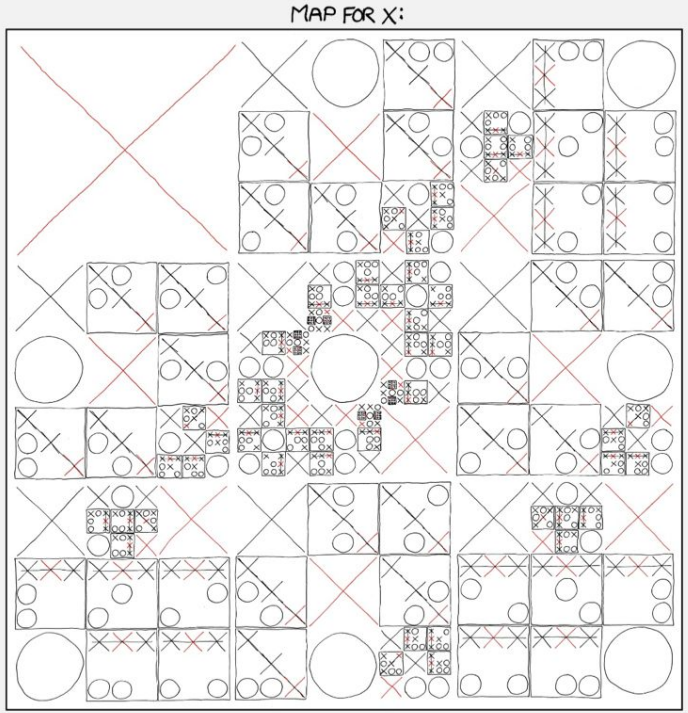
Monte Carlo Tree Search

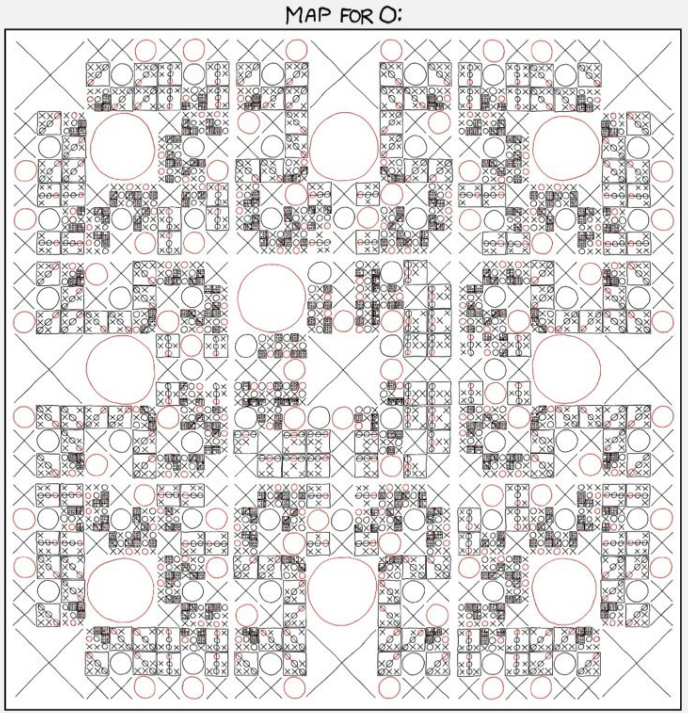
Djordje Grbic

A little recap

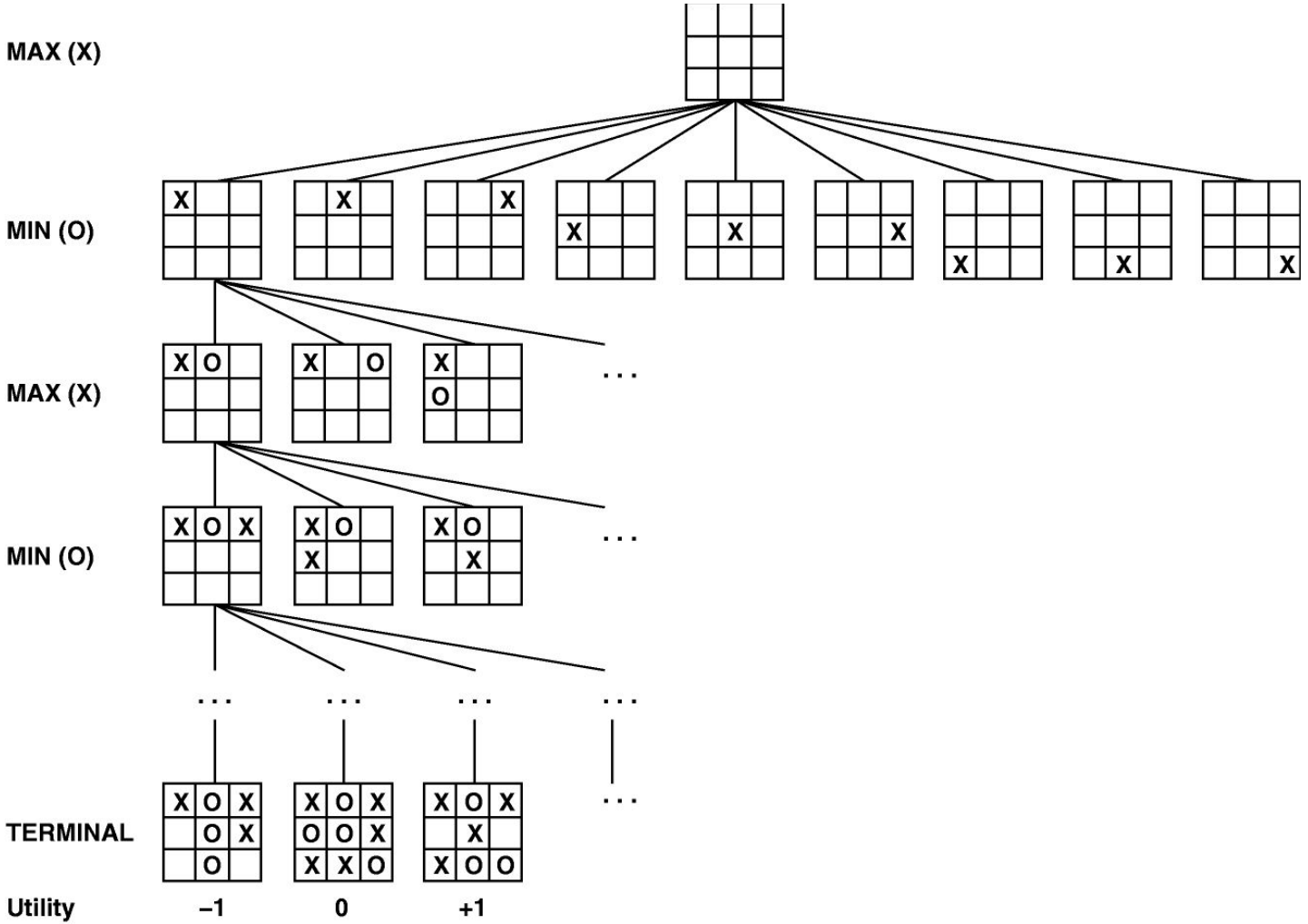
Tree search algorithms

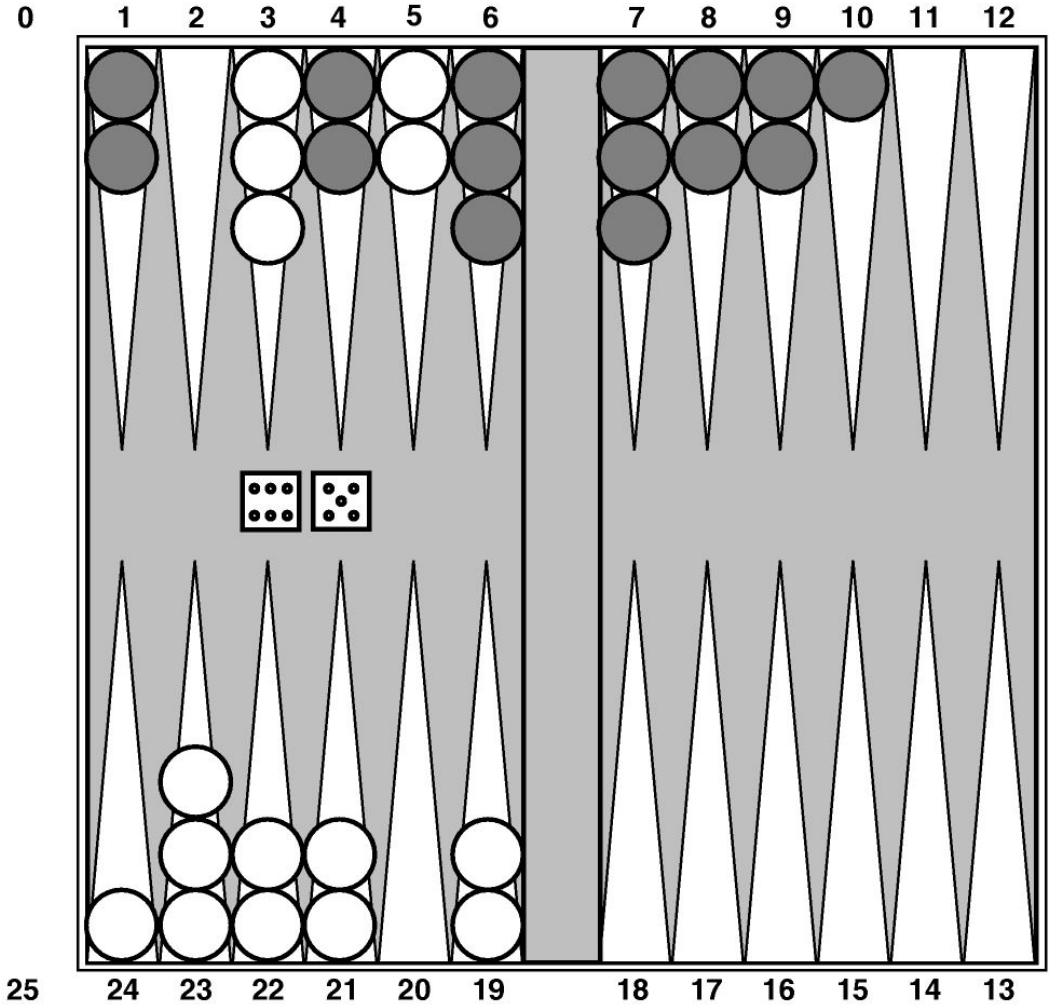
TREE SEARCH—THE NAÏVE APPROACH – Optimal game

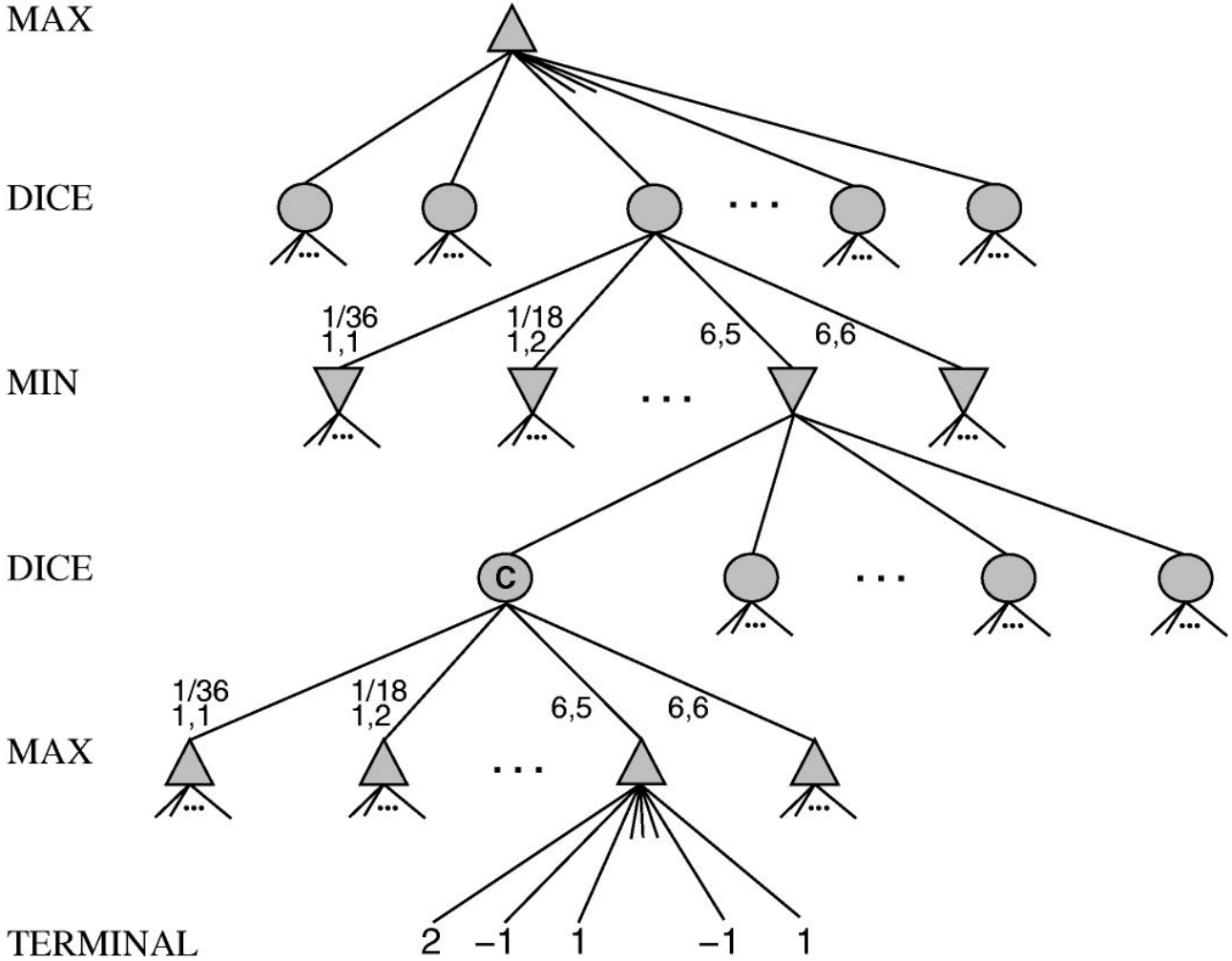




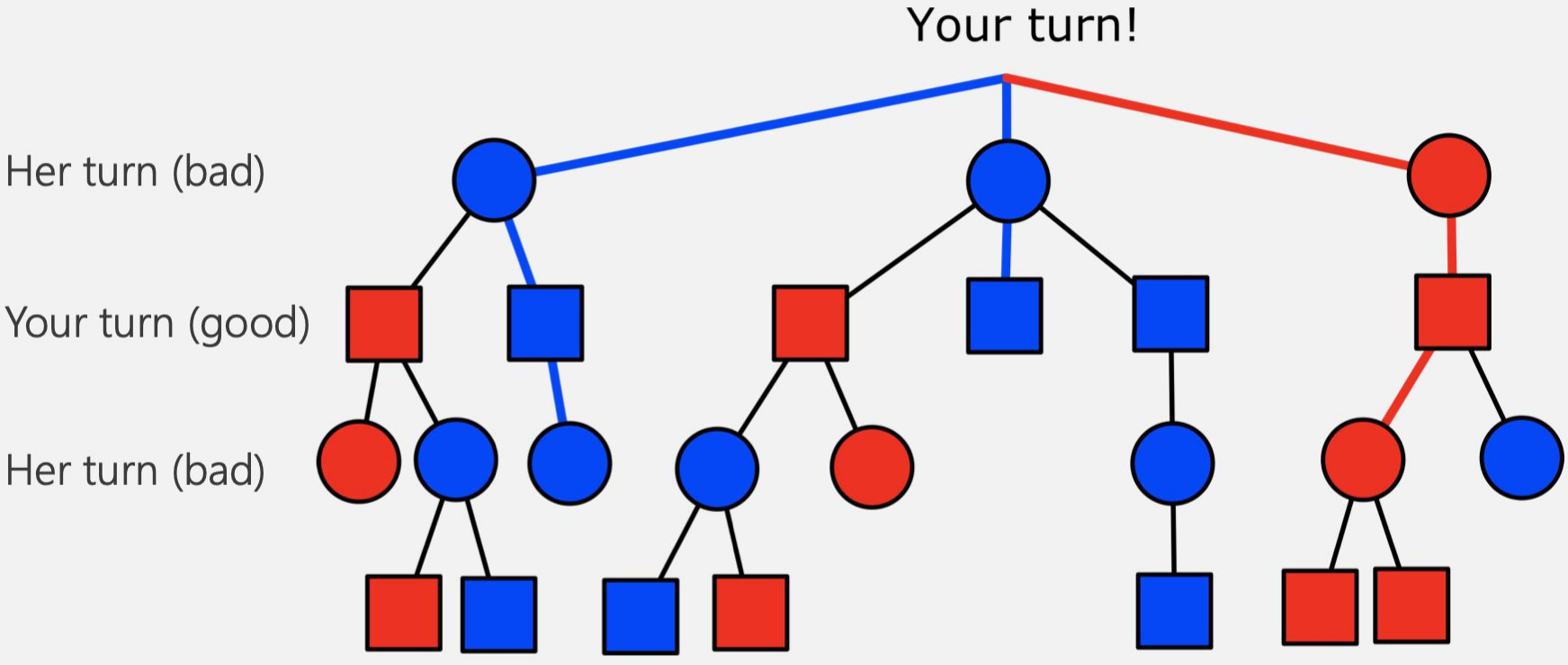
https://xkcd.com/832/


MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
–1
0
+1

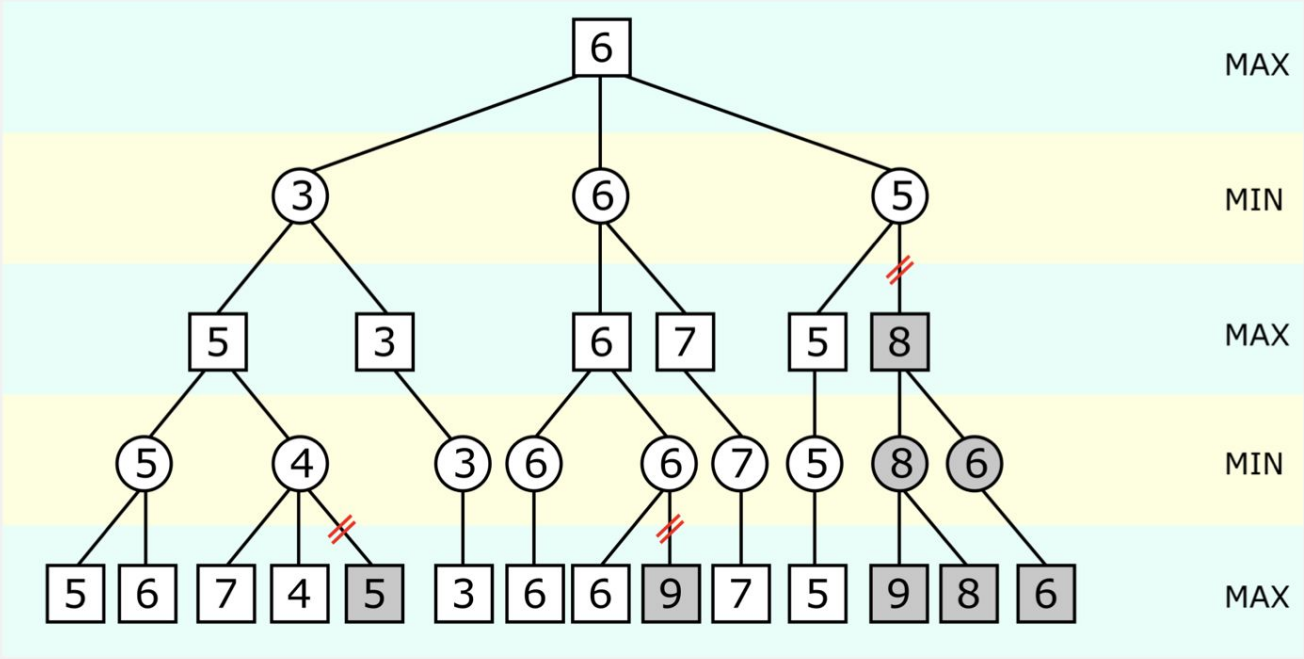

0
1
2
3
4
5
6
7
8
9
10
11
12
25
24
23
22
21
20
19
18
17
16
15
14
13


MAX
DICE
MIN
DICE
MAX
TERMINAL
1/36
1,1
1/18
1,2
6,5
6,6
C
2
−1
1
−1
1

MINIMAX



ALPHA-BETA PRUNING



STRIPS

- Objects

  J (for Jaguar, a car!), G (for Garage, a place)

- Facts about the world

  At(x), Have(x), Sells(x, y)

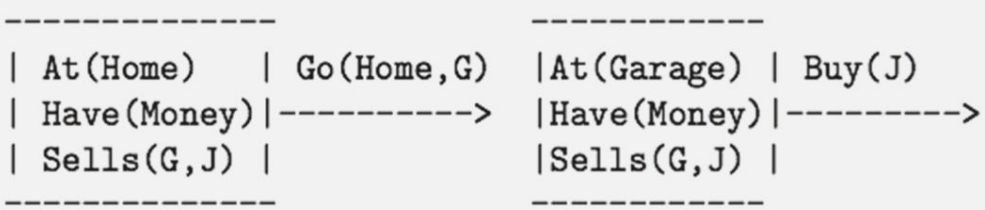
- Goals

  Have(J)

- Actions

  Buy(x), Go(x, y)

STRIPS—FORWARD PLANNING

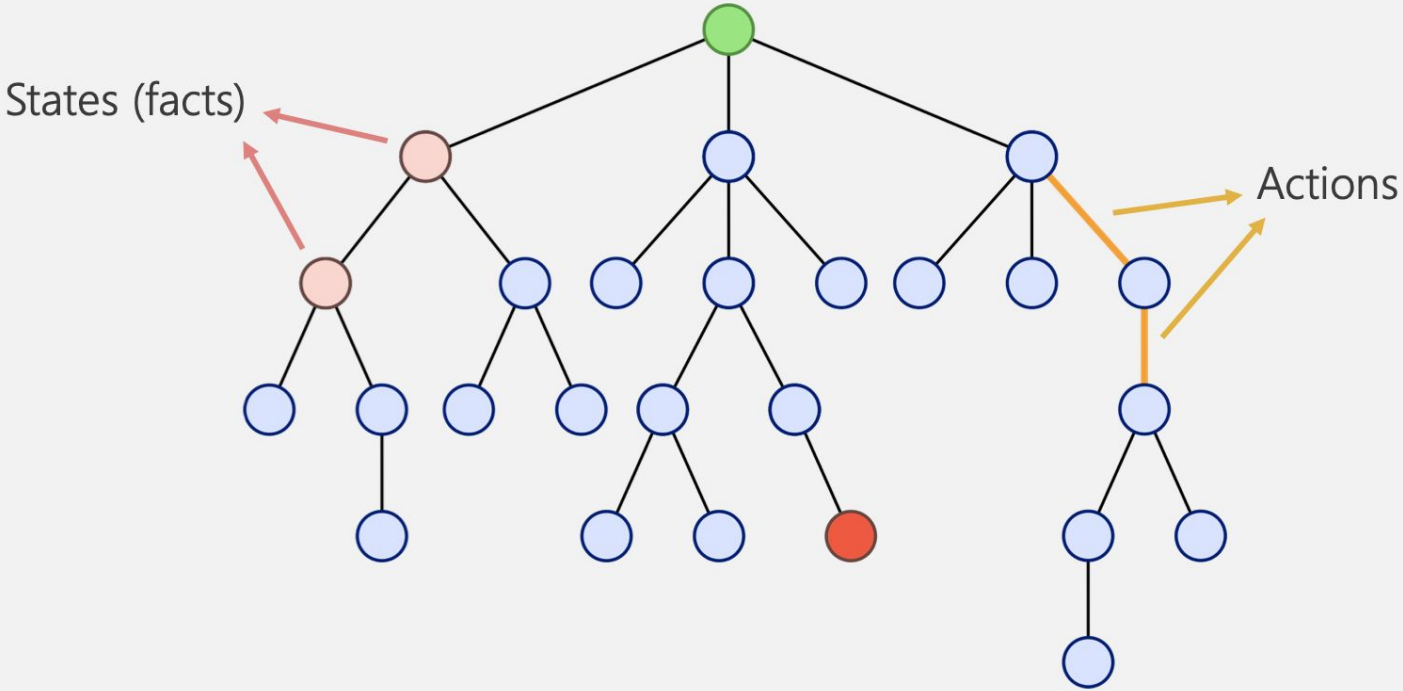


Go(Home,Home) applicable and does not change the state)

Buy(x) not available for any x (don't have Sells(Home, x))

Go(Garage,Home) and Go(Garage,Garage) also available

FINDING THE SEQUENCE



Monte Carlo | Tree Search
Random | Tree Search

(Because the city of Monte Carlo is known for gambling, which involves random chance.)

State of the art planning algorithm

MENU

nature

Published: 27 January 2016

Mastering the game of Go with deep neural networks and tree search

David Silver ✉, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis ✉

Nature **529**, 484–489(2016) | Cite this article


AlphaGo
Lee Sedol


특별대국 제5차전
이세돌 : 알파고
LEE SEDOL
00:18:50
ALPHAGO
00:44:49

(Yonhap Interview) Go master Lee says he quits unable to win over AI Go players

"Even if I become the number one, there is an entity that cannot be defeated," he said in an interview with Yonhap News Agency in Seoul on Monday.

SHARE

REPORT

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

David Silver[1,2,*,†], Thomas Hubert[1,*], Julian Schrittwieser[1,*], Ioannis Antonoglou[1], Matthew Lai[1], Arthur Guez[1], Marc Lancto…

+ See all authors and affiliations

Science 07 Dec 2018:
Vol. 362, Issue 6419, pp. 1140-1144
DOI: 10.1126/science.aar6404

The New York Times

ESSAY

One Giant Step for a Chess-Playing Machine

The stunning success of AlphaZero, a deep-learning algorithm, heralds a new age of insight — one that, for humans, may not last long.

"I always wondered how it would be if a superior species landed on earth and showed us how they played chess," Peter Heine Nielsen told the BBC.

"Now I know."

Entire human chess knowledge learned and surpassed by DeepMind's AlphaZero in four hours

"The implications go far beyond my beloved chessboard... Not only do these self-taught expert machines perform incredibly well, but we can actually learn from the new knowledge they produce."
- GARRY KASPAROV

Goal: Pick the action in the current state with highest expected reward

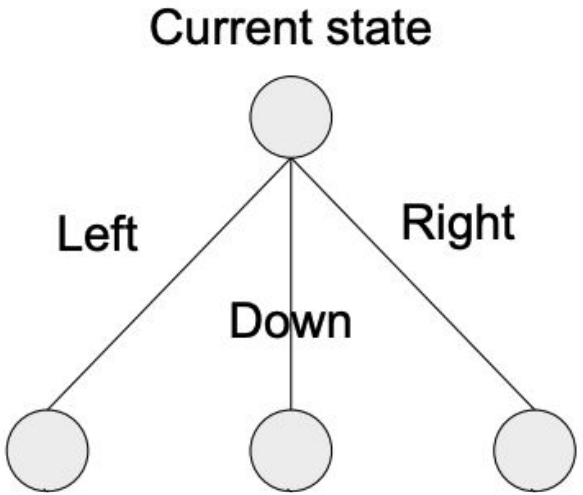
Goal: Pick the action in the current state with highest expected reward

Minimax / Expectimax is optimal, so use if you can.

But what if tree is too large? E.g. Go:

- ~150 moves per game
- With ~250 possible moves on average per move.
- 250^150 ~ 10^360 moves to consider
- Approx 10^80 particles in the observable universe!

Flat Monte Carlo Search

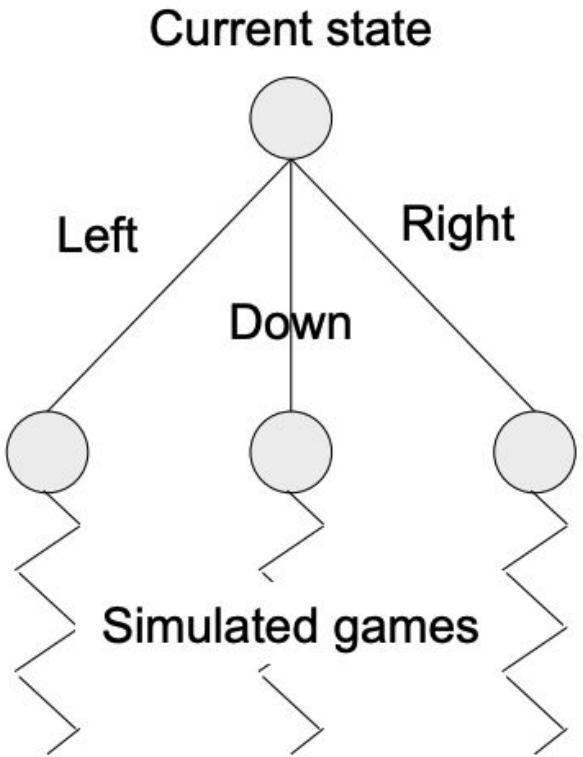


Start with a very simple algorithm, and build on it

How do you evaluate reward of action?

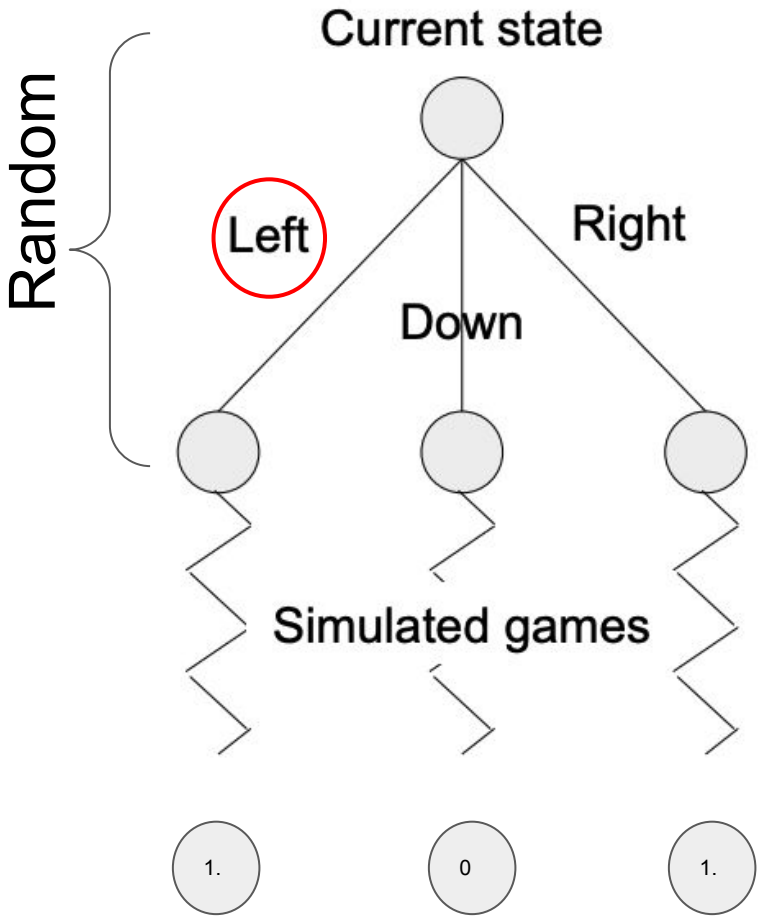
Just simulate game from that action onwards!

Flat Monte Carlo Search



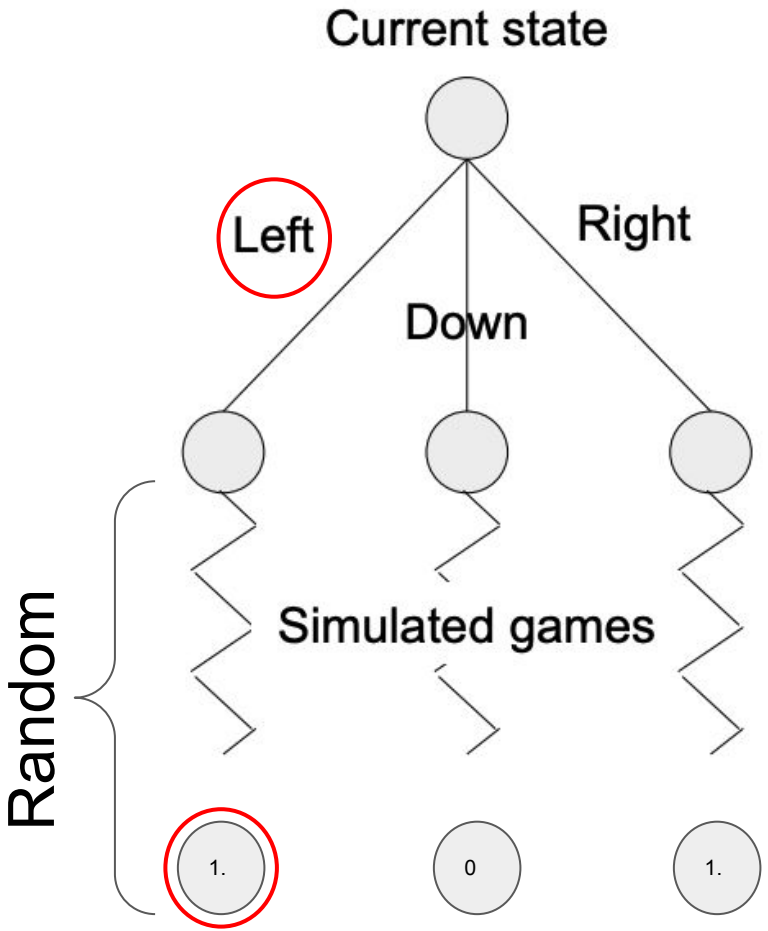
Random sample which actions to simulate

Flat Monte Carlo Search



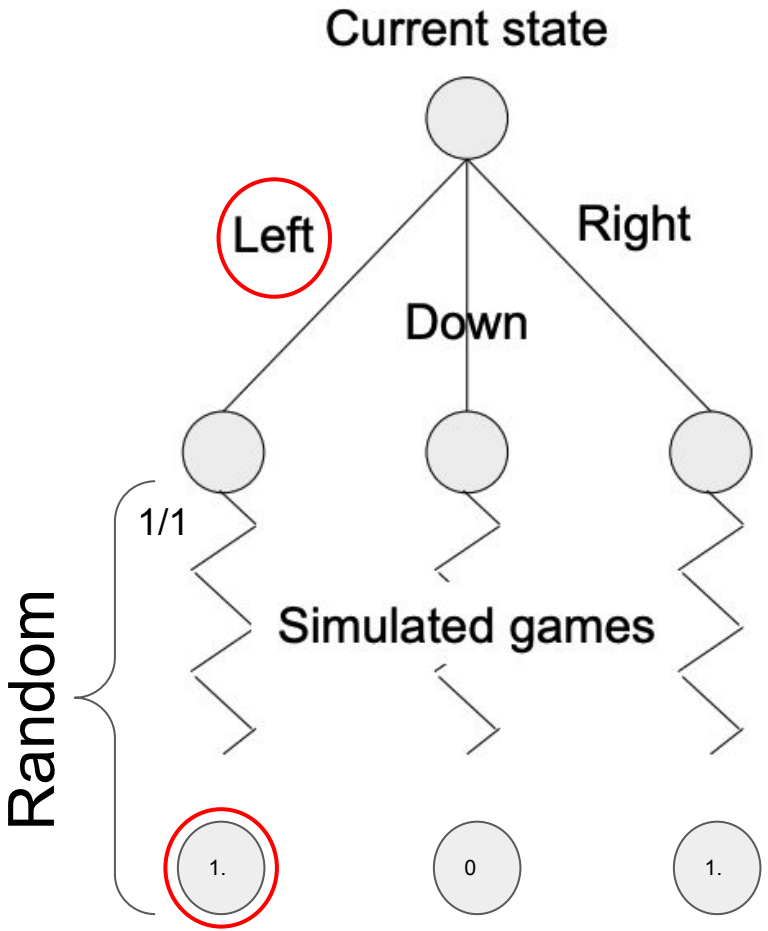
Then take random actions (random policy) until terminal state

Flat Monte Carlo Search



Record how many wins you had from that state.

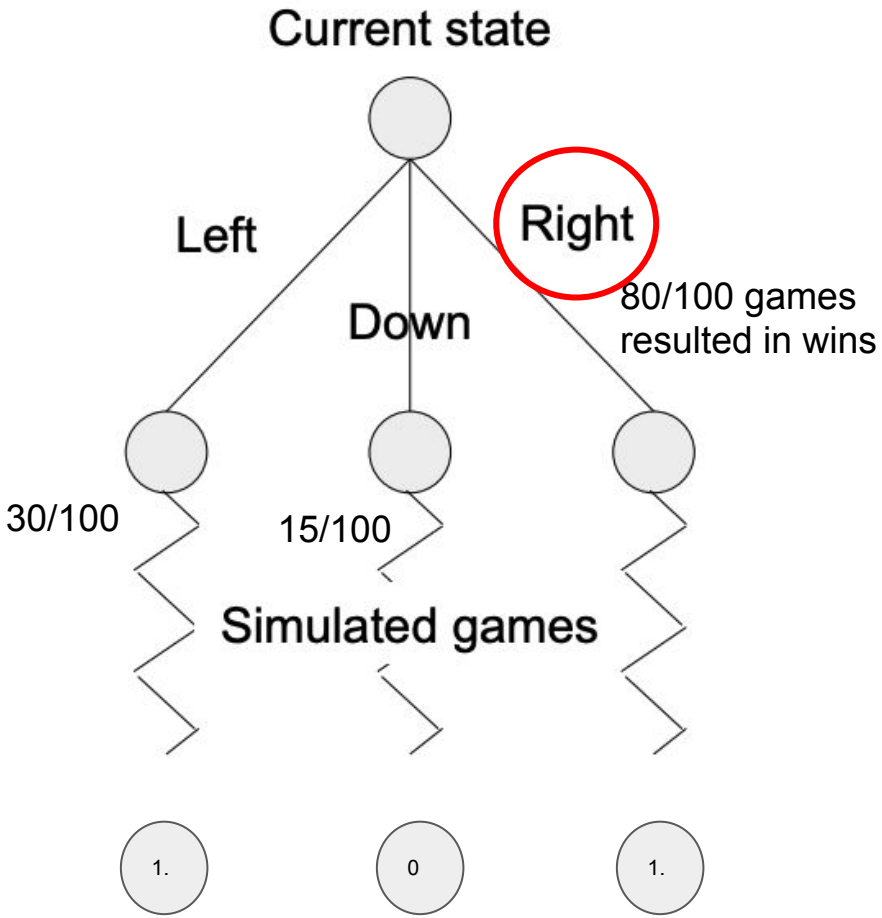
Flat Monte Carlo Search



Repeat many times.

Finally pick action with highest average reward

Flat Monte Carlo Search

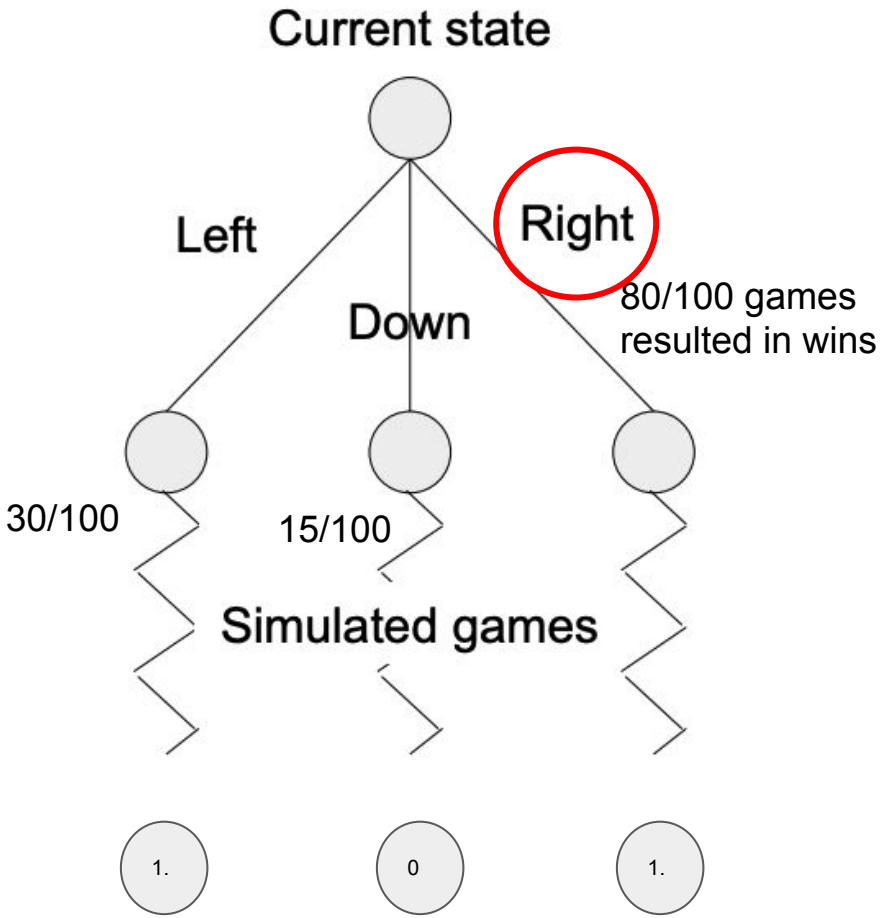


Pros?

Cons?

Discuss (5 min)

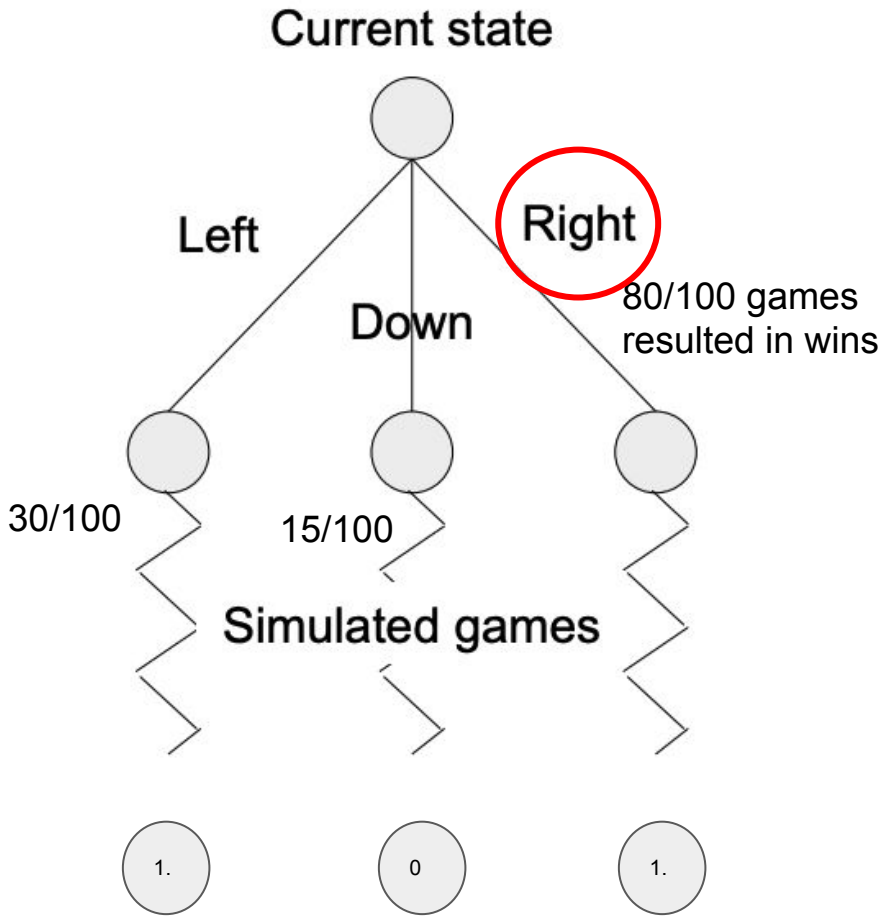
Flat Monte Carlo Search



Pros

- Anytime
- No heuristics
- Support parallelization

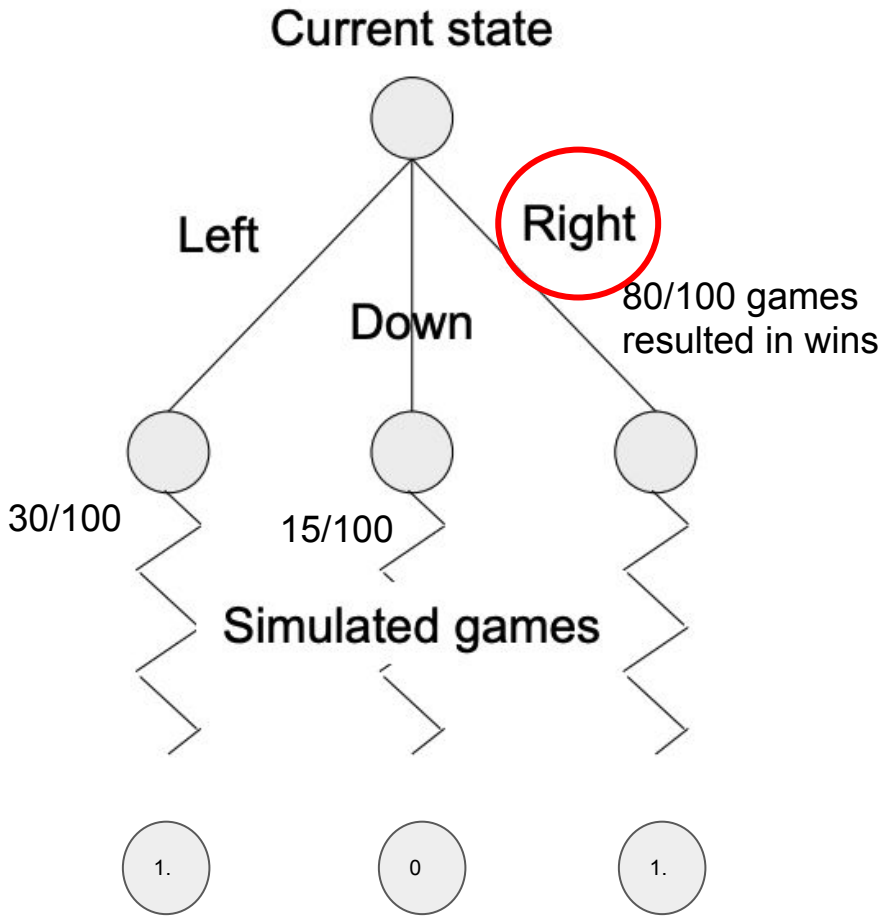
Flat Monte Carlo Search



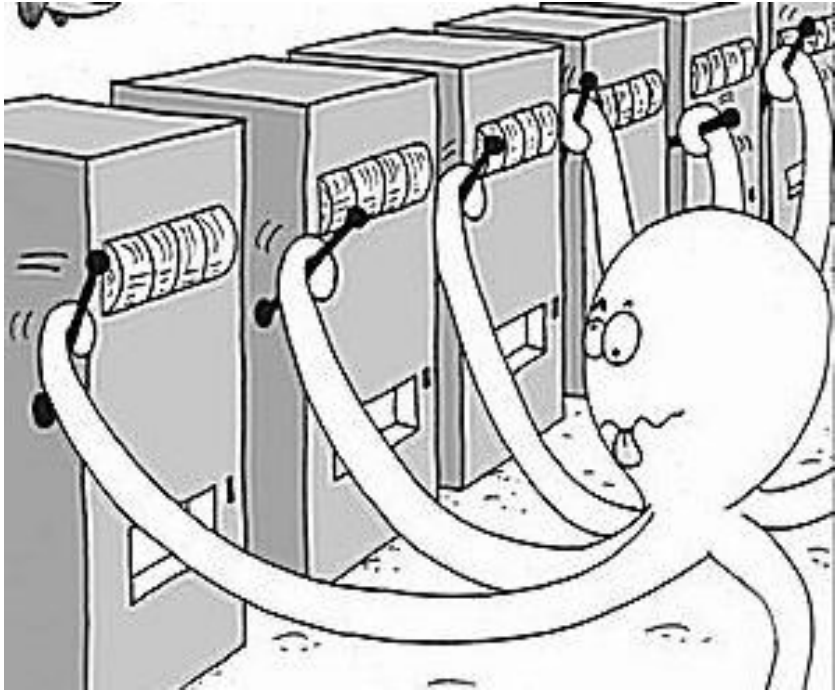
Cons

- Random rollouts likely underestimates your own (and opponents) policy.
- Spends lots of rollouts on less promising states
- Throws away lots of information

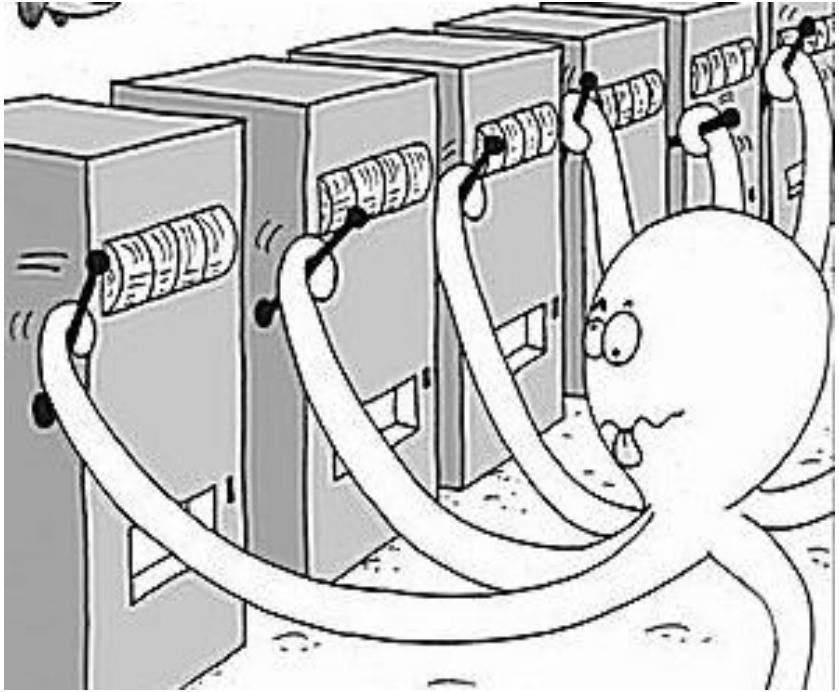
Flat Monte Carlo Search



Multi-armed bandits



Multi-armed bandits



Multiple slot machines (bandits)

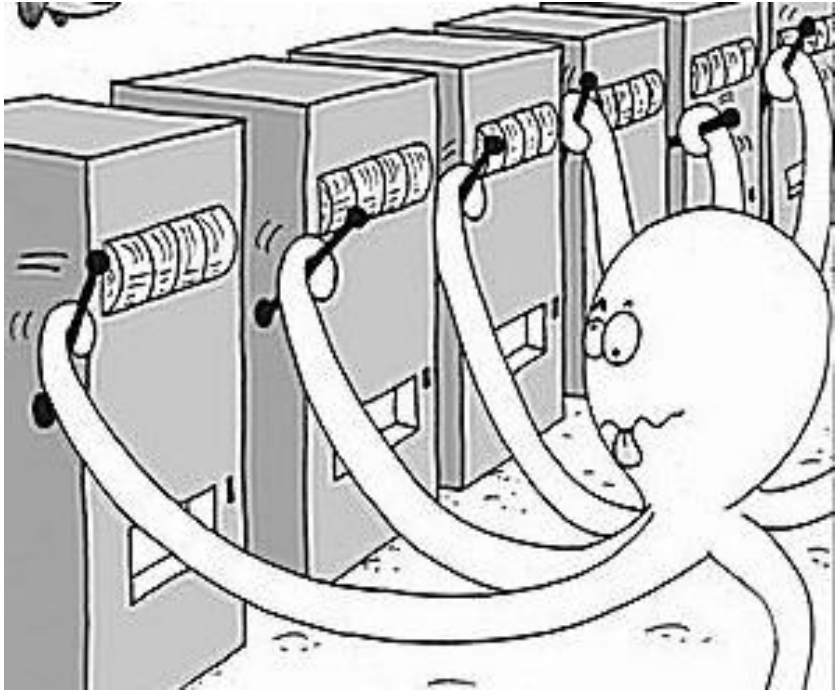
Each slot machine has different reward distribution.

Each step you pull one arm, get **random** reward from reward distribution.

Goal: maximize accumulated reward

Which arm should you pull?

Multi-armed bandits



Example

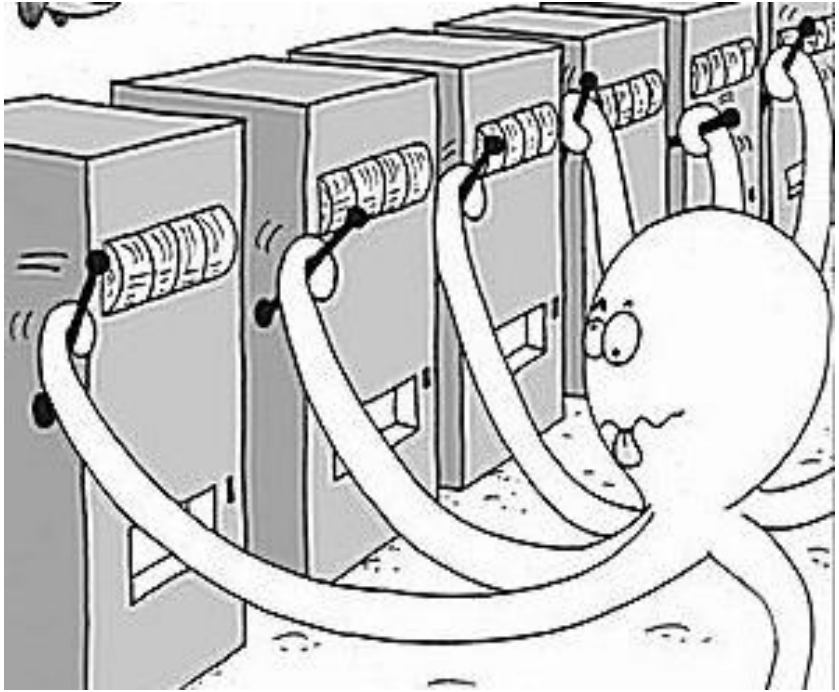
3 slot machines (A, B, C).
Observed rewards so far:

A: [2, 2, 2]
B: [-3, 7]
C: [1, -1]

What arm do you pull if you have **one** more pull? Why?

Discuss in groups (3 min)

Multi-armed bandits



Example

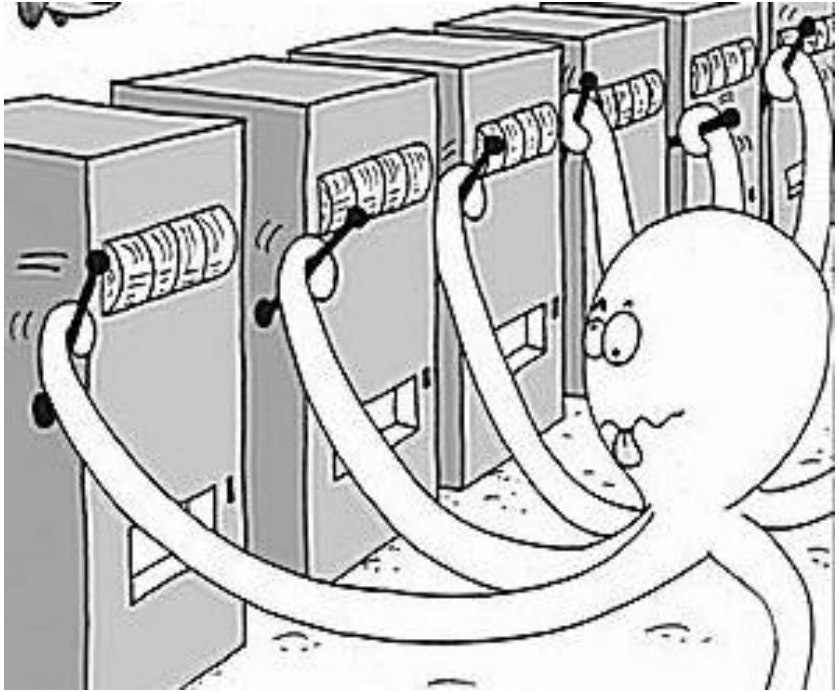
3 slot machines (A, B, C).
Observed rewards so far:

A: [2, 2, 2]
B: [-3, 7]
C: [1, -1]

What if you have 100 more pulls? What's your strategy?

Discuss in groups (5 min)

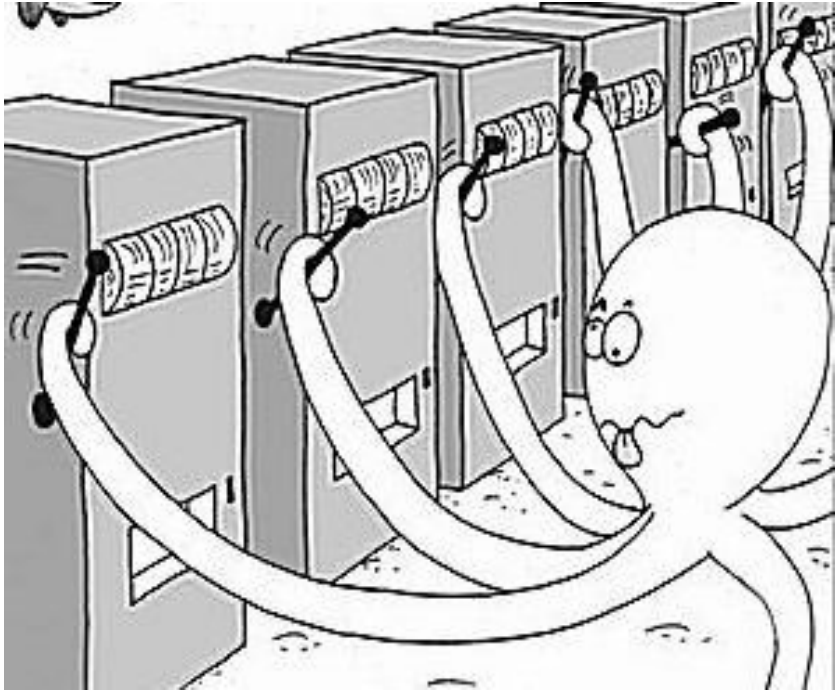
Multi-armed bandits



Exploration vs. Exploitation

Tradeoff between exploring to make better choices in the future, and exploiting to make good choices now.

Multi-armed bandits



Exploration vs. Exploitation

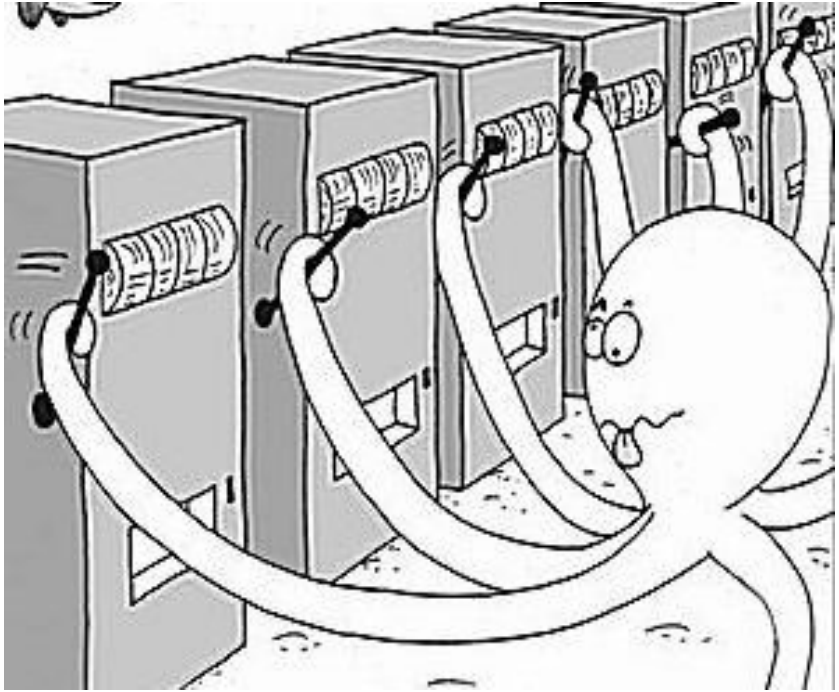
Greedy

Pick the arm with the best average reward.

Minimal exploration.

Maximum exploitation.

Multi-armed bandits



Exploration vs. Exploitation

Uniform random.

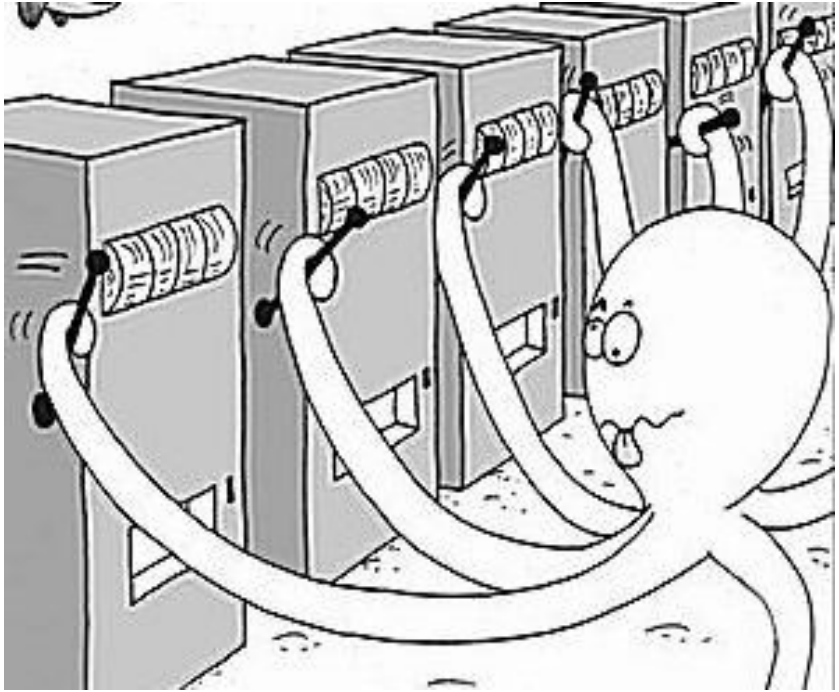
Just pick arms at random.

Maximum exploration.

No exploitation.

No use of information gained through exploration!

Multi-armed bandits



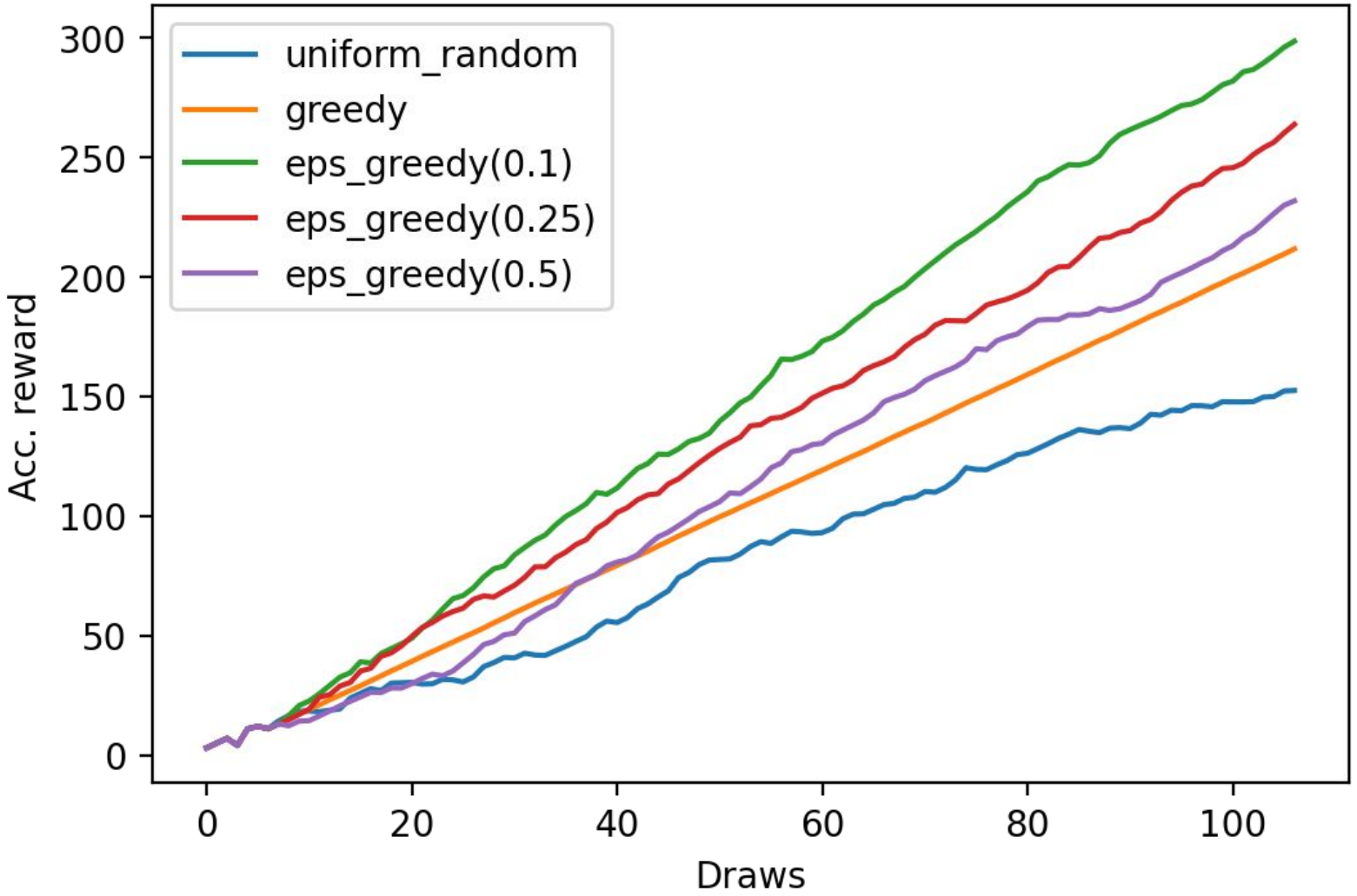
Exploration vs. Exploitation

ε-greedy

Pick at random with epsilion probability

Otherwise pick greedy.

ε controls exploration-exploitation tradeoff.


uniform_random
greedy
eps_greedy(0.1)
eps_greedy(0.25)
eps_greedy(0.5)
Acc. reward
Draws
0
50
100
150
200
250
300
0
20
40
60
80
100

Upper Confidence Bound (UCB) strategy

2 bandits (A, B)

Either 0 or 1 reward.

Let's look at how our belief about the bandits change with draws.

No Draws. No information about the bandit performance → We believe all rewards equally likely.

Upper Confidence Bound (UCB) strategy

draws = {
“A”: 2/2,
“B”: 1/2
}

A is clearly better. Right?

So we’ll draw more from A

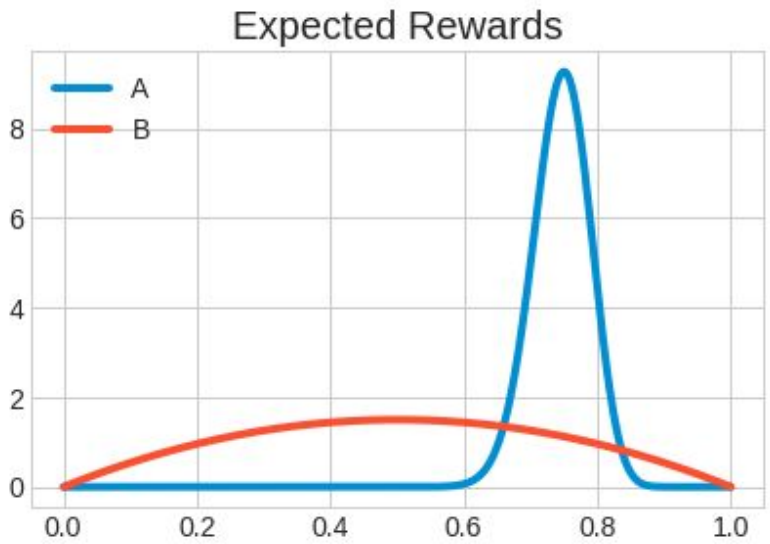
Upper Confidence Bound (UCB) strategy

draws = {
“A”: 6/8,
“B”: 1/2
}

A still looks better. Right?

So we’ll draw more from A

Upper Confidence Bound (UCB) strategy



draws = {
    “A”: 75/100,
    “B”: 1/2
}

Now we’re very certain A gives around 0.75 reward on average.

But we’re very uncertain about B.

B could actually be better!

Upper Confidence Bound (UCB) strategy

draws = {
“A”: 75/100,
“B”: 1/2
}

Formalize this intuition w. upper confidence bounds.

95% confidence bound → 5% chance it’s better than this.

Upper Confidence Bound (UCB) strategy

Select the bandit with the highest α-confidence bound (e.g. 95%, etc).

Optimal under some conditions*

α is a parameter you usually choose empirically.

Higher α → more exploration

** Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem.", 2002*

Upper Confidence Bounds (UCB1): (Auer et al (2002)).
Choose arm *j* so as to maximise:

$$\mathrm{UCB1} = \overline{X}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

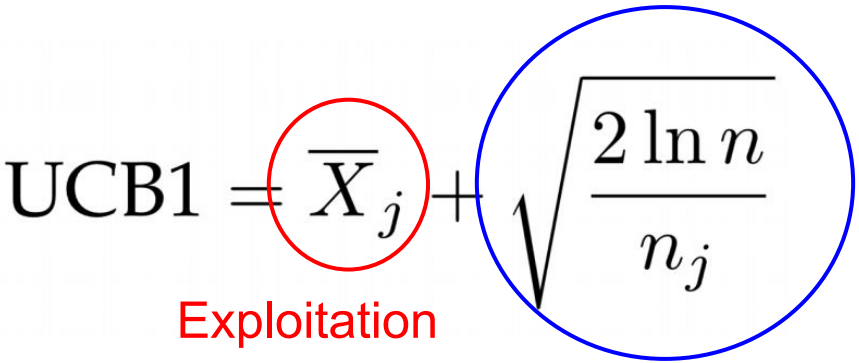
where X_j is the average reward from arm *j*

n = number of plays/pulls so far

n_j = number of times arm j was pulled

** Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem.", 2002*

Upper Confidence Bounds (UCB1): (Auer et al (2002)).
Choose arm *j* so as to maximise:



where X_j is the average reward from arm *j*

n = number of plays/pulls so far

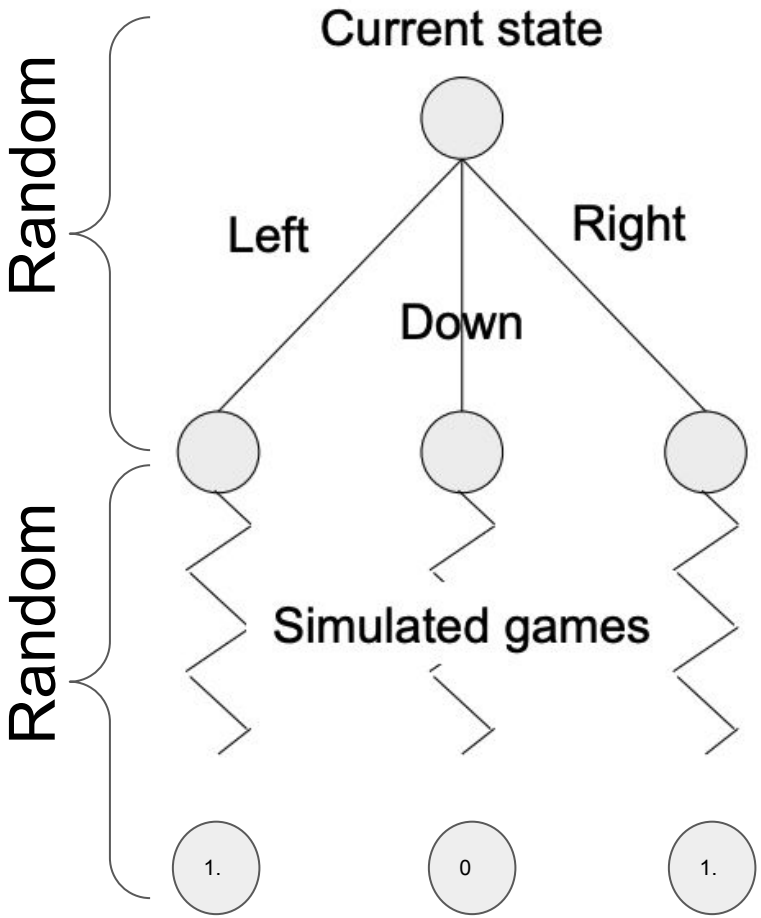
n_j = number of times arm j was pulled

** Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem.", 2002*

This is a bandit problem!

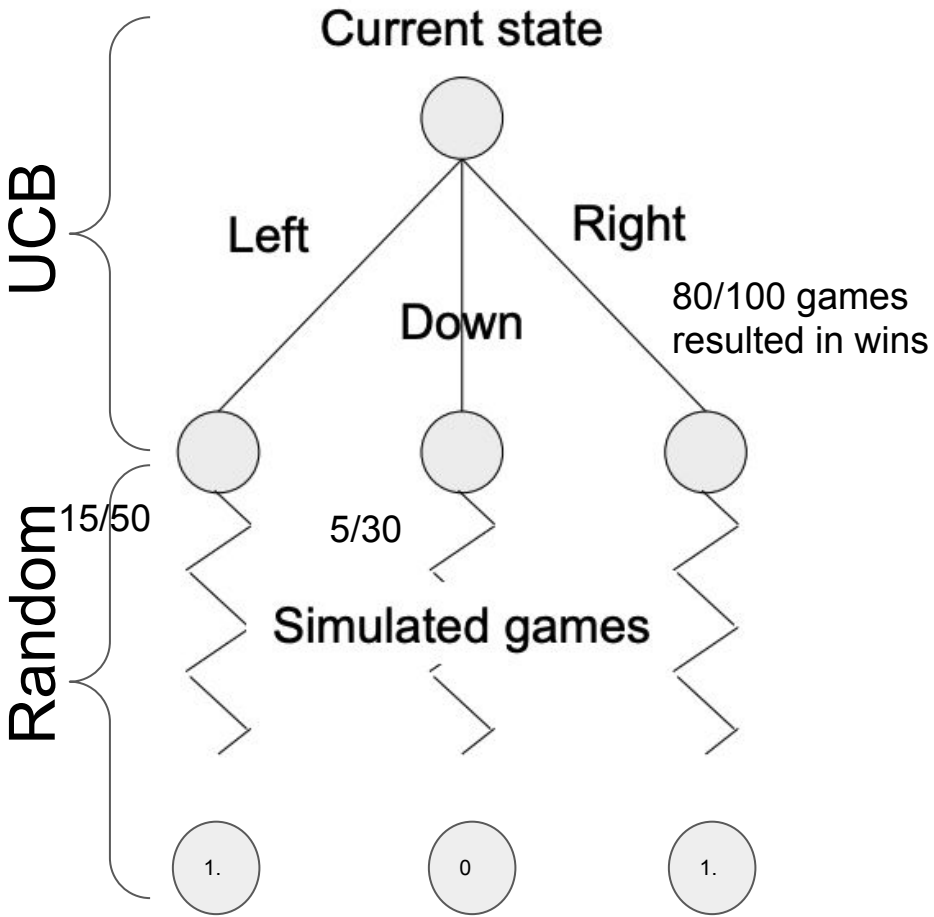
So we can do better than random!

Flat Monte Carlo Search



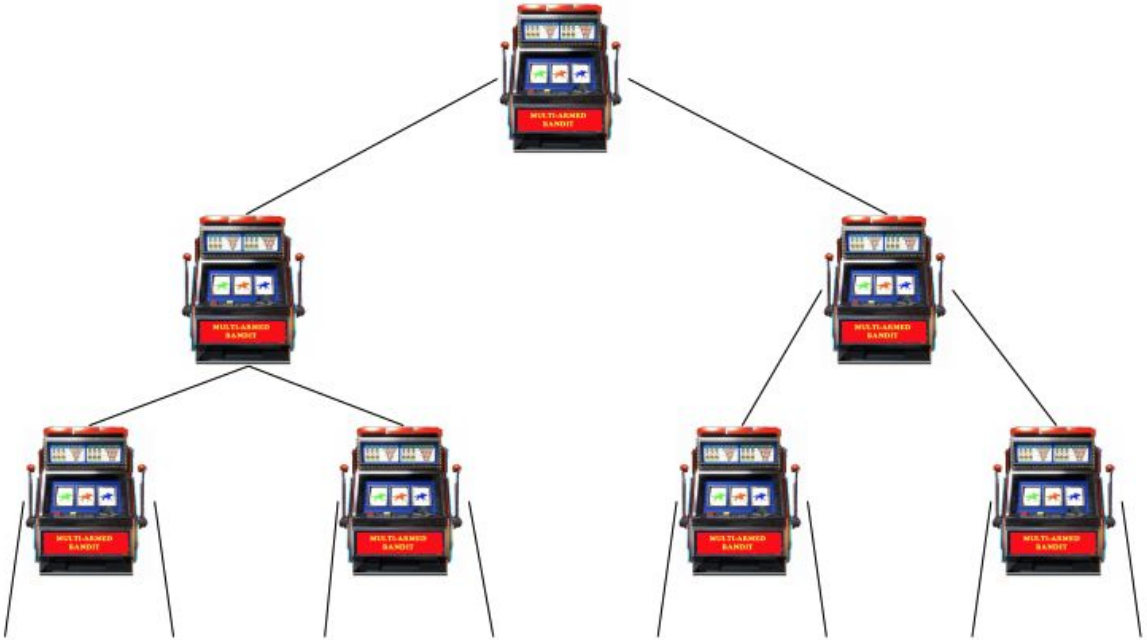

READY!

Flat UCB

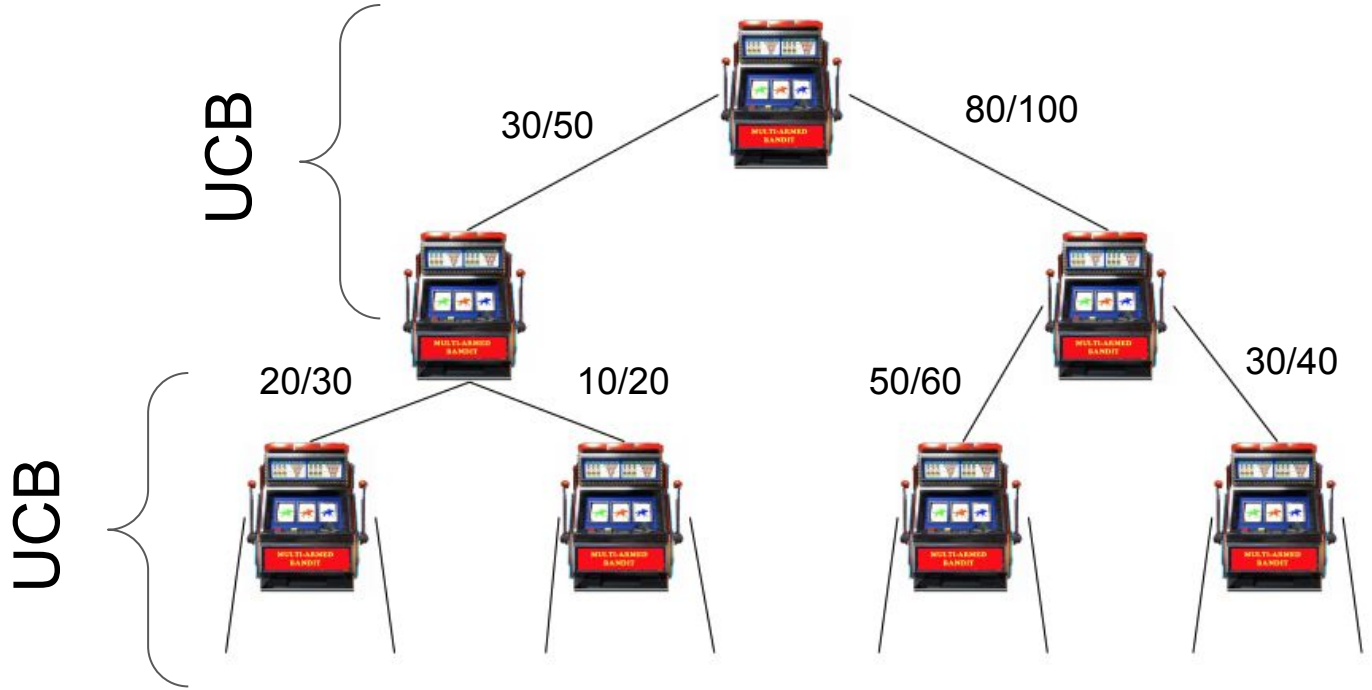

Current state
UCB
Left
Down
Right
80/100 games
resulted in wins
Random
15/50
5/30
Simulated games
1.
0
1.

MCTS

Game decisions are sequential

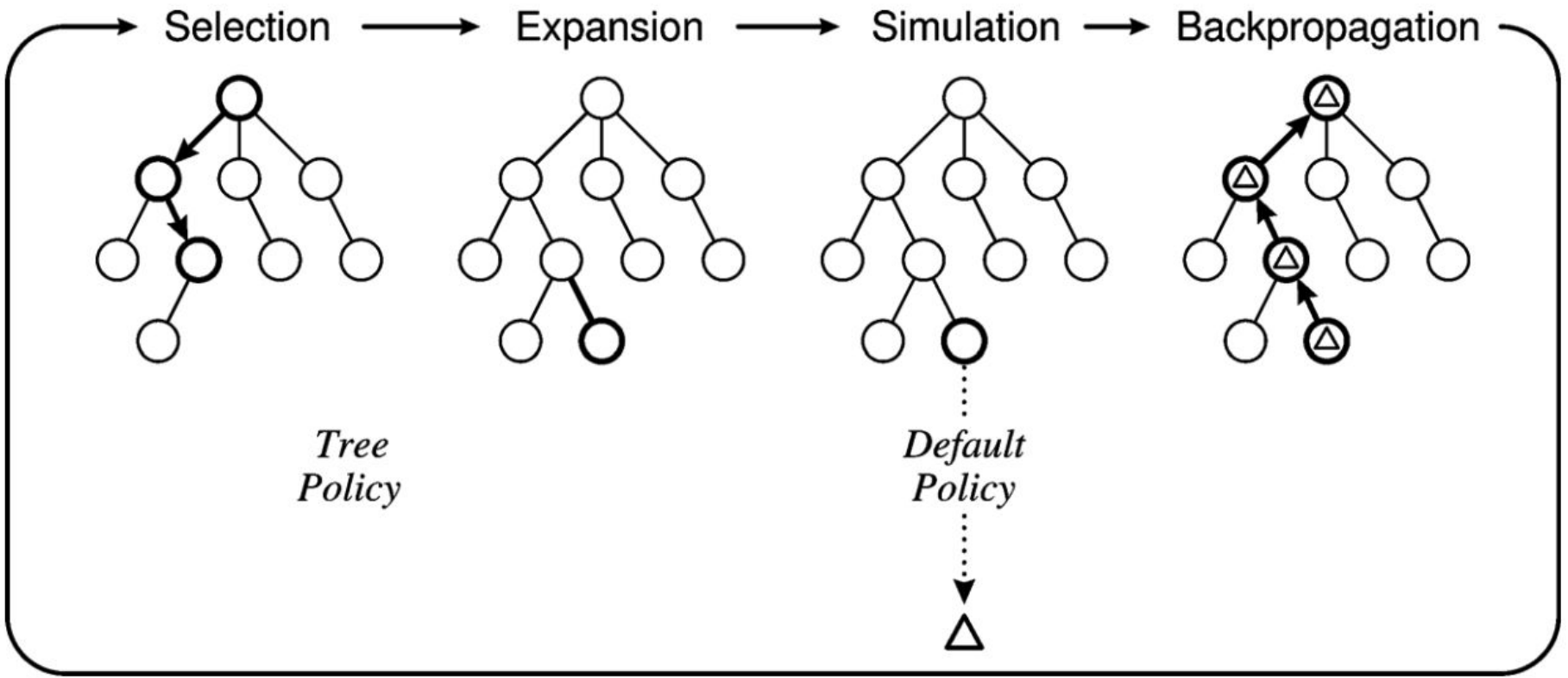


A tree of bandits!

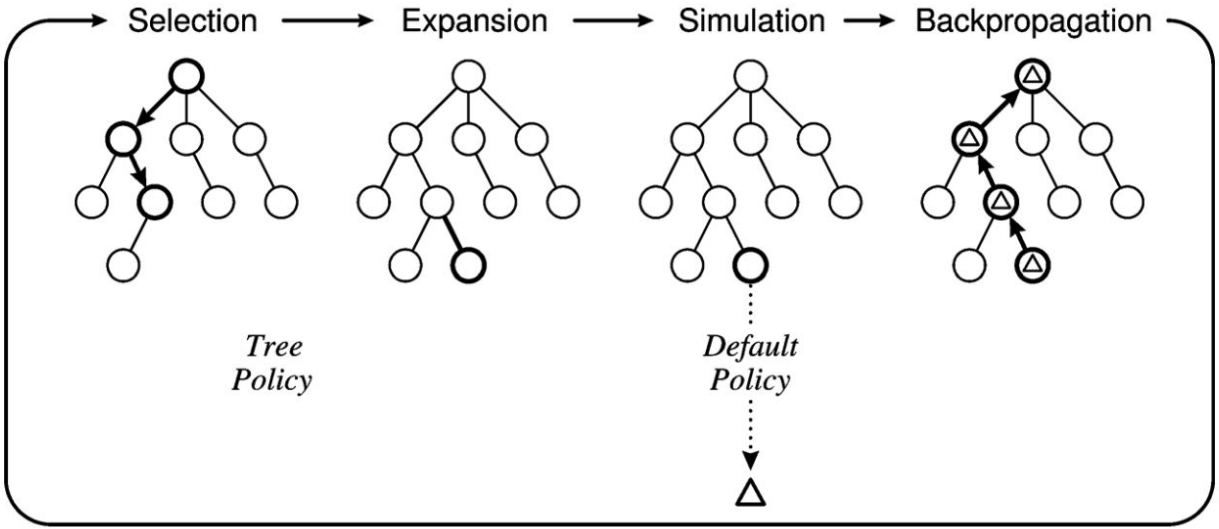


A tree of bandits!

Monte Carlo Tree Search



Browne, Cameron B., et al. "A survey of monte carlo tree search methods." 2012

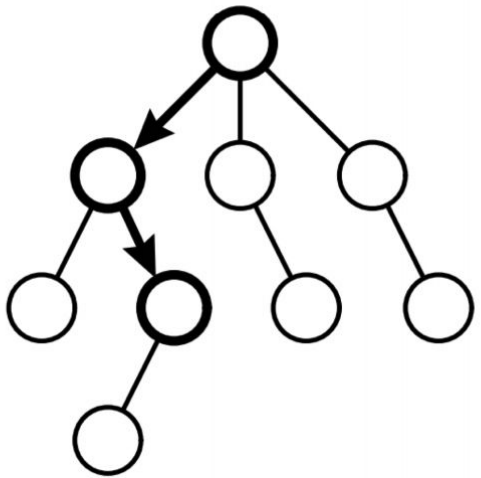


function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0))

Browne, Cameron B., et al. "A survey of monte carlo tree search methods." 2012

Tree policy: Select most *urgent* node. → Selection

```
function TreePolicy(v)
    while v is nonterminal do
        if v not fully expanded then
            return Expand(v)
        else
            v ← BestChild(v, Cp)
    return v
```



E.g. UCB1, ε-greedy..

function EXPAND(v)

choose $a \in$ untried actions from $A(s(v))$

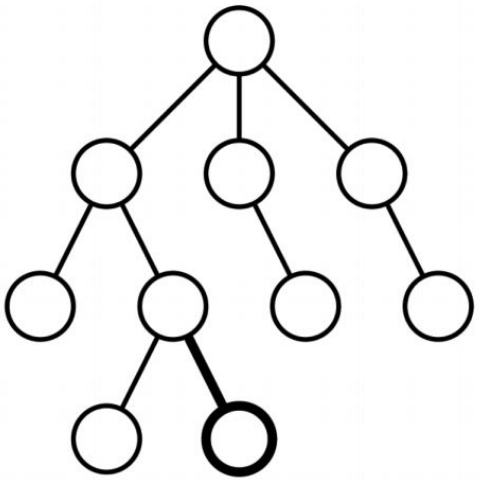
add a new child v' to v

with $s(v') = f(s(v), a)$

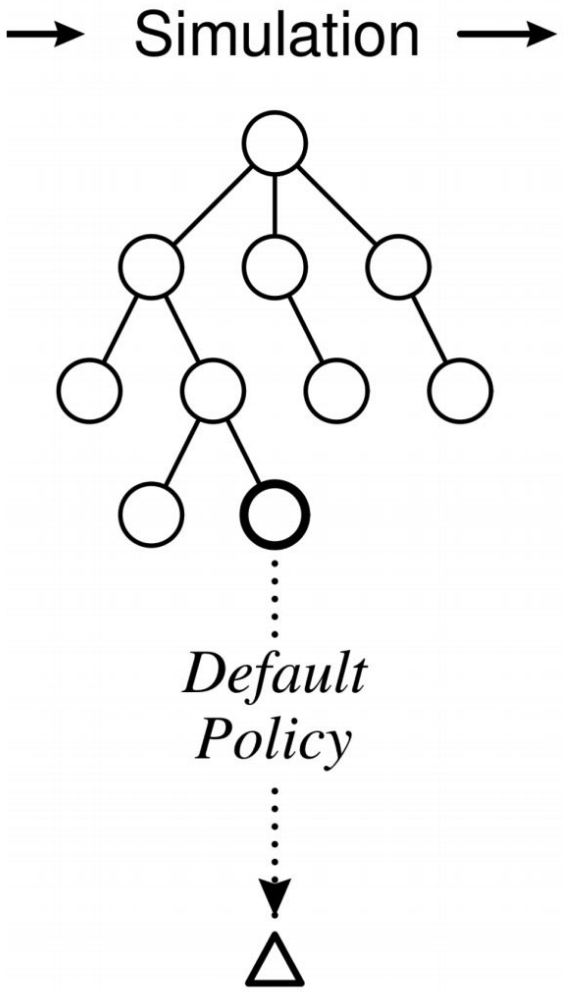
and $a(v') = a$

return v'

→ Expansion ←



```
function DefaultPolicy(s)
    while s is non-terminal do
        choose a ∈ A(s) uniformly at random
        s ← f(s, a)
    return reward for state s
```



The standard is to do action selection with uniform randomness.

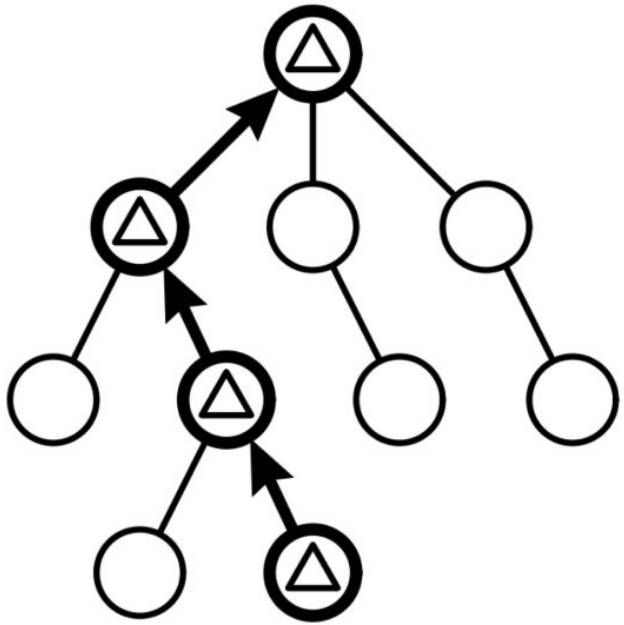
f(s, a) is a forward model that given a state *s* and action *a* returns the resulting state.

function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

v is the node returned by TreePolicy

Back up the value from *v* to the root. Increment visit count as well

Backpropagation



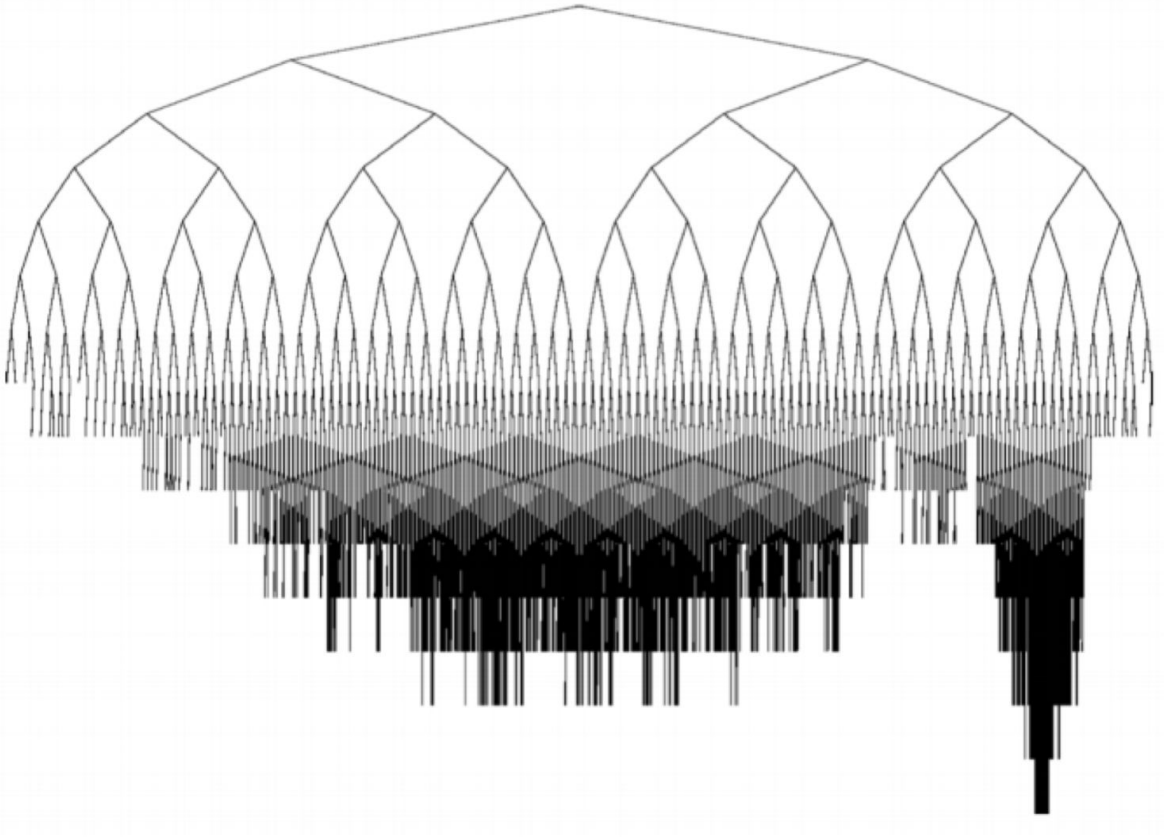
Upper Confidence for Trees (UCT)

function BESTCHILD(v, c)

return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

"Just" UCB1, but proven to converge to optimal (minimax) policy w. enough rollouts!

- *L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning,", 2006*
- *L. Kocsis, C. Szepesvári, and J. Willemson, "Improved Monte-Carlo search,", 2006.*



R. Coquelin, Pierre-Arnaud and Munos, “Bandit Algorithms for Tree Search,” in Proc. Conf. Uncert. Artif. Intell. Vancouver, Canada: AUAI Press, 2007,

MCTS

Multi-player games

Simplest method:

Each node stores a vector of rewards, and the selection procedure seeks to maximise the UCB1 value calculated using the appropriate component of the reward vector.

For two players, one value is often used - but negated for player 2.

Multiple-MCTS:

There is a tree for each player, and the search descends and updates all of these trees simultaneously, using statistics in the tree for the relevant player at each stage of selection.

MCTS

Enhancements

Selection/expansion:

- All moves as first (AMAF) / RAVE
- Action ordering
- Action pruning

Parallelization:

- Leaf**:** Concurrent rollouts
- Root**:** Multiple trees in parallel
- Tree**:** Multiple threads one same tree

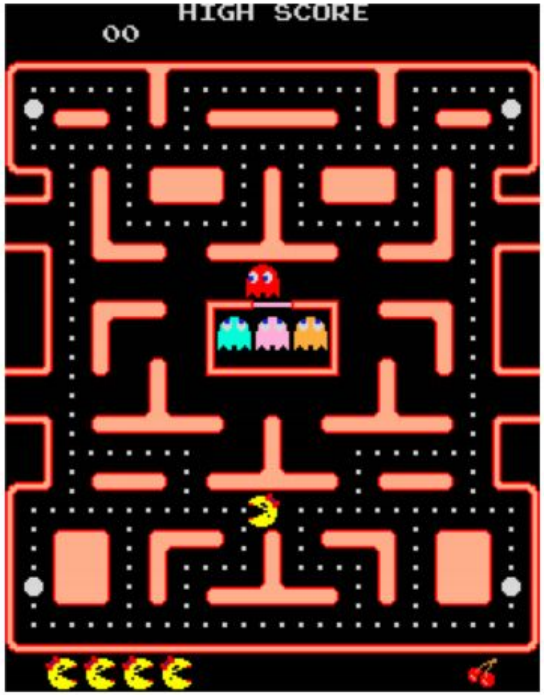
Simulation:

- Domain knowledge in rollouts - more expensive
- *Depth-limit* with state evaluation function

Examples

Ms. Pac-Man

- End state might not be reached
- Small branching factor
- Stochastic forward model (ghost decisions are unknown)



Examples

Ms. Pac-Man

Looking back at MCTS main characteristics:

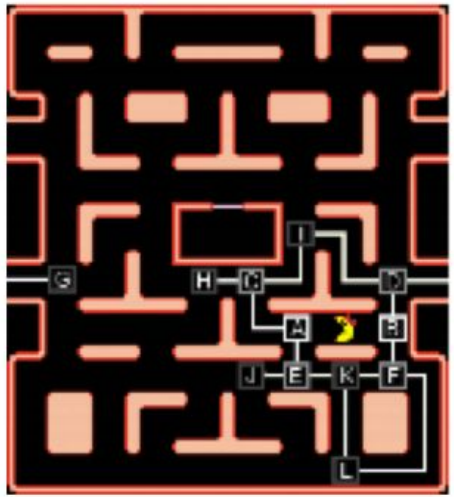
1. **Scalable.** Can tackle complex games better than other tree search algorithms.
2. **No state evaluation function needed**
3. **Anytime:** We can stop the algorithm at anytime and ask for an action.

Is MCTS suited for Ms. Pac-Man?

Examples

Ms. Pac-Man: Pepels and Winand

1. Store several rewards in each node (ghost score, pill score, survival rate) - allows the selection of different tactics
2. Bounded search depth with variation
3. Not completely random playouts - e.g. use fixed ghost behaviors
4. Endgame database
5. Use intersections/junctions as nodes in the tree - not every step



Conclusion

MCTS is a *stochastic* tree-search algorithm that:

- produces an *unbalanced* search-tree due to a mix of **exploration** and **exploitation** - e.g. using UCB1.
  - Can overcome larger complexities.
- uses **rollouts** as heuristic - no state evaluation function needed.
  - In most games, a combination of rollouts and state evaluation function works best.
- has A LOT of variations and enhancements; find many of them in (Browne 2012).
  - AMAF, RAVE, determinations methods, multi-MCTS etc..