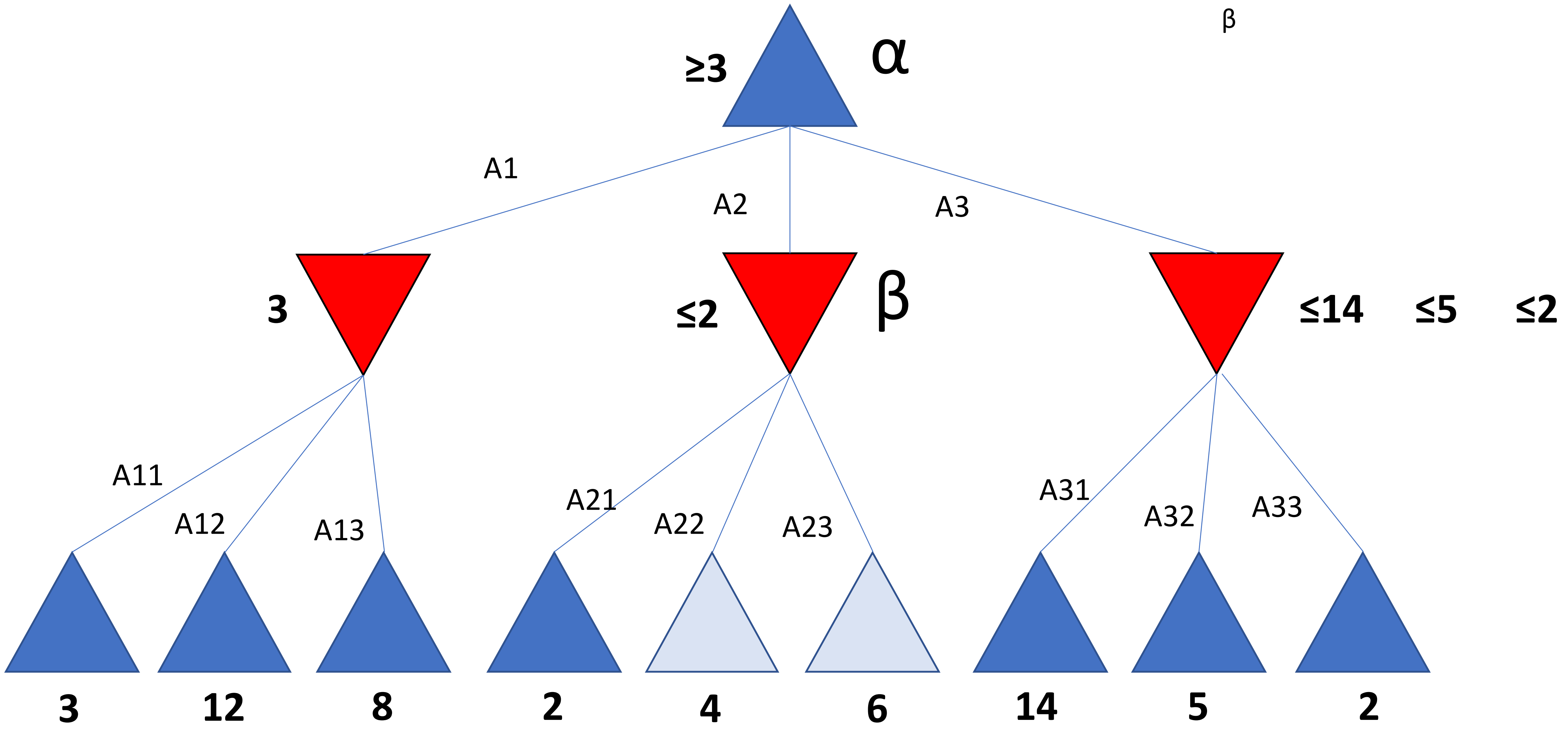
α-β Pruning

```
if (α > β):
// Stop Searching leaves of β
```



α-β Pruning

if (α > β):
// Stop Searching leaves of β



Monte-Carlo Tree Search and Go

2008: MoGo becomes a ‘dan’ level in 9x9 Go

This had some hard-codes.

If you are attacked (atari – one move off being taken, try to save your stones)

Else if there is an empty location with no neighbours, try play there

Else if there is a specific pattern, play a specific move

Else if you can capture stones

Else some randomness

2008-2012 MoGo & MoGoTW make incremental improvements beating higher and higher level players

2015: AlphaGo beats Fan Hui (European Champion, 2-dan, 5-0)

2016: AlphaGo beats Lee Sedol (World Grandmaster, 9-dan, 4-1)

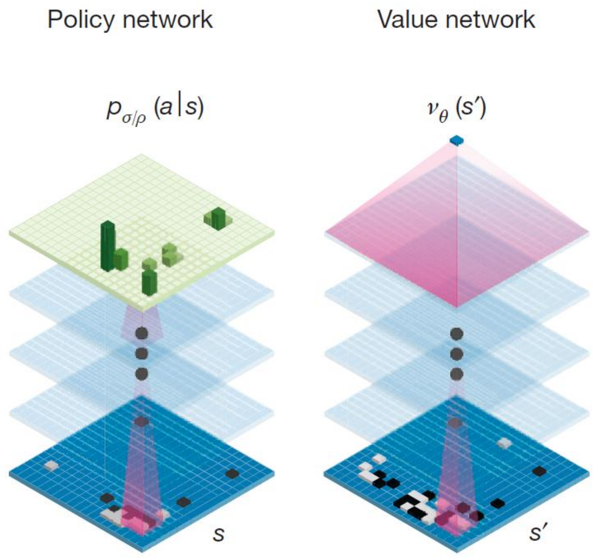
AlphaGo? How does it work?

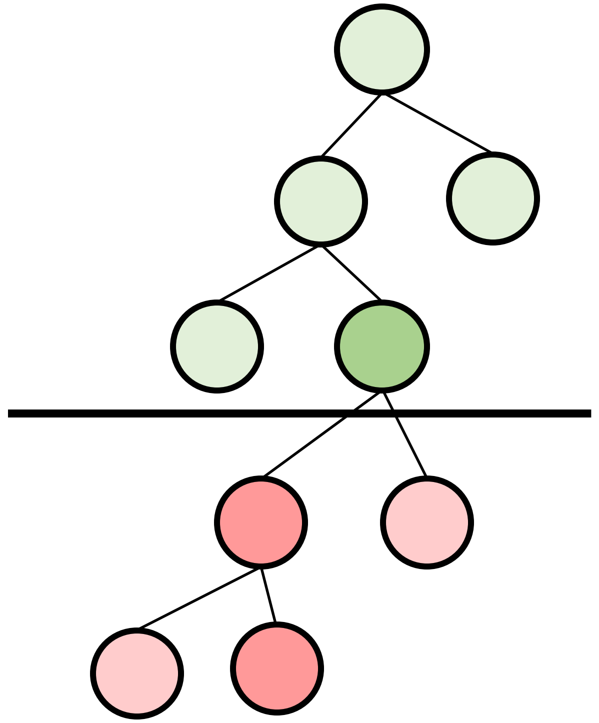
Use Machine Learning

- Used supervised learning on professional games followed by unsupervised learning on self-play

What is it learning?

- Given the current board, what is a good move? – by policy network
  - Narrow down the search space
- What board position will win the game? – by value network
  - More accurate and efficient evaluation of a board position





AlphaZero and so on...

AlphaZero (2017)

- Does more than just GO (Shogi, Chess)
- No pretraining on human expert games, completely self-play from scratch
- Merged single policy and utility prediction network

MuZero (2019)

- No knowledge of the game rules, can play Atari Game

EfficientZero (2021)

- Higher sample efficiency, reduced computational cost.

Questions?