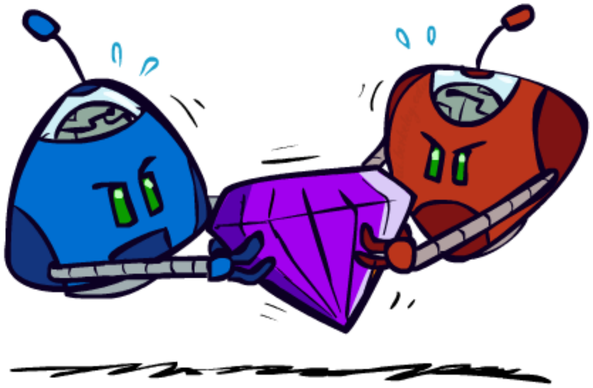
CMSC 471: Games
MCTS

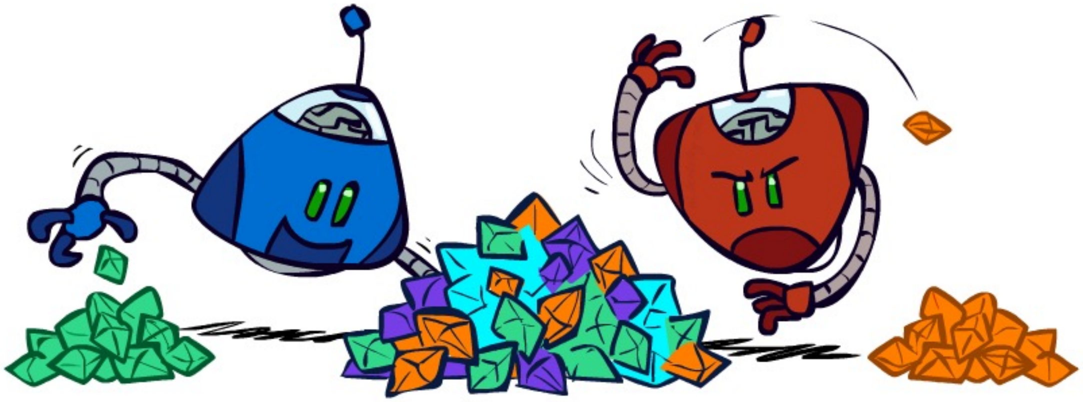
KMA Solaiman

ksolaima@purdue.edu

Some materials adopted from slides by Dan Klein and Pieter Abbeel, Stuart Russell and Dawn Song at UC Berkeley
[http://ai.berkeley.edu]

Zero-Sum Games

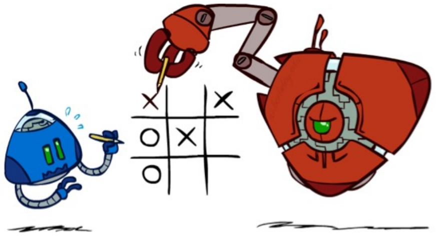


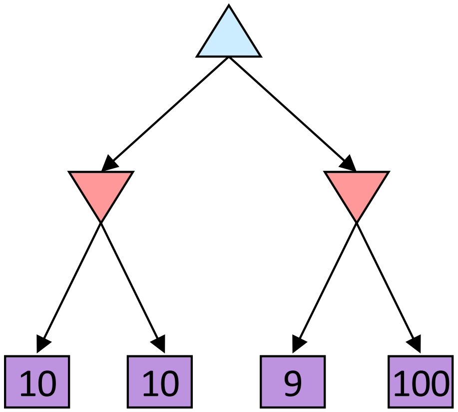


- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - **Lets us think of a single value that one maximizes and the other minimizes**
  - Adversarial, pure competition

- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

Chance outcomes in trees

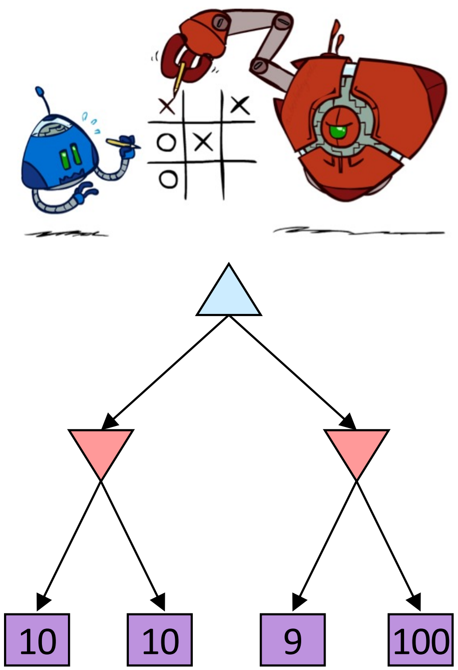




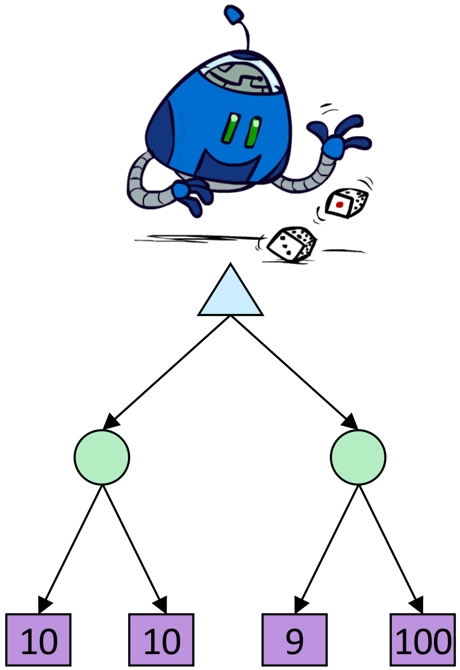
Tictactoe, chess

Minimax

Chance outcomes in trees

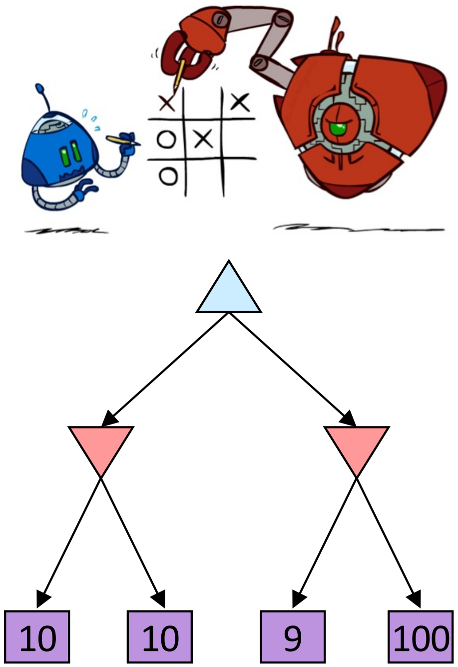


Tictactoe, chess
Minimax



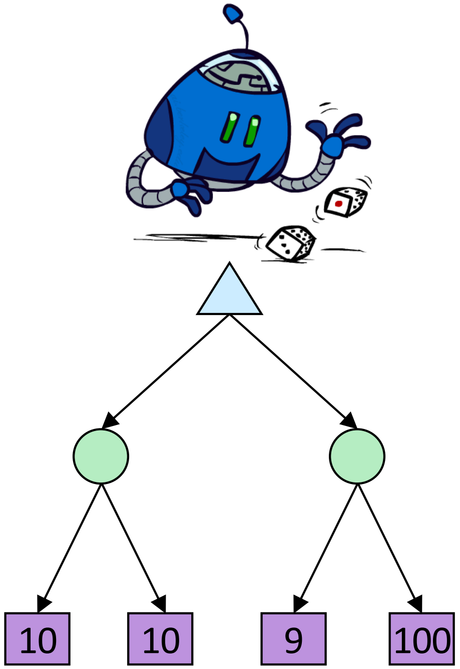
Tetris, investing
Expectimax

Chance outcomes in trees



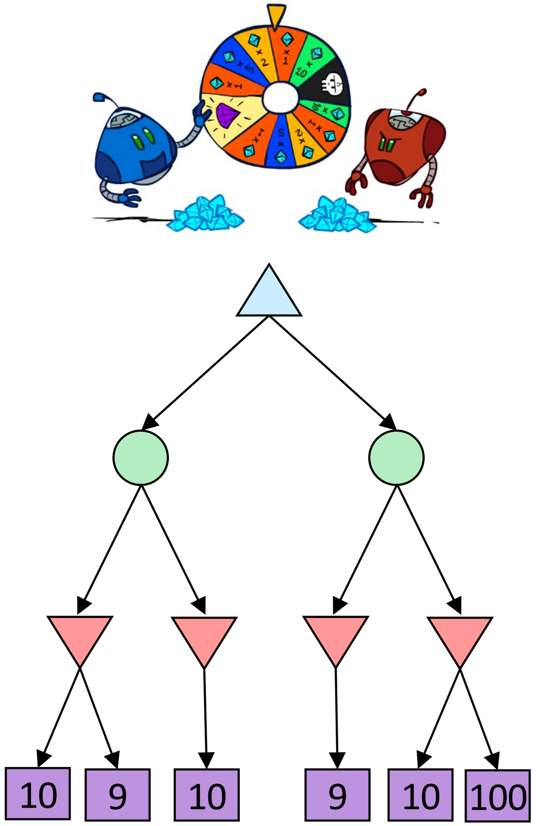
Tictactoe, chess

Minimax



Tetris, investing

Expectimax

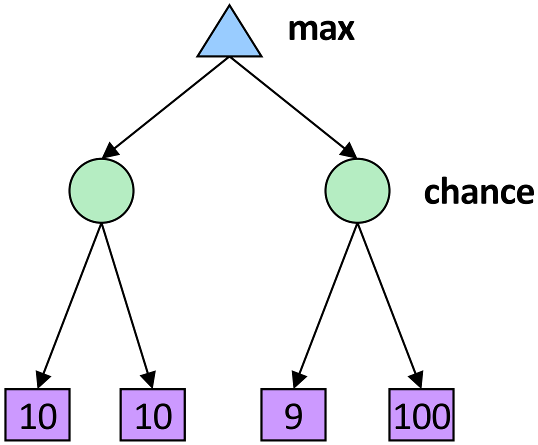


Backgammon, Monopoly

Expectiminimax

Expectimax Search

- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip
- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes



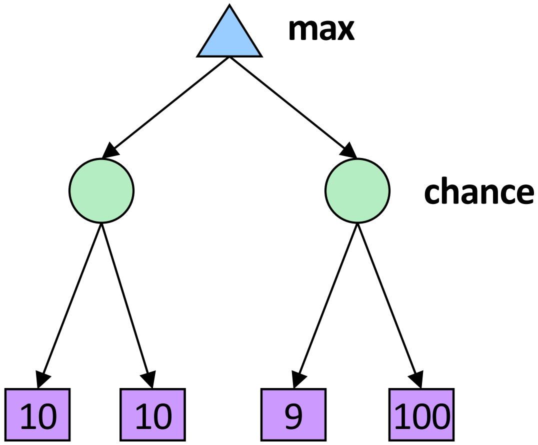
Expectimax Search

- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip

- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes

- Expectimax search: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children

- Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes



Expectimax Pseudocode

```
def value(state):
        if the state is a terminal state: return the state's utility
        if the next agent is MAX: return max-value(state)
        if the next agent is EXP: return exp-value(state)
```

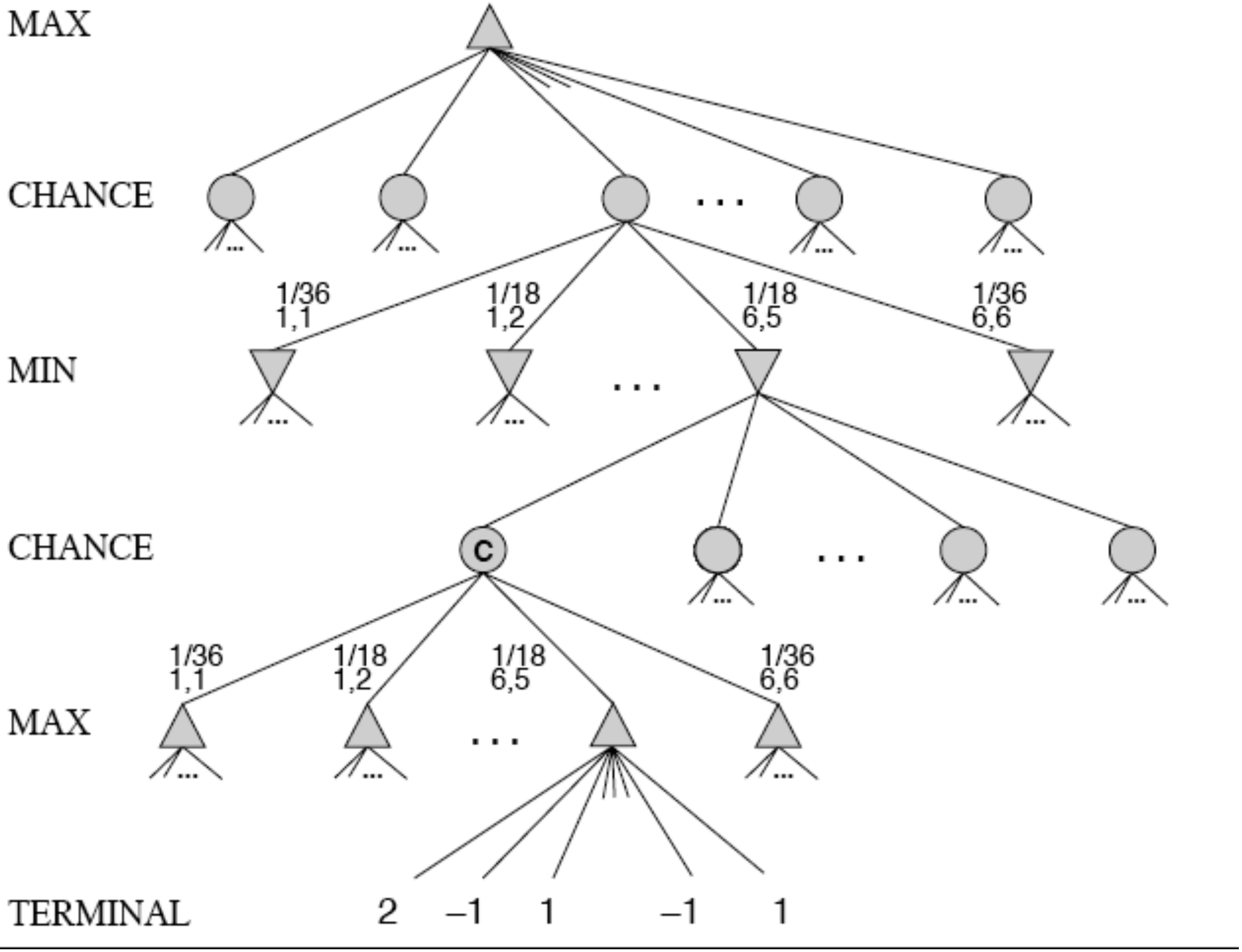
Expectimax Pseudocode

```
def value(state):
        if the state is a terminal state: return the state's utility
        if the next agent is MAX: return max-value(state)
        if the next agent is EXP: return exp-value(state)
```

```
def max-value(state):
      initialize v = -∞
      for each successor of state:
            v = max(v, value(successor))
      return v
```

```
def exp-value(state):
      initialize v = 0
      for each successor of state:
                p = probability(successor)
            v += p * value(successor)
      return v
```

MiniMax trees with Chance Nodes



High-Performance Game Programs

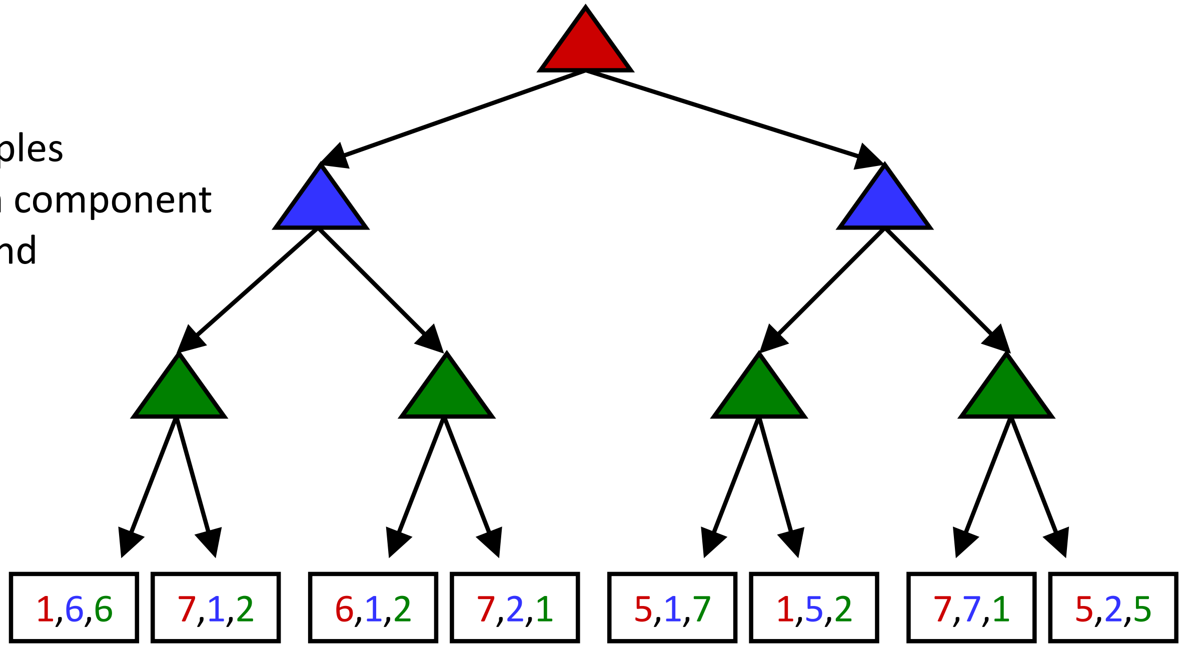
- Many programs based on alpha-beta + iterative deepening + extended/singular search + transposition tables + huge databases + ...
- Chinook searched all checkers configurations with ≤ 8 pieces to create endgame database of 444 billion board configurations
- Methods general, but implementations improved via many specifically tuned-up enhancements (e.g., the evaluation functions)

Other Issues

- Multi-player games, no alliances
  - E.g., many card games, like Hearts
- Multi-player games with alliances
  - E.g., Risk
  - More on this when we discuss game theory
  - Good model for a social animal like humans, where we must balance cooperation and competition

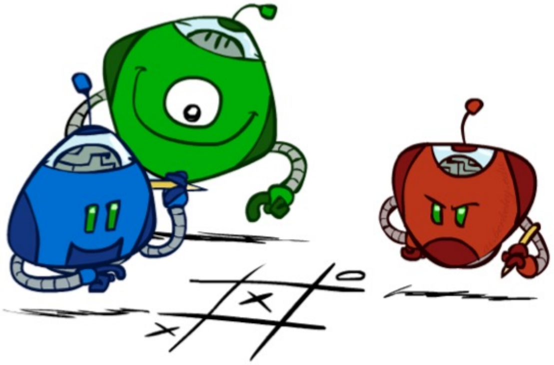
Multi-Agent Utilities

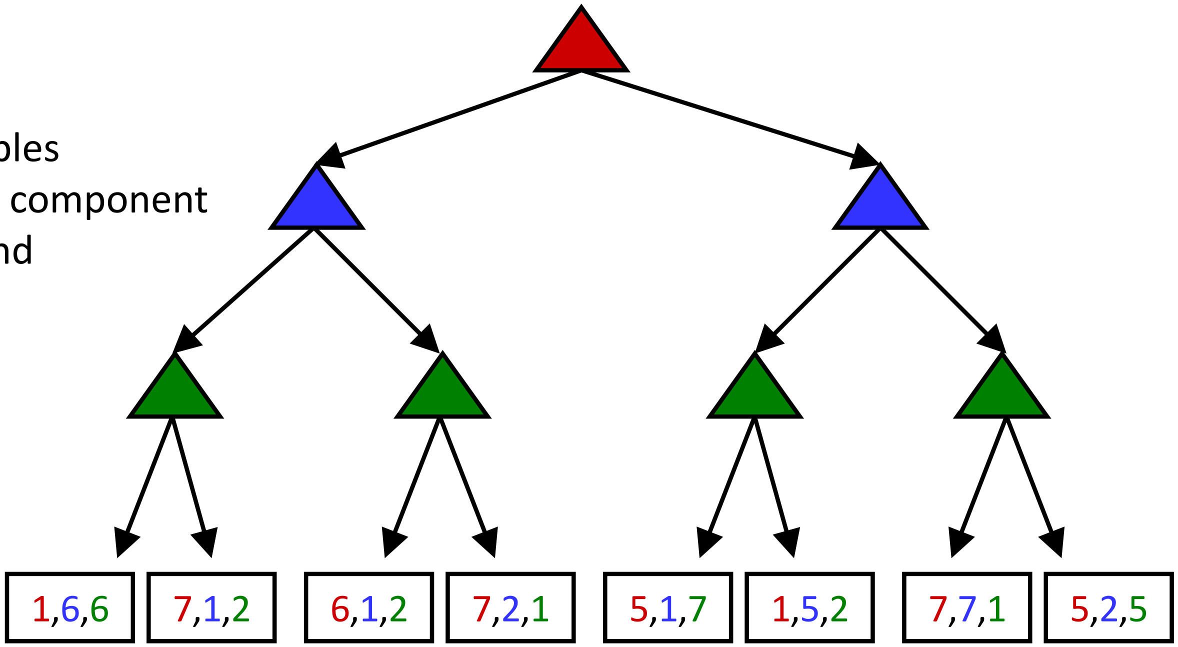
- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…



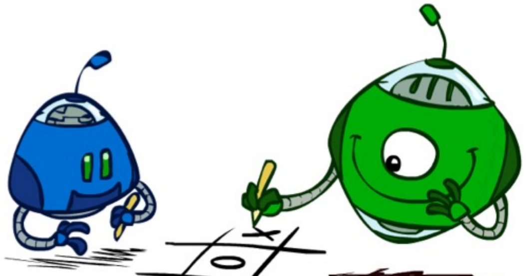
Multi-Agent Utilities

- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…

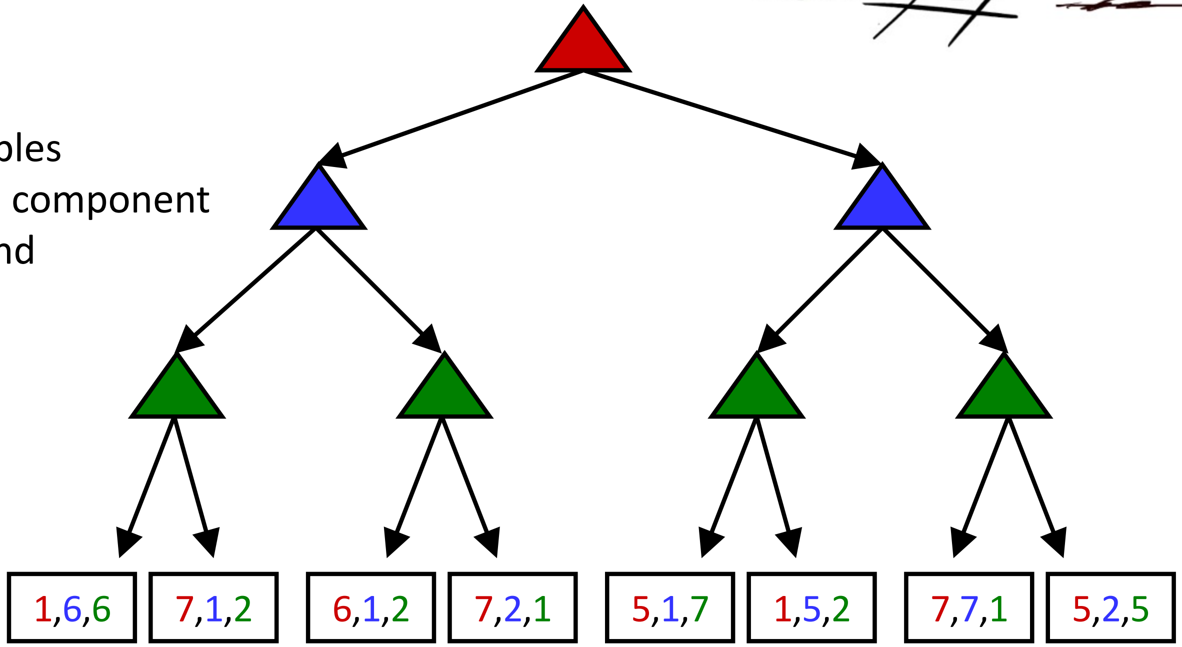


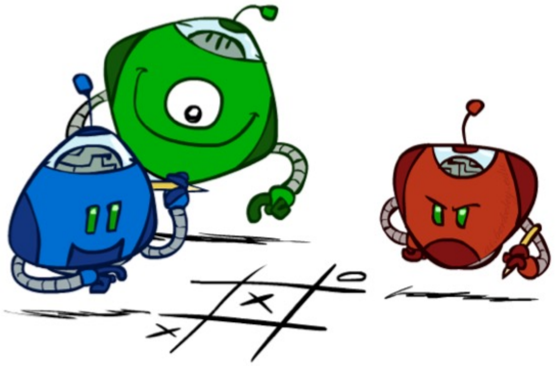


Multi-Agent Utilities



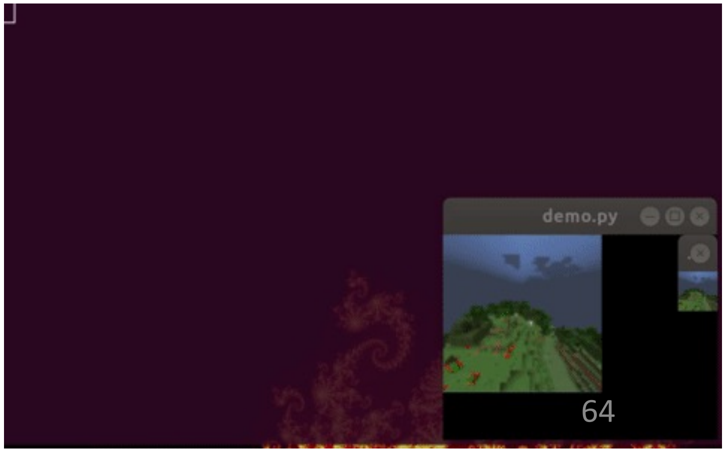
- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…





AI and video Games

- Many games include agents run by the game program as
  - Adversaries, in first person shooter games
  - Collaborators, in a virtual reality game
  - E.g.: AI bots in Fortnite Chapter 2
- Some games used as AI/ML challenges or learning environments
  - MineRL: train bots to play Minecraft
  - MarioAI: train bots for Super Mario Bros



AlphaGO

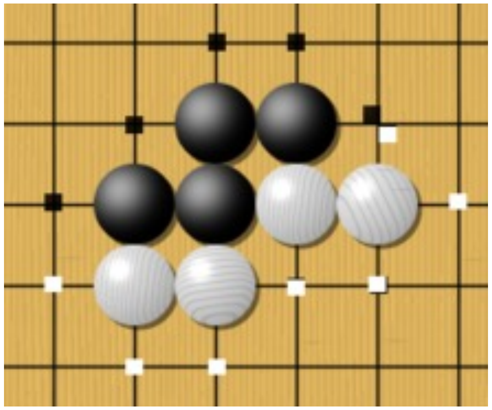
- Developed by Google’s DeepMind
- Beat top-ranked human grandmasters in 2016
- Used Monte Carlo tree search over game tree

  expands search tree via random sampling of search space
- *Science* Breakthrough of the year runner-up

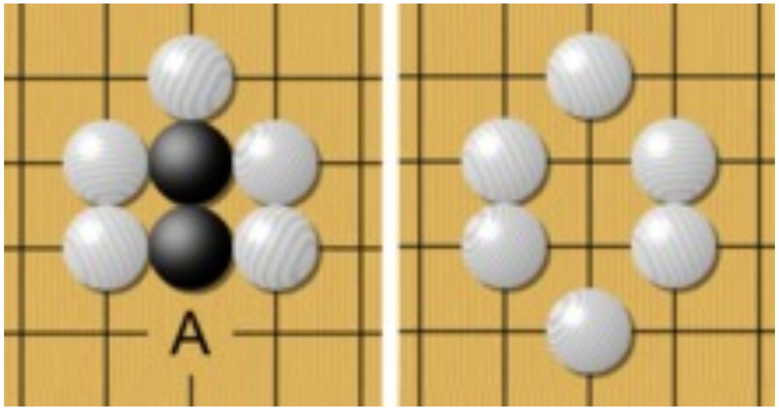
  Mastering the game of Go with deep neural networks and tree search, Silver et al., *Nature*, 529:484–489, 2016
- Match with grandmaster Lee Sedol in 2016 was subject of award-winning 2017 AlphaGo

Go - the game

- Played on 19x19 board; black vs. white stones
- Huge state space $O(b^d)$: chess:~35^{80}, go: ~250^{150}
- Rule: Stones on board must have an adjacent open point ("liberty") or be part of connected group with a liberty. Groups of stones losing their last liberty are removed from the board.



liberties



capture

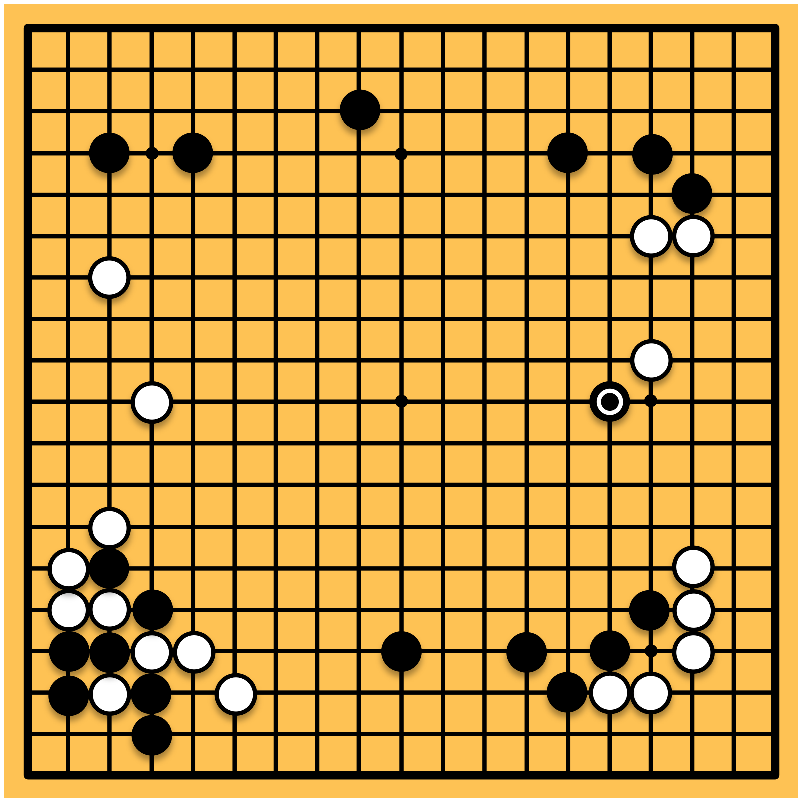
Monte Carlo Tree Search

- Methods based on alpha-beta search assume a fixed horizon
  - Pretty hopeless for Go, with $b > 300$
- MCTS combines two important ideas:
  - ***Evaluation by rollouts*** – play multiple games to termination from a state s (using a simple, fast rollout policy) and count wins and losses
  - ***Selective search*** – explore parts of the tree that will help improve the decision at the root, regardless of depth

Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

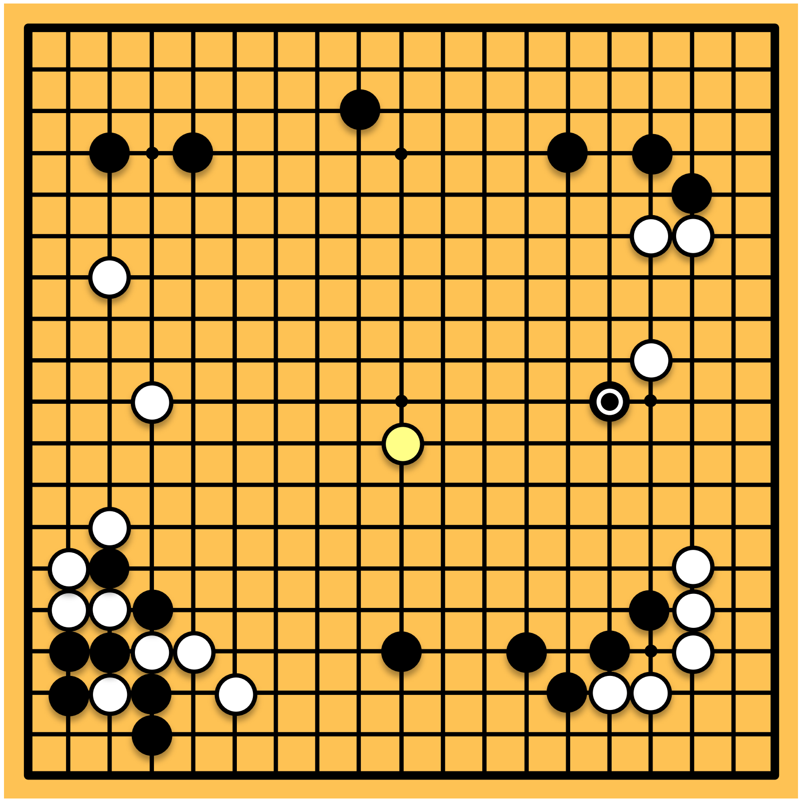
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

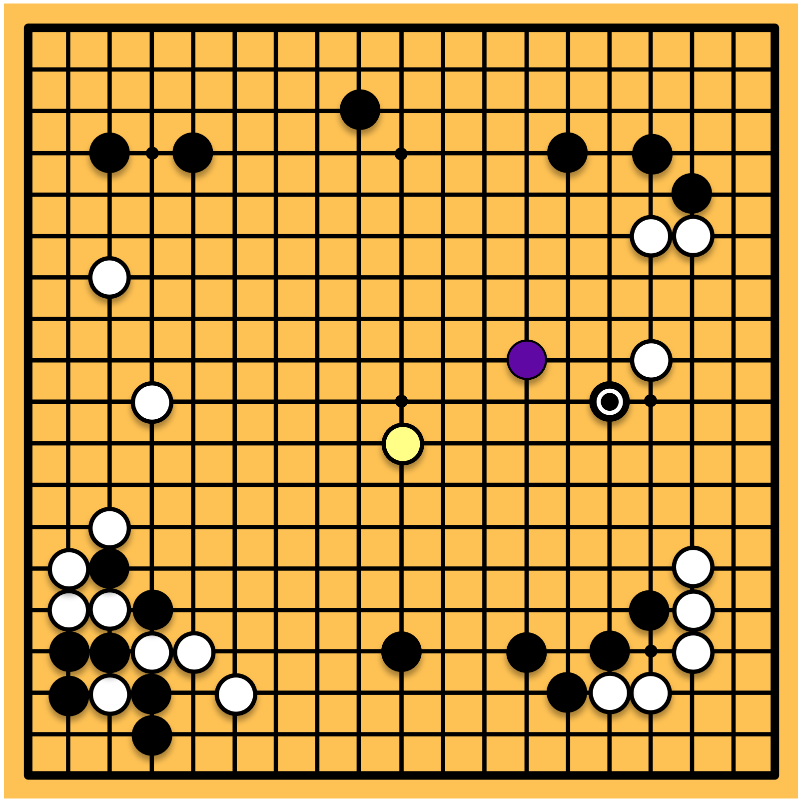
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

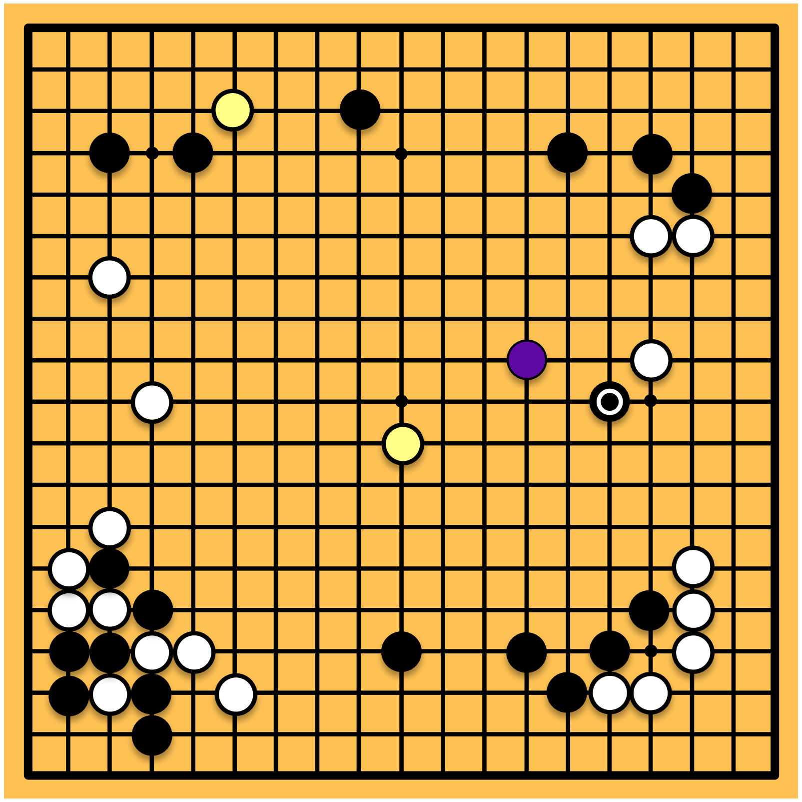
“Move 37”



Rollouts

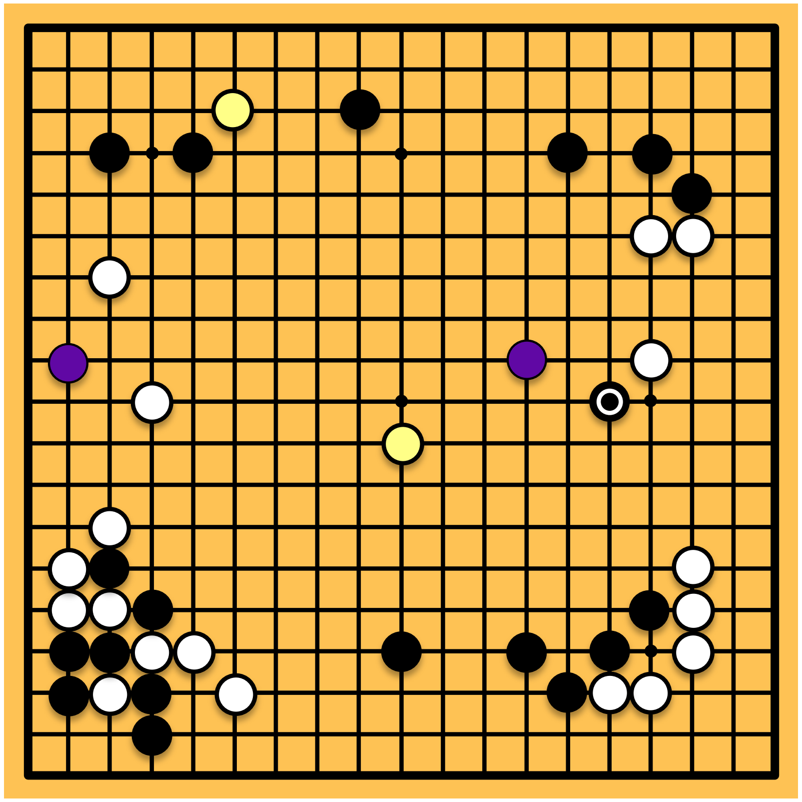
- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

“Move 37”



Rollouts

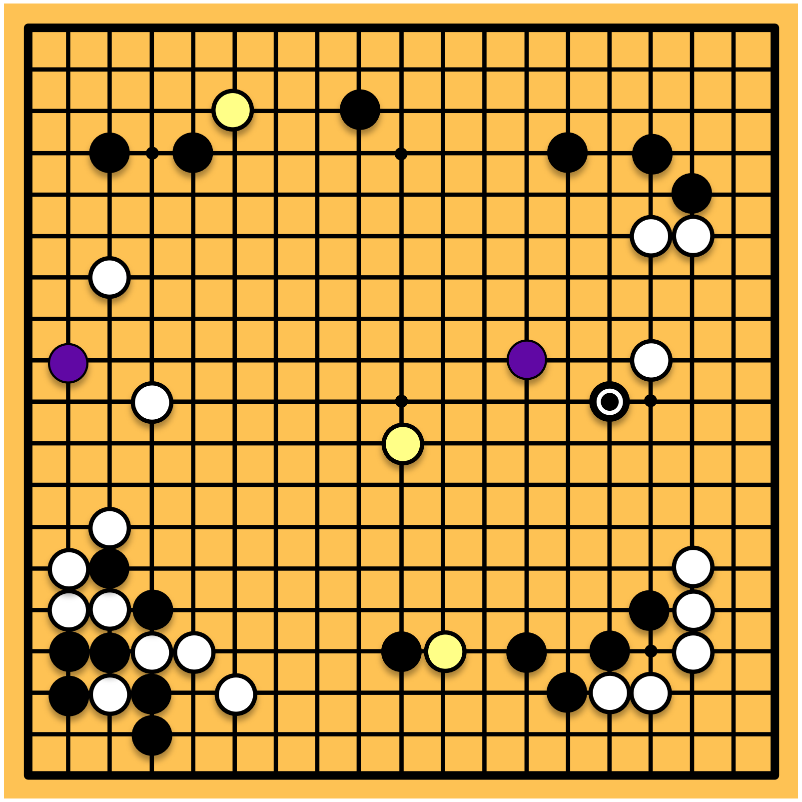
- For each rollout:
  - – Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - – Record the result


“Move 37”

Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

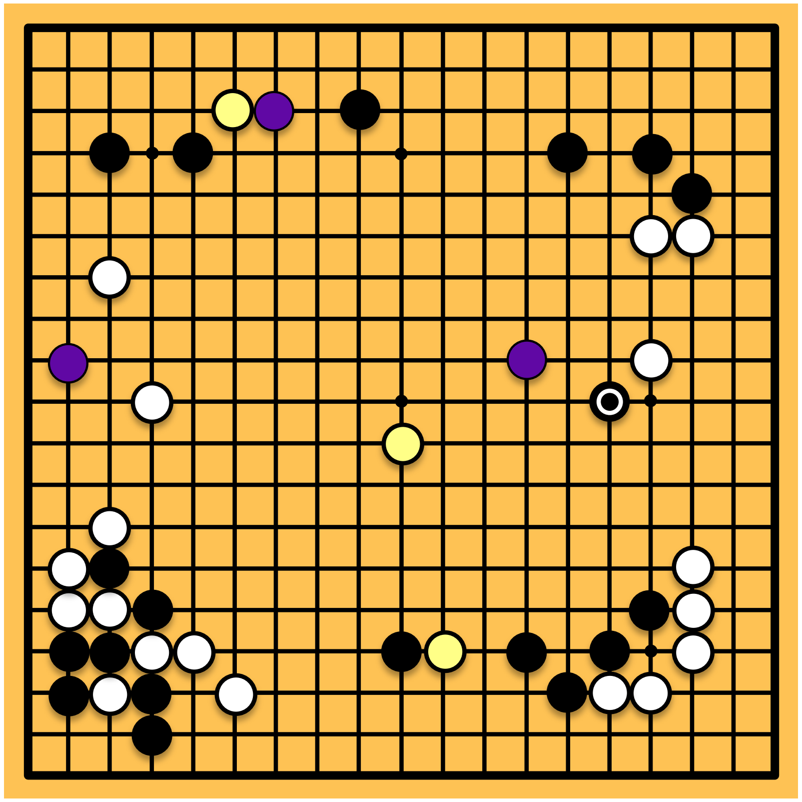
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

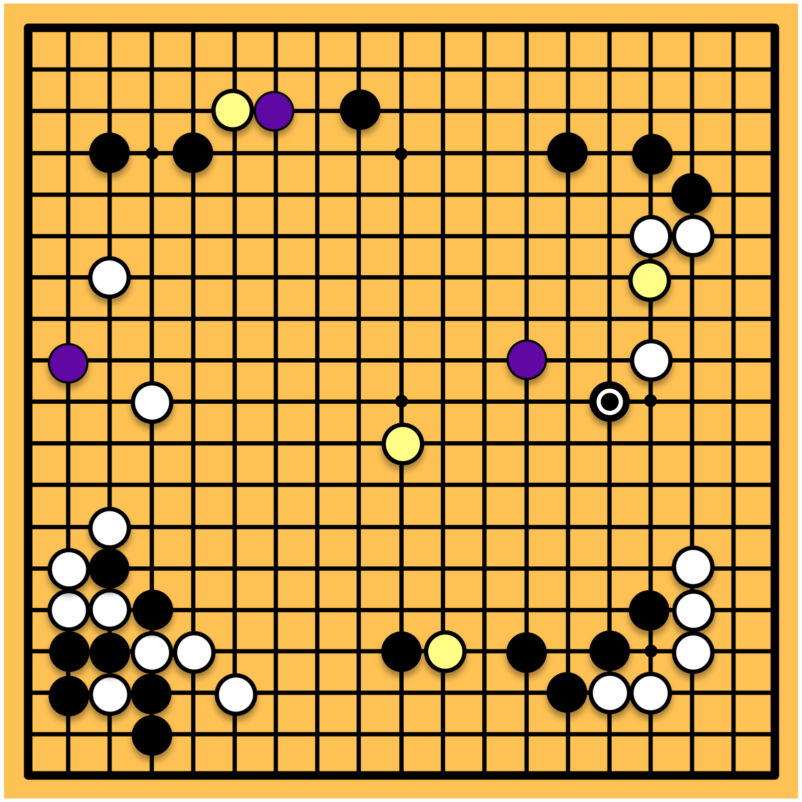
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

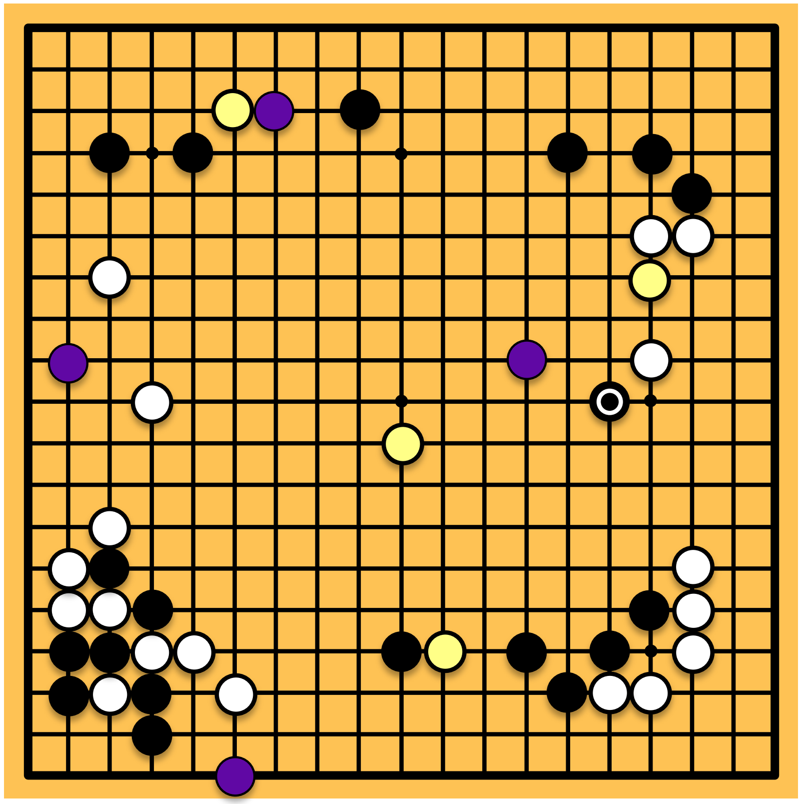
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

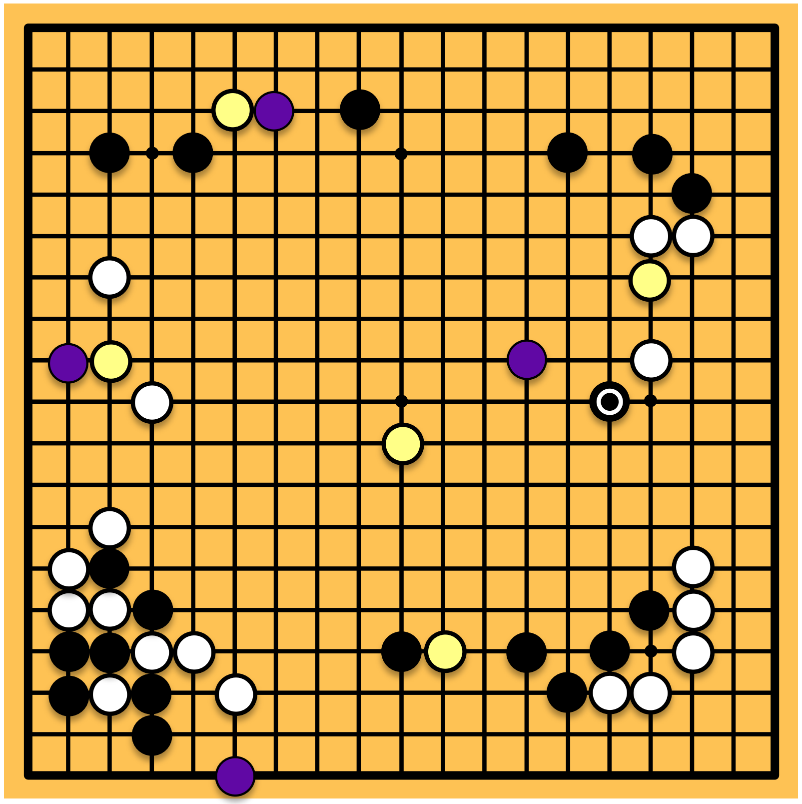
“Move 37”



Rollouts

- For each rollout:
  - – Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - – Record the result

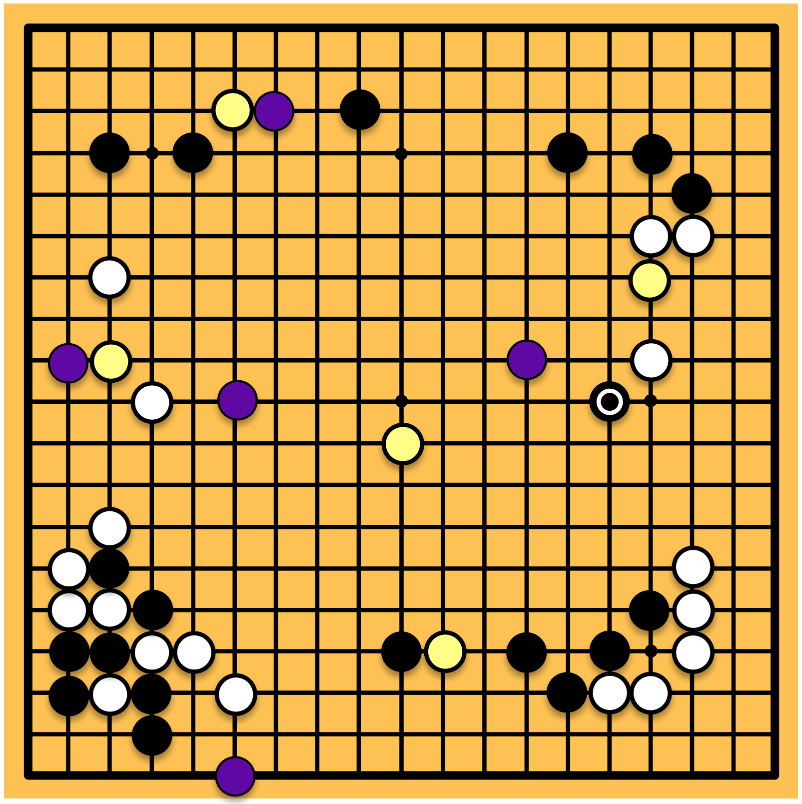
“Move 37”



Rollouts

- For each rollout:
  - – Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - – Record the result

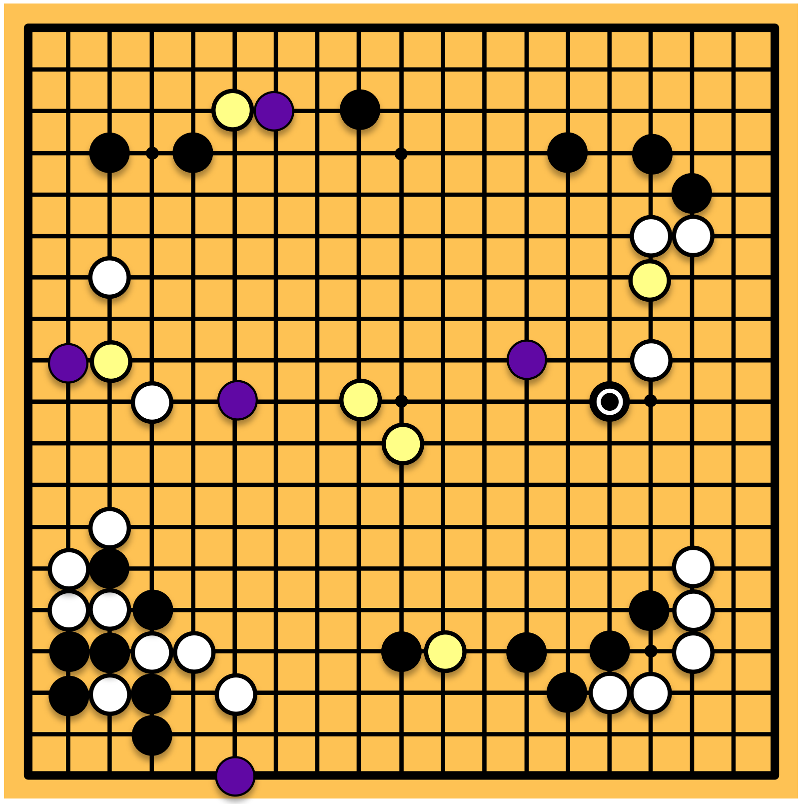
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

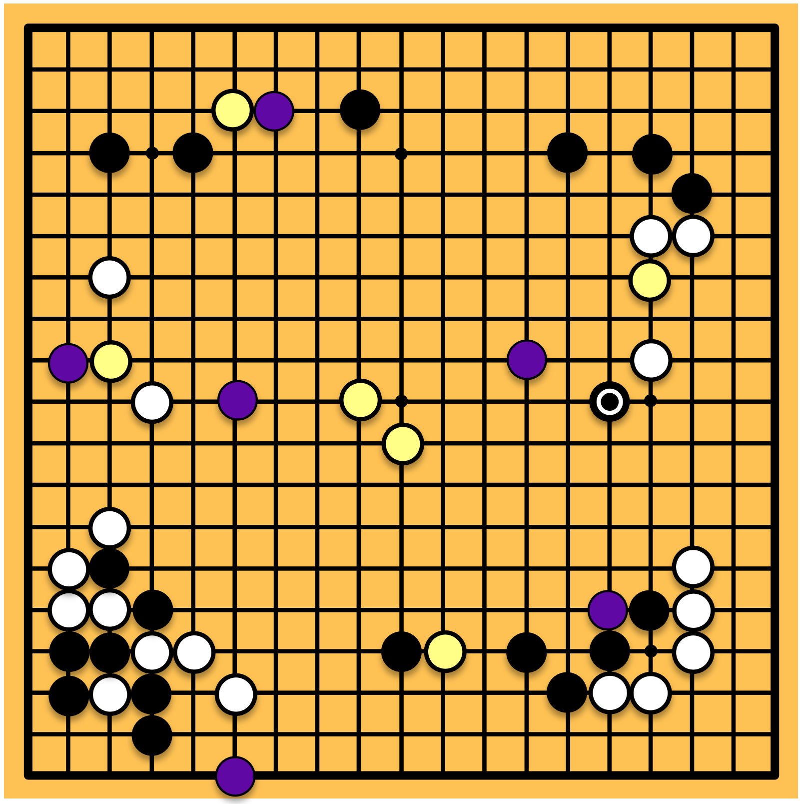
“Move 37”



Rollouts

- For each rollout:
  - – Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - – Record the result

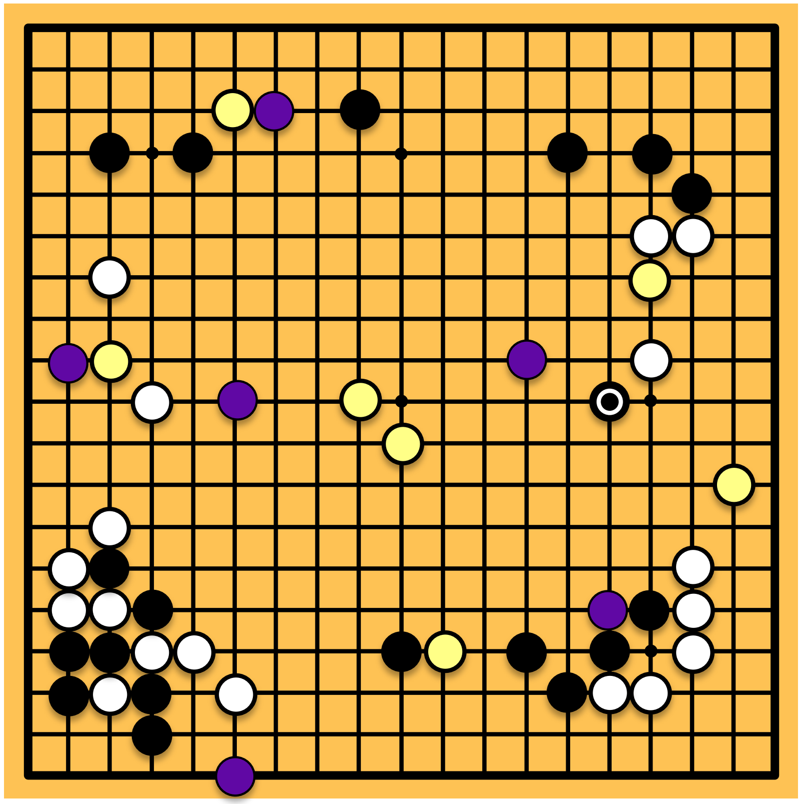
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

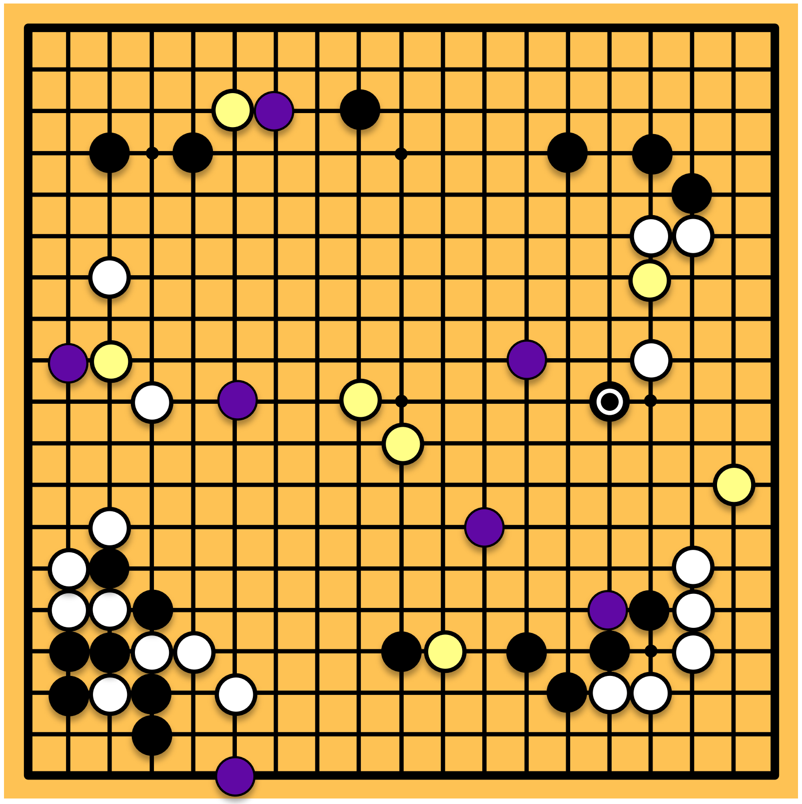
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

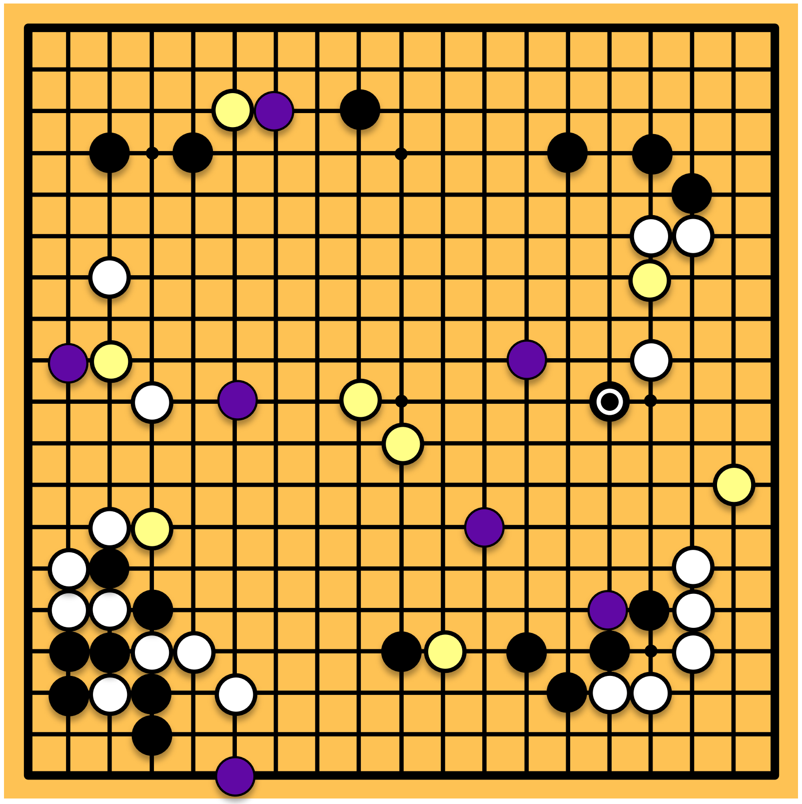
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

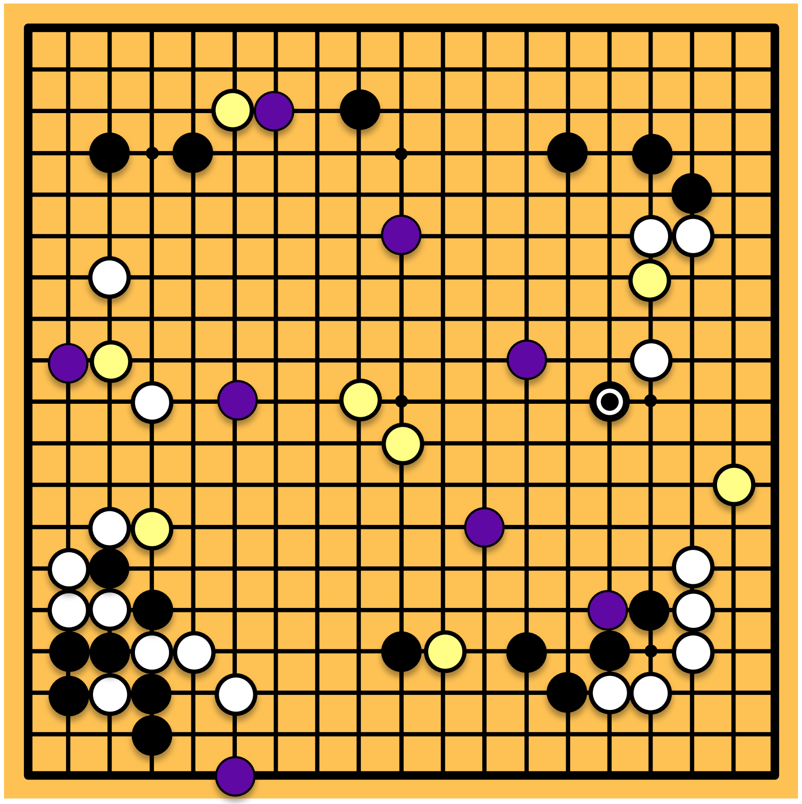
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

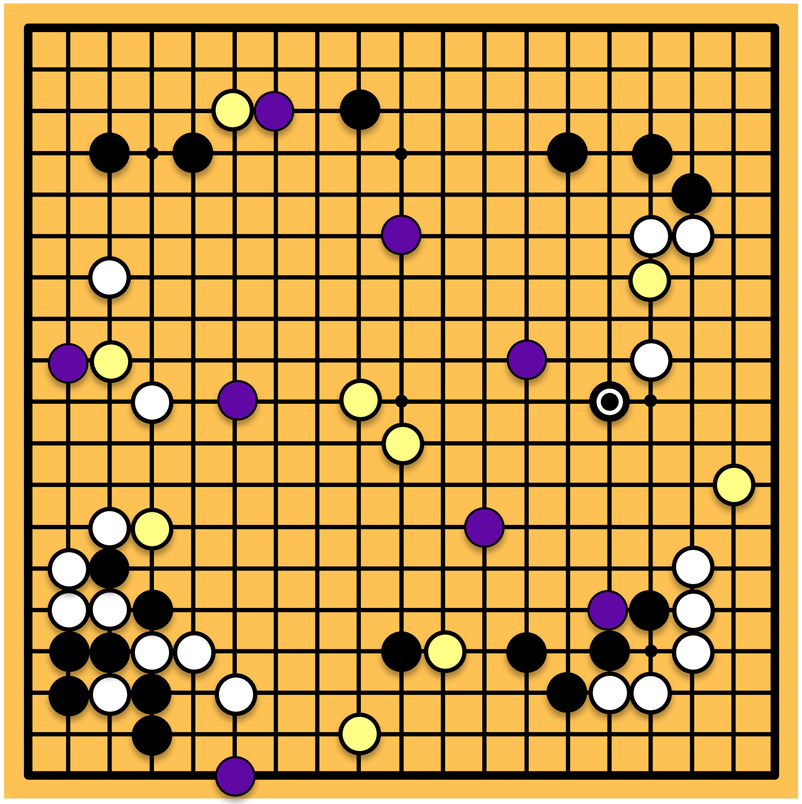
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

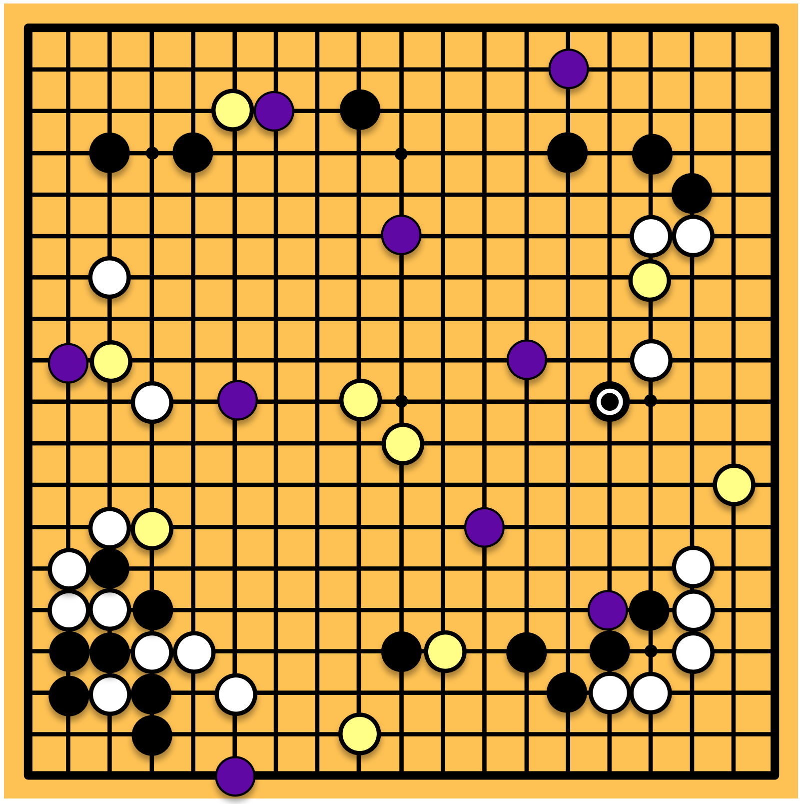
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

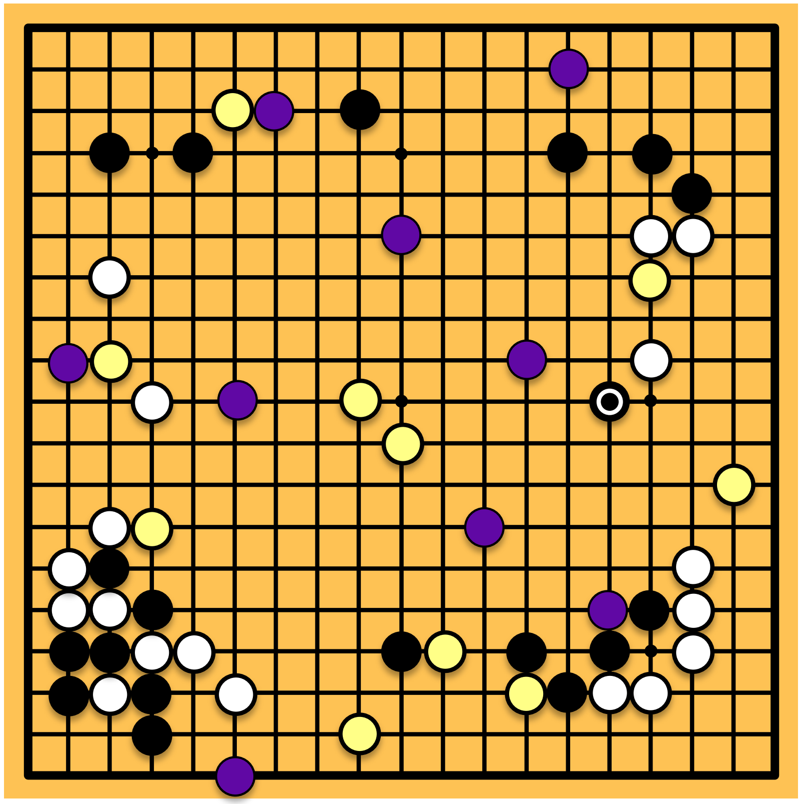
“Move 37”



Rollouts

- For each rollout:
  - – Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - – Record the result

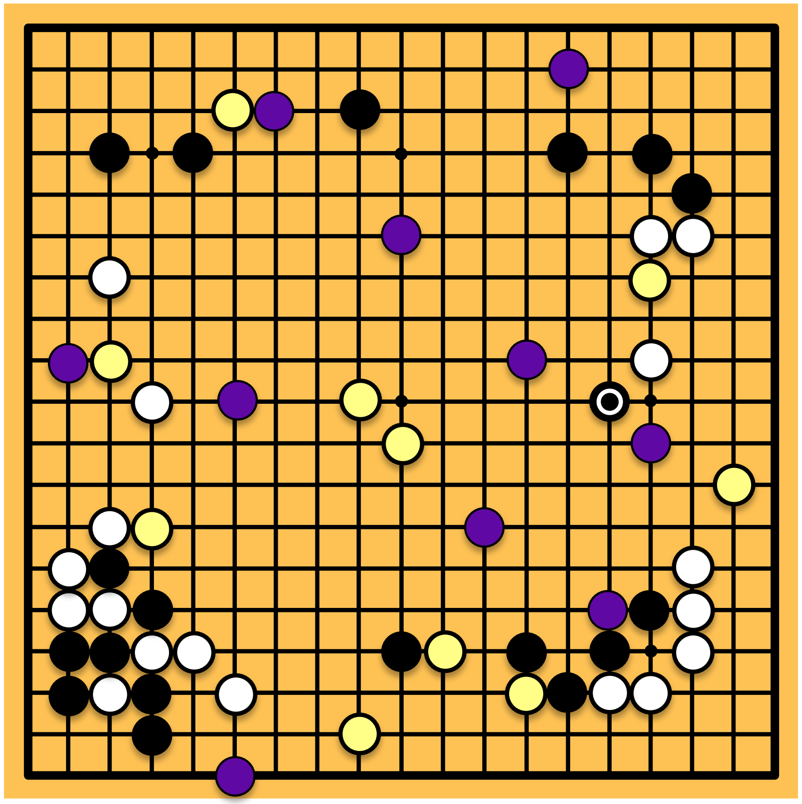
“Move 37”



Rollouts

- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result

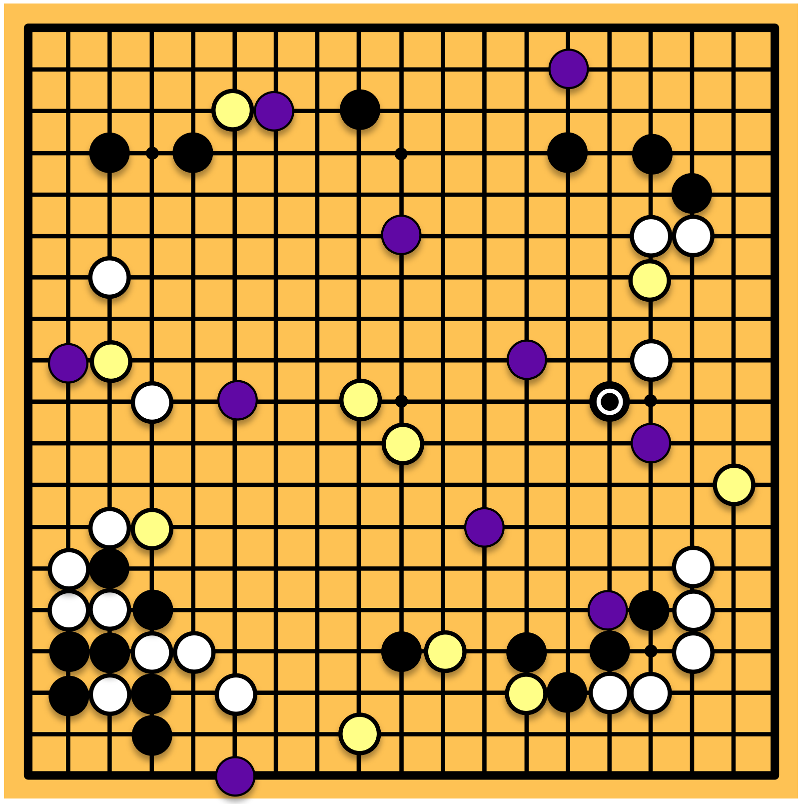
“Move 37”



Rollouts

- For each rollout:
  - – Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - – Record the result
- Fraction of wins correlates with the true value of the position!

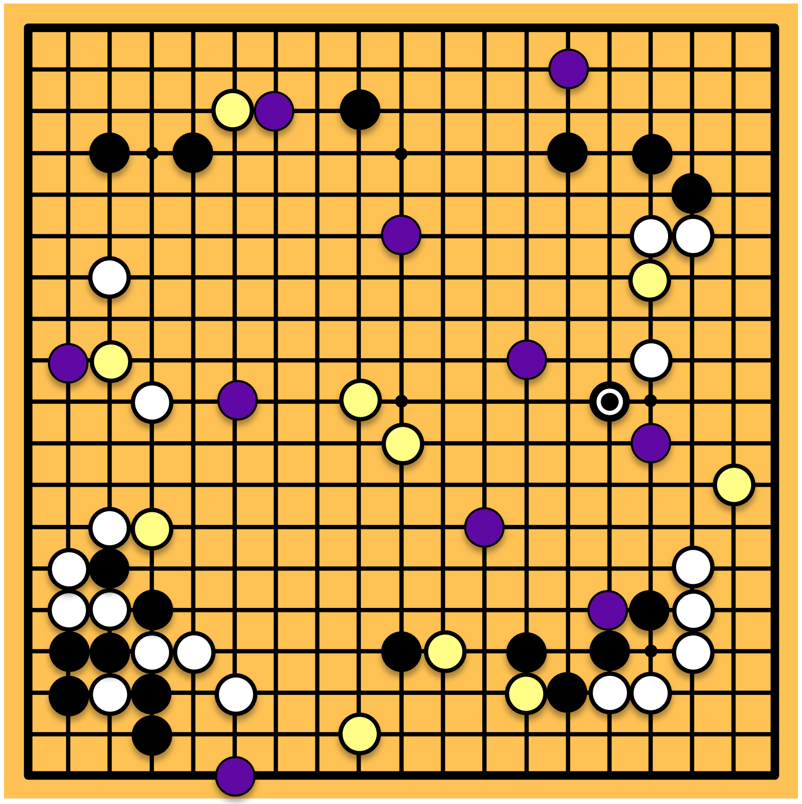
“Move 37”



Rollouts

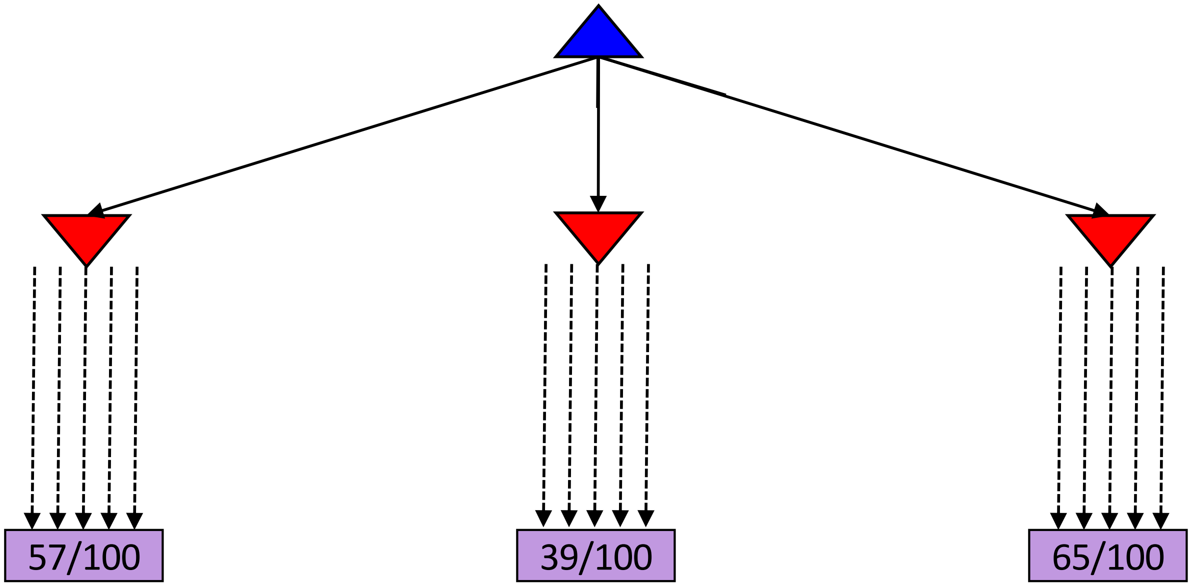
- For each rollout:
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result
- Fraction of wins correlates with the true value of the position!
- Having a “better” rollout policy helps

“Move 37”



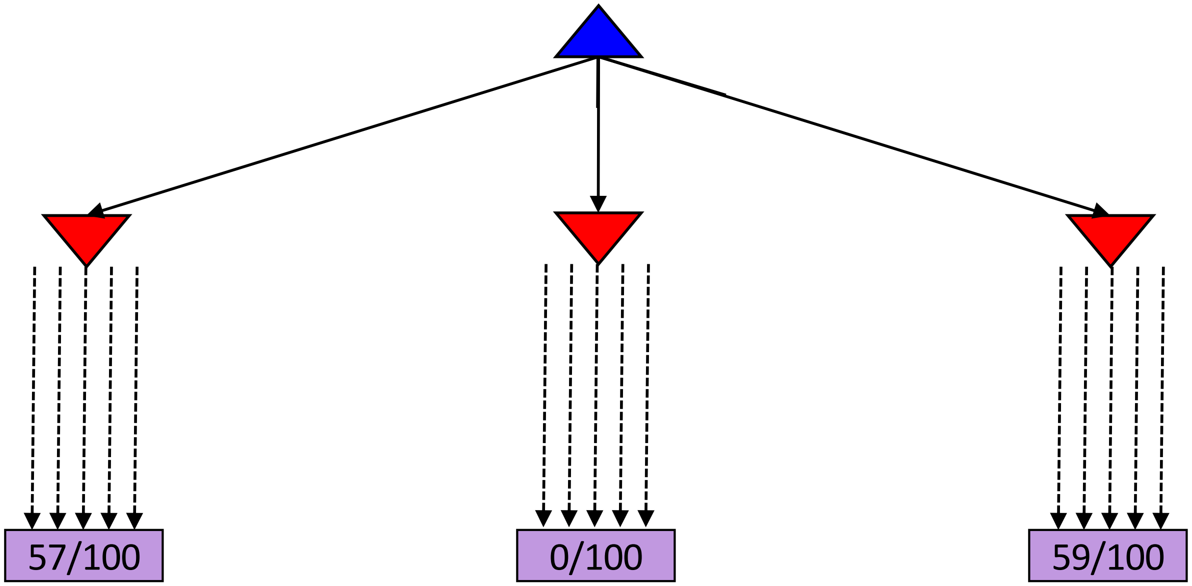
MCTS Version 0

- Do *N* rollouts from each child of the root, record fraction of wins
- Pick the move that gives the best outcome by this metric



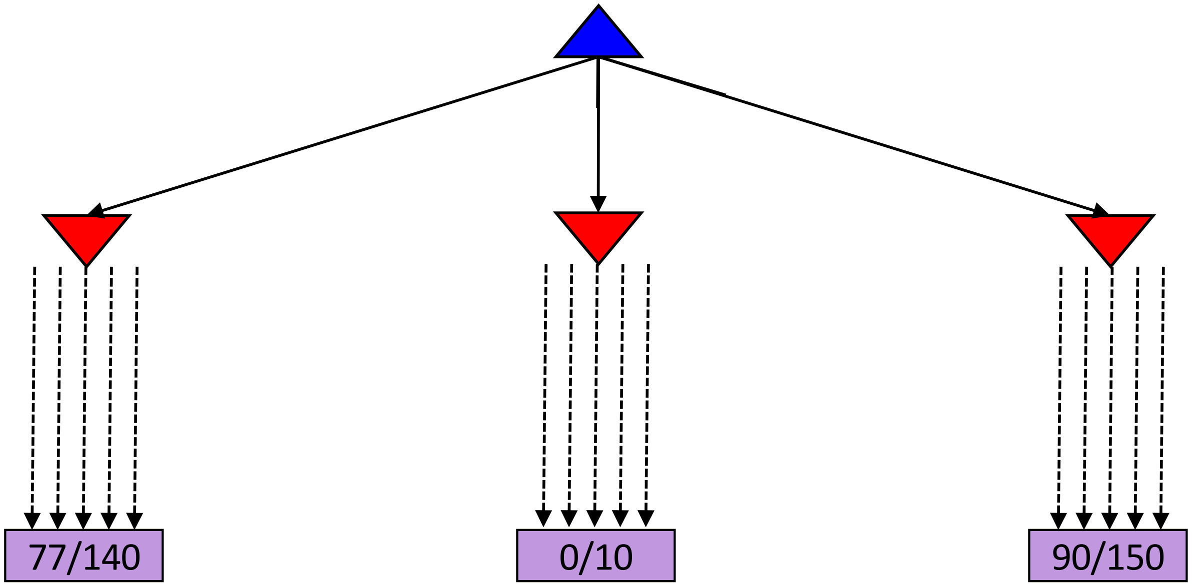
MCTS Version 0

- Do *N* rollouts from each child of the root, record fraction of wins
- Pick the move that gives the best outcome by this metric



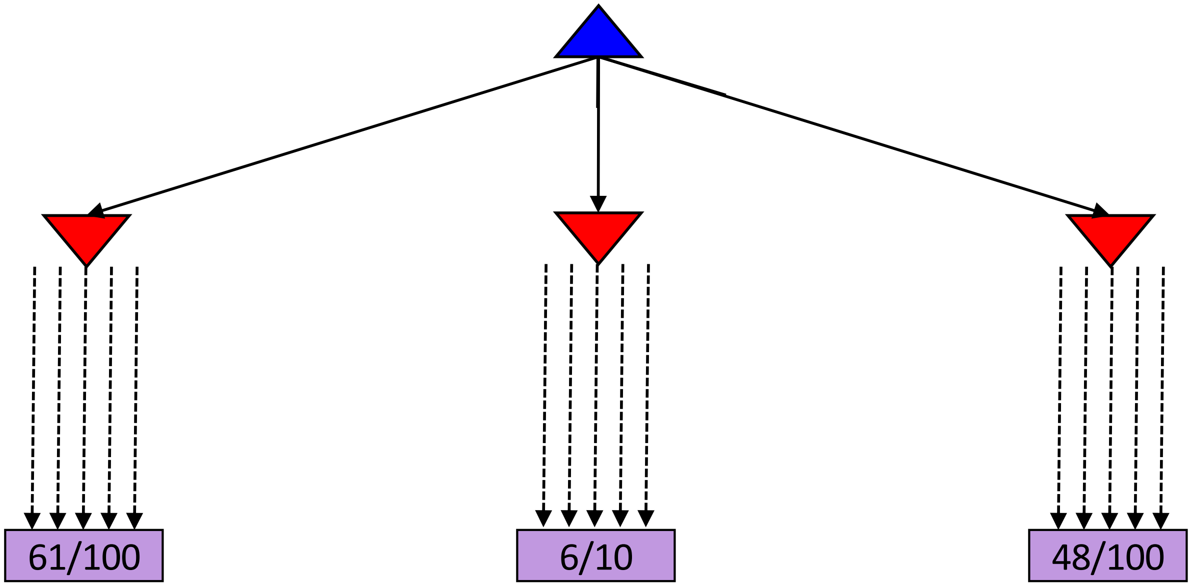
MCTS Version 0.9

- Allocate rollouts to more promising nodes



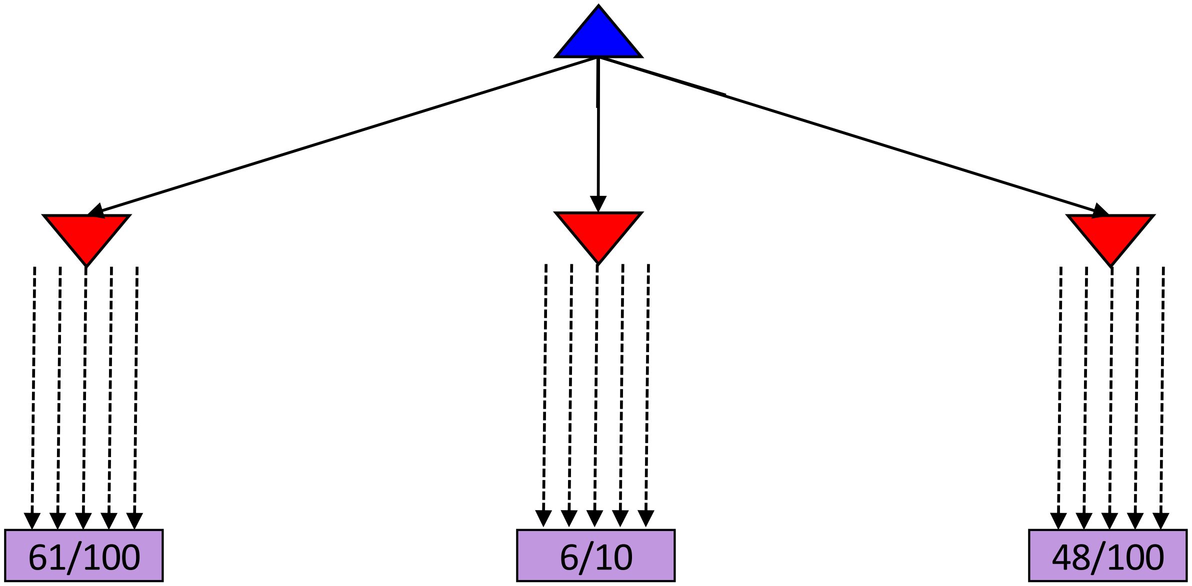
MCTS Version 0.9

- Allocate rollouts to more promising nodes



MCTS Version 1.0

- Allocate rollouts to more promising nodes
- Allocate rollouts to more uncertain nodes



Upper Confidence Bounds (UCB) heuristics

- UCB1 formula combines "promising" and "uncertain":
  - C is a parameter we choose to trade off between two terms

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{Parent}(n))}{N(n)}}$$

- $N(n)$ = number of rollouts from node *n*
- $U(n)$ = total utility of rollouts (# wins) for player of Parent(*n*)
  - Keep track of both N and U for each node

Upper Confidence Bounds (UCB) heuristics

- UCB1 formula combines "promising" and "uncertain":
  - C is a parameter we choose to trade off between two terms

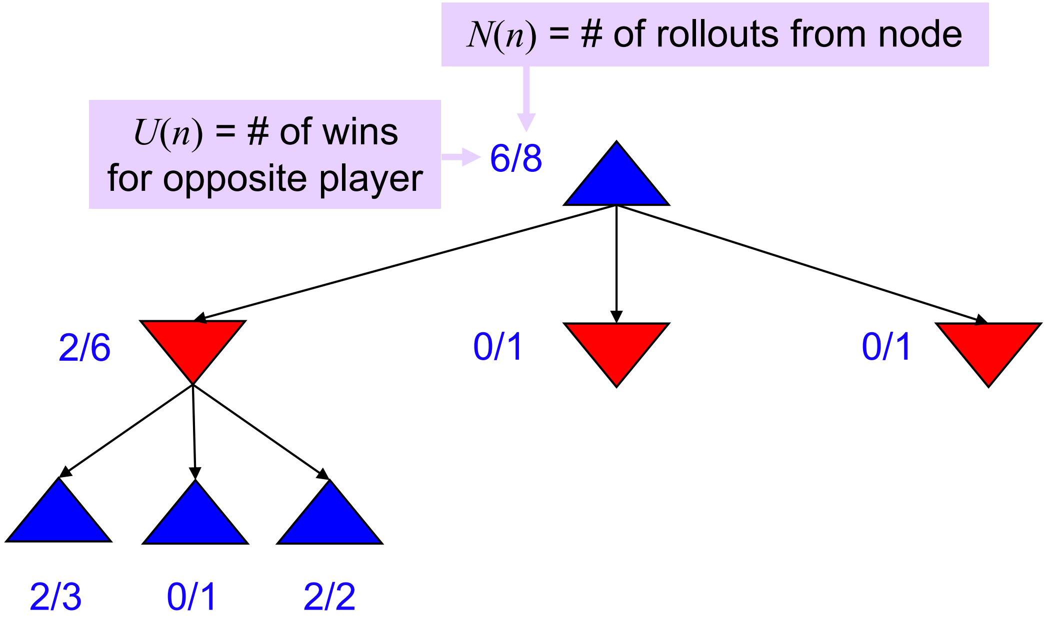
$$UCB1(n) = \frac{U(n)}{N(n)} + \boxed{C \times \sqrt{\frac{\log N(\text{Parent}(n))}{N(n)}}}$$

- High for small N
- Low for large N

- $N(n)$ = number of rollouts from node *n*
- $U(n)$ = total utility of rollouts (# wins) for player of Parent(*n*)
  - Keep track of both N and U for each node

MCTS Algorithm

- Repeat until out of time:
  - **Selection:** recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion:** add a new child *c* to *n*
  - **Simulation:** run a rollout from *c*
  - **Backpropagation:** update U and N counts from *c* back up to the root

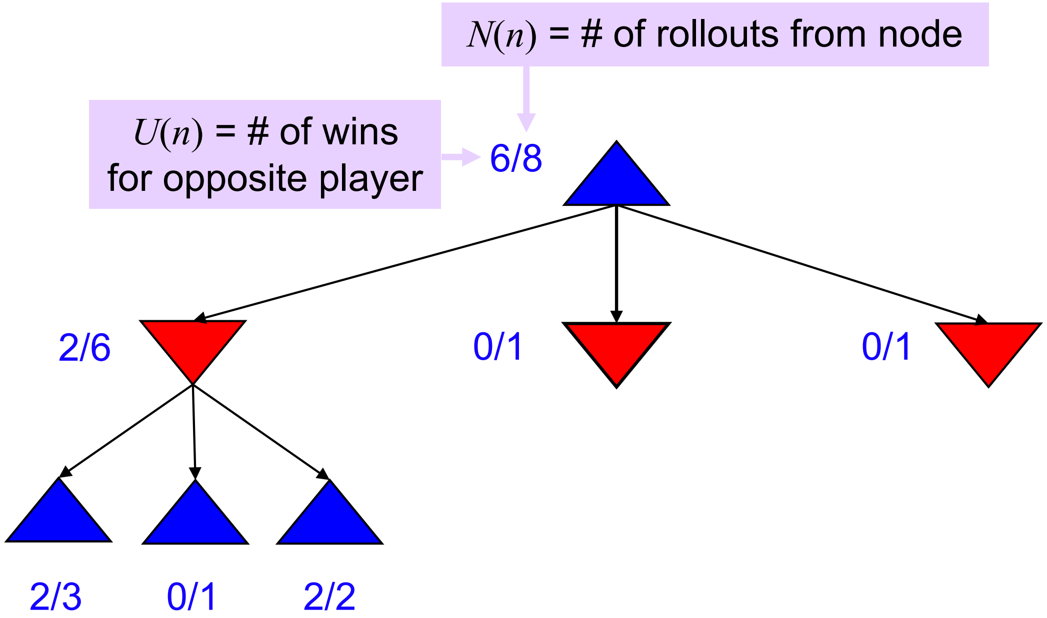

$N(n)$ = # of rollouts from node

$U(n)$ = # of wins for opposite player

MCTS Algorithm

- Repeat until out of time:
  - **Selection**: recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion:** add a new child *c* to *n*
  - **Simulation:** run a rollout from *c*
  - **Backpropagation:** update U and N counts from *c* back up to the root

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{Parent}(n))}{N(n)}}$$



$N(n)$ = # of rollouts from node

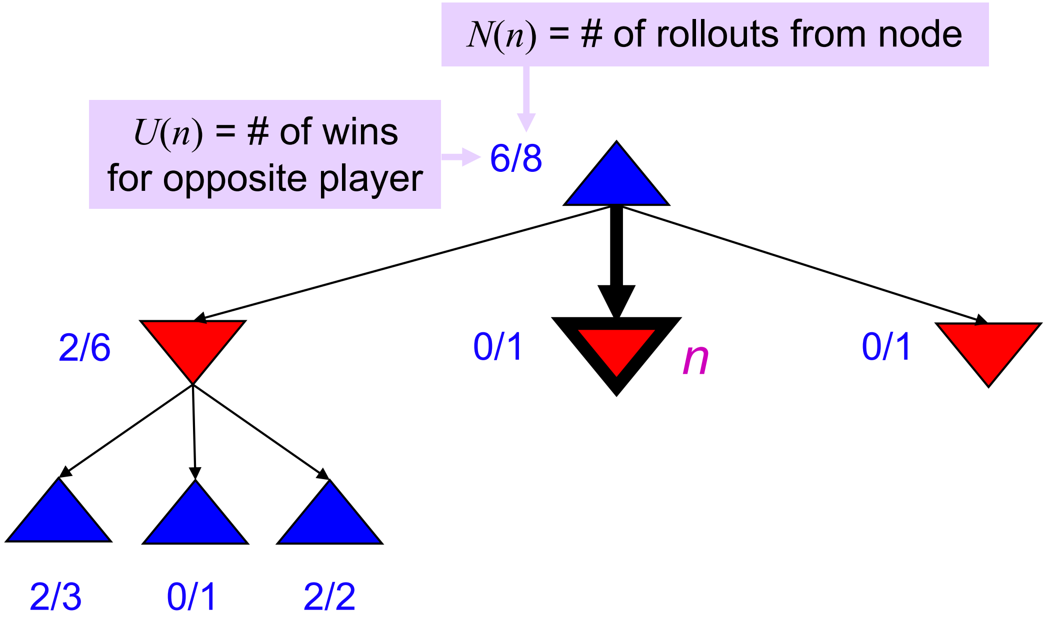
$U(n)$ = # of wins for opposite player

For 3 red nodes above the UCB values (with C=1) are:

$$\frac{2}{6} + \sqrt{\frac{\log 8}{6}} \quad 1.06 \qquad \frac{0}{1} + \sqrt{\frac{\log 8}{1}} \quad 1.75 \qquad \frac{0}{1} + \sqrt{\frac{\log 8}{1}}$$

MCTS Algorithm

- Repeat until out of time:
  - **Selection**: recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion:** add a new child *c* to *n*
  - **Simulation:** run a rollout from *c*
  - **Backpropagation:** update U and N counts from *c* back up to the root



$N(n)$ = # of rollouts from node

$U(n)$ = # of wins for opposite player

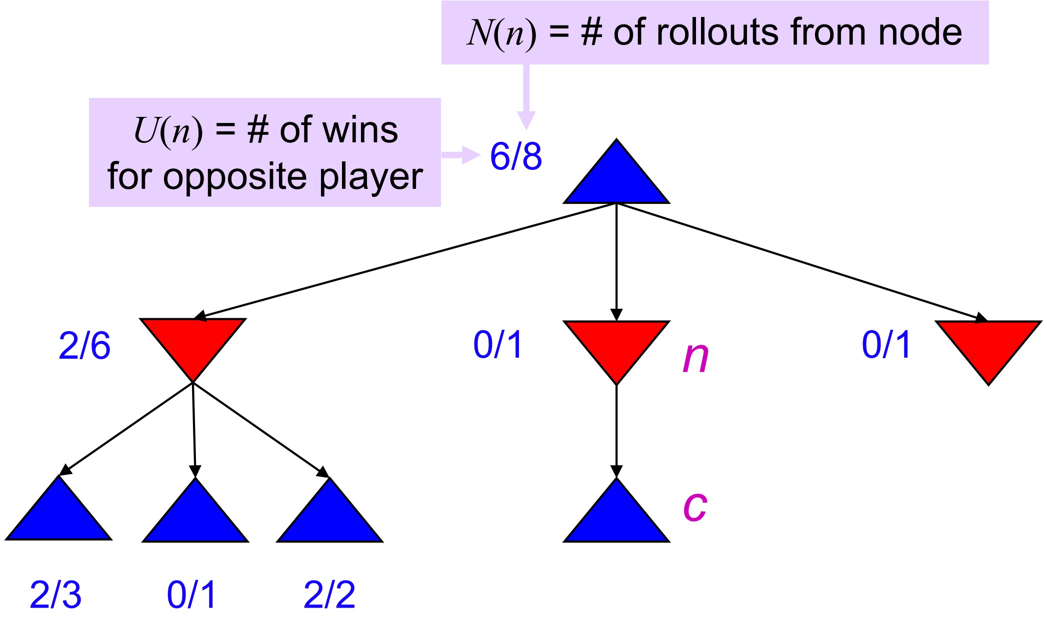
$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{Parent}(n))}{N(n)}}$$

For 3 red nodes above the UCB values (with C=1) are:

$$\frac{2}{6} + \sqrt{\frac{\log 8}{6}} \qquad \frac{0}{1} + \sqrt{\frac{\log 8}{1}} \qquad \frac{0}{1} + \sqrt{\frac{\log 8}{1}}$$

MCTS Algorithm

- Repeat until out of time:
  - **Selection:** recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion**: add a new child *c* to *n*
  - **Simulation:** run a rollout from *c*
  - **Backpropagation:** update U and N counts from *c* back up to the root

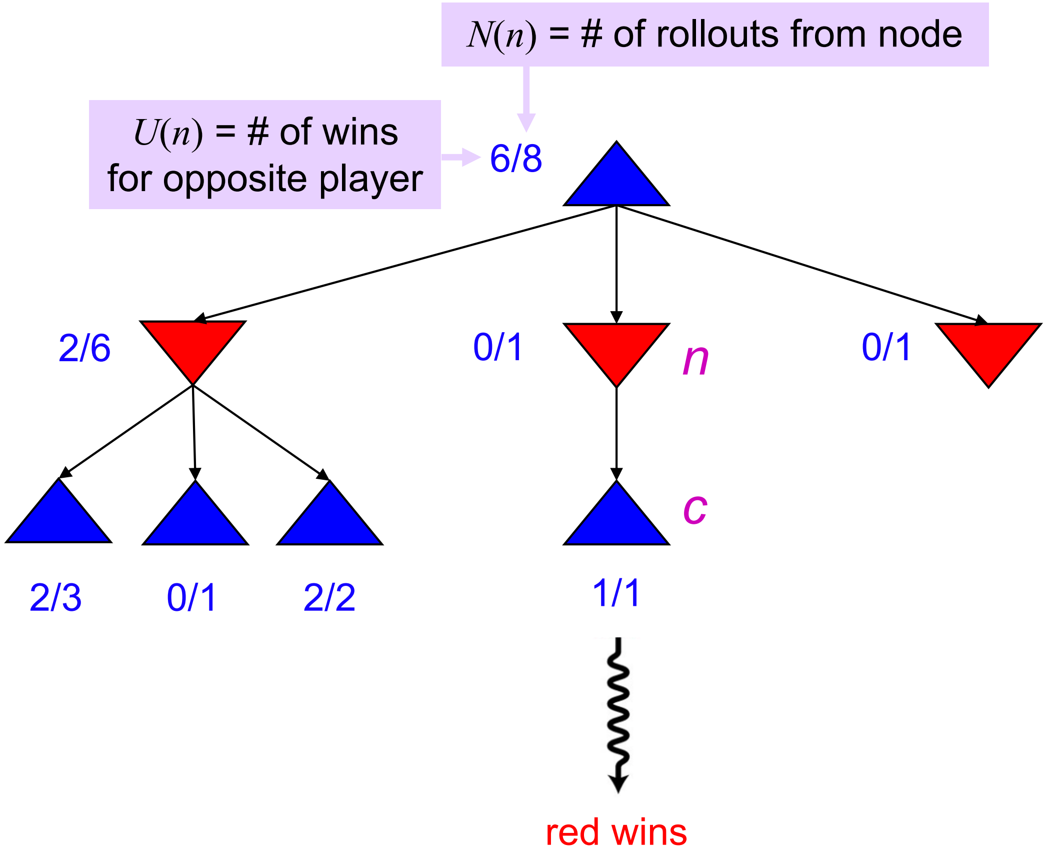


$N(n)$ = # of rollouts from node

$U(n)$ = # of wins for opposite player

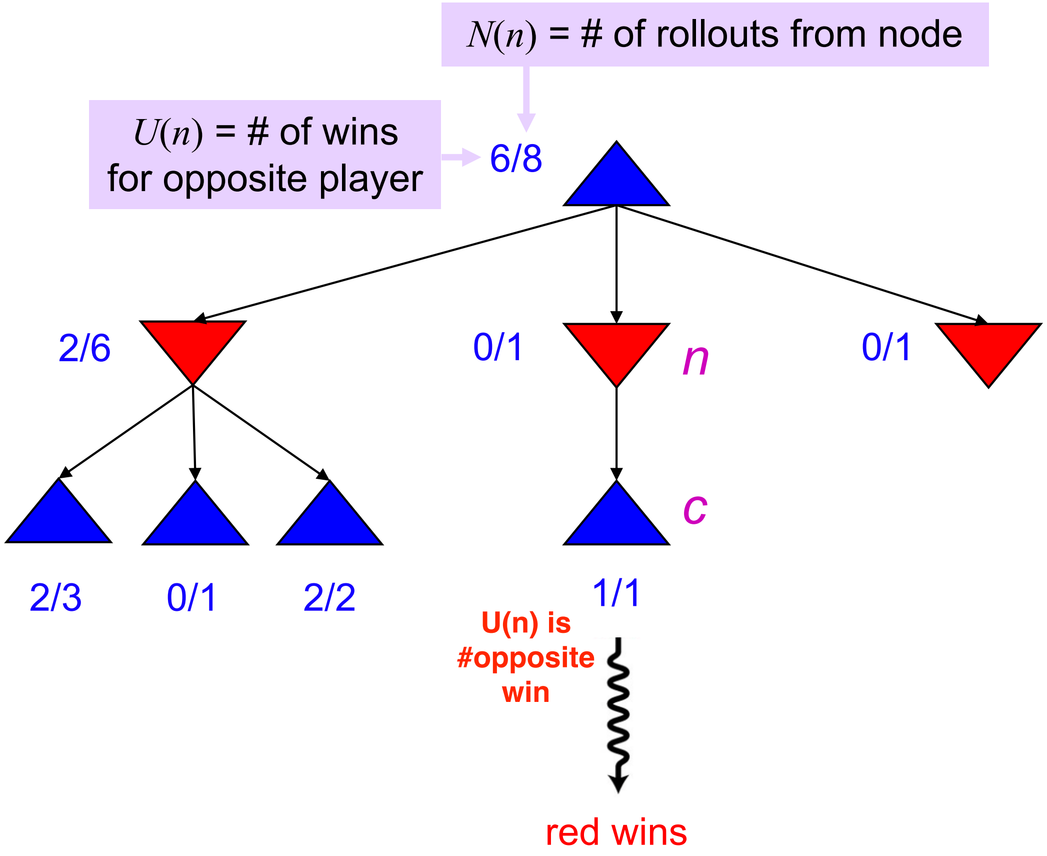
MCTS Algorithm

- Repeat until out of time:
  - **Selection:** recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion:** add a new child *c* to *n*
  - **Simulation**: run a rollout from *c*
  - **Backpropagation:** update U and N counts from *c* back up to the root



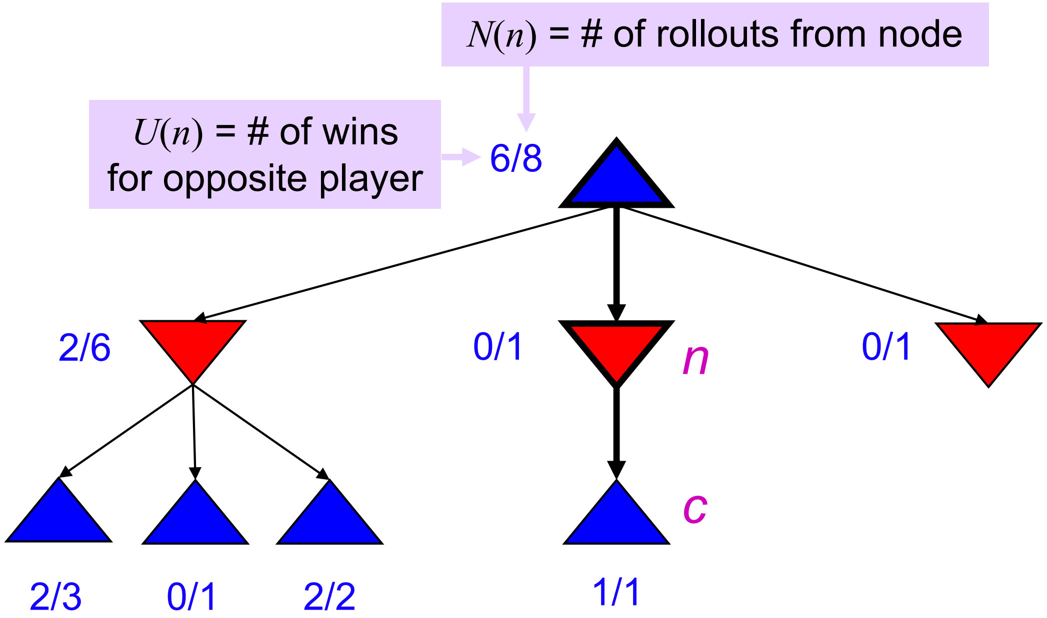
MCTS Algorithm

- Repeat until out of time:
  - **Selection:** recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion:** add a new child *c* to *n*
  - **Simulation**: run a rollout from *c*
  - **Backpropagation:** update U and N counts from *c* back up to the root



MCTS Algorithm

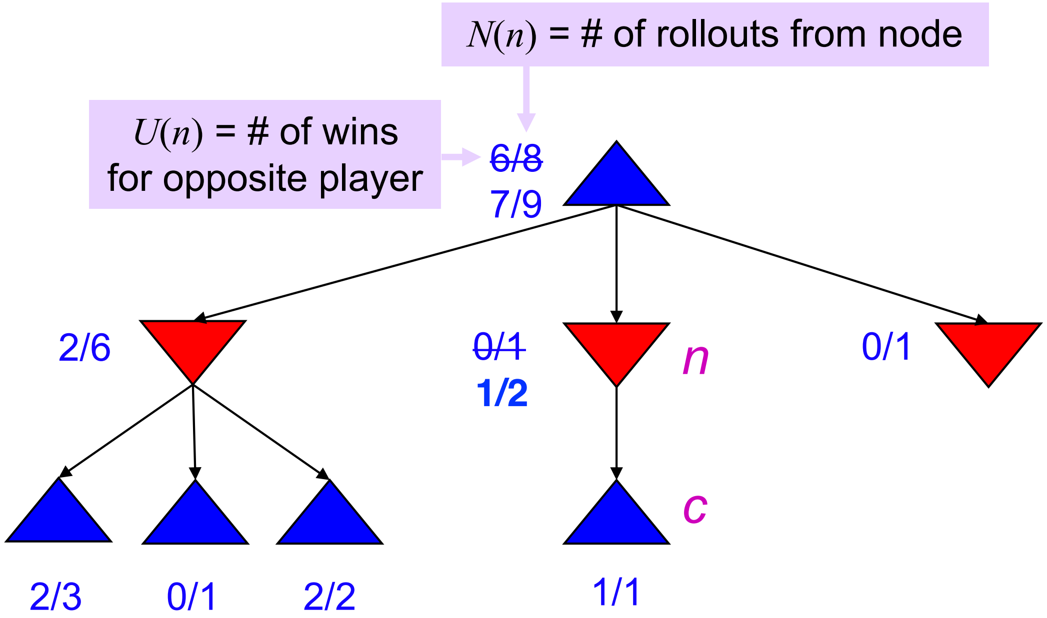
- Repeat until out of time:
  - **Selection:** recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion:** add a new child *c* to *n*
  - **Simulation:** run a rollout from *c*
  - **Backpropagation**: update U and N counts from *c* back up to the root


$N(n)$ = # of rollouts from node

$U(n)$ = # of wins for opposite player

MCTS Algorithm

- Repeat until out of time:
  - **Selection:** recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion:** add a new child *c* to *n*
  - **Simulation:** run a rollout from *c*
  - **Backpropagation**: update U and N counts from *c* back up to the root

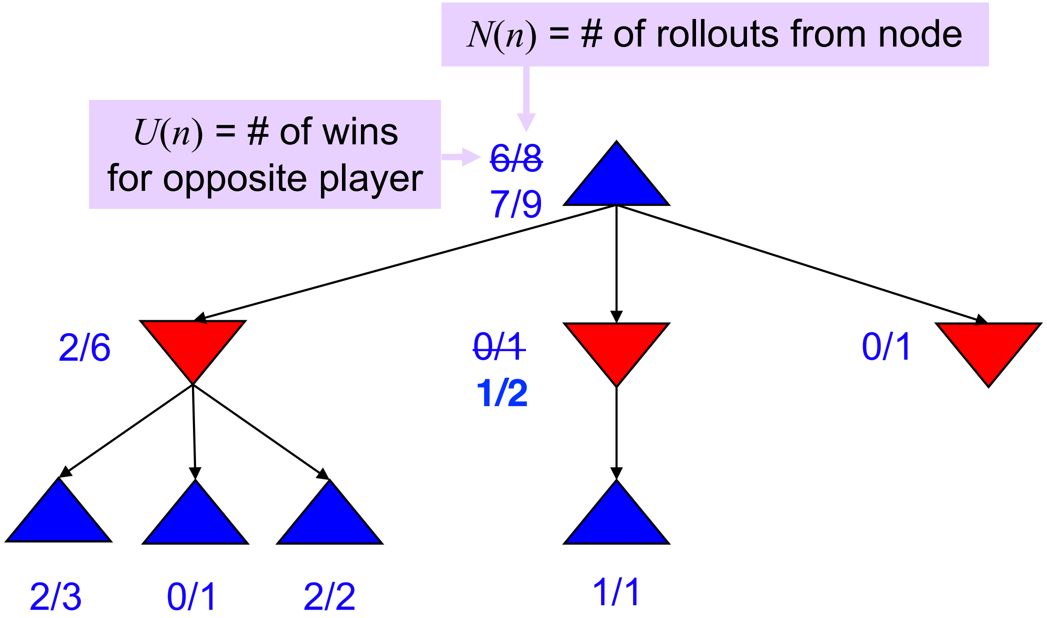


$N(n)$ = # of rollouts from node

$U(n)$ = # of wins for opposite player

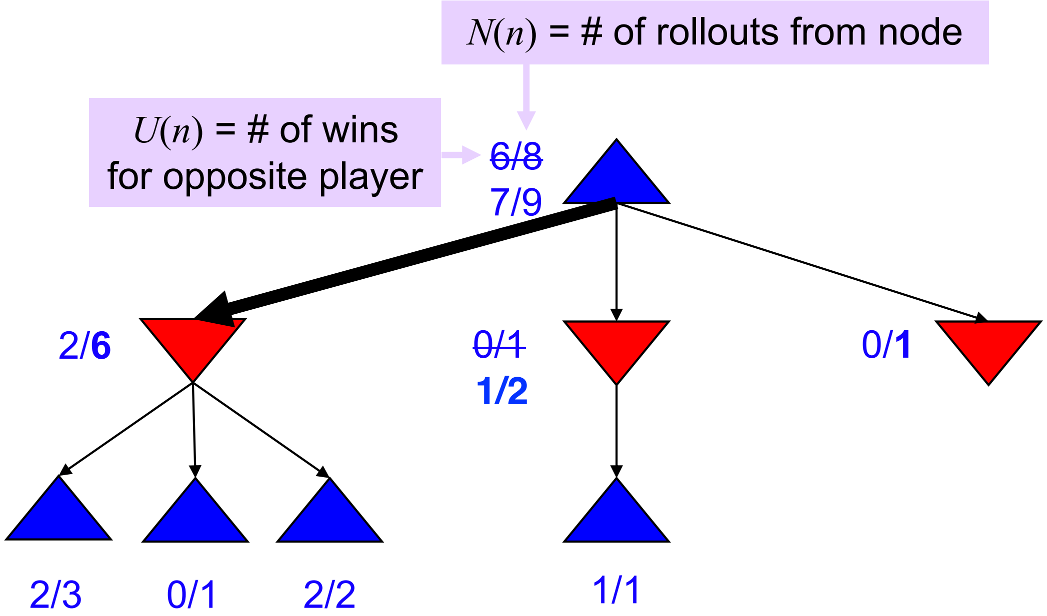
MCTS Algorithm

- Repeat until out of time:
  - **Selection:** recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion:** add a new child *c* to *n*
  - **Simulation:** run a rollout from *c*
  - **Backpropagation:** update U and N counts from *c* back up to the root
- Choose the action leading to the child with highest *N*



MCTS Algorithm

- Repeat until out of time:
  - **Selection:** recursively apply UCB to choose a path down to a leaf node *n*
  - **Expansion:** add a new child *c* to *n*
  - **Simulation:** run a rollout from *c*
  - **Backpropagation:** update U and N counts from *c* back up to the root
- Choose the action leading to the child with highest *N*



MCTS Summary

- MCTS is currently the most common tool for solving hard search problems
- Why?
  - Time complexity independent of b and m
  - No need to design evaluation functions (general-purpose & easy to use)
- Solution quality depends on number of rollouts N
  - Theorem: as $N \rightarrow \infty$ UCT selects the minimax move
- Example of using random sampling in an algorithm
  - Broadly called *Monte Carlo* methods
- MCTS can be improved further with machine learning

Why is there no min or max?????

- “Value” of a node, $U(n)/N(n)$, is a weighted ***sum*** of child values!

Why is there no min or max?????

- "Value" of a node, $U(n)/N(n)$, is a weighted ***sum*** of child values!
- Idea: as $N \rightarrow \infty$, the vast majority of rollouts are concentrated in the best child(ren), so weighted average $\rightarrow$ max/min

Why is there no min or max?????

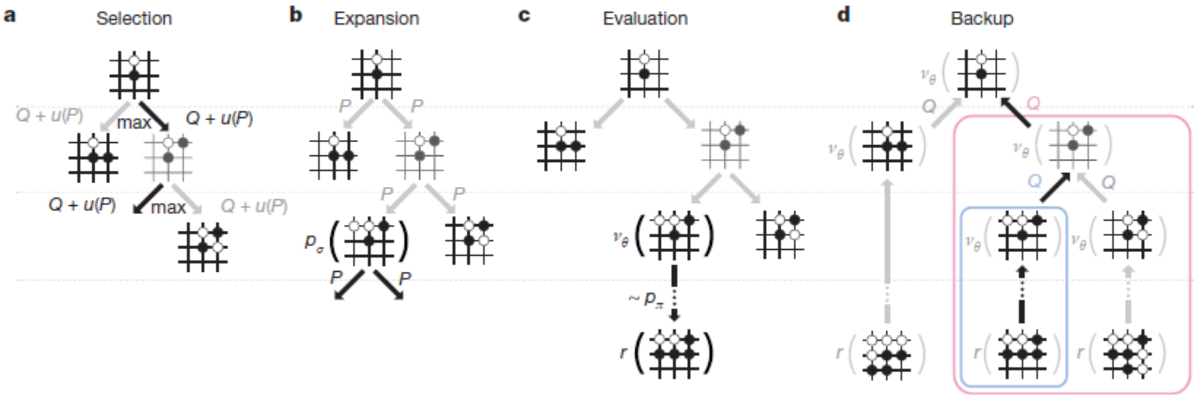
- “Value” of a node, $U(n)/N(n)$, is a weighted ***sum*** of child values!
- Idea: as $N \to \infty$, the vast majority of rollouts are concentrated in the best child(ren), so weighted average → max/min
- Theorem: as $N \to \infty$ UCT selects the minimax move

Why is there no min or max?????

- "Value" of a node, $U(n)/N(n)$, is a weighted ***sum*** of child values!
- Idea: as $N \rightarrow \infty$, the vast majority of rollouts are concentrated in the best child(ren), so weighted average → max/min
- Theorem: as $N \rightarrow \infty$ UCT selects the minimax move
  - (but N never approaches infinity!)

AlphaGo implementation

- Trained deep neural networks (13 layers) to learn **value function** and **policy function**
- Performs Monte Carlo game search
  - –explore state space like minimax
  - –random "rollouts"
  - –simulate probable plays by opponent according to policy function

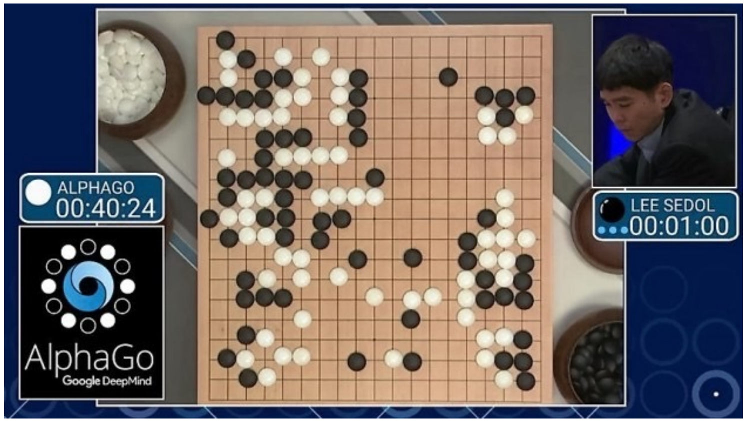


AlphaGo implementation

- Hardware: 1920 CPUs, 28O GPUs
- Neural networks trained in two phases over 4-6 weeks
- **Phase 1:** supervised learning from database of 30 million moves in games between two good human players
- **Phase 2:** play against versions of self using reinforcement learning to improve performance

MCTS + Machine Learning: AlphaGo

- Monte Carlo Tree Search with additions including:
  - Rollout policy is a neural network trained with reinforcement learning and expert human moves
  - In combination with rollout outcomes, use a trained value function to better predict node’s utility



[Mastering the game of Go with deep neural networks and tree search. Silver et al. Nature. 2016]

Summary

- Games require decisions when optimality is impossible
  - Bounded-depth search and approximate evaluation functions

Summary

- Games require decisions when optimality is impossible
    - Bounded-depth search and approximate evaluation functions
- Games force efficient use of computation
    - Alpha-beta pruning, MCTS

Summary

- Games require decisions when optimality is impossible
  - Bounded-depth search and approximate evaluation functions
- Games force efficient use of computation
  - Alpha-beta pruning, MCTS
- Game playing has produced important research ideas
  - Reinforcement learning (checkers)
  - Iterative deepening (chess)
  - Rational metareasoning (Othello)
  - Monte Carlo tree search (chess, Go)
  - Solution methods for partial-information games in economics (poker)

Summary

- Games require decisions when optimality is impossible
  - Bounded-depth search and approximate evaluation functions
- Games force efficient use of computation
  - Alpha-beta pruning, MCTS
- Game playing has produced important research ideas
  - Reinforcement learning (checkers)
  - Iterative deepening (chess)
  - Rational metareasoning (Othello)
  - Monte Carlo tree search (chess, Go)
  - Solution methods for partial-information games in economics (poker)
- Video games present much greater challenges – lots to do!
  - $b = 10^{500}$, $|S| = 10^{4000}$, $m = 10{,}000$, partially observable, often > 2 players