ARTIFICIAL INTELLIGENCE

MONTE CARLO TREE SEARCH

A heuristic search algorithm for some kinds of decision processes, most notably those employed in software that plays board games.

SUPERVISED BY:

DR./ Mohamed Hussien

ARTIFICIAL INTELLIGENCE

Team Members

1. Ali Mohamed Ali
2. Omar Ahmed Abd-Alaziz
3. Marios Magid Atef
4. Hagar Ali El-Refaai
5. Hana Ahmed Nabhan

Monte Carlo tree search

AGENDA

- Introduction
- Principle of operation
- The 4 Phases of MCTS

1. Selecting
2. Expansion
3. Simulation
4. Backpropagation

- Exploration and Exploitation
- UCT Formula
- Flowchart
- Pseudocode
- Worked Example

1. Initial state
2. First iteration
3. second iteration
4. Third iteration

- Advantages & Disadvantages
- Common Q&A

Introduction

Definition

Monte Carlo Tree Search (MCTS) is a simulation-based tree search algorithm that uses random sampling to make decisions. It is widely used in game AI (such as Go, Chess, and Tic-Tac-Toe) and in complex planning problems.

Key Idea:

Build a search tree incrementally using random simulations to estimate the value of moves

Introduction

Where It's Used:

Used in in game play.

o MCTS was introduced in 2006 for computer Go

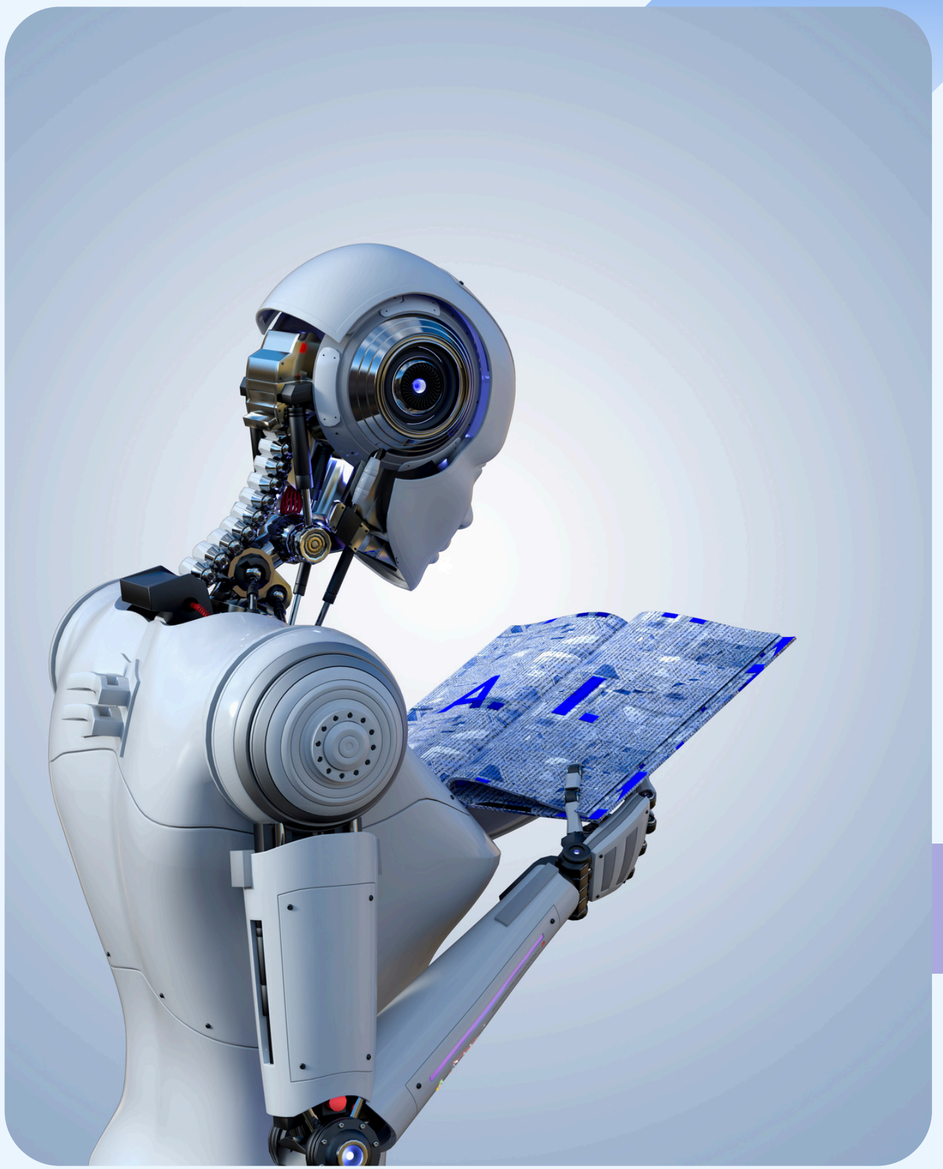
o Used in board games like chess and shogi

o Used in games with incomplete information such as bridge and poker

o Used in real-time video games such as Total War: Rome II's implementation of the high level campaign A1.

Principle of operation

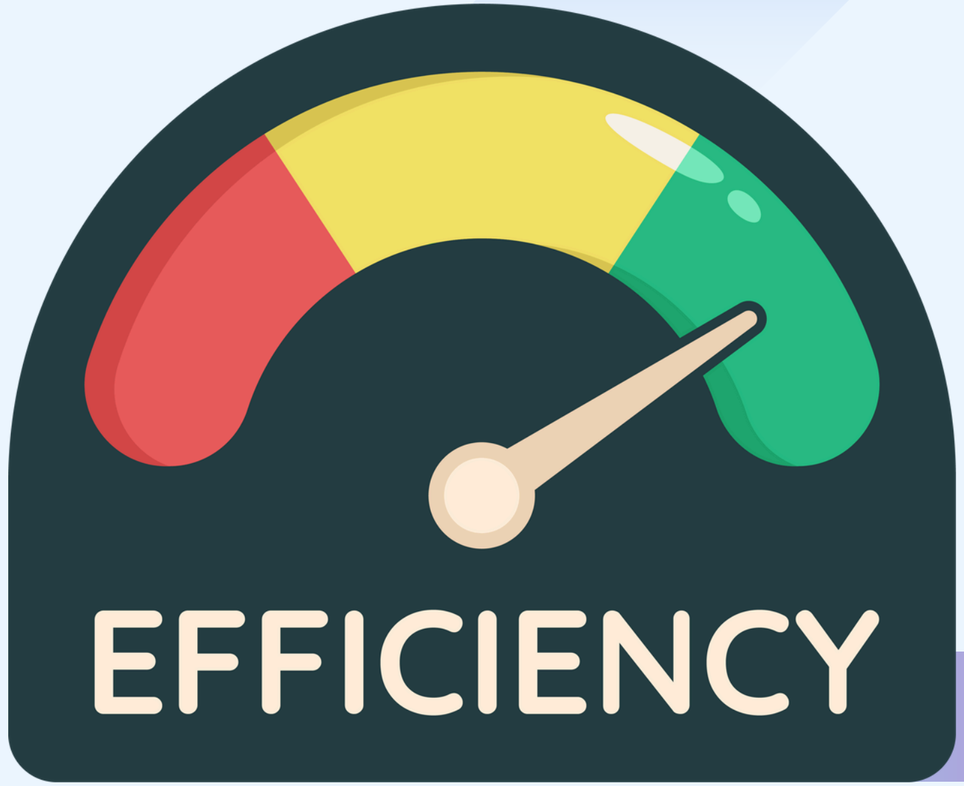
- The focus of MCTS is on the analysis of the most promising moves.
- It expands the search tree based on random sampling of the search space.
- The application of Monte Carlo tree search in games is based on many playouts, also called roll-outs.
- In each playout, the game is played out to the very end by selecting moves at random.



Monte Carlo tree search

Principle of operation

- The final game result of each playout is then used to weight the nodes in the game tree so that better nodes are more likely to be chosen in future playouts.
- The efficiency of this method—called Pure Monte Carlo Game Search—often increases with time as more playouts are assigned to the moves that have frequently resulted in the current player's victory according to previous playouts.



Monte Carlo tree search

The 4 Phases of MCTS

Each round of Monte Carlo tree search consists of four steps

Selecting

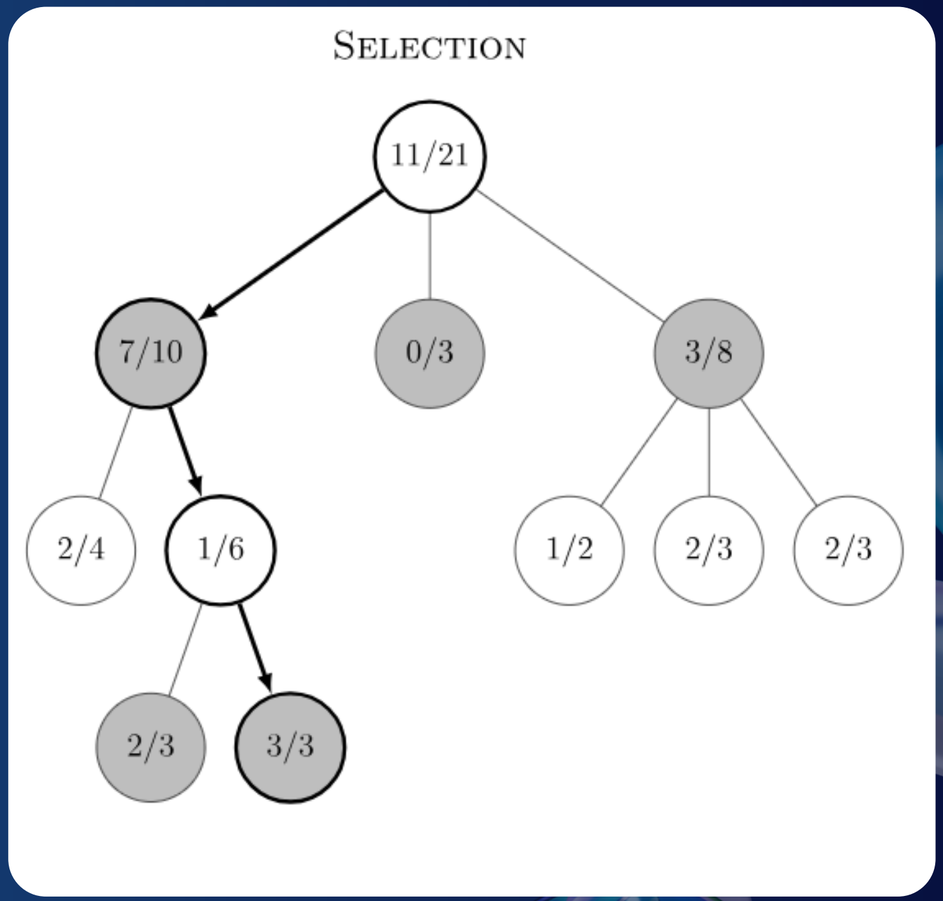
Expansion

Simulation

Backpropagation

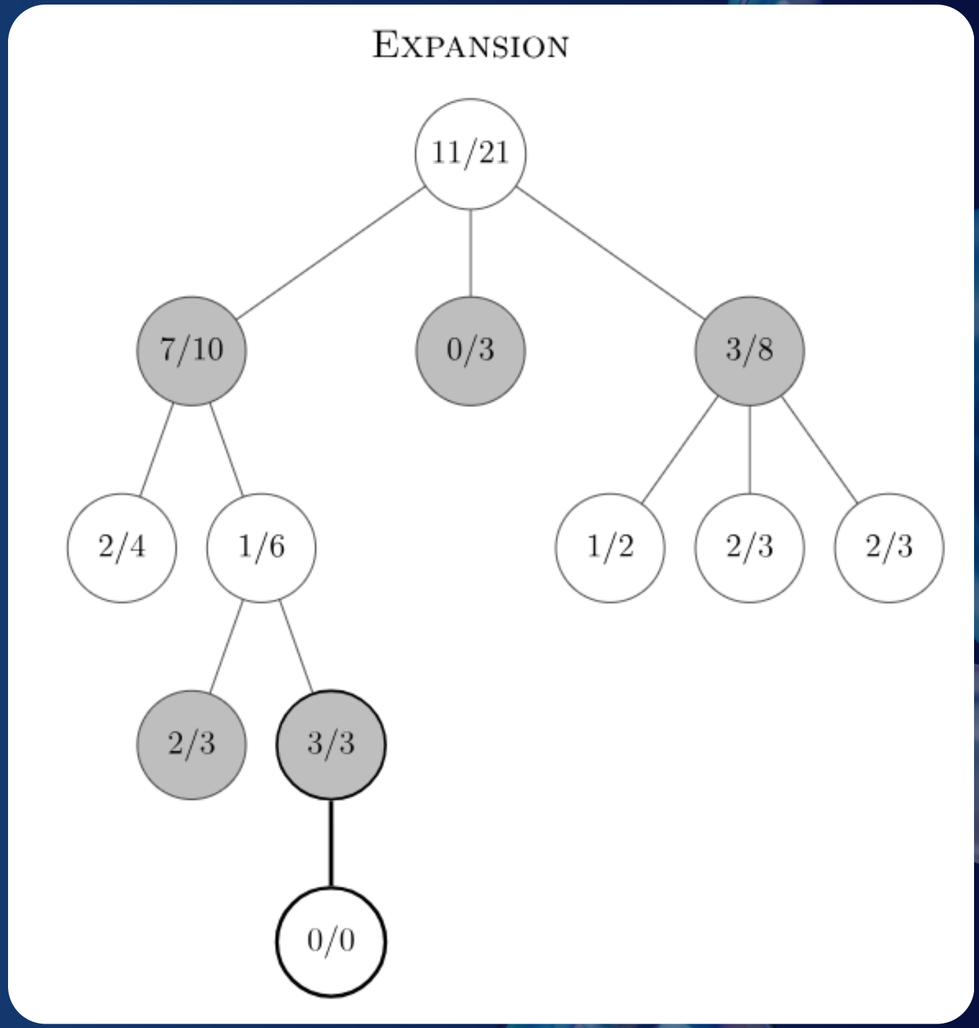
1- Selecting

- Start from root R and select successive child nodes until a leaf node L is reached. The root is the current game state and a leaf is any node that has a potential child from which no simulation (playout) has yet been initiated.
- The section says more about a way of biasing choice of child nodes that lets the game tree expand towards the most promising moves, which is the essence of Monte Carlo tree search.



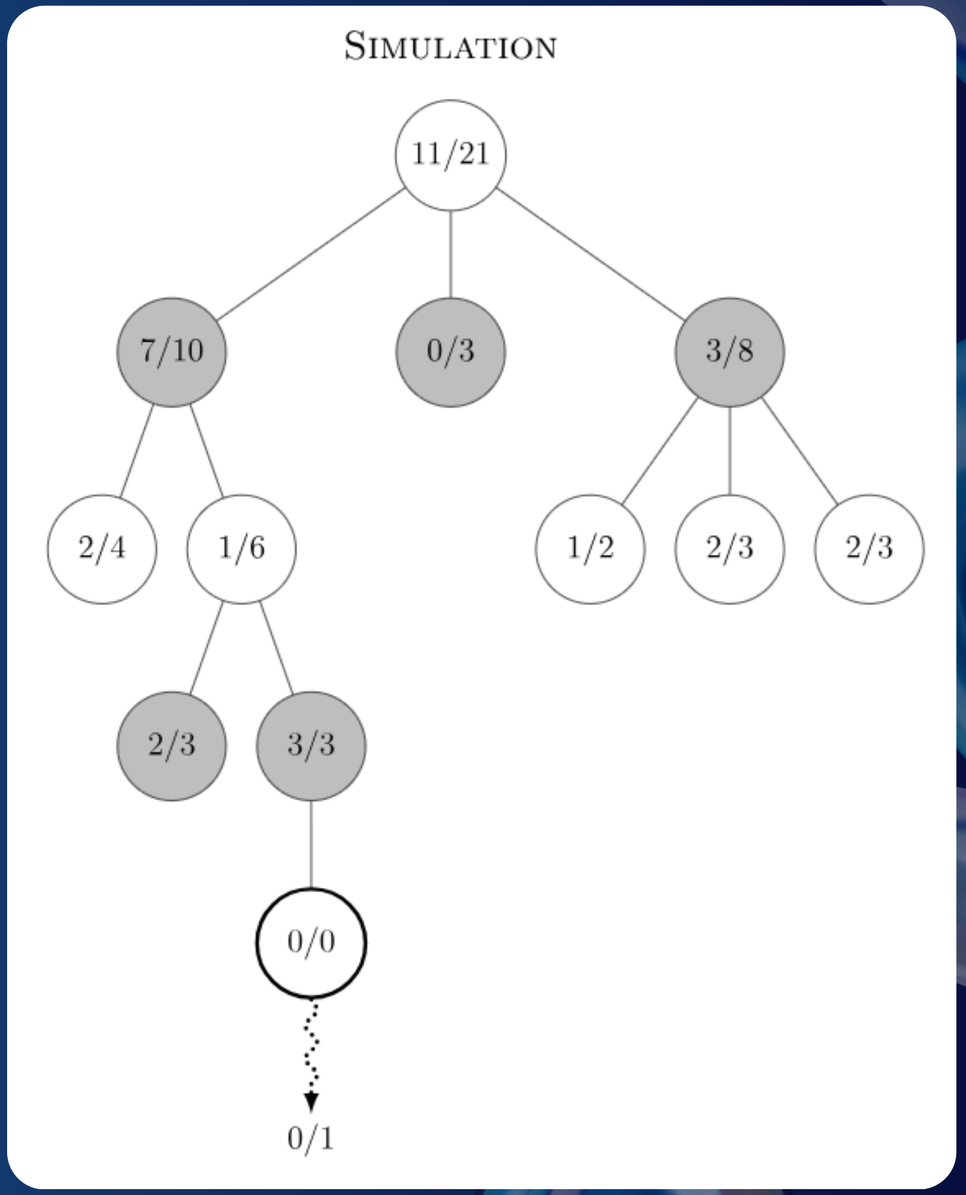
2- Expansion

- Unless L ends the game decisively (e.g. win/loss/draw) for either player, create one (or more) child nodes and choose node C from one of them. Child nodes are any valid moves from the game position defined by L.



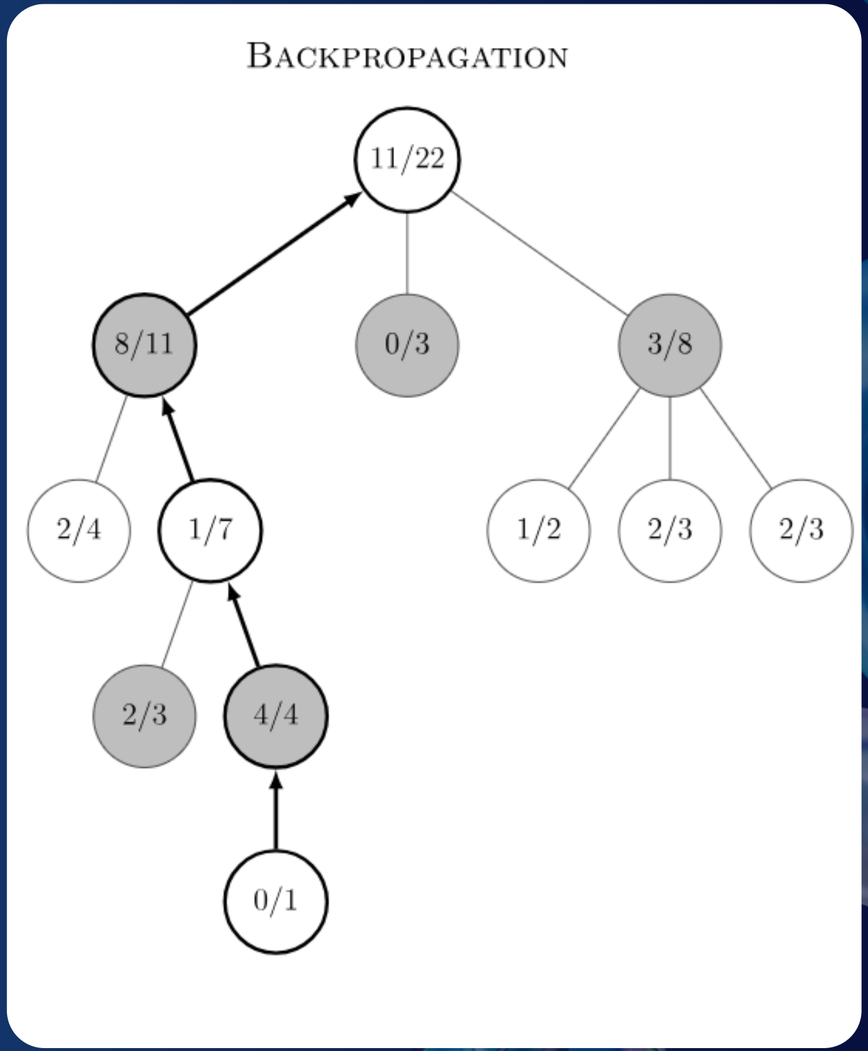
3- Simulation

- Complete one random playout from node C. This step is sometimes also called playout or rollout. A playout may be as simple as choosing uniform random moves until the game is decided (for example in chess, the game is won, lost, or drawn).



4- Backpropagation

- Use the result of the playout to update information in the nodes on the path from C to R.



Exploration and Exploitation

- The main difficulty in selecting child nodes is maintaining some balance between the exploitation of deep variants after moves with high average win rate and the exploration of moves with few simulations.

- The first formula for balancing exploitation and exploration in games, called UCT (Upper Confidence Bound)

UCT Formula (Upper Confidence Bound)

$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

- w_i → stands for the number of wins for the node considered after the i-th move
- n_i → stands for the number of simulations for the node considered after the i-th move
- N_i → stands for the total number of simulations after the i-th move run by the parent node of the one considered
- c → is the exploration parameter—theoretically equal to √2; in practice usually chosen empirically

UCT Formula (Upper Confidence Bound)

The UCT formula decides which child node to explore next by combining:

Exploitation

How good is this move based on past simulations?

$$\frac{w_i}{n_i}$$

Exploration

How much we haven't tried this move yet?

$$c.\sqrt{\frac{lnN}{n_i}}$$

Balanced Decision Making:

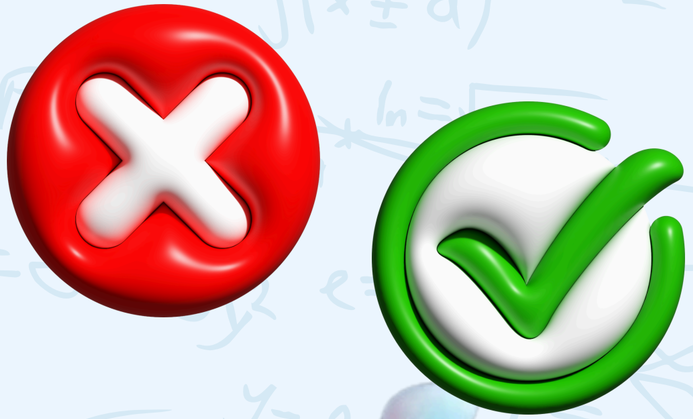
- If a node has been visited a lot, its exploration term becomes smaller → it won't be favored unless it has high win rate.
- If a node has been visited rarely, the exploration term becomes larger → we encourage trying new paths.

Why is it important?

Without UCT, MCTS might either:

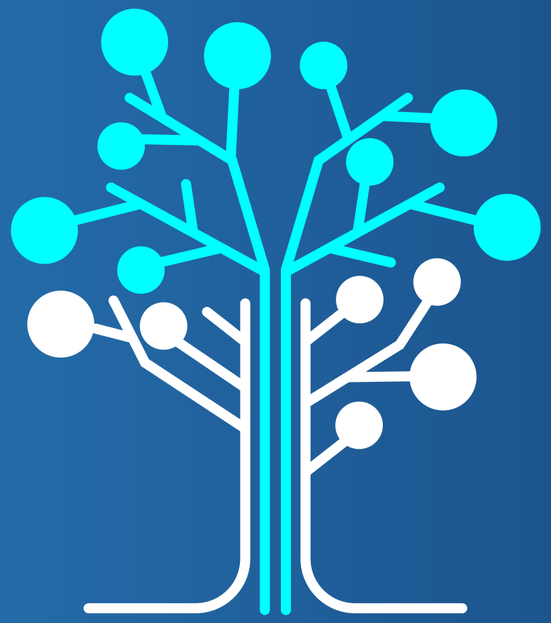
- Get stuck exploiting a move that looked good early on (without trying others).
- Or waste time exploring too many bad moves.

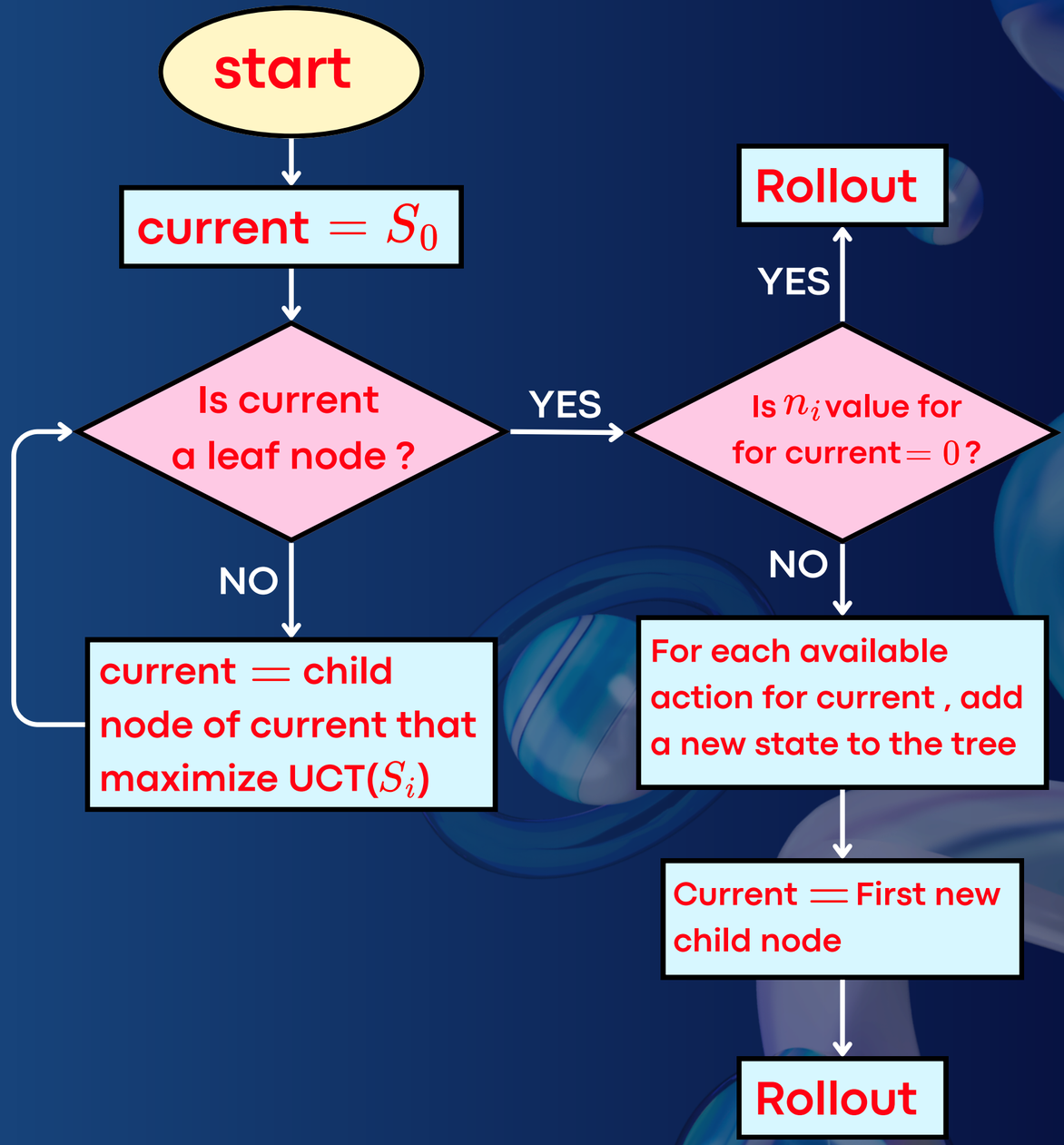
UCT gives the best of both worlds. 🌍



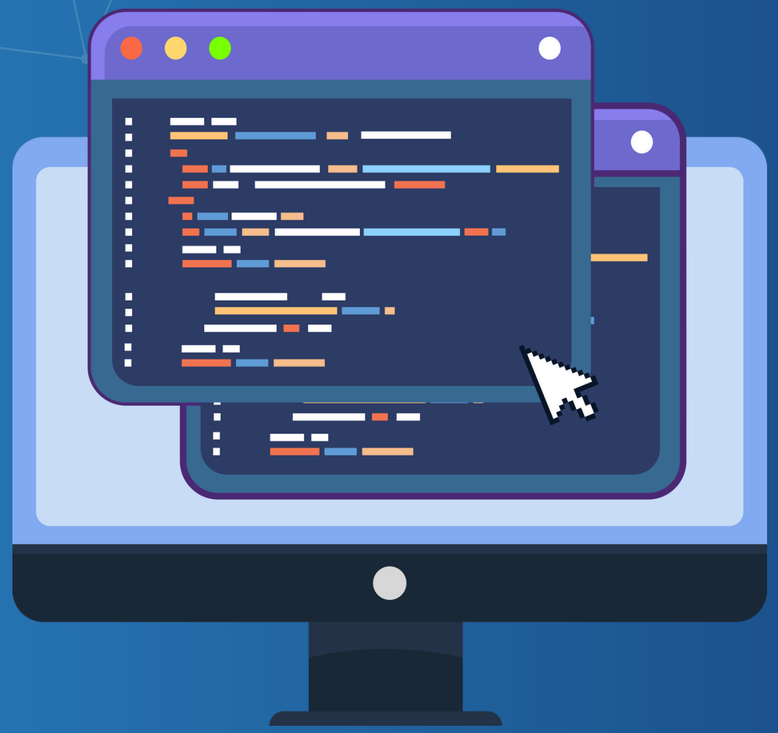
FLOWCHART

Algorithm for tree traversal and node Expansion





Pseudocode

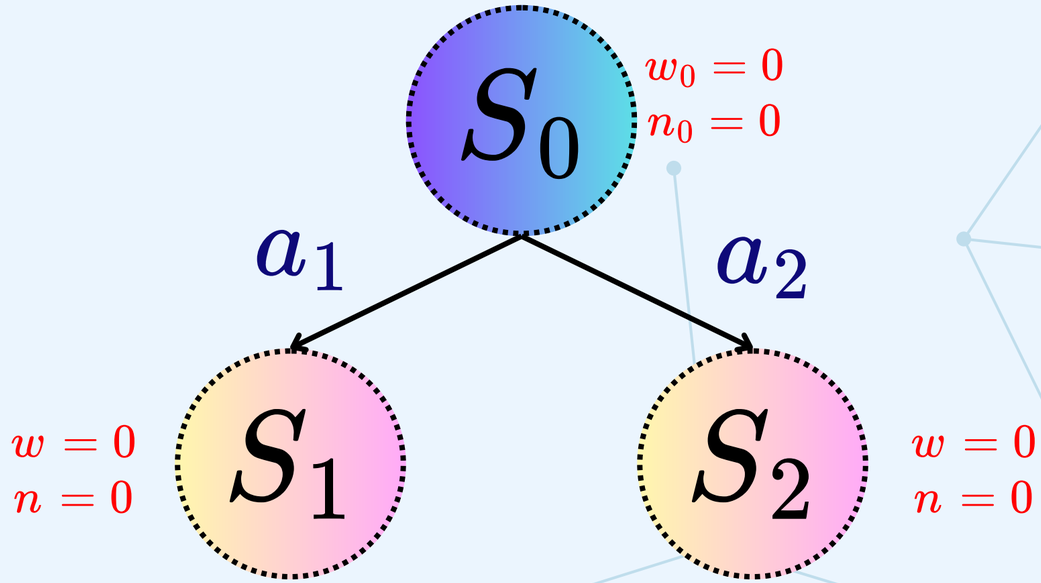



```
function TreeTraversalAndExpansion(root):
    current ← root  // S₀

    while current is not a leaf node:
        current ← child of current that maximizes UCT

    if n(current) = 0:  // node has never been visited
        return Rollout(current)

    else:
        for each available action from current:
            new_state ← result of applying action
            Add new_state as a child node to current

        current ← first new child node added
        return Rollout(current)
```

```
function Rollout(si):
    loop forever:
        if si is a terminal state:
            return value(si)

        ai ← random choice from available actions of si
        si ← simulate(si, ai)
```

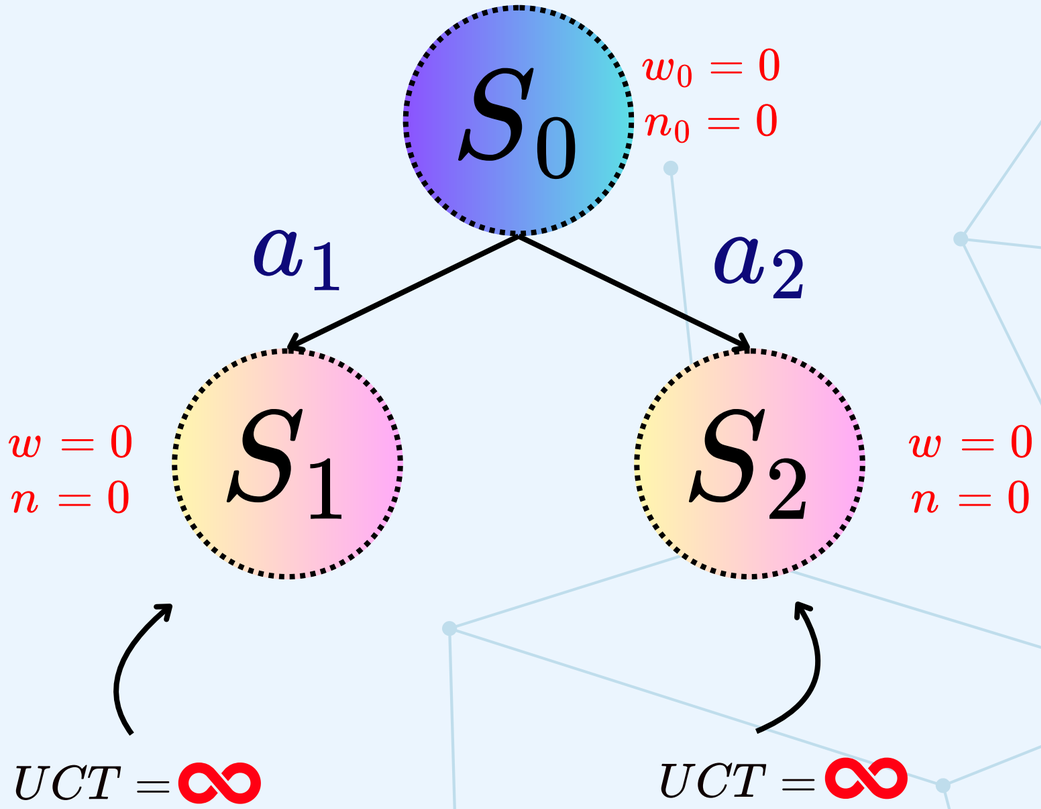
Initial state

$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$



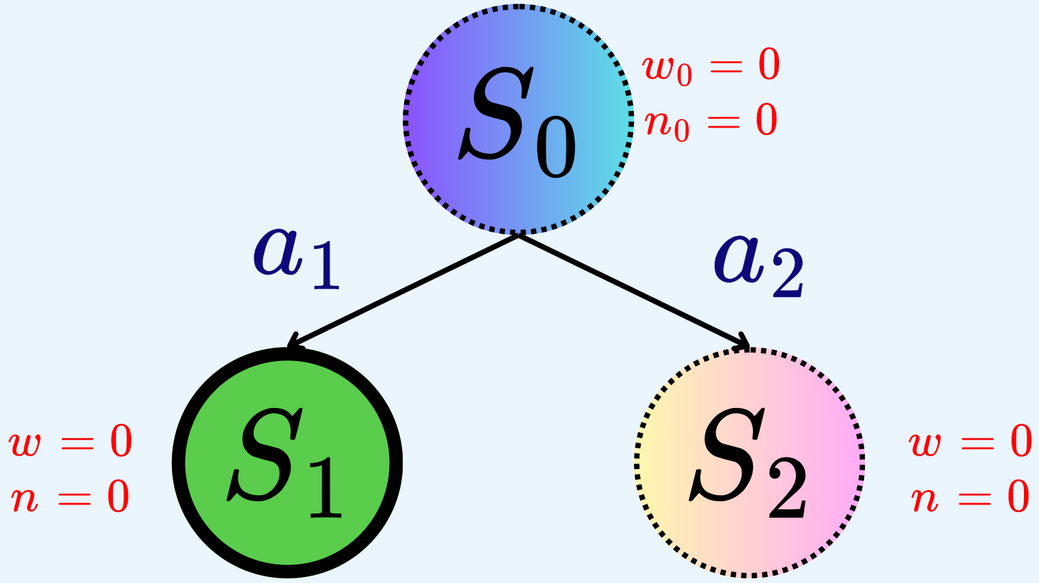
First Iteration

$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

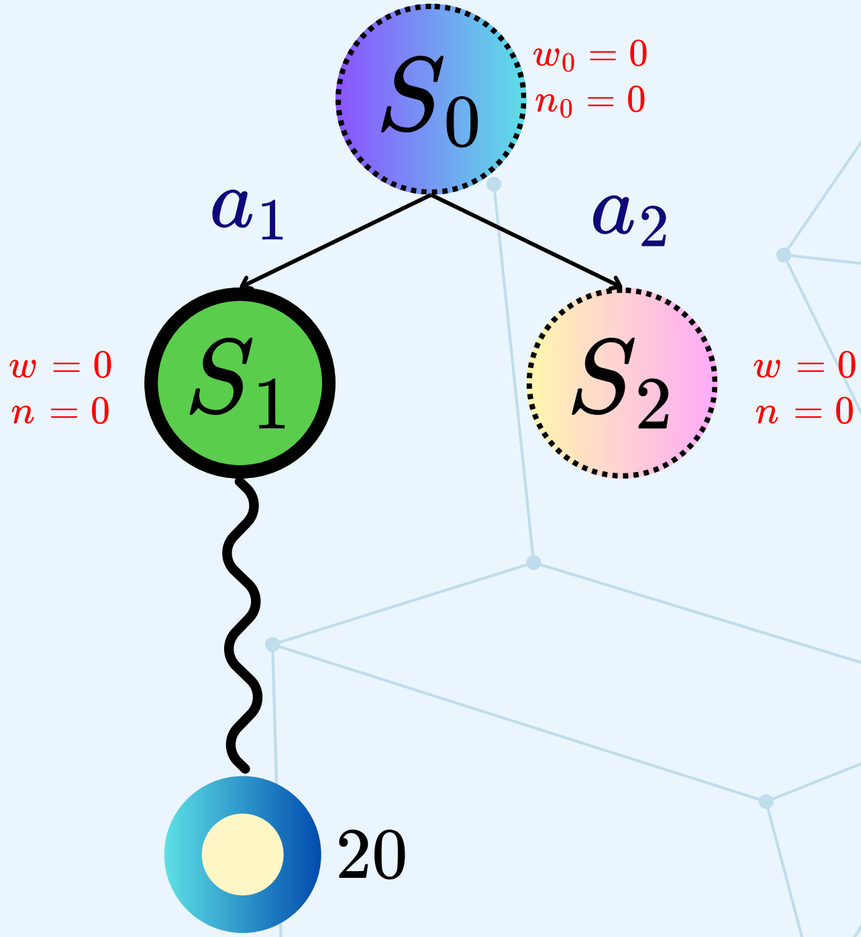


CHOOSE RANDOM

First Iteration

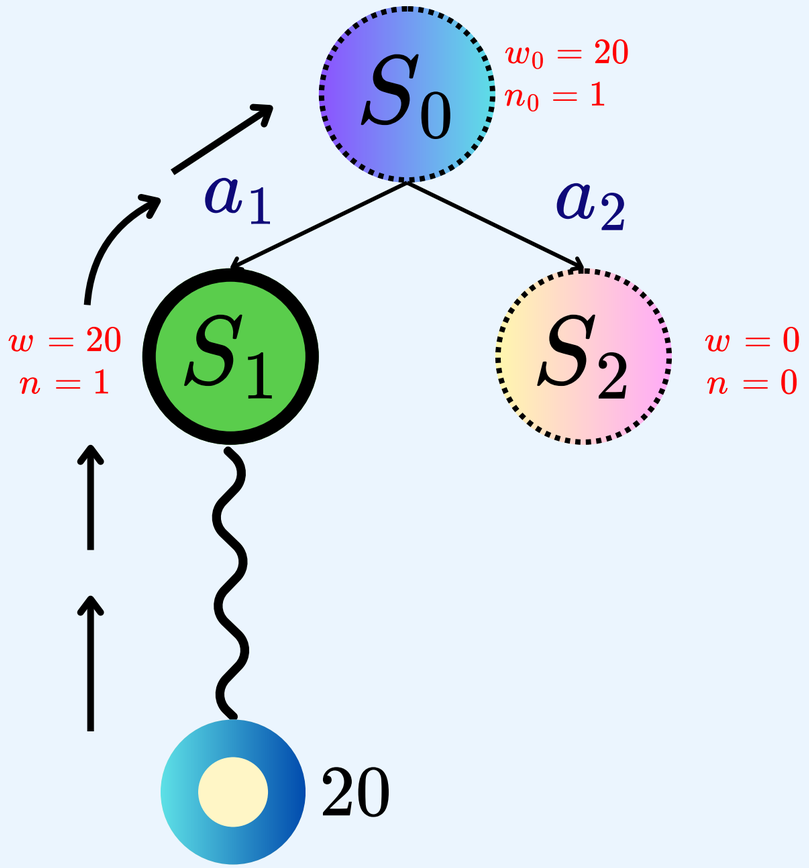


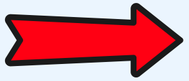
ROLLOUT !

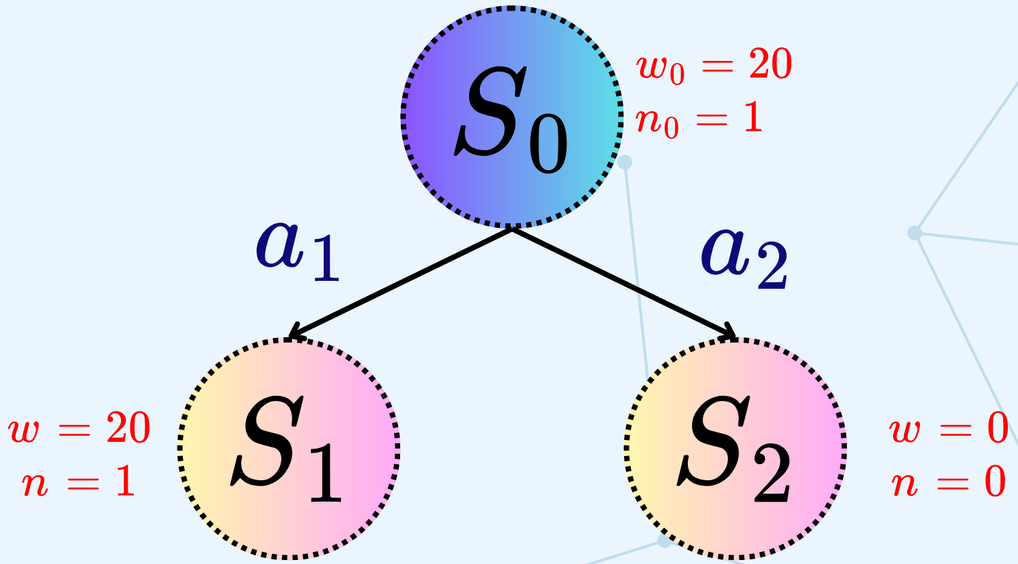


Since S1 has not been visited yet

First Iteration

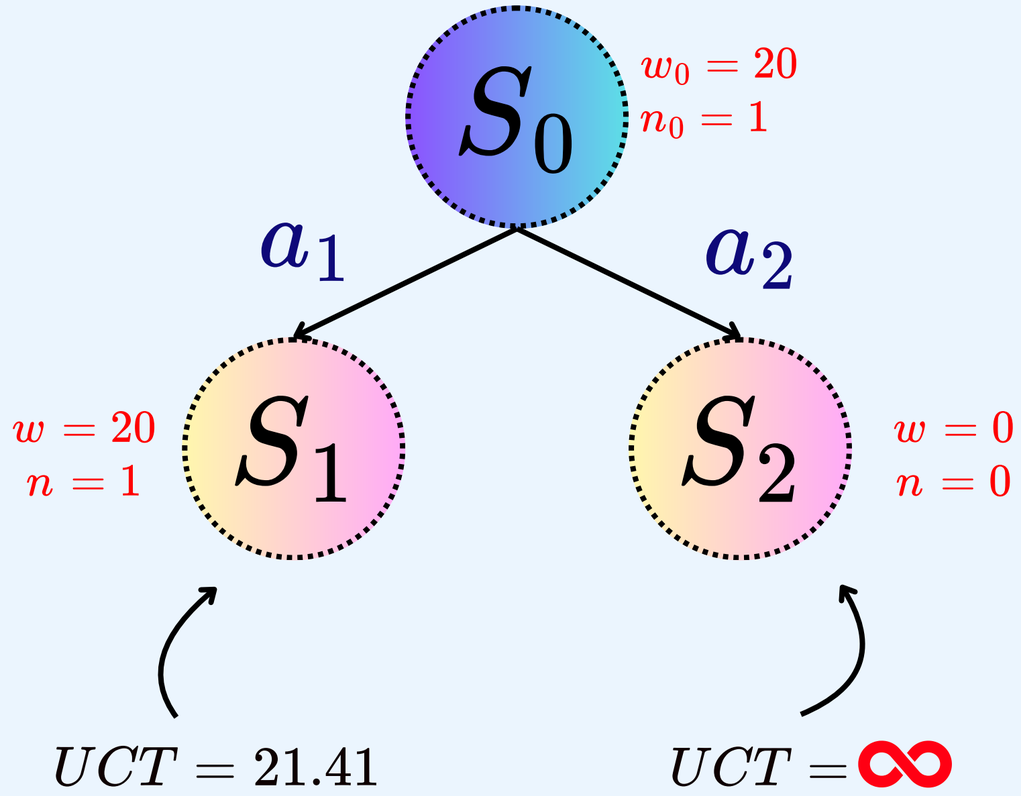




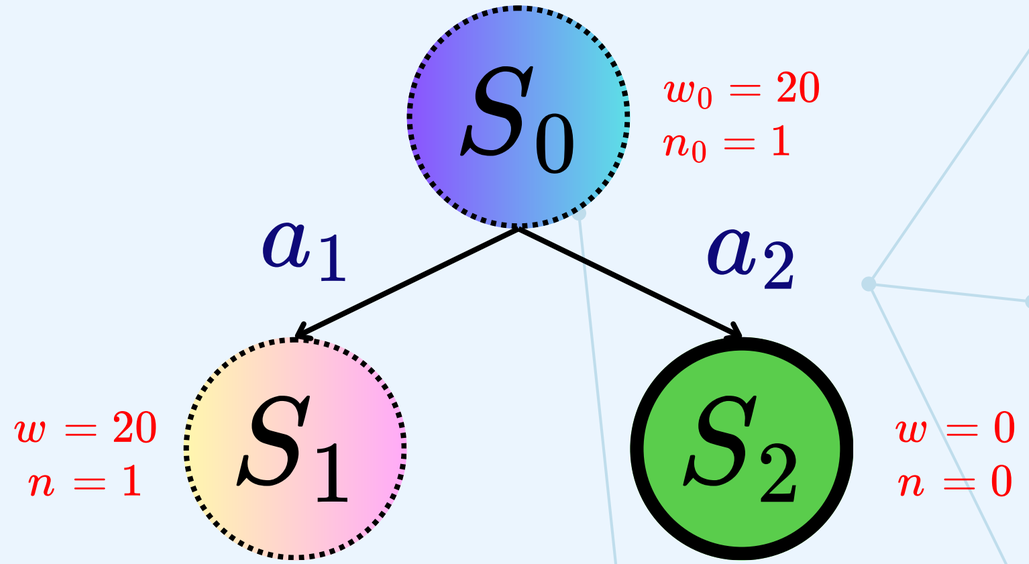


End of first iteration

Second Iteration

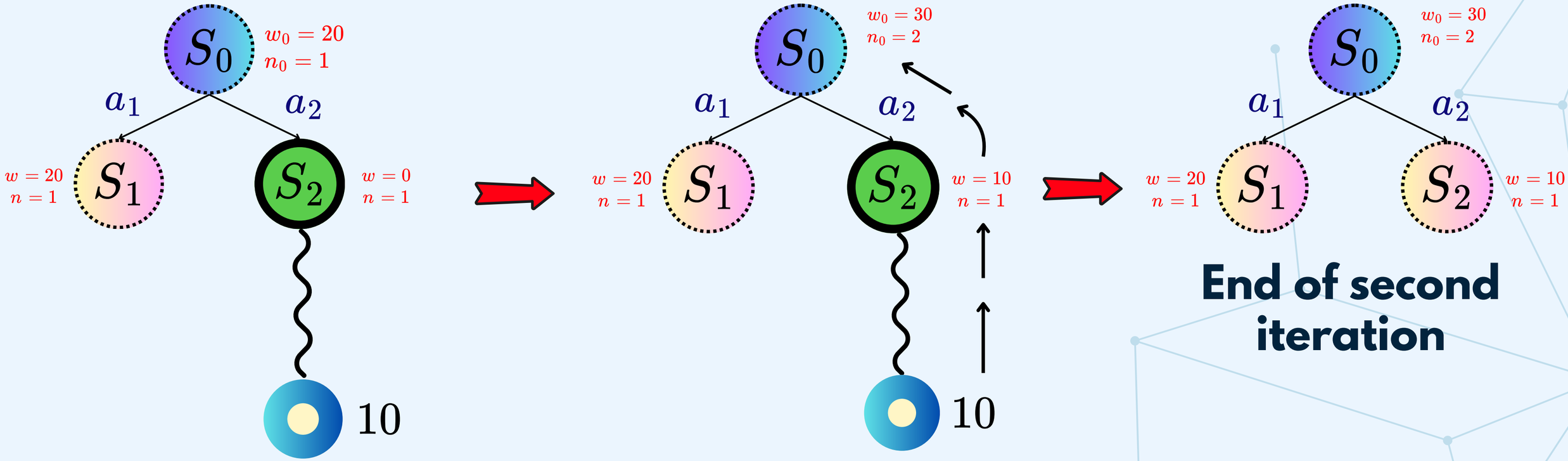


$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$



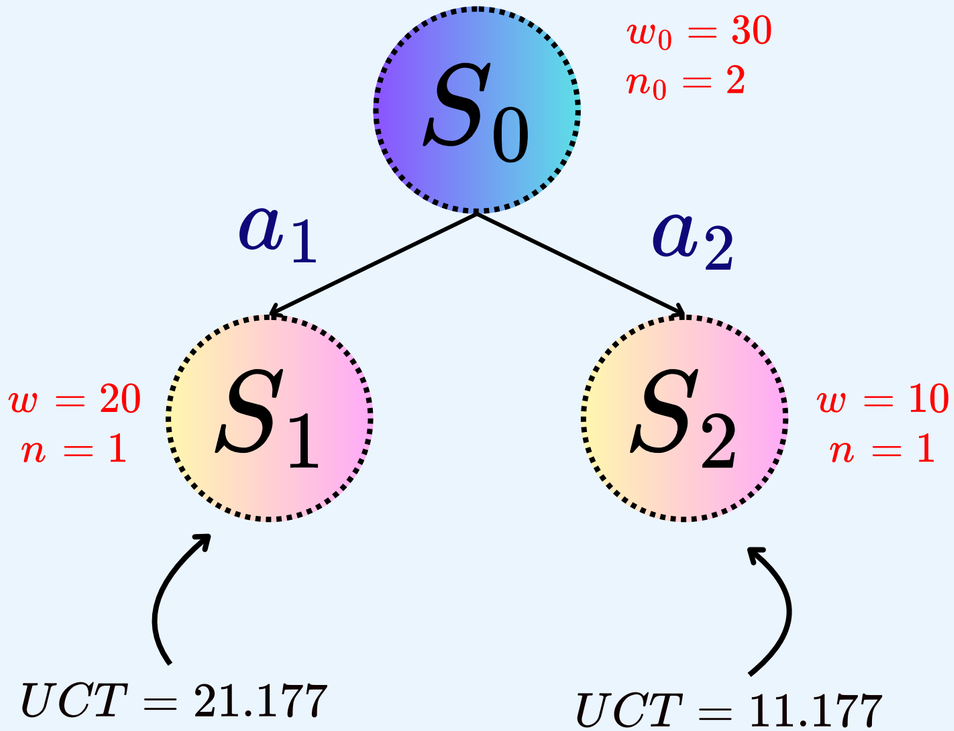
Select S2 for second iteration

Second Iteration

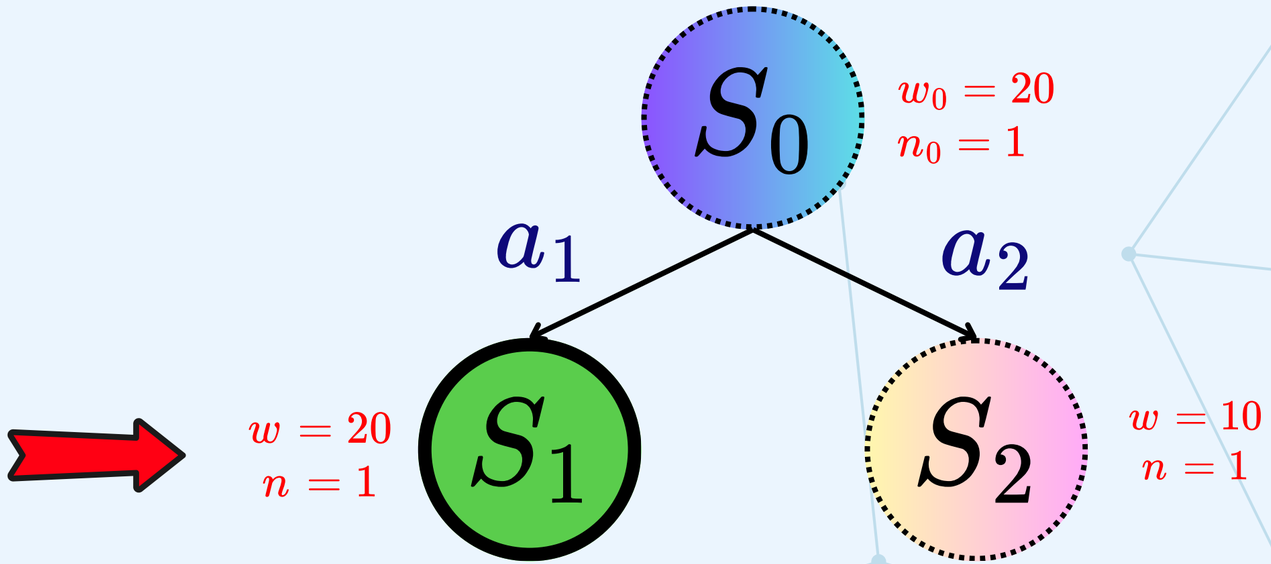


End of second iteration

Third Iteration

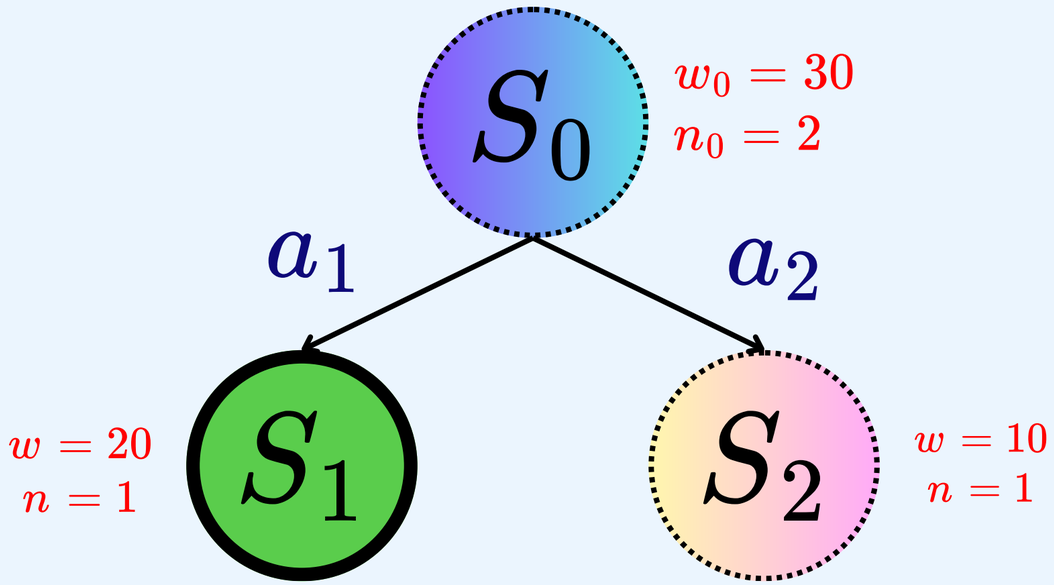


$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

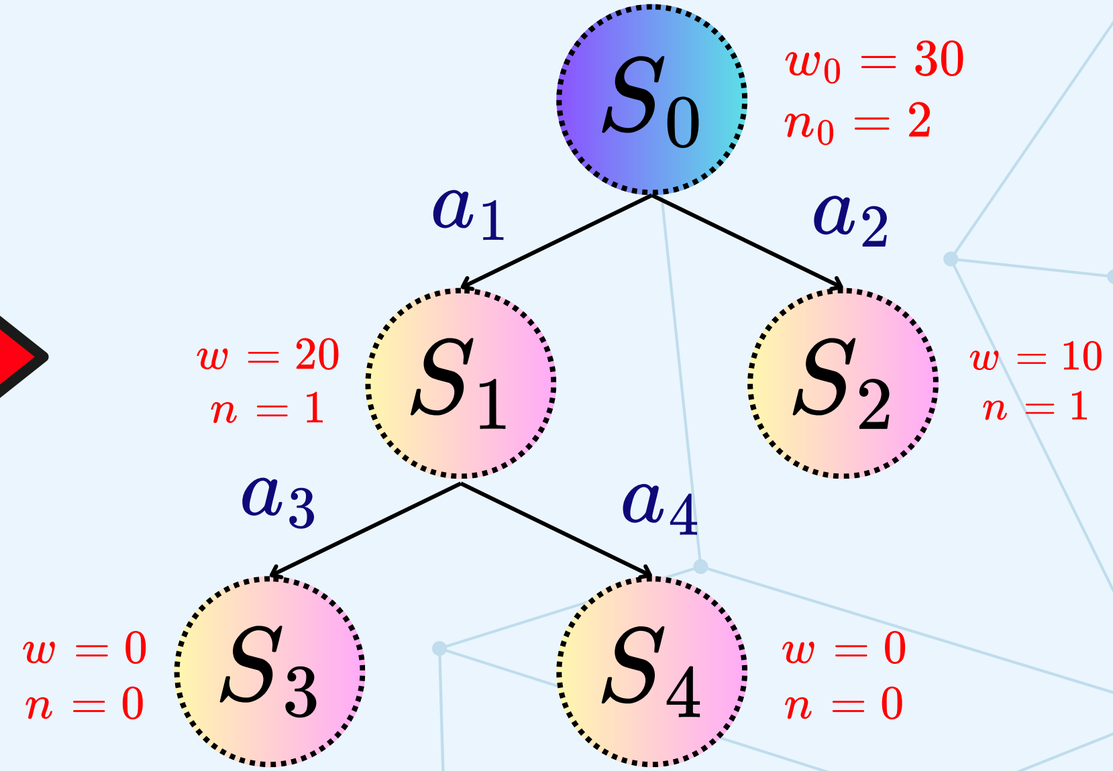


Select S1 for third iteration

Third Iteration

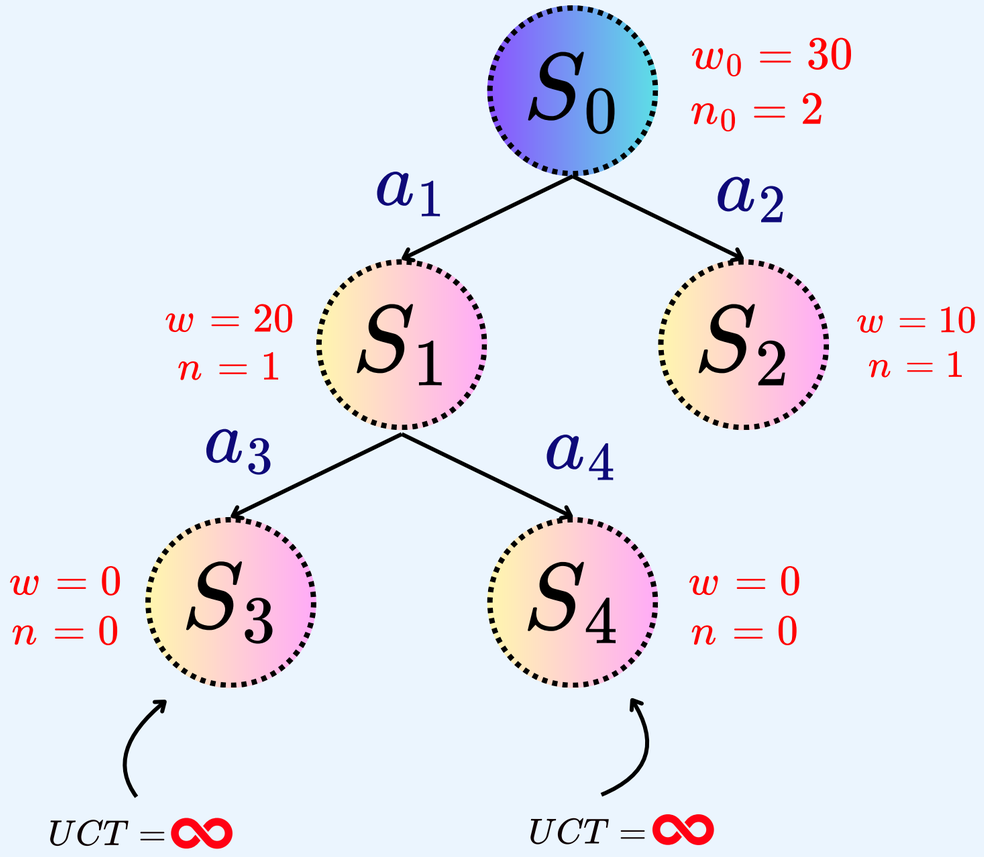


EXPAND !

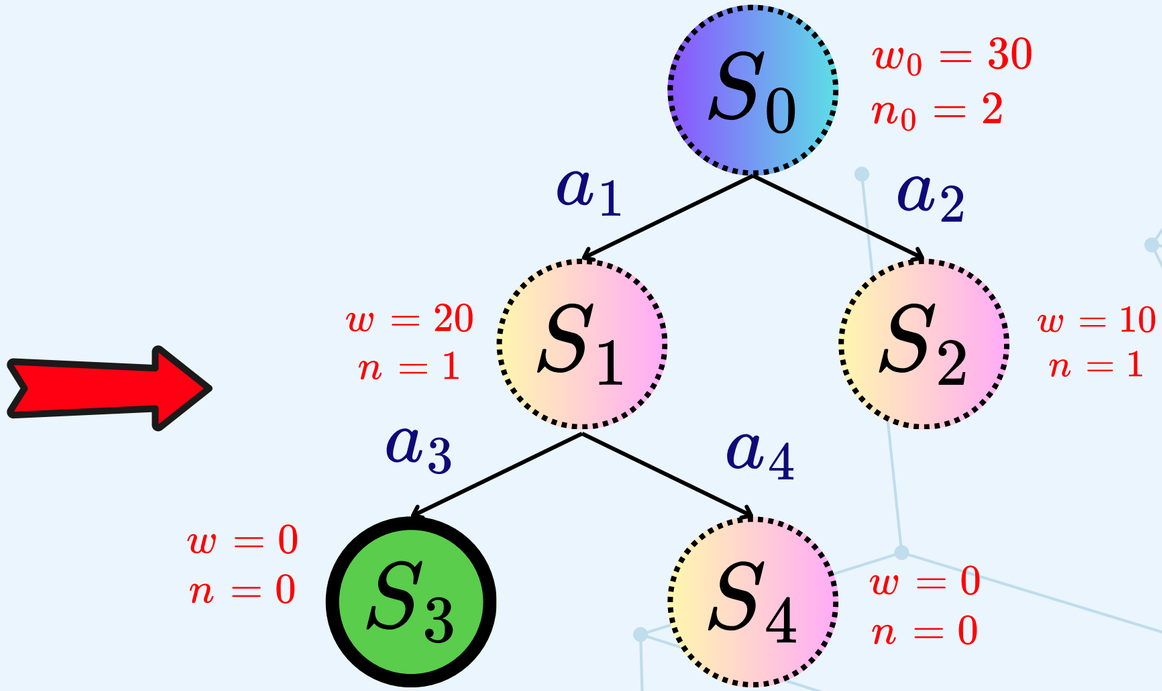


Since S1 has been visited before

Third Iteration



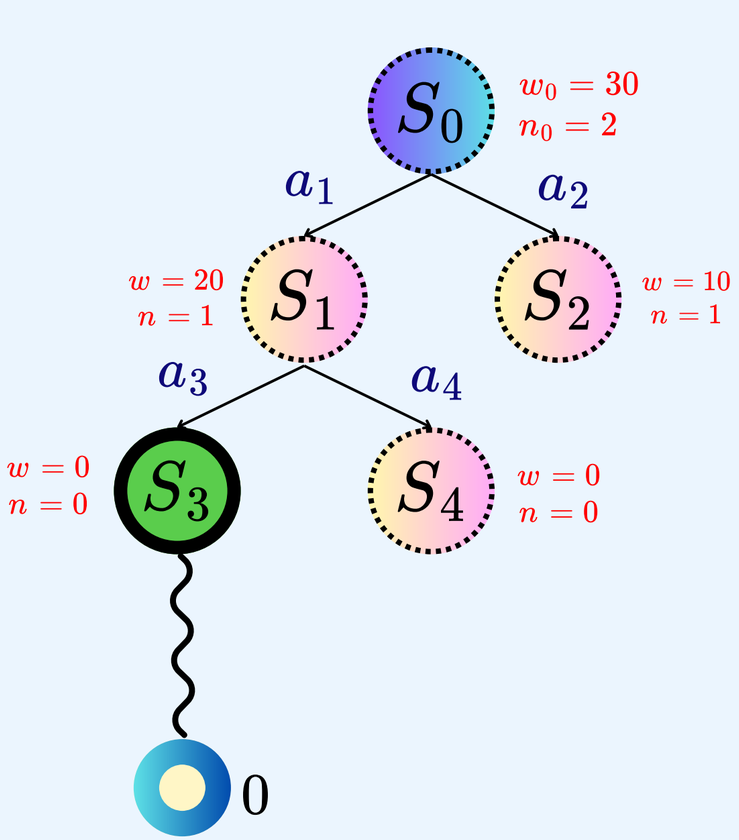
CHOOSE RANDOM

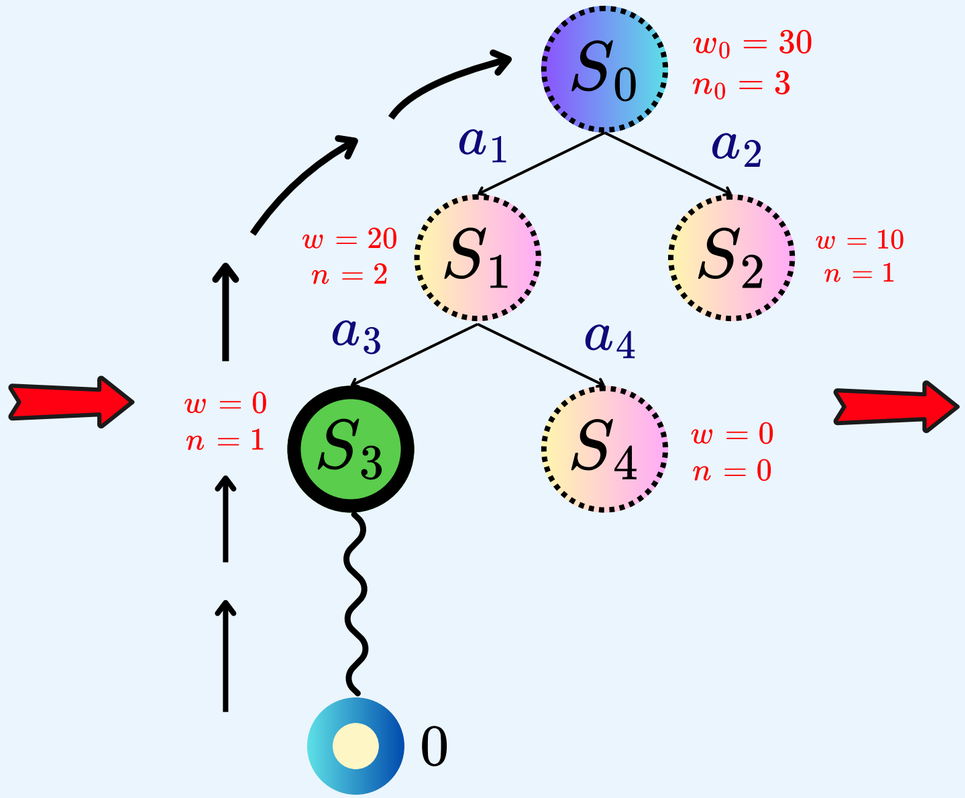


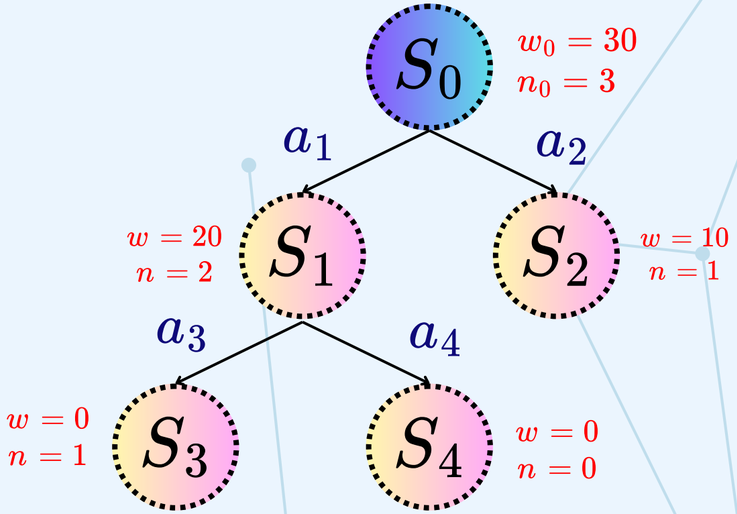
Since S1 has not been visited yet

ROLLOUT !

Third Iteration





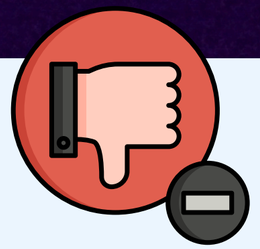


End of Third iteration

Advantages and Disadvantages

Advantages:

- Does not require a heuristic evaluation function
- Works well in large or complex state spaces
- Dynamically balances exploration and exploitation



Disadvantages:

- Slow if you don’t have enough time for many simulations
- Randomness makes it inconsistent unless repeated many times
- May miss deep tactical threats if simulation depth is shallow

Monte Carlo tree search

Common Q&A

Q1: What's the difference between MCTS and Minimax?

- Minimax explores the full tree with evaluation at leaves
- MCTS explores promising paths using simulations, no evaluation function needed

Q2: Can MCTS be used in non-game scenarios?

MCTS also used in:

- Robotics
- Network path optimization
- Resource allocation

Q3: How many iterations should we run?

As many as your time allows! The more simulations, the better the result.
or when are sure about what is the best action

THANK YOU!