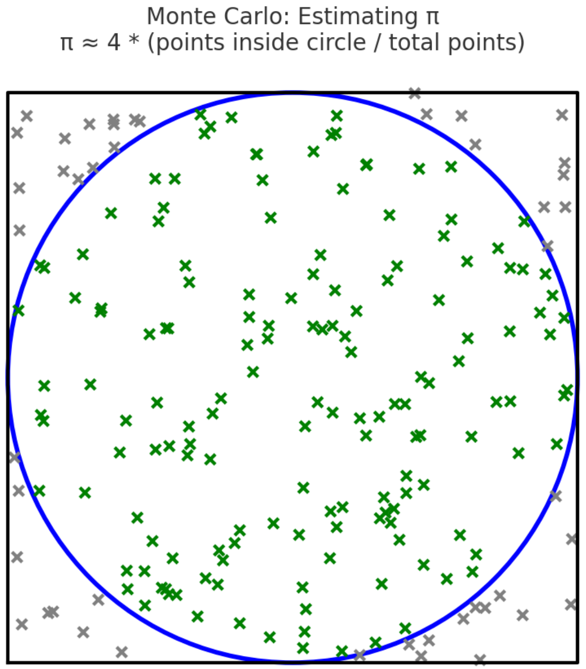
Monte Carlo Tree Search

Monte Carlo

- It refers to algorithms using random numbers (pseudo random variables) to solve computation problems.
- Is it the name of a person? No,it is the name of a city famous for gambling.
- Classic applications:
  - Estimation of π
  - Monte Carlo integration

Monte Carlo Application:Estimation Of π

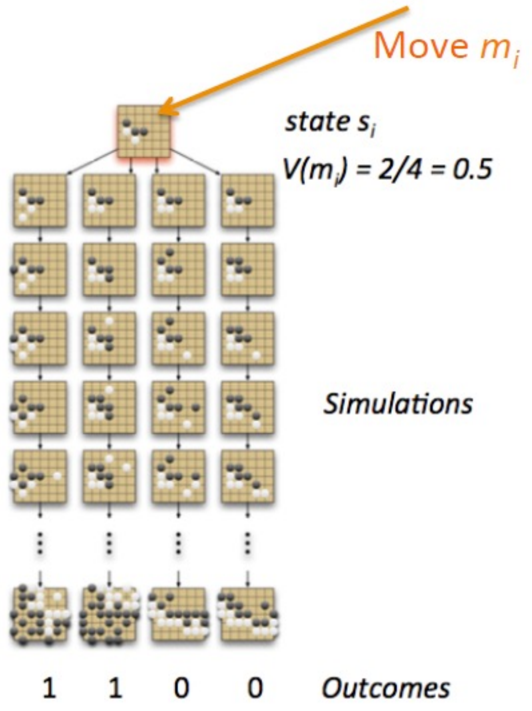
- Generate random points (x, y) in [0,1].
- Count how many points fall inside the circle centered at (0.5,0.5) with radius 0.5.
- $\pi \approx 4 \times$ (points inside circle/ total points).



Trying to Apply Monte Carlo to Games

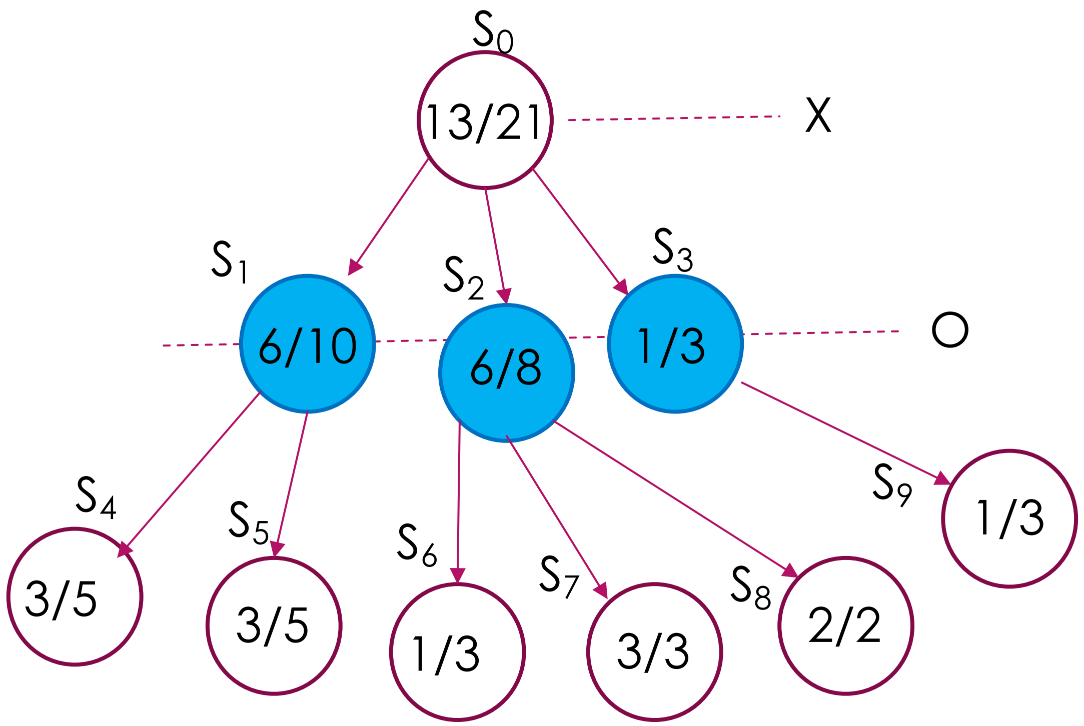
Naive Idea:

- Run many random play-outs of the game.
- Estimate the win rate of each move by averaging outcomes.
- Choose the move with the highest estimated win rate.



Limits

- Random play-outs ignore the decision structure of the game tree.
  Example:X will select node S_1(6/8), but as X's opponant O, O may select S_6(1/3), not S_7(3/3) or S_8(2/2).
- Some moves are explored too little, others too much.
  Example: S_3(1/3) only explored 3 times while its brother S_1 explored 10 , S_2 explored 8.
- No mechanism to balance exploration vs exploitation.



How to improve

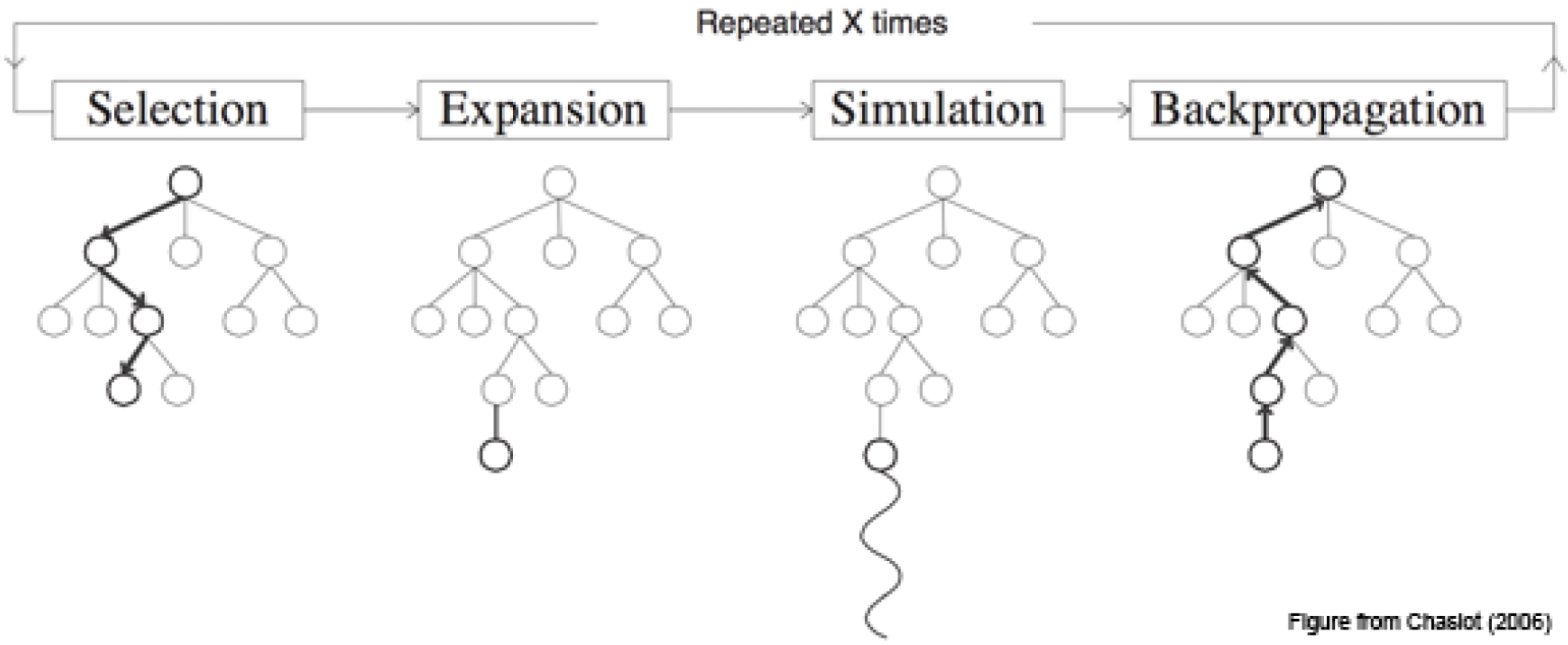
- Incorporating adversarial thinking, assumes an opponent who minimizes our advantage, and thus searches for robust strategies.
- Prioritizes promising regions for precise evaluation, rapidly dismisses unpromising areas with minimal simulation.
- Balance between exploration and exploitation.

MCTS can do all above!

Monte Carlo Tree Search (MCTS)

- **A Game AI Algorithm**
  - Decision Making under Uncertainty
  - widely used in AlphaGo, Game Bots
- **Core Idea**
  - Evaluate moves through random simulations
  - Build search tree incrementally
  - Balance exploration vs exploitation
- **Advantages**
  - No domain knowledge required
  - Easy parallelization
  - Anytime algorithm (can stop anytime)

MCTS Basic Process



- Selection: from root node, recursively apply a child selection policy to descend through the tree until Þnd the most urgent expandable leaf node.
- Expansion: Add one or more child nodes.
- Simulation: Run random simulation from the new node to terminal state.
- Backpropagation: The simulation result is" backed up" through the selected nodes to update their statistics.

Example: Selection and Expansion

Initialization:

0/0 Root node: X

Selection:

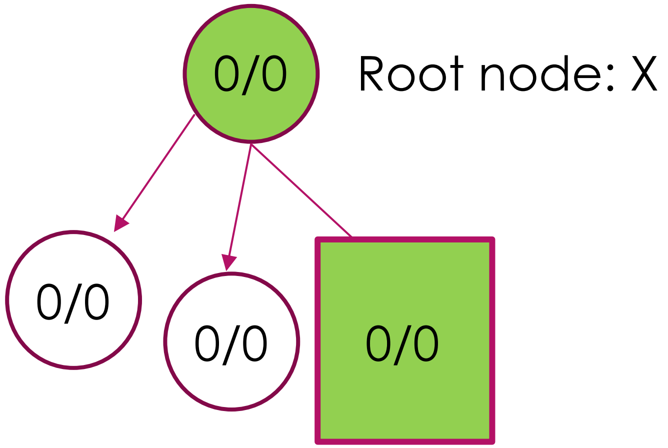
0/0 Root node: X

Current node has no child info, select itself.

What selection policy to select a child if it has child?

Expansion:

Add one or more child nodes.
Return a new node.



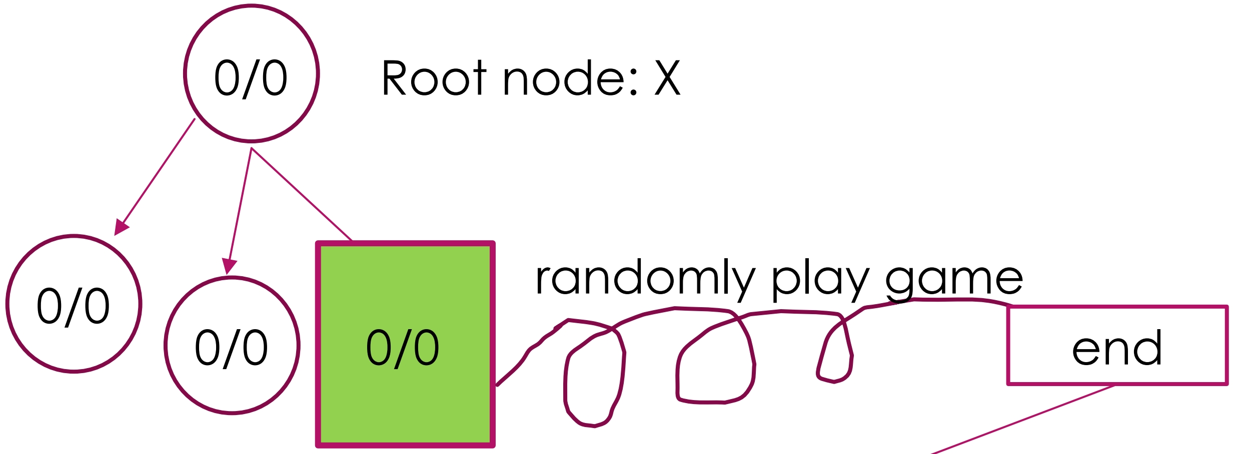
If there's more than one child, pick one at random or by selection policy.

Example: Simulation and Backpropagation

Simulation

Use random strategy to play from the child state until game ends.

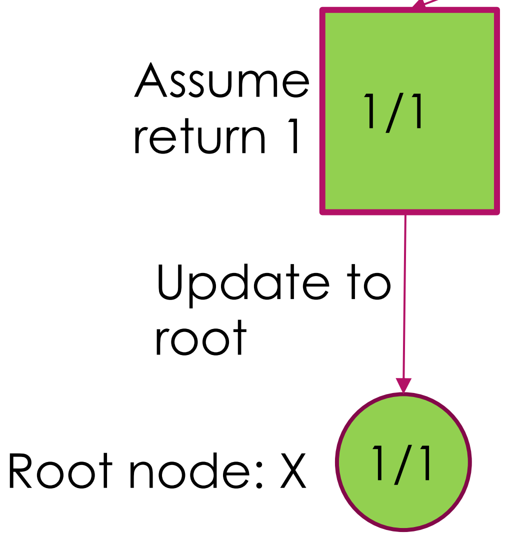
Return the result of the simulated game.

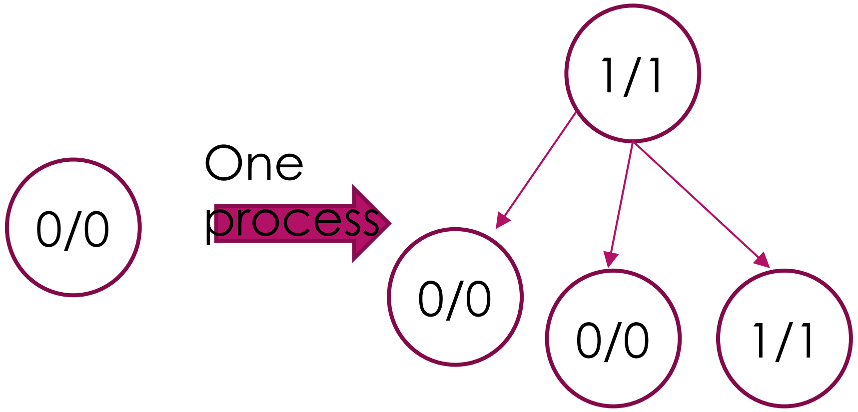


Return 1 for X win, 0 for X loss

Backpropagation

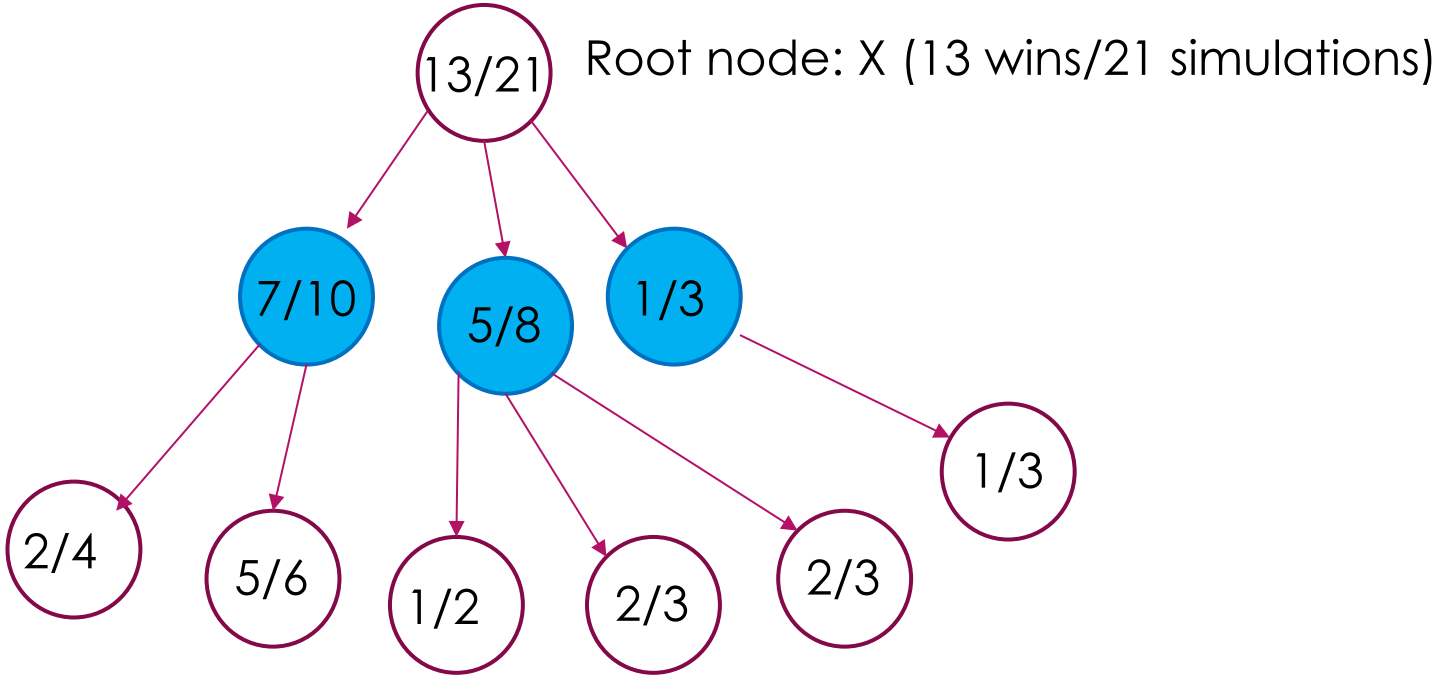
Update win rates and visit counts from leaf to root.





Simulation Results Statistics

Assume the process repeat 21 times, and X win 13 times:

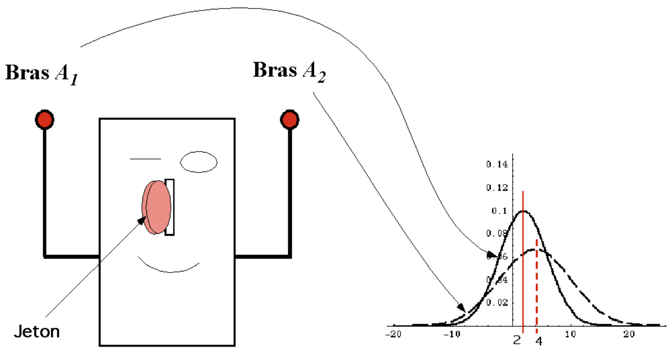


Based on the above tree, we go on to discuss the **selection policy**.

Multi-Armed Bandit Problem(MAB)

Assumption

- Choice among several arms
- Each arm pull is independent of other pulls
- Each arm has unknown reward distribution



Which arm has the best average payoff?

Regret in Multi-Armed Bandit (MAB)

- Optimal strategy: always pull the best arm
- Actual strategy: mix of exploration and exploitation
- Regret = Optimal reward – Actual reward

Example: Regret

A

P(A wins) = 60%

B

P(B wins) = 55%

C

P(C wins) = 40%

A sequence of pulls:
Pull A 300 times, win 170 times
Pull B 300 times, win 150 times
Pull C 400 times, win 165 times

Best sequence of pulls:
Pull A 1000 times, win 600 times

Regret = 60% - 485/1000 = 0.115

How To Minimize regret ?

- Actual strategy: mix of exploration and exploitation
  - Exploitation: choose arm with the highest win rate
  - Exploration: explore the arm with the highest uncertainty

Need to **balance** exploration and exploitation

↓ ***Uniform*** *policy*

↓ ***Greedy*** *policy*

Upper Confidence Bound (UCB)

- UCB1 formula[Aueretal.2002]
  - First,try each arm once
  - Then at each time step, choose arm i that maximizes the UCB1 formula:

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

value estimate

tunable parameter

total number of trials

num trials for arm i

Very sensitive to C

Upper Confidence Tree(UCT)=MCTS+UCB

Apply UCB to MCTS selection phase:

$$\frac{U_i}{N_i} + C\sqrt{\frac{In(N)}{N_i}}$$

U_i : Number of wins for the child node i

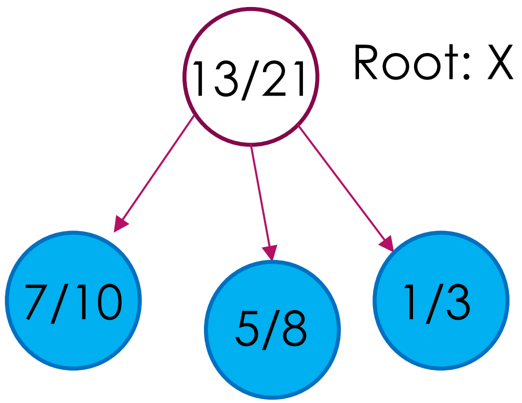
N_i : Number of simulations for the child node i

N : Total number of simulations

C : coefficient (tunable parameter), exploration-exploitation trade-off

UCT Calculation Example (X Turn)

Root node: N = 21



set C =10
child node1:
UCT = 7/10 +10 $* \sqrt{\ln(21)/10}$
≈ 6.22

child node 2:
UCT = 5/8 +10 $* \sqrt{\ln(21)/8}$
≈ 6.80

child node 3:
UCT = 1/3 +10 $* \sqrt{\ln(21)/3}$
≈ **10.40 (selected, high C favors exploration)**

set C = 0.5
child node 1:
UCT = 7/10 +0.5 $* \sqrt{\log(21)/10}$
≈ **0.976 (selected, low C favors exploitation)**

child node 2:
UCT = 5/8 +0.5 $* \sqrt{\log(21)/8}$
≈0.934

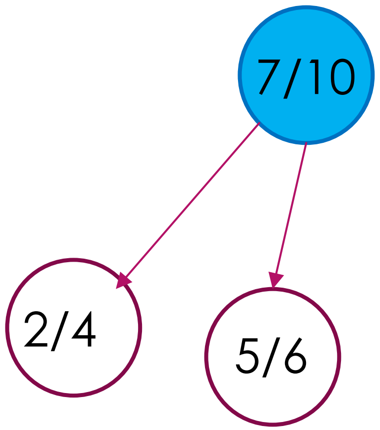
child node 3:
UCT = 1/3 +0.5 $* \sqrt{\log(21)/3}$
≈0.836

Assume set C = 10, from the root,
select the child : 

Now, current parent node: N = 10

child node 1： UCB = $2/4 + 10*\sqrt{\ln(10)/4} \approx 8.09$

child node 2： UCB = $5/6 + 10*\sqrt{\ln(10)/6} \approx 7.03$



Right? **No**

UCT Calculation Example (Opponent's Turn)

During opponent's turn, invert win rate in UCB formula:

$$\left(1-\frac{U_i}{N_i}\right)+C\sqrt{\frac{In(N)}{N_i}}$$

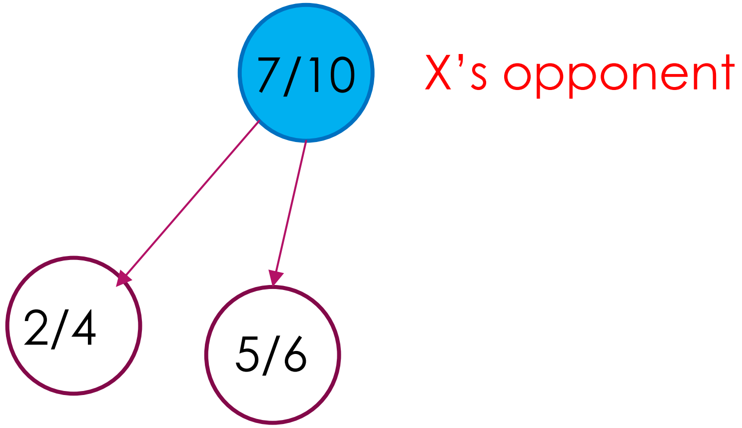
Because opponent wants to minimize your win rate.

Calculation Example:

Current parent node: N = 10

child node 1： UCB = (1 – 2/4) + 10*$\sqrt{\ln(10)/4}$ = 8.09

child node 2：UCB = (1-5/6) + 10*$\sqrt{\ln(10)/6}$ = 6.36



Limitations of MCTS

- Quality depends on simulations
- May need many iterations
- Parameter sensitivity
- Memory intensive

Summary

- ✓ MCTS is a decision-making method requiring no prior knowledge.
- ✓ Applications: Game AI, path planning, automated reasoning
- ✓ UCT balances exploration and exploitation for efficient decision-making
- ✓ Future Directions: Deep learning integration, Real-time applications, Multi-agent systems.