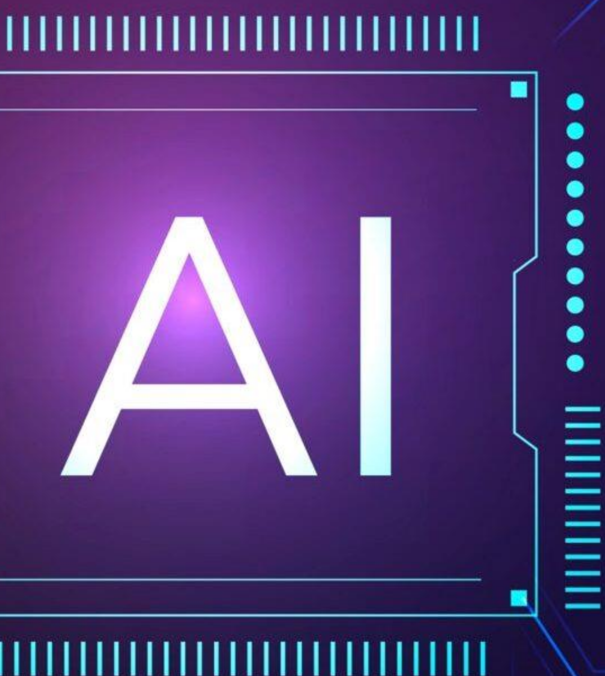


MONTE CARLO TREE SEARCH

Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

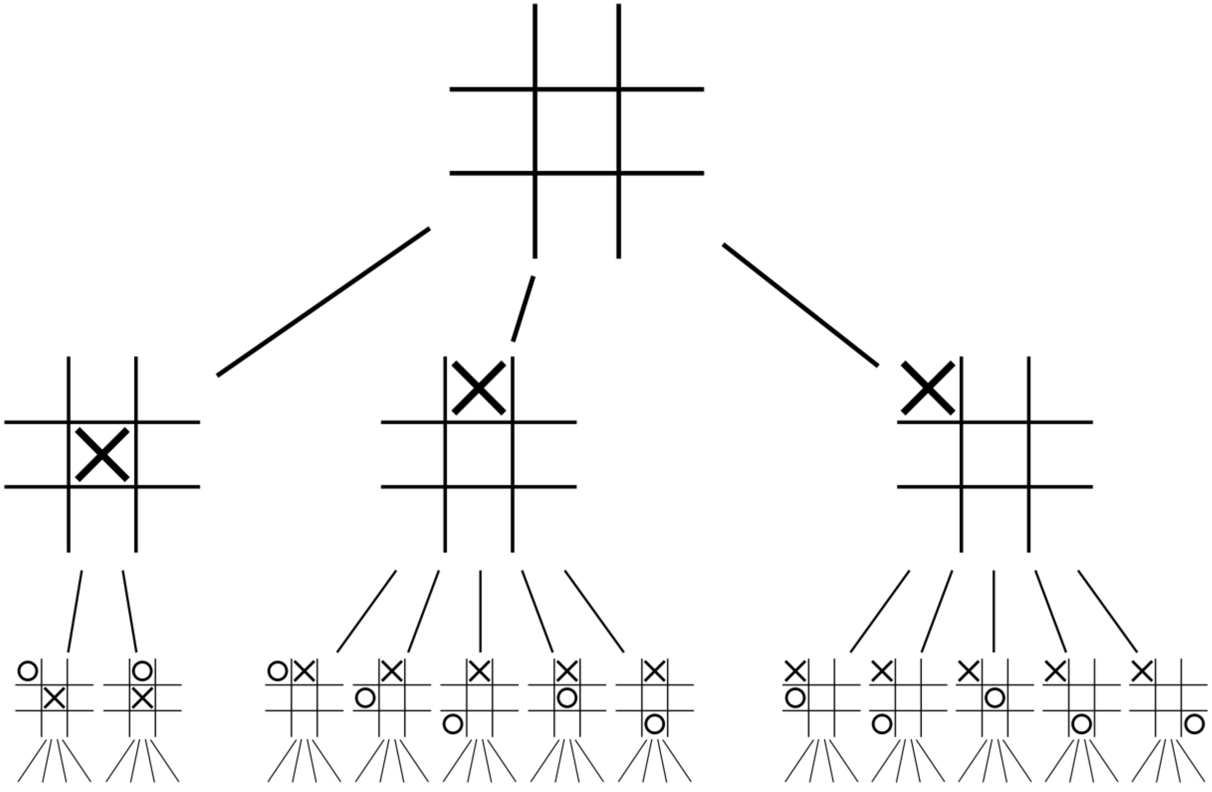
Outline

- An introduction to MCTS
- A complete walkthrough with example
- A deeper insight to MCTS

Monte Carlo tree search

Game tree and search strategies

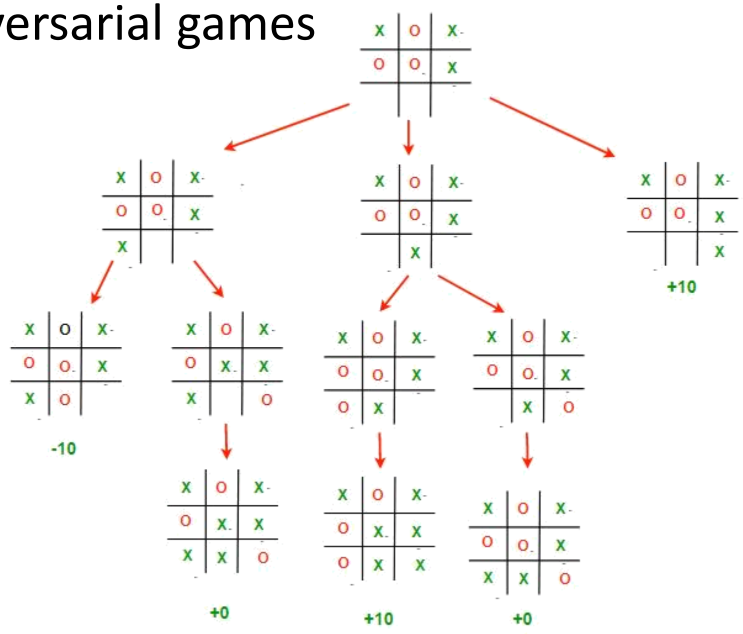
- A **game tree** represents a hierarchy of moves in a game.
  - Each node of a game tree represents a particular state in a game.
  - A move makes a transition from a node to its successors.



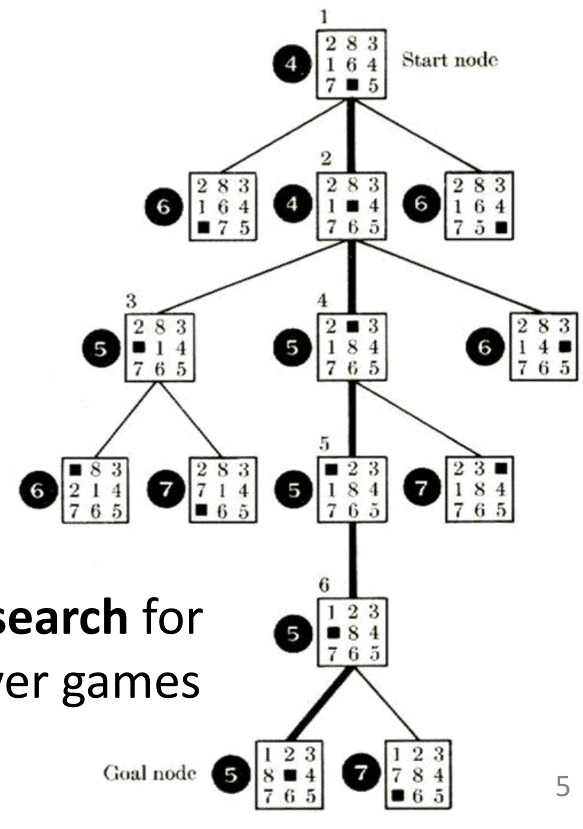
Game tree and search strategies

- Many AI problems can be cast as search problems, which are solved by finding the best plan, model or function.
- A search algorithm finds the best path to win the game.

Minimax search for two-players adversarial games

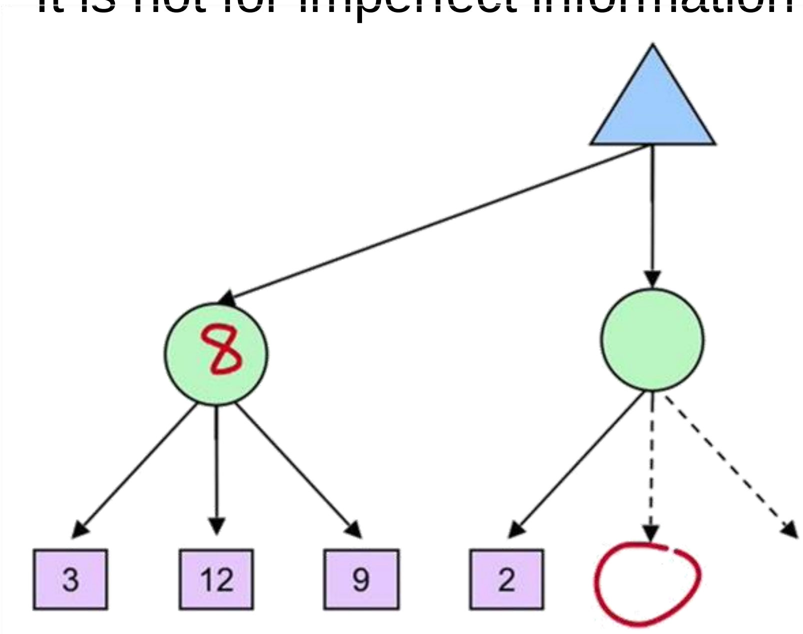


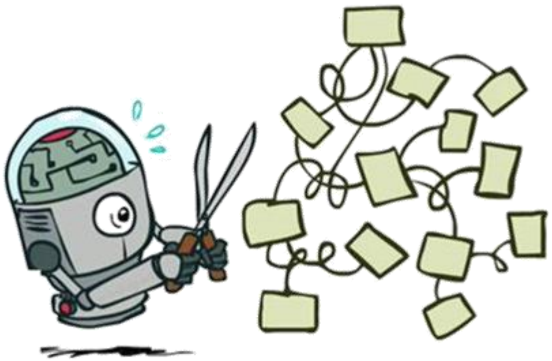
Best-first search for single-player games



The limitations of Minimax

- Minimax explores all the nodes available → infeasible for complex games in a finite amount of time.
  - It is not for imperfect information and stochastic games either.

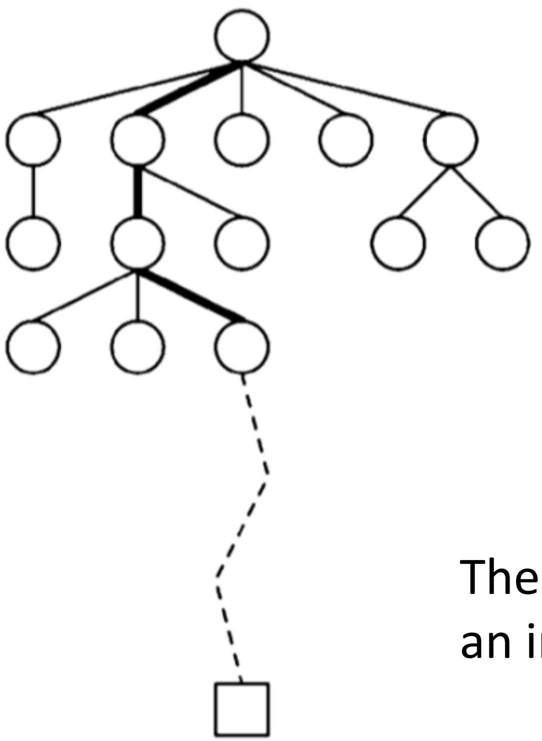


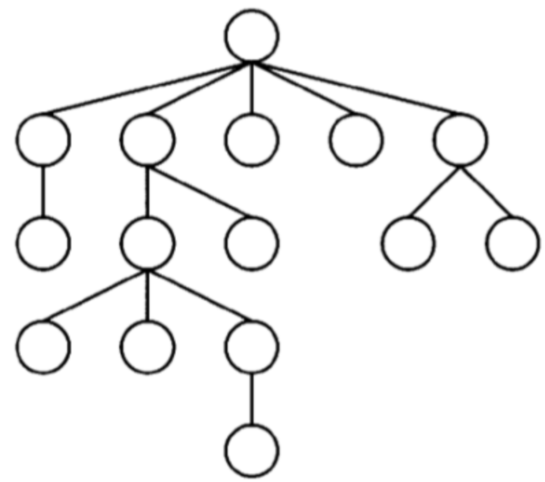


- Expectimax generalizes minimax to stochastic games.
  - The pruning is harder because of chances nodes.

Monte Carlo tree search (MCTS)

- MCTS is a heuristic search method that links the precision of tree search with the generality of random sampling.
- It finds optimal decisions in a domain by taking random samples in the decision space to grow the search tree.





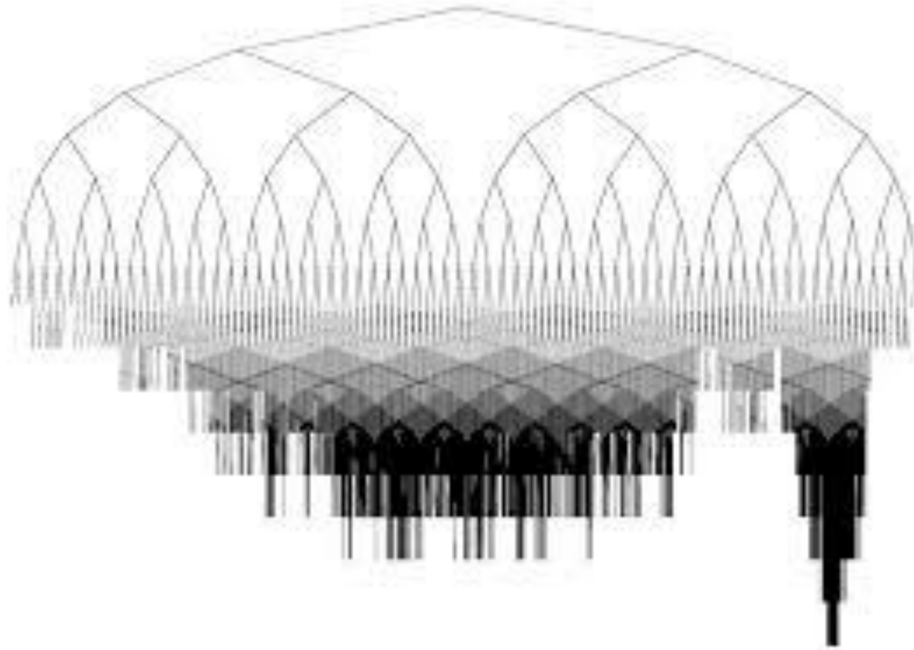
The basic MCTS process: A tree is built in an incremental and asymmetric manner.

MCTS: Fundamental idea

- MCTS progressively builds a partial game tree, guided by the results of previous exploration on the same tree.

The longer the tree grows, the more accurate the estimates become.

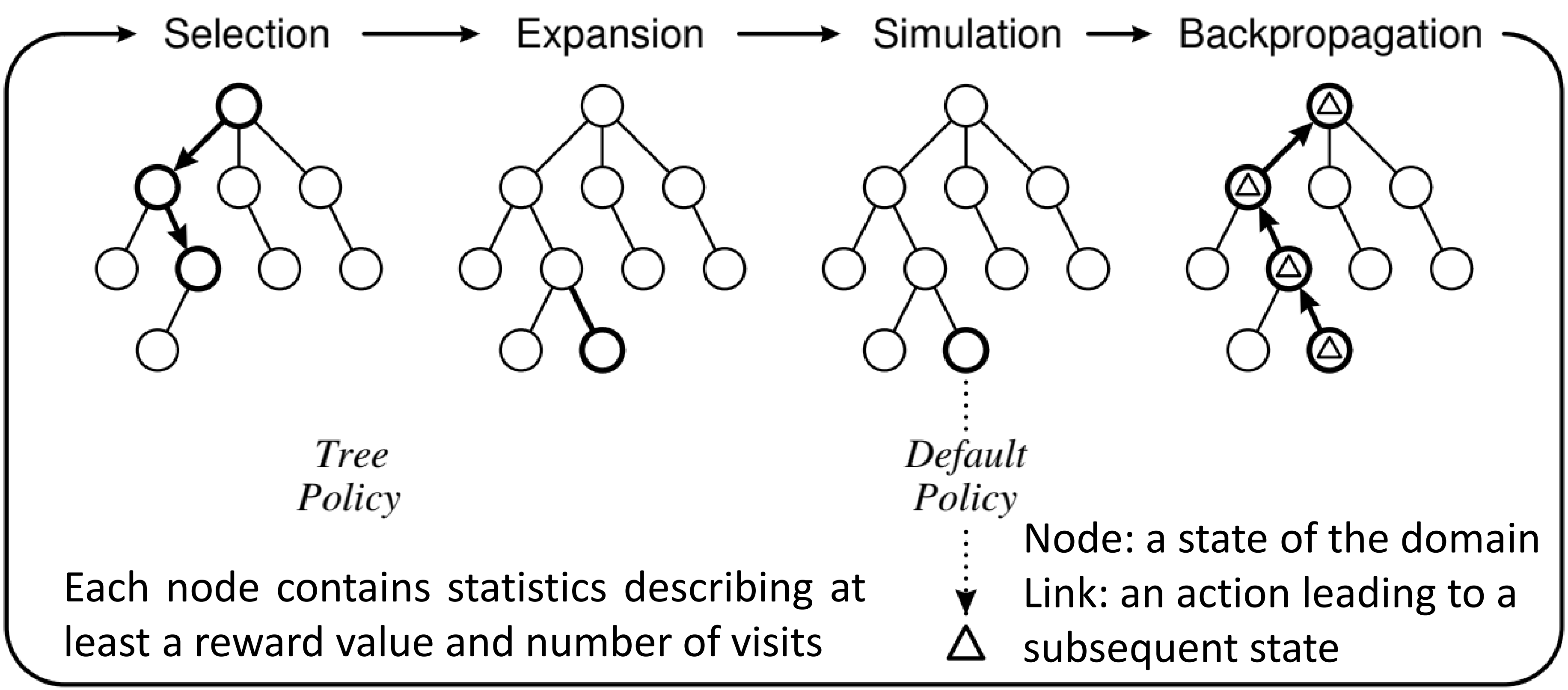
The tree grows in an asymmetric manner, heading towards more promising moves.



- The tree is used estimate the values of moves via random simulation → adjust the policy towards a best-first strategy.

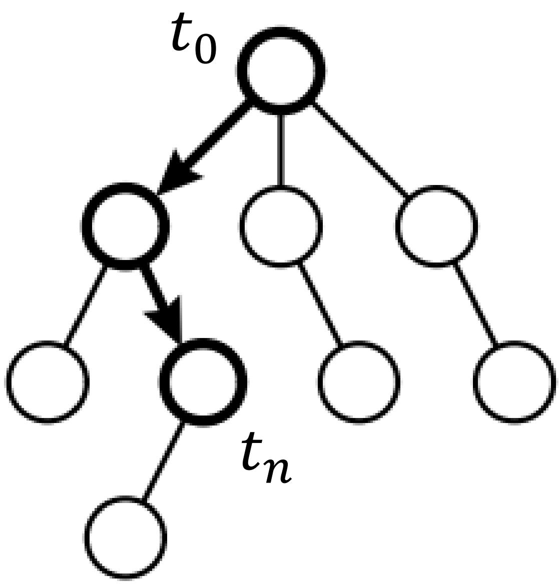
MTCS: Steps in an iteration

- MCTS iteratively grows a search tree within some predefined computational budget (e.g., time, memory, or iteration).



MCTS: Selection step

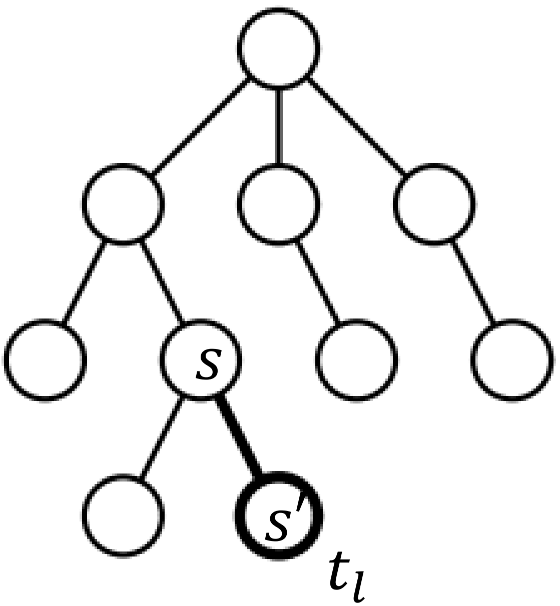
1. Selection: recursively apply a child selection policy (*tree policy*), from the root till the most urgent expandable node.



Starting at the root node t_0, we reach the node t_n.

MCTS: Expansion step

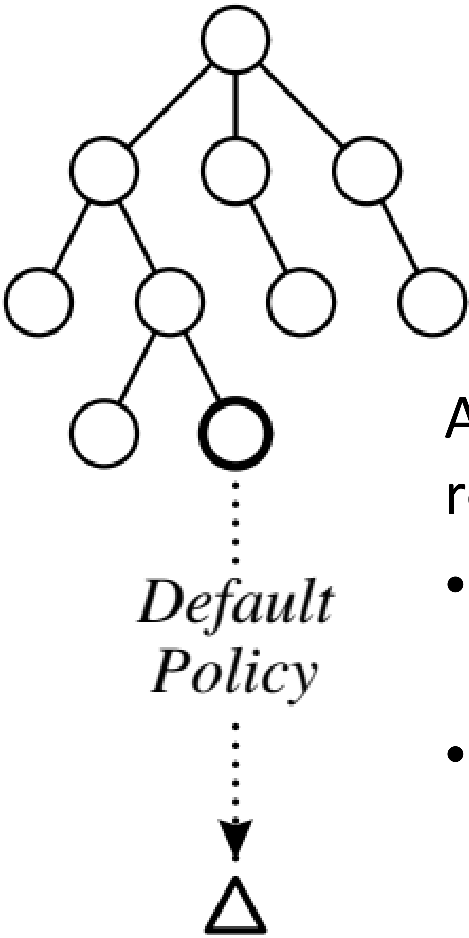
2. Expansion: expand the tree by adding one (or more) children, according to the available actions.



An unvisited action a from state s is selected and a new leaf node t_l is added to the tree.

MCTS: Simulation step

3. Simulation: run a simulated play from the new node(s) using the *default policy* to produce an outcome.

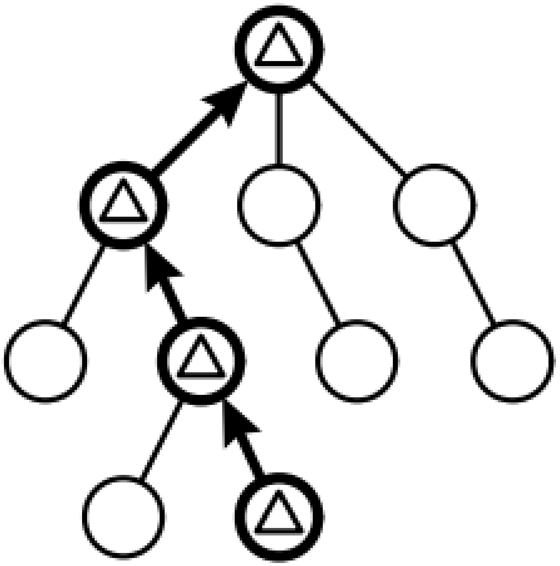


A simulation is run from the node t_l to produce a reward value Δ, which may be

- A discrete (win/draw/loss) result or continuous reward value for simple domains
- A vector of reward values relative to each agent p for more complex multiagent domains.

MCTS: Backpropagation step

4. Backpropagation: back up the simulation result through the selected nodes to update their statistics.



The reward value Δ is backed up to update the nodes along the path.

For each node, its visit count is incremented, and its average reward (or Q-value) updated according to Δ.

- As soon as the computation budget is reached or there is an interruption, the search terminates.

Monte Carlo tree search (MCTS)

- *Tree policy:* select or create a leaf node from the nodes in tree (selection and expansion).
- *Default policy:* Play out the domain from a given non-terminal state to produce a value estimate (simulation).

Algorithm 1 General MCTS approach.

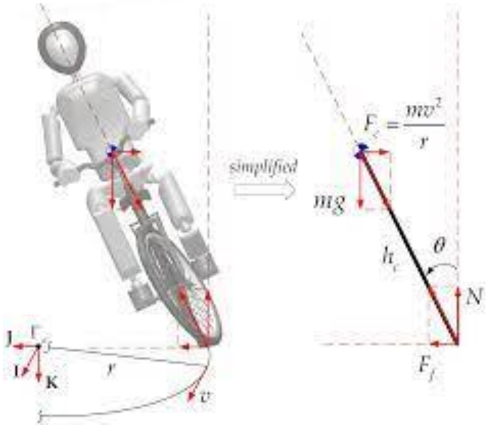
function $\textsc{MctsSearch}(s_0)$
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow \textsc{TreePolicy}(v_0)$
    $\Delta \leftarrow \textsc{DefaultPolicy}(s(v_l))$
    $\textsc{Backup}(v_l, \Delta)$
  return $\boxed{a(\textsc{BestChild}(v_0))}$

the action a that leads to the best child of the root node v_0

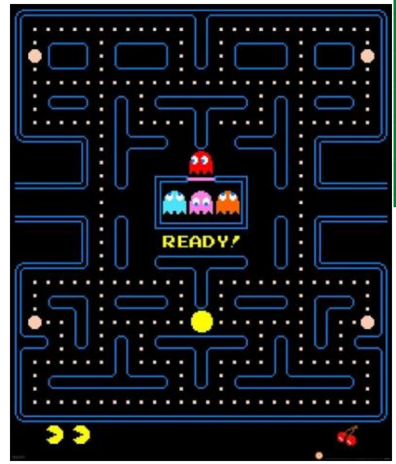
MCTS: Child selection

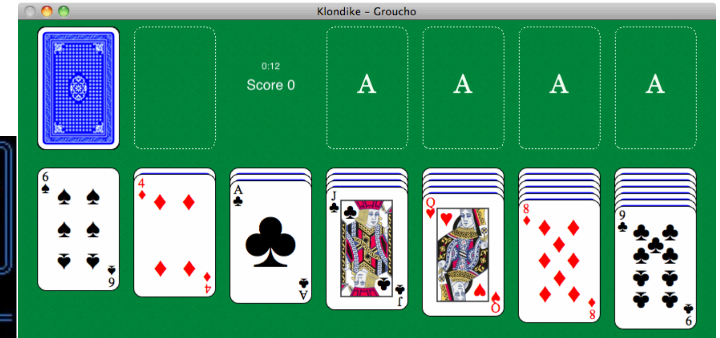
- An action a of the root t_0 is selected by some mechanism.
- *Max child:* select the root child with the highest reward.
- *Robust child:* select the most visited root child.
- *Max-Robust child:* select the root child with both the highest visit count and the highest reward.
  - If none exist, search until an acceptable visit count is achieved.
- *Secure child:* select the child which maximizes a lower confidence bound.

MCTS: Applications

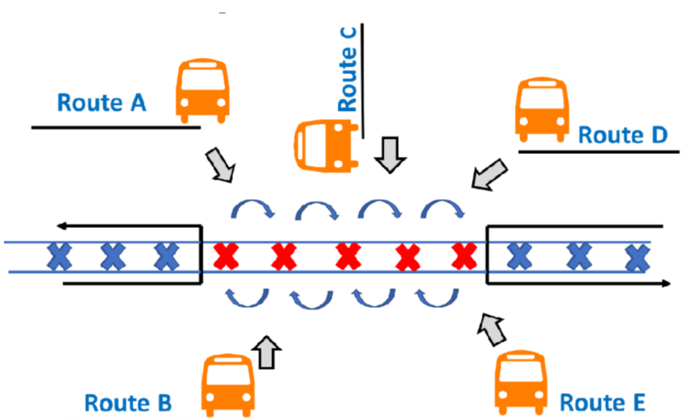


Physics Simulations





Realtime games and
Nondeterministic games



Bus regulation problem

Travelling Salesman Problem

A complete walkthrough with example

Upper confidence bound value

- The upper confidence bound for a node is defined as

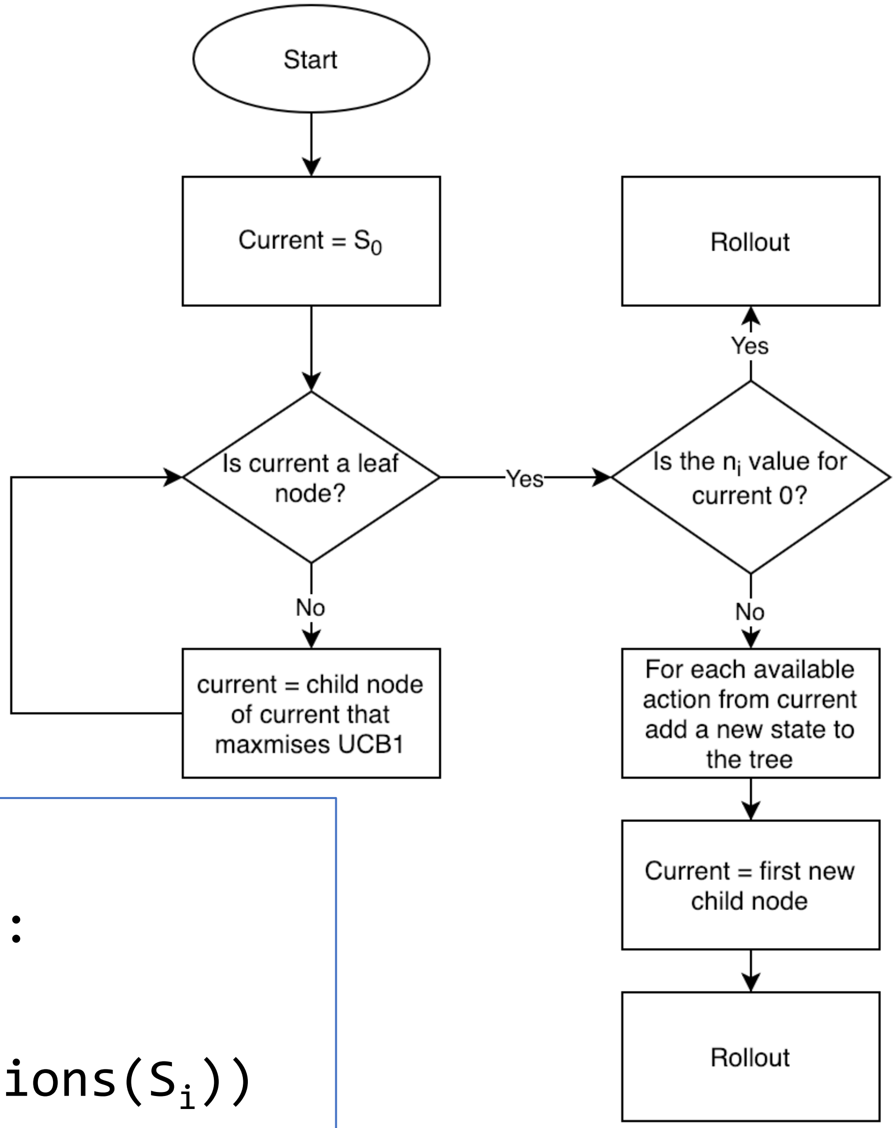
$$\mathbb{UCB}\mathbf{1} = \boldsymbol{V_i} + \boldsymbol{C}\sqrt{\frac{\ln \boldsymbol{N}}{\boldsymbol{n_i}}}$$

  - V_i: the value estimate of the node, which is the average reward of all nodes beneath this node
  - C is a tunable bias parameter (in this example, $C = 2$)
  - N: number of times the **parent node** has been visited
  - n_i: number of times the **current node** has been visited

- UCB1 can be served as a tree policy.

Roll out

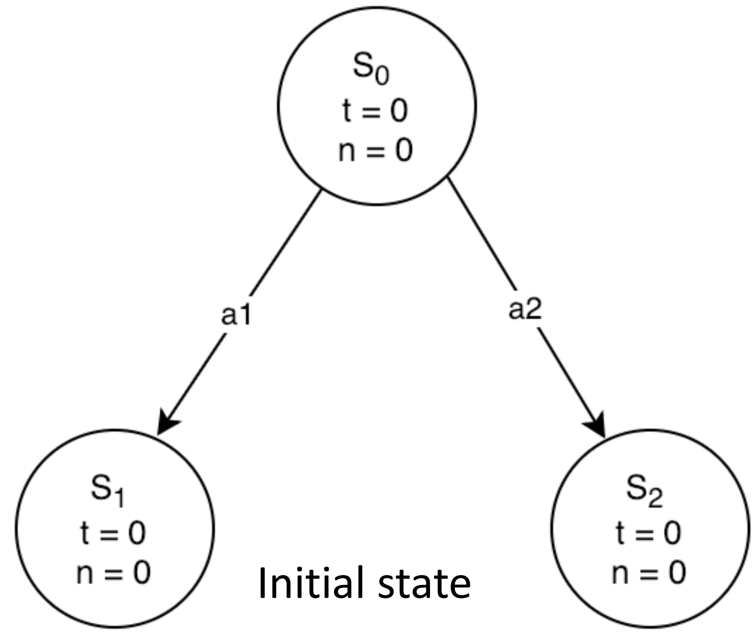
- Randomly pick an action at each step and simulate the action to receive an average reward when the game ends




```
Loop:
   if Sᵢ is a terminal state:
      return Value(Sᵢ)
   Aᵢ = random(available_actions(Sᵢ))
   Sᵢ = Simulate(Sᵢ, Aᵢ)
Until a terminal state is reached.
```

An example: Iteration 1

- Let's start with an initial state S_0.
- The actions, a_1 and a_2, lead to states S_1 and S_2, each has total score t and a number of visits n.



Initial state

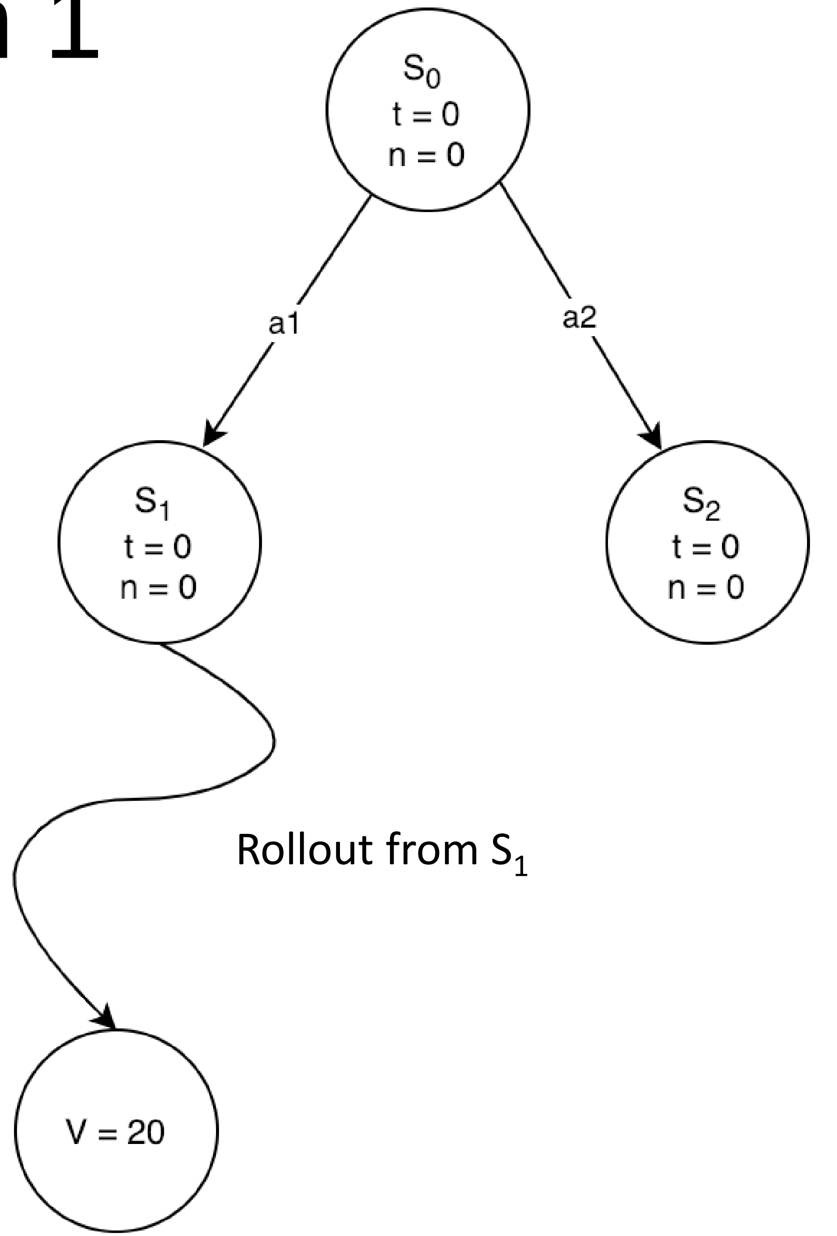
- *Q: How do we choose between the two child nodes?*
- A: Calculate the UCB values for both the child nodes and take whichever node that maximizes the UCB1 value.

  $\rightarrow$ Simply take the first node when no node has been visited yet.

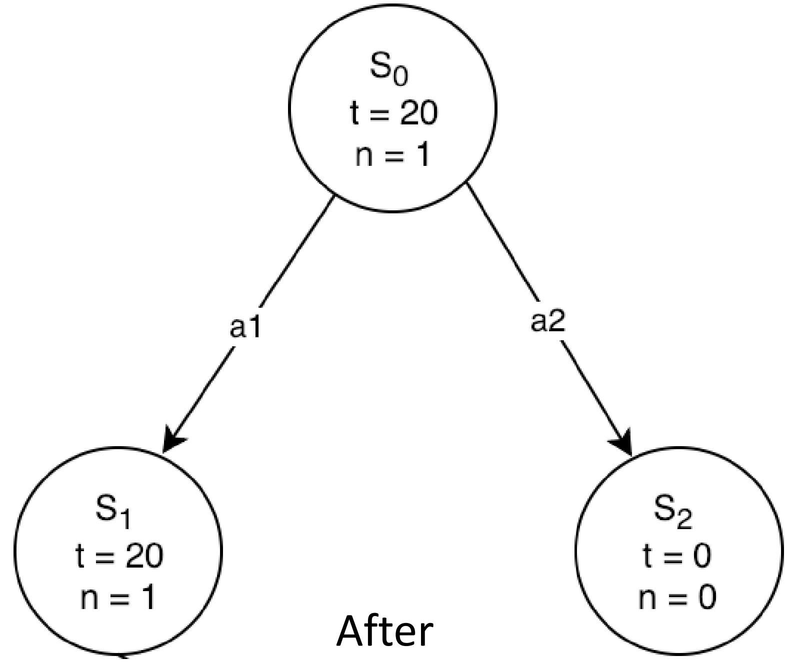
An example: Iteration 1

- The leaf node taken has not been visited before.

  $\rightarrow$ Do a rollout all the way down to the terminal state.
- Let's say the value of this rollout is 20 (i.e., just an example)



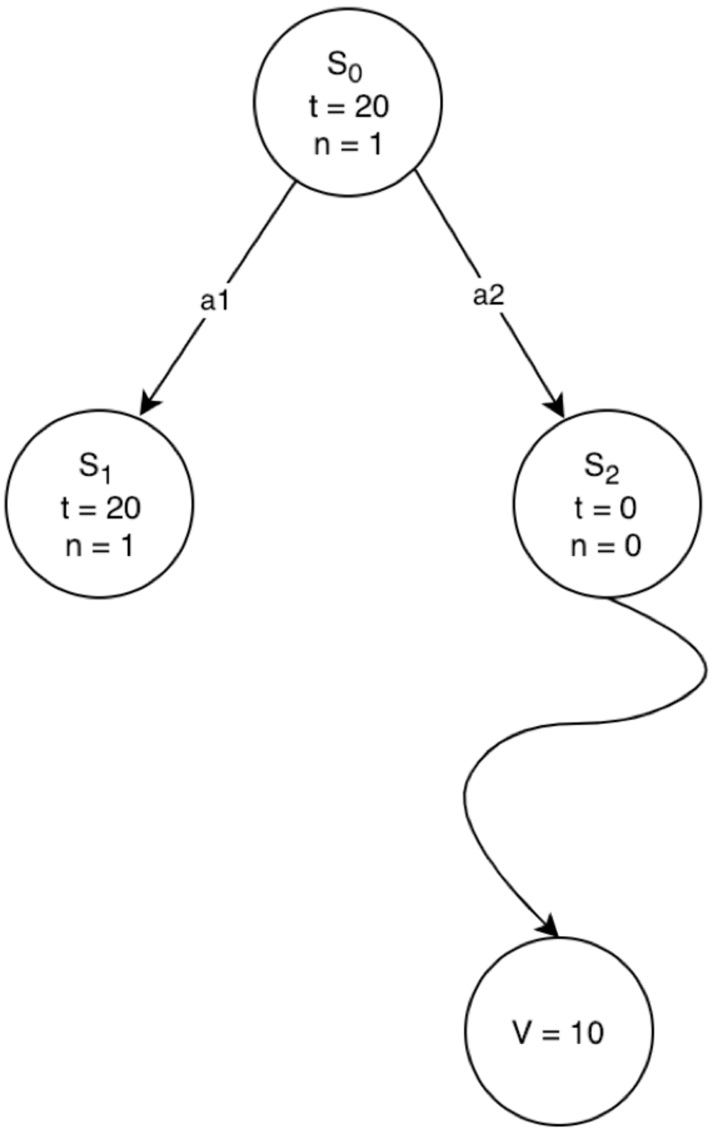
An example: Iteration 1

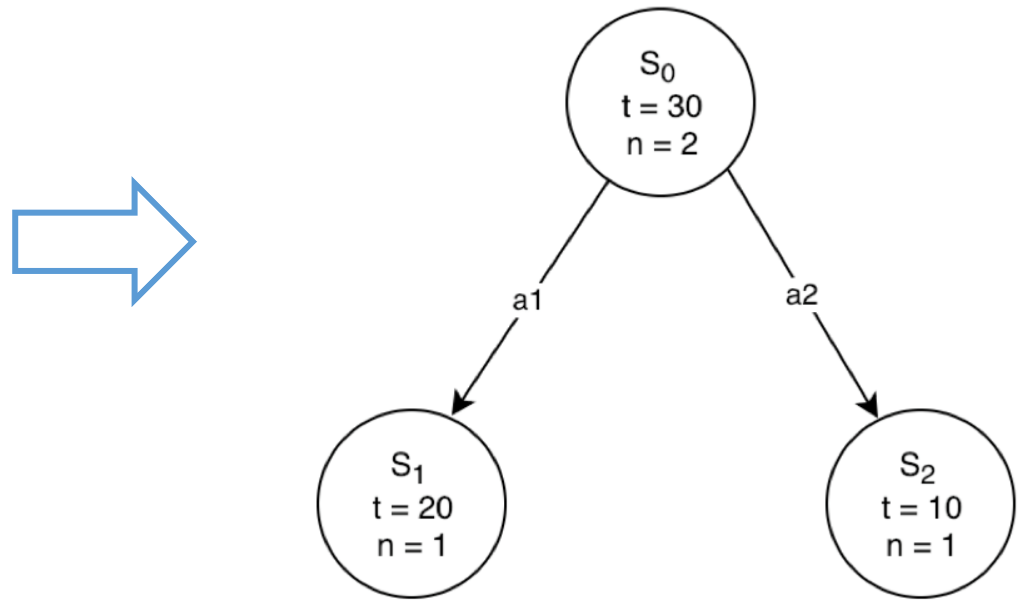


After backpropagation

- The value 20 is backed up all the way to the root.
- So now, $t = 20$ and $n = 1$ for nodes S_1 and S_0.
- That's the end of the first iteration

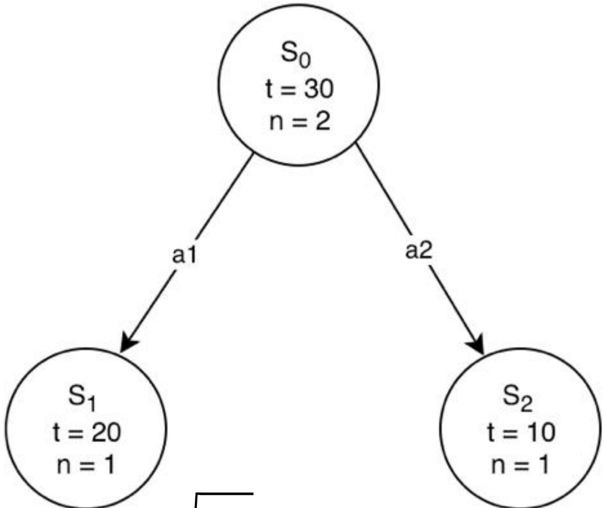
An example: Iteration 2





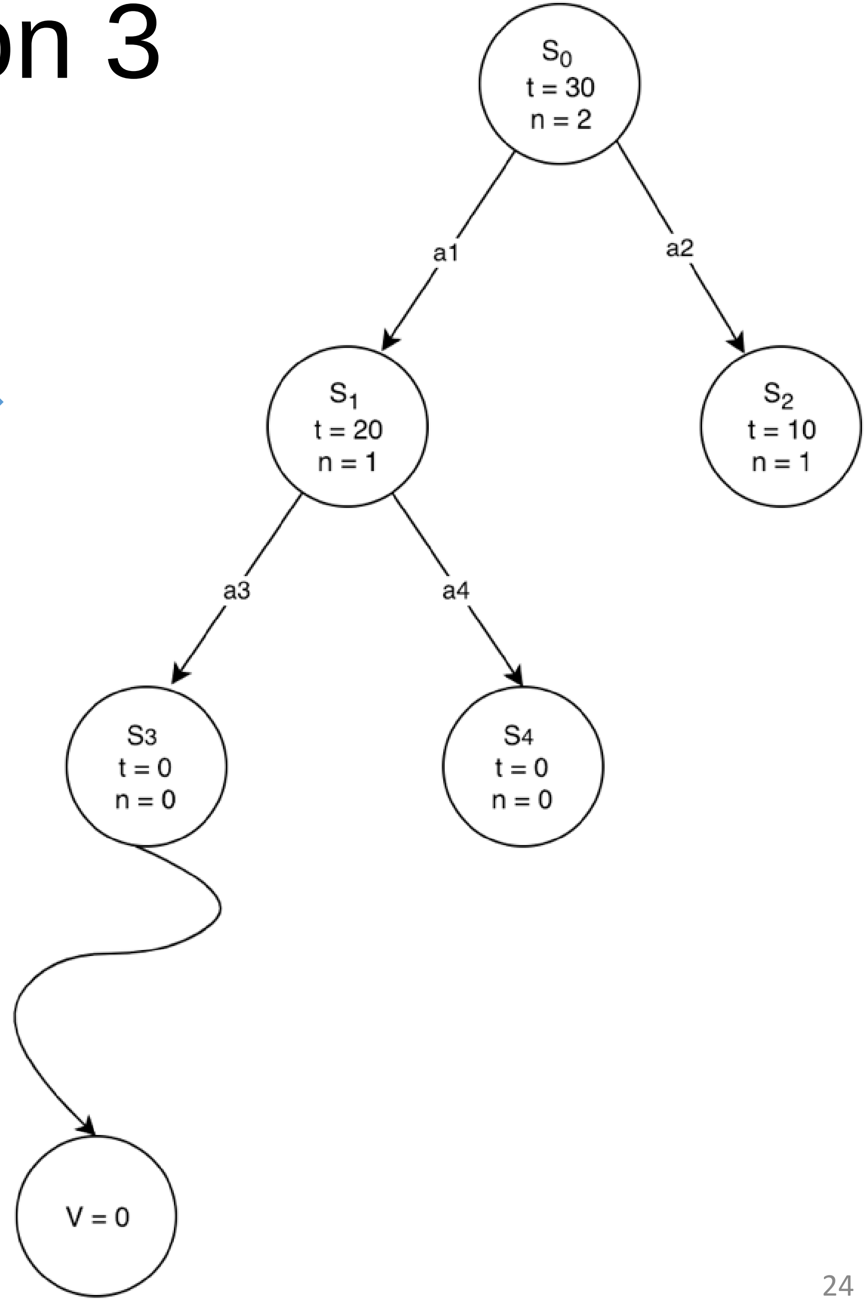
- The UCB values are 20 for S_1 and infinity for $S_2 \rightarrow$ visit S_2 next.
- Rollout at S_2 to get to the value 10.
- The value 10 is then backed up to the root $\rightarrow$ the value at root node now is 30

An example: Iteration 3



$$UCB1 = 20 + 2\sqrt{\frac{ln2}{1}} = 21.67$$

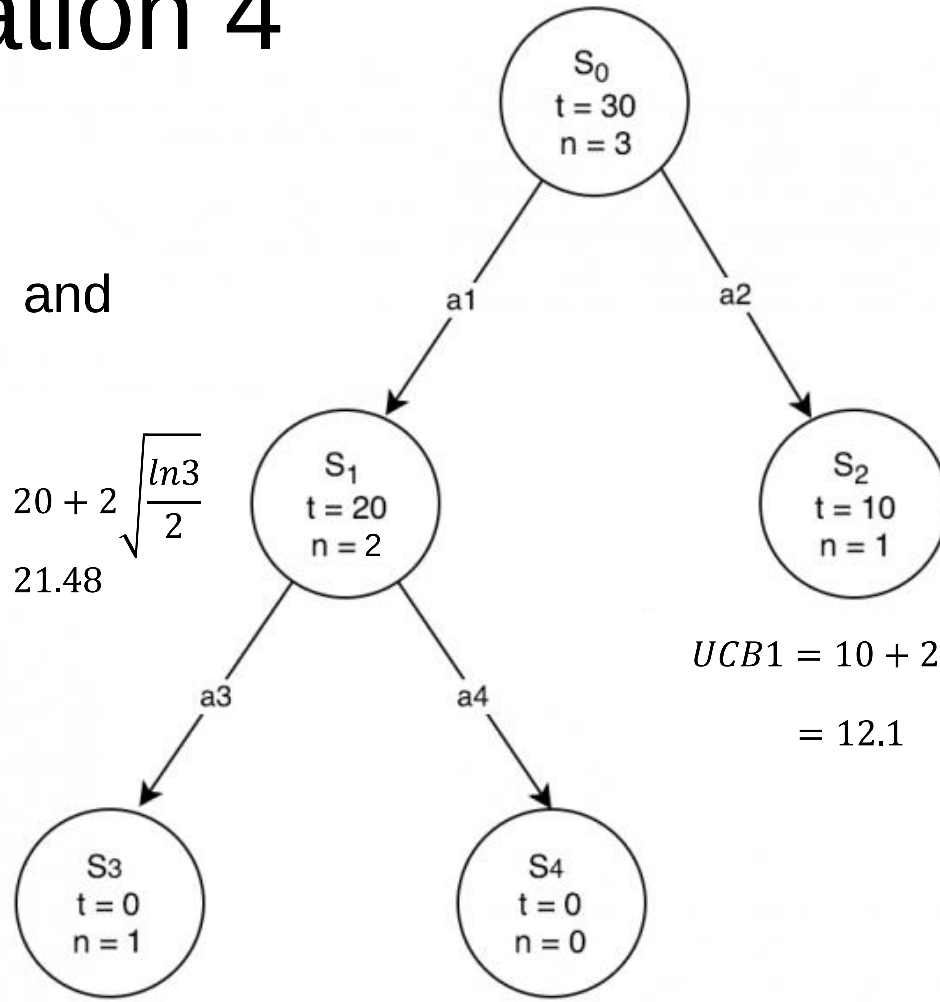
$$UCB1 = 10 + 2\sqrt{\frac{ln2}{1}} = 11.67$$



- S_1 has a higher UCB1 value
  $\rightarrow$ the expansion will be done here
- We do a rollout from S_3 and get
  a value of 0 at the leaf node

An example: Iteration 4

- S_1 has a higher UCB1 value
- The UCB values are 0 for S_3 and infinity for S_4 $\rightarrow$ visit S_4 next.



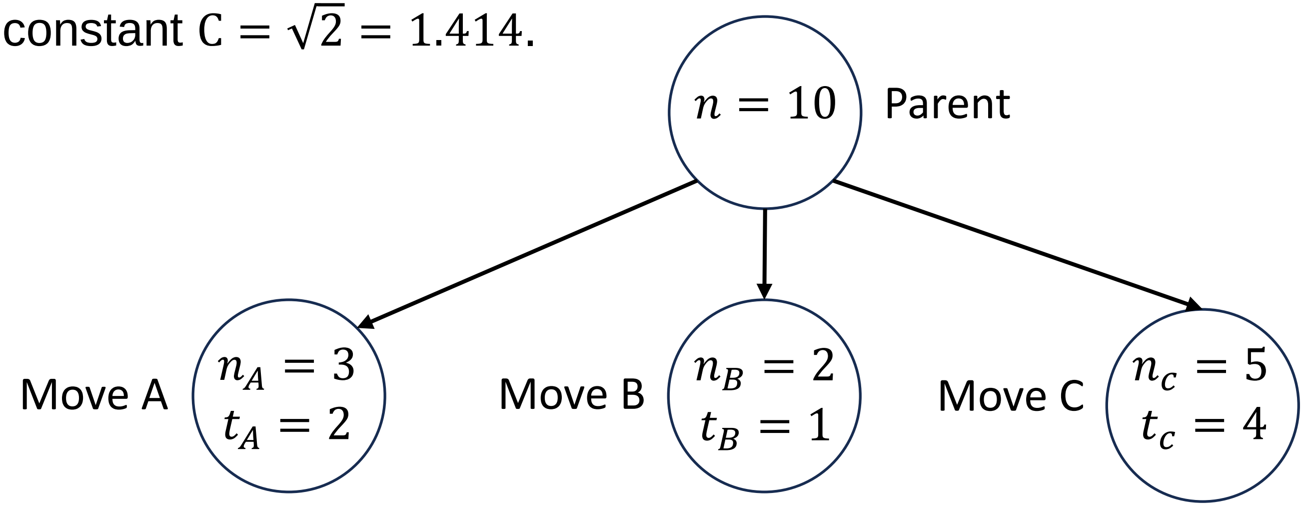
$$UCB1 = 20 + 2\sqrt{\frac{ln3}{2}}$$
$$= 21.48$$

$$UCB1 = 10 + 2\sqrt{\frac{ln3}{1}}$$
$$= 12.1$$

- A rollout is done till the leaf node to get the value and backpropagate.

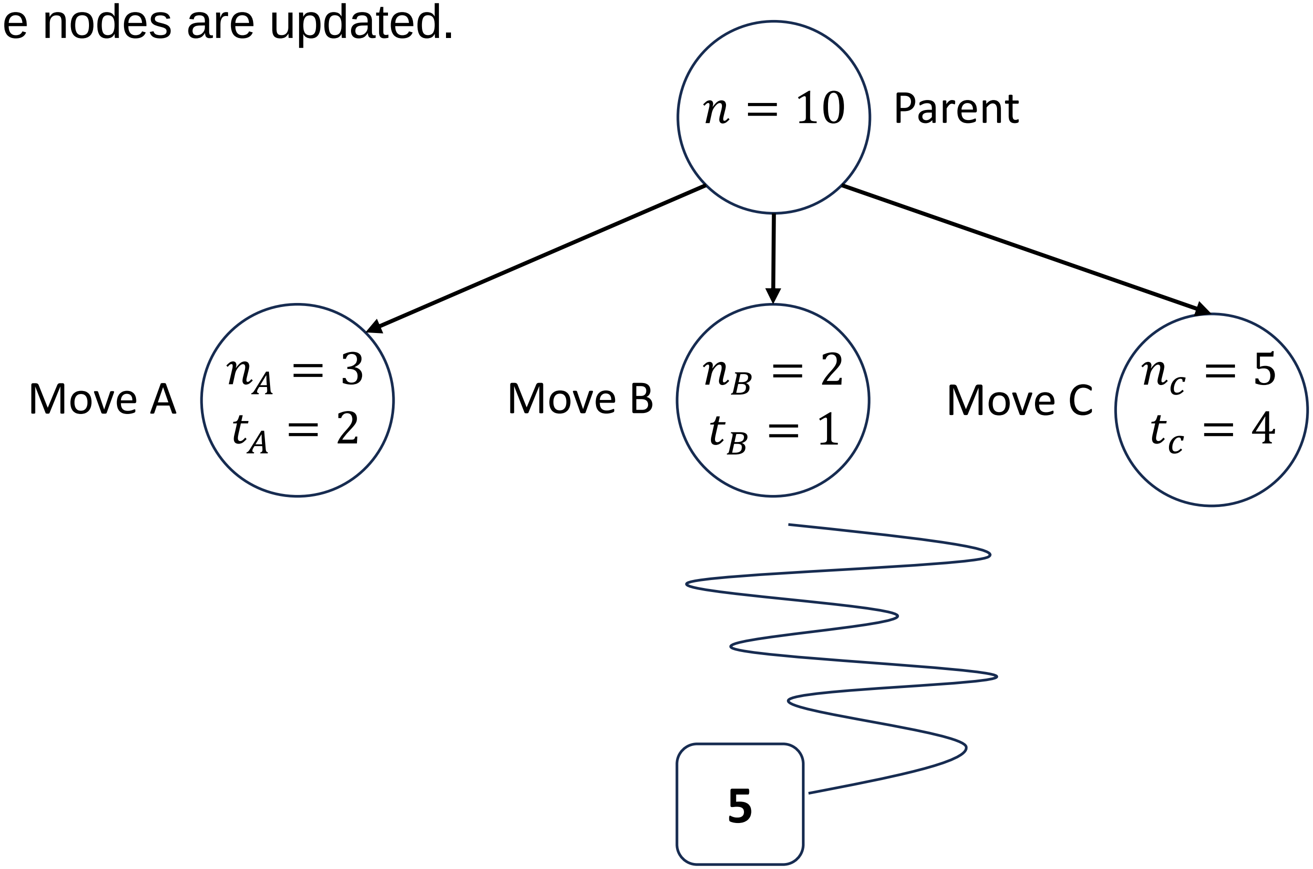
Quiz 01: UCB1 Calculation

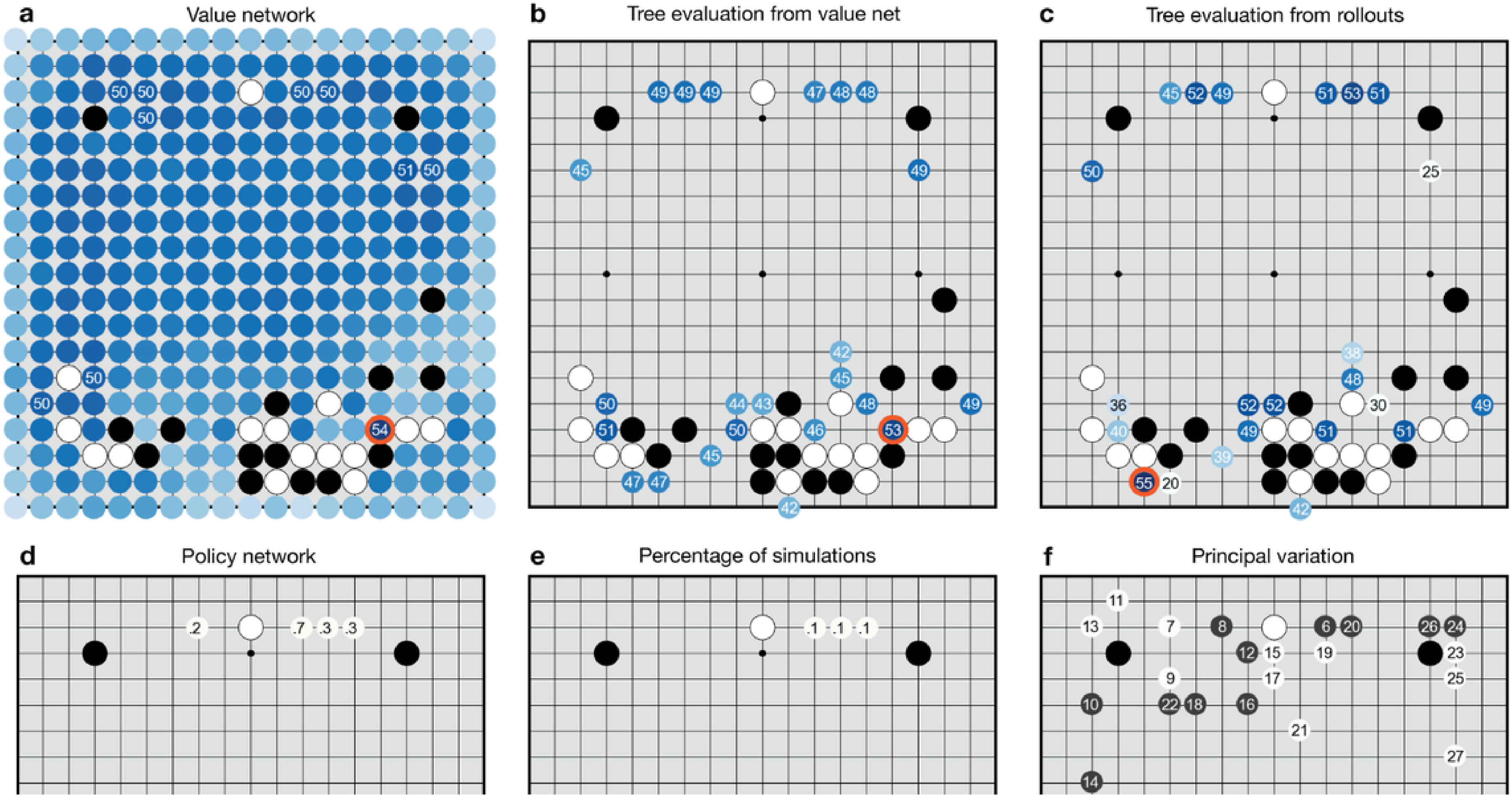
- Let's calculate MCTS with UCB1 using an example from a simple game like Tic-Tac-Toe.
- We have three possible moves (nodes) to explore: A, B, and C.
- The total number of times the parent node has been visited is $n = 10$.
- Move A has been visited $n_A = 3$ times and has a total reward of $t_A = 2$.
- Move B has been visited $n_B = 2$ times and has a total reward of $t_B = 1$.
- Move C has been visited $n_C = 5$. times and has a total reward of $t_C = 4$.
- The exploration constant $\mathrm{C} = \sqrt{2} = 1.414$.



Quiz 01: UCB1 Calculation

- Let's perform a roll-out from the parent all the way down to the terminal state.
- Assume that any expansion will generate two more successor, tie nodes are resolved from left to right, and the value of this rollout is 5.
- Show how the nodes are updated.

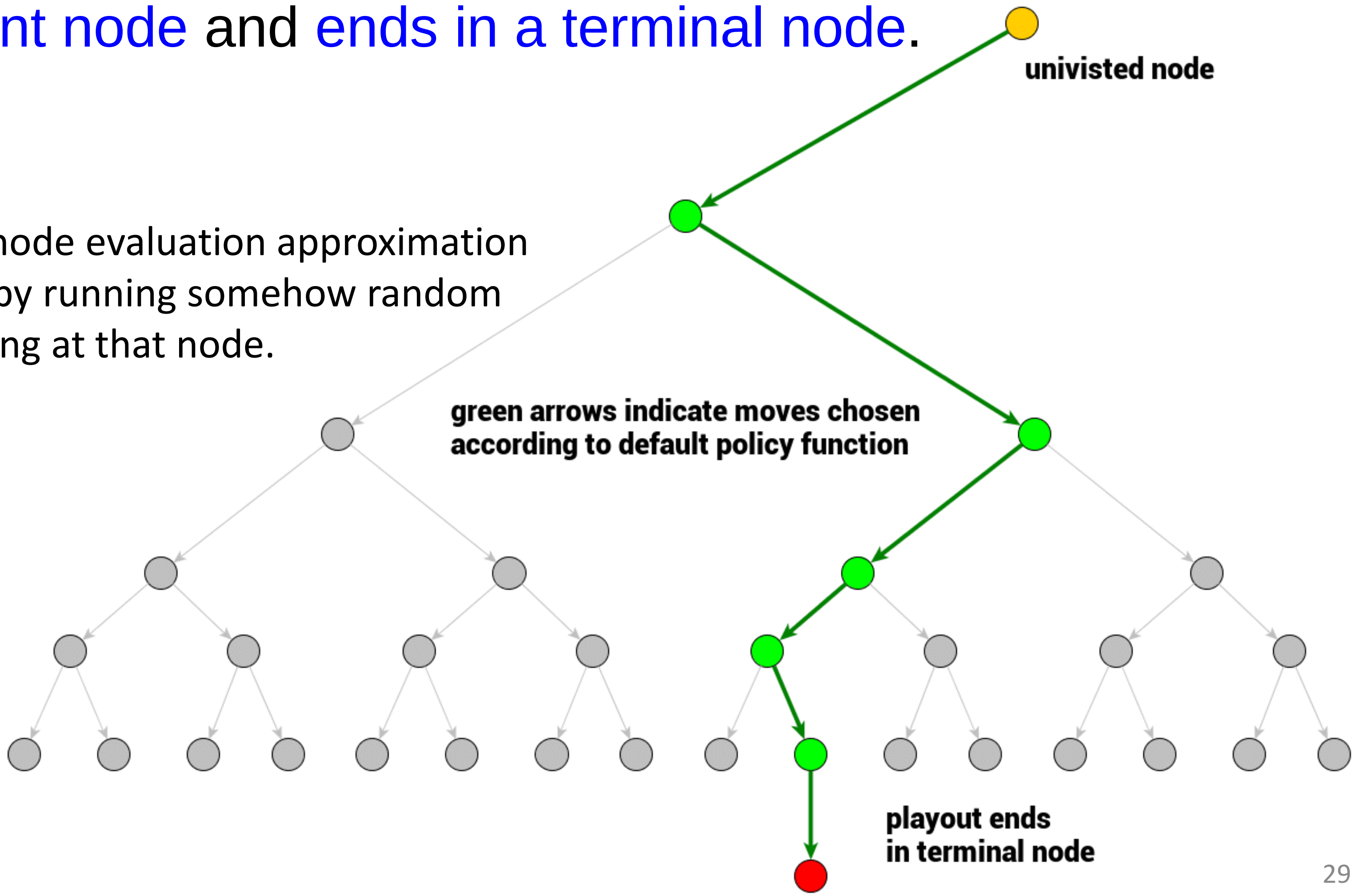




A deeper insight into MCTS

Simulation / Playout

- Simulation (or playout) is a sequence of moves that starts in a current node and ends in a terminal node.



It is a tree node evaluation approximation computed by running somehow random game starting at that node.

Simulation / Playout

- During the simulation, the rollout policy function consumes a game state $\boldsymbol{s_i}$ and produces the next move/action $\boldsymbol{a_i}$.
  - The default rollout policy function is a uniform random.
  - In practice it is designed to be fast to allow many simulations being played quickly.
- Simulation always results in an evaluation.
  - It is a win, loss or a draw for the games, but generally any value is a legit result of a simulation.

Simulation in AlphaGo and AlphaZero

- In AlphaGo, the evaluation of the leaf S_L is defined a

$$V(S_L) = (1-\alpha)v_0(S_L) + \alpha z_L$$

  - z_L: a standard rollout evaluation with custom fast rollout policy, which is a shallow SoftMax neural network with handcrafted features.
  - v_0: a Value Network that evaluates positions by a 13-layer CNN, trained on 30mils distinct positions extracted from self-plays.

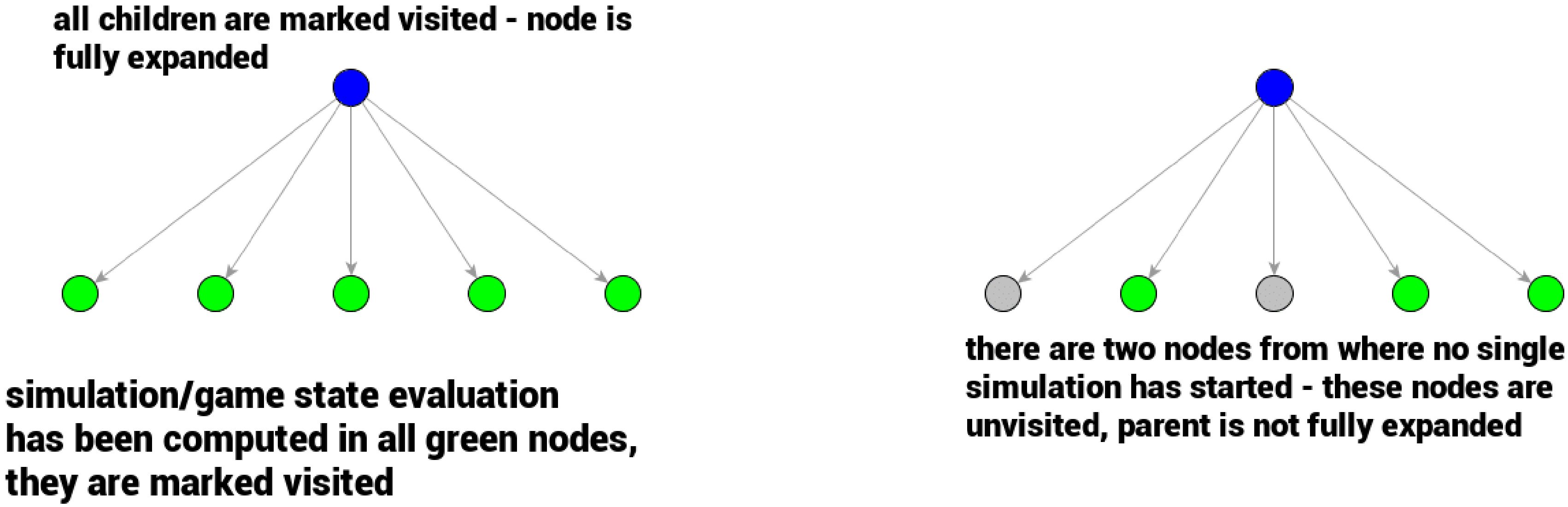
- In AlphaZero, a 19-layer CNN residual network directly rates the node.

$$V(S_L) = f_0(S_L)$$

  - It outputs both position evaluation and moves probability vector.

Node expansion on the game tree

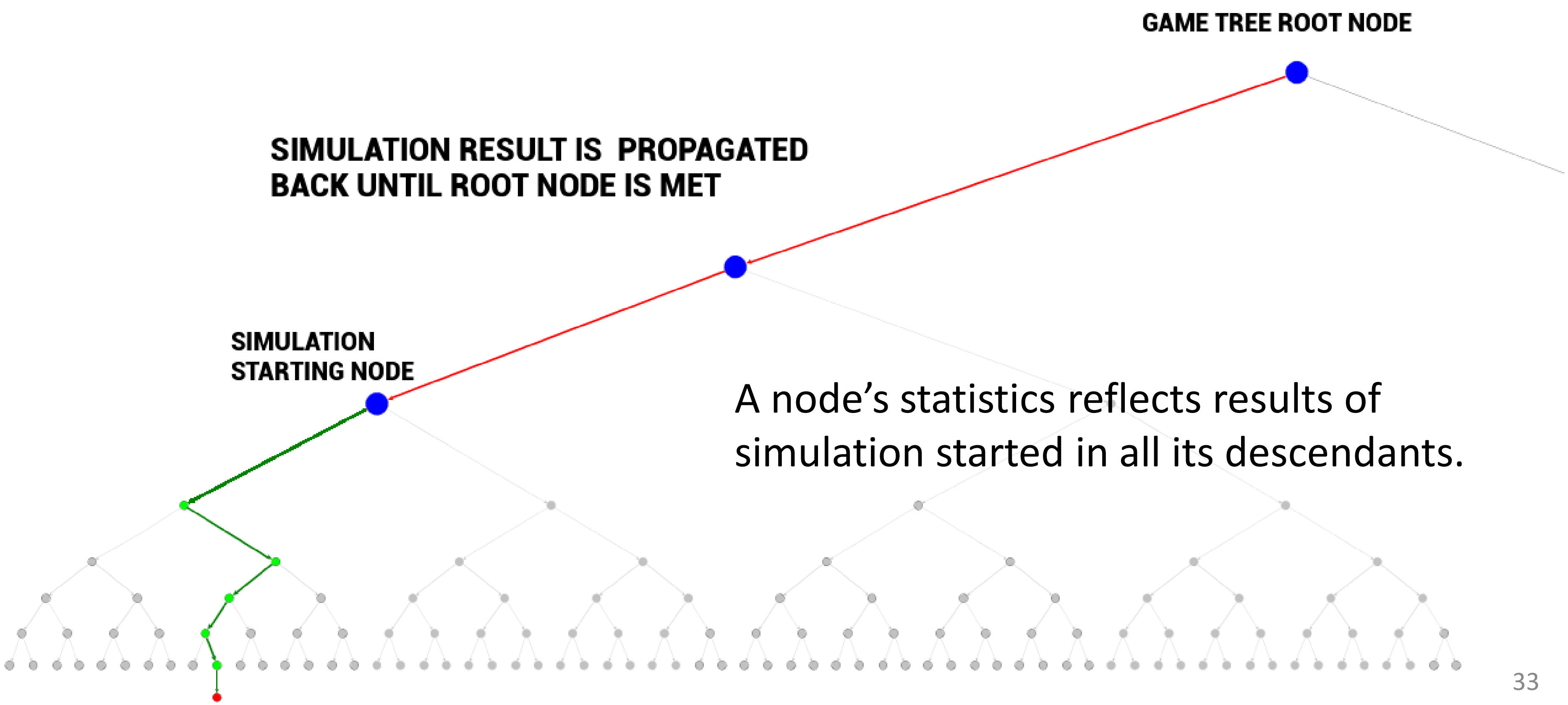
- A node is considered visited if a playout has been started in that node, i.e., has been evaluated at least once.



- A node is fully expanded if all its children are visited.

Backpropagation

- Backpropagation carries simulation results up to the root.
- For every node on the path, certain statistics are updated.



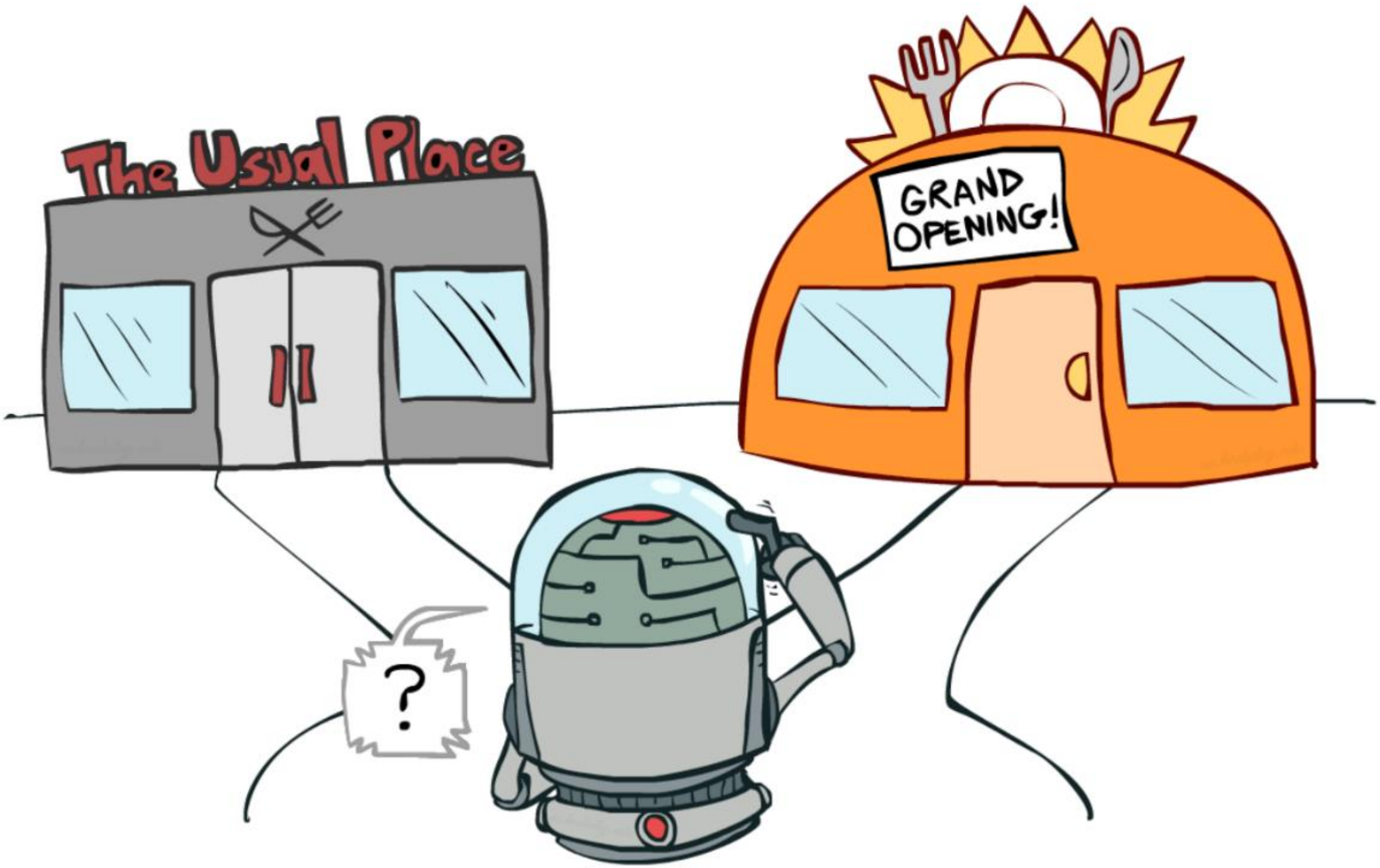
A node's statistics reflects results of simulation started in all its descendants.

Statistics in a node v

- $Q(v)$: the total simulation reward
  - In a simplest form, it is a sum of simulation results that passed through considered node.
- $N(v)$: the total number of visits
  - That is, a counter of how many times a node has been on the backpropagation path.

- $Q(v)$ indicates how promising the node is and $N(v)$ refers to how intensively explored it has been.
- These values are maintained for every visited node.

Exploration – Exploitation Dilemma



Nodes with high reward are good candidates to follow (exploitation) but those with low number of visits may be interesting too (because they are not explored well).

Upper Confidence Bound for trees

- Upper Confidence Bound for trees (UCT) lets us choose the next node among visited nodes to traverse through.

$$\mathbb{UCT}(\boldsymbol{v_i}, \boldsymbol{v}) = \frac{\boldsymbol{Q}(\boldsymbol{v_i})}{\boldsymbol{N}(\boldsymbol{v_i})} + \boldsymbol{C}\sqrt{\frac{\boldsymbol{ln}(\boldsymbol{N}(\boldsymbol{v}))}{\boldsymbol{N}(\boldsymbol{v_i})}}$$

  - C is a tunable bias parameter ($C = 1/\sqrt{2}$ for rewards in [0,1]).

- There is an essential balance between the first (exploitation) and second (exploration) terms.
- The exploration term ensures that each child has a nonzero probability of selection.

Algorithm 2 The UCT algorithm.

function $\textsc{UctSearch}(s_0)$
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow \textsc{TreePolicy}(v_0)$
    $\Delta \leftarrow \textsc{DefaultPolicy}(s(v_l))$
    $\textsc{Backup}(v_l, \Delta)$
  return $a(\textsc{BestChild}(v_0, 0))$

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

function $\textsc{TreePolicy}(v)$
  while v is nonterminal **do**
    if v not fully expanded **then**
      return $\textsc{Expand}(v)$
    else
      $v \leftarrow \textsc{BestChild}(v, Cp)$
  return v

function $\textsc{Expand}(v)$
  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'

function $\textsc{DefaultPolicy}(s)$
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function $\textsc{Backup}(v, \Delta)$
  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

MCTS pseudo-code

```
def monte_carlo_tree_search(root):
    while resources_left(time, computational power):
        leaf = traverse(root) # leaf = unvisited node
        simulation_result = rollout(leaf)
        backpropagate(leaf, simulation_result)
    return best_child(root)

def traverse(node):
    while fully_expanded(node):
        node = best_uct(node)
    return pick_univisted(node.children) or node # in case no
children are present / node is terminal

def rollout(node):
    while non_terminal(node):
        node = rollout_policy(node)
    return result(node)

def rollout_policy(node):
    return pick_random(node.children)

def backpropagate(node, result):
   if is_root(node) return
   node.stats = update_stats(node, result)
   backpropagate(node.parent)

def best_child(node):
    pick child with highest number of visits
```

List of references

- Browne, Cameron B., et al. "A survey of monte carlo tree search methods." IEEE Transactions on Computational Intelligence and AI in games 4.1 (2012): 1-43.
- Introduction to Monte Carlo Tree Search: The Game-Changing Algorithm behind DeepMind’s AlphaGo

  https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/
- Monte Carlo tree search: beginners guide

  https://int8.io/monte-carlo-tree-search-beginners-guide/
- Bruno Bouzy. Monte-Carlo Tree Search (MCTS) for Computer GO. Lecture notes for AOA class, Université Paris Descartes.

  https://helios.mi.parisdescartes.fr/~bouzy/Doc/AA2/MCTSGo-Bouzy.pdf


...the end.