Reinforcement Learning

Monte-Carlo Tree Search (MCTS)

Or how to combine learning and planning

Akka Zemmari

Introduction to Monte-Carlo Tree Search (MCTS)

- Monte-Carlo Tree Search (MCTS) is a heuristic search algorithm.

Introduction to Monte-Carlo Tree Search (MCTS)

- Monte-Carlo Tree Search (MCTS) is a heuristic search algorithm.
- Combines random sampling with systematic tree expansion.

Introduction to Monte-Carlo Tree Search (MCTS)

- Monte-Carlo Tree Search (MCTS) is a heuristic search algorithm.
- Combines random sampling with systematic tree expansion.
- Widely used in board games (e.g., Go, Chess) and sequential decision-making problems.

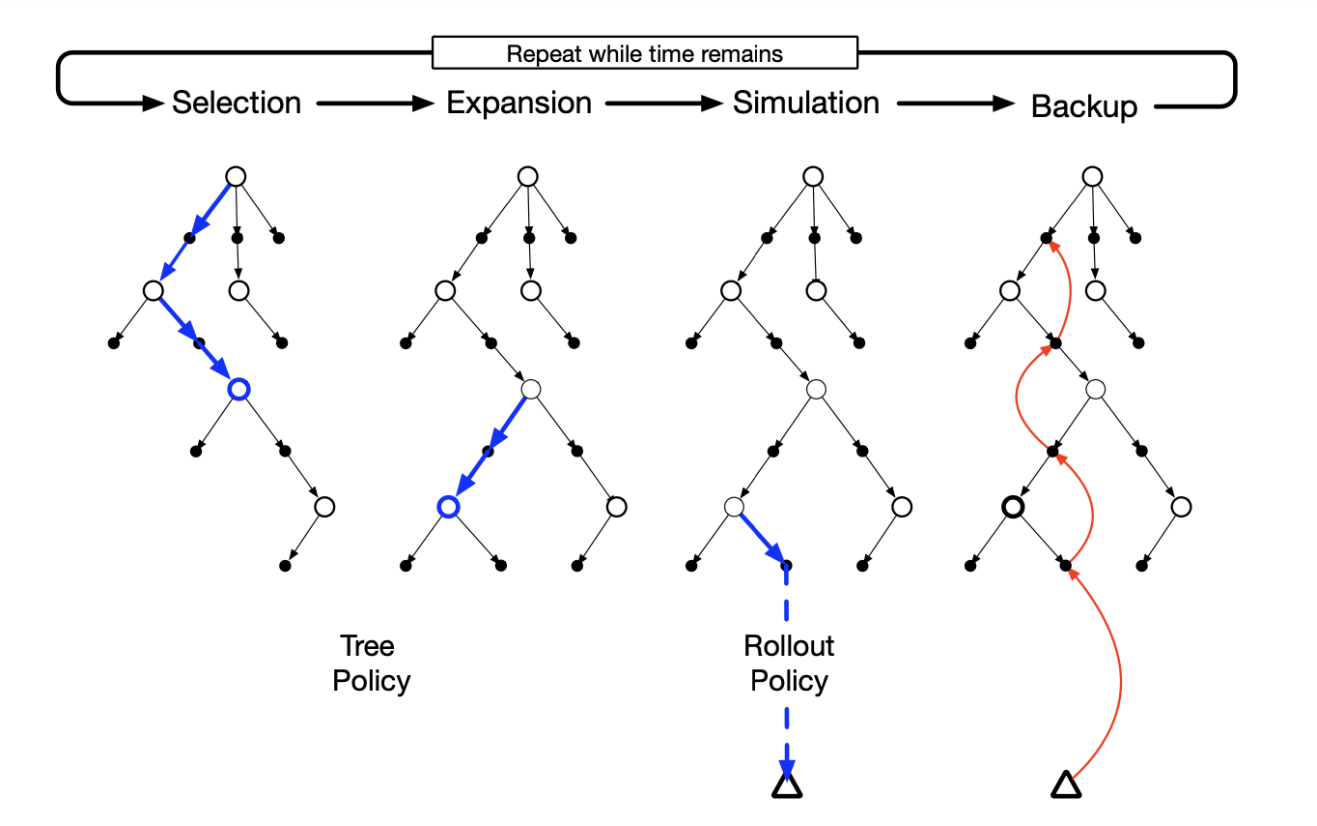
Introduction to Monte-Carlo Tree Search (MCTS)

- Monte-Carlo Tree Search (MCTS) is a heuristic search algorithm.
- Combines random sampling with systematic tree expansion.
- Widely used in board games (e.g., Go, Chess) and sequential decision-making problems.
- Effective for large and complex state spaces.

Core Idea of MCTS

- Incrementally builds a search tree based on simulations.

Monte Carlo Tree Search[1] in a Nutshell



[1] Sutton et al.

Core Idea of MCTS

- Operates in four key steps:
    1. **Selection**: Traverse the tree to select a promising node.

Core Idea of MCTS

- Operates in four key steps:
  1. **Selection**: Traverse the tree to select a promising node.
  2. **Expansion**: Add a new child node to the tree.

Core Idea of MCTS

- Operates in four key steps:
  1. **Selection**: Traverse the tree to select a promising node.
  2. **Expansion**: Add a new child node to the tree.
  3. **Simulation**: Perform random simulations from the new node.

Core Idea of MCTS

- Operates in four key steps:
  1. **Selection**: Traverse the tree to select a promising node.
  2. **Expansion**: Add a new child node to the tree.
  3. **Simulation**: Perform random simulations from the new node.
  4. **Backpropagation**: Update node values based on simulation results.

Step 1: Selection

- Traverse the tree using a selection policy (e.g., UCT).
- Upper Confidence (Bounds) for trees (UCT):

$$UCT = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}}$$

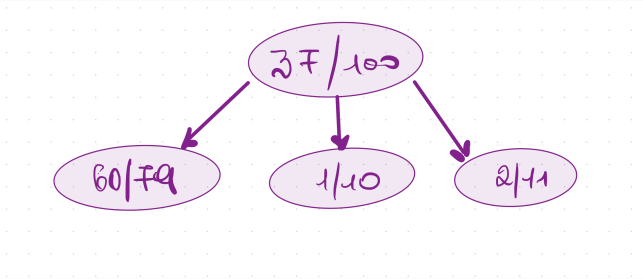
where:

- w_i: Number of wins for node i,
- n_i: Number of visits for node i,
- N: Total visits to the parent node,
- C: Exploration hyperparameter.

Step 1: Selection

- Traverse the tree using a selection policy (e.g., UCT).
- Upper Confidence (Bounds) for trees (UCT):

$$UCT = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}}$$



where:

- w_i: Number of wins for node i,
- n_i: Number of visits for node i,
- N: Total visits to the parent node,
- C: Exploration hyperparameter.

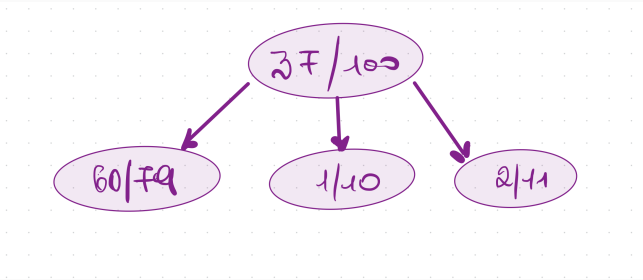
Step 1: Selection

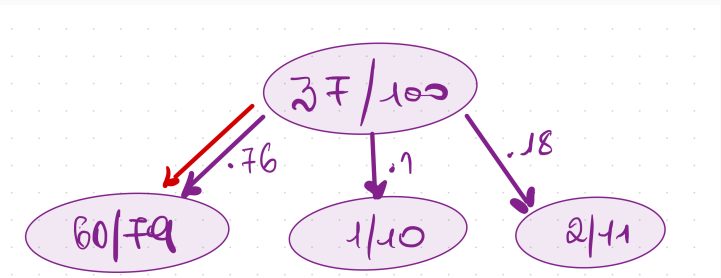
- Traverse the tree using a selection policy (e.g., UCT).
- Upper Confidence (Bounds) for trees (UCT):

$$UCT = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}}$$

where:

- w_i: Number of wins for node i,
- n_i: Number of visits for node i,
- N: Total visits to the parent node,
- C: Exploration hyperparameter.





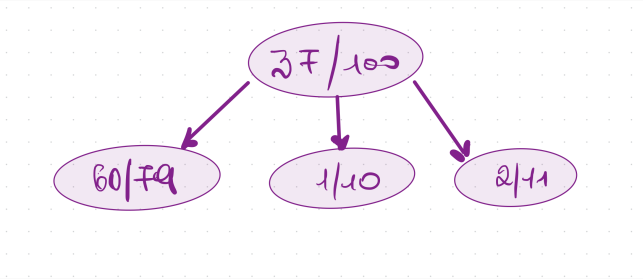
Step 1: Selection

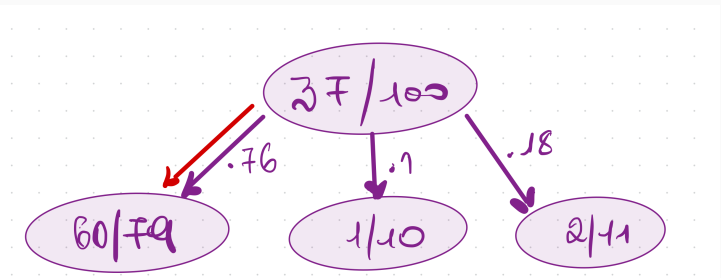
- Traverse the tree using a selection policy (e.g., UCT).
- Upper Confidence (Bounds) for trees (UCT):

$$UCT = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}}$$

where:

- w_i: Number of wins for node i,
- n_i: Number of visits for node i,
- N: Total visits to the parent node,
- C: Exploration hyperparameter.





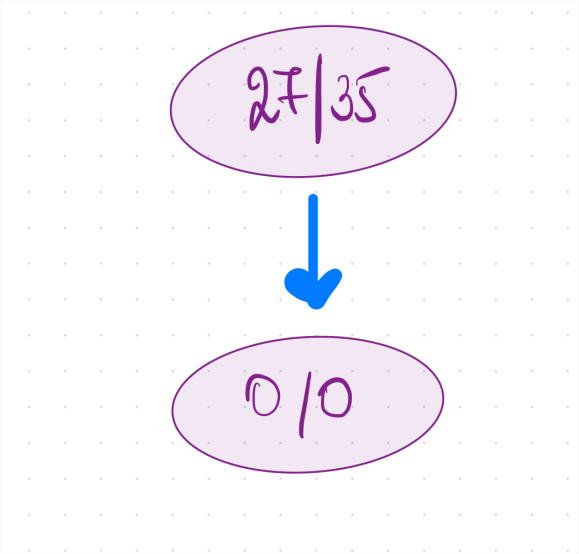
Remark: In the illustration, we do not realy use the UCB formula, but the idea is the same.

Step 2: Expansion

- Expand the tree by adding a new child node to the selected node.
- Allows the algorithm to explore previously unvisited states.

Step 2: Expansion

- Expand the tree by adding a new child node to the selected node.
- Allows the algorithm to explore previously unvisited states.

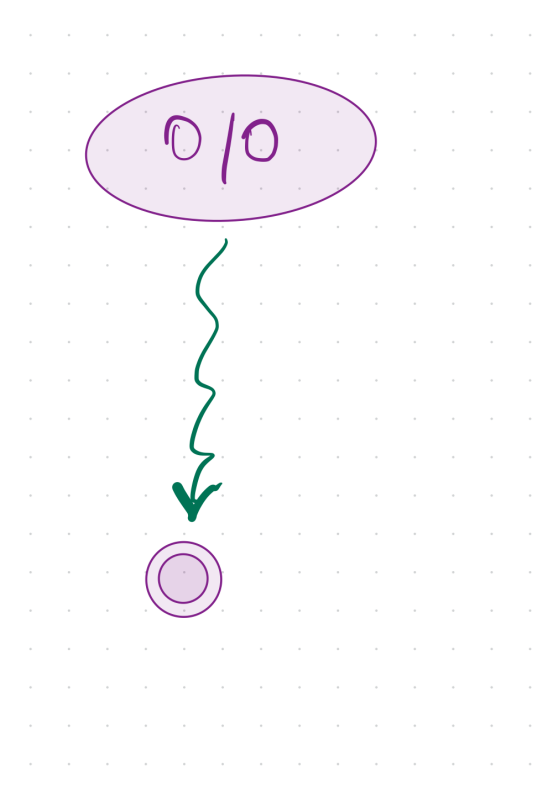


Step 3: Simulation

- Perform a random simulation (rollout) from the newly added node.
- Continue until reaching a terminal state.
- Evaluate the outcome of the simulation (e.g., win, loss, draw).

Step 3: Simulation

- Perform a random simulation (rollout) from the newly added node.
- Continue until reaching a terminal state.
- Evaluate the outcome of the simulation (e.g., win, loss, draw).

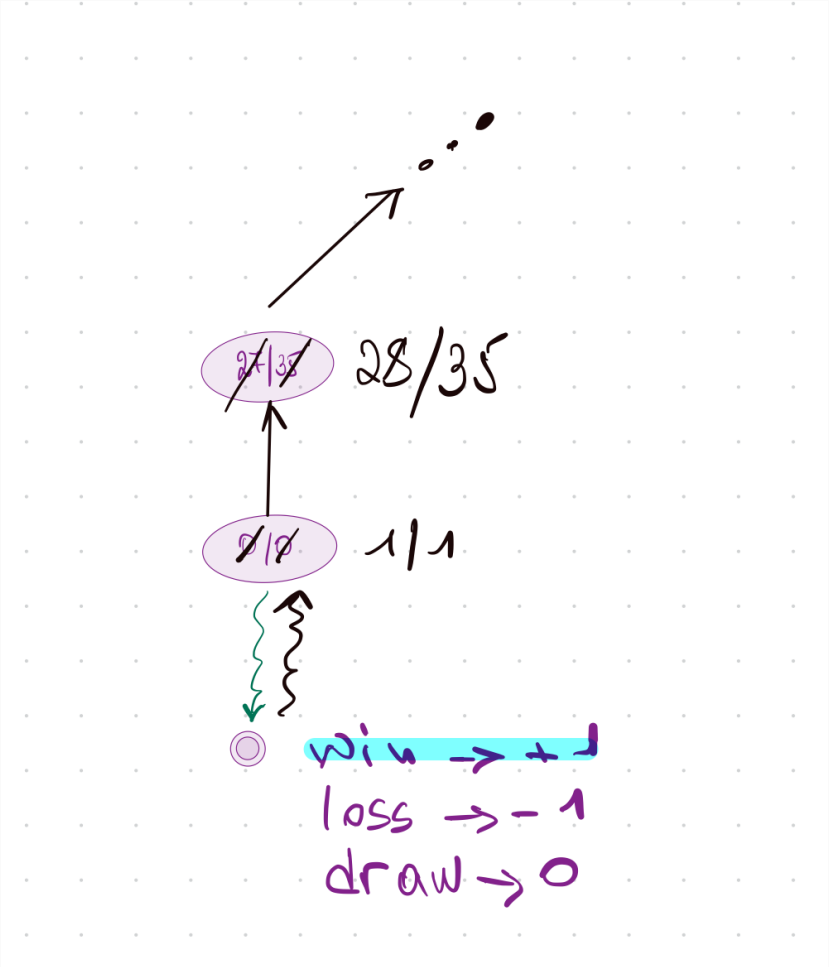


Step 4: Backpropagation

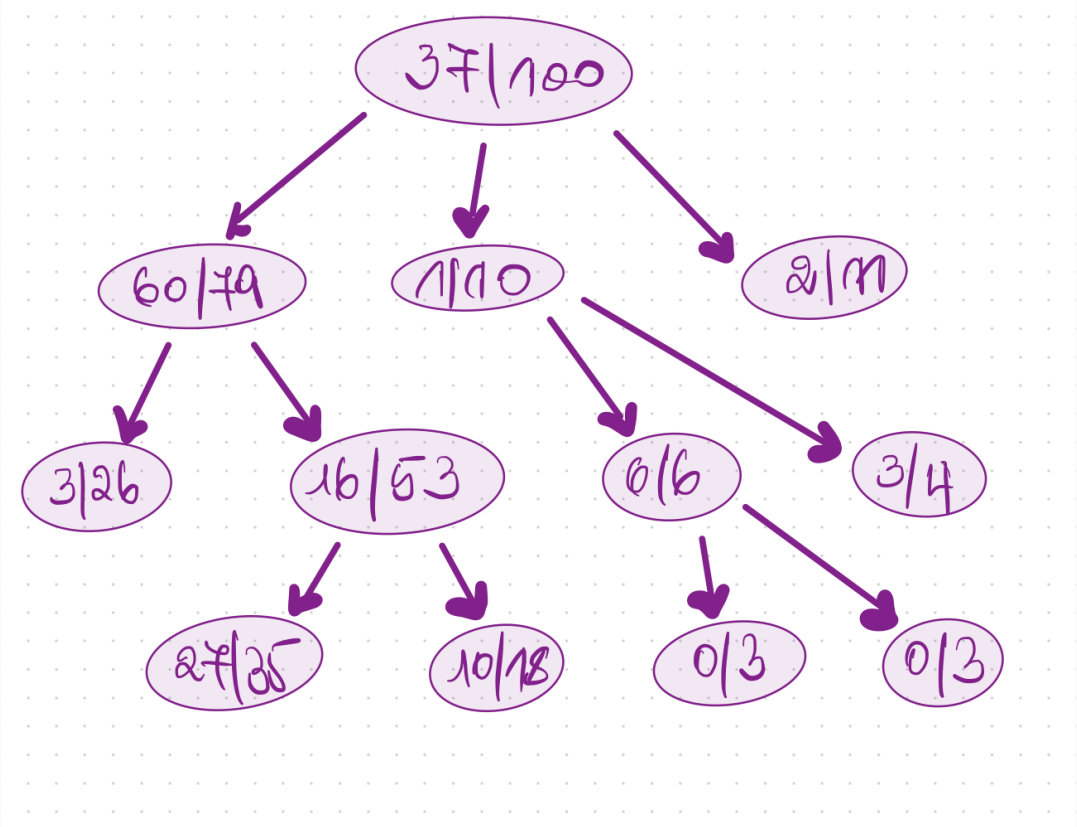
- Update the statistics of nodes along the path from the simulated node to the root.
- Adjust win/loss ratios and visit counts.

Step 4: Backpropagation

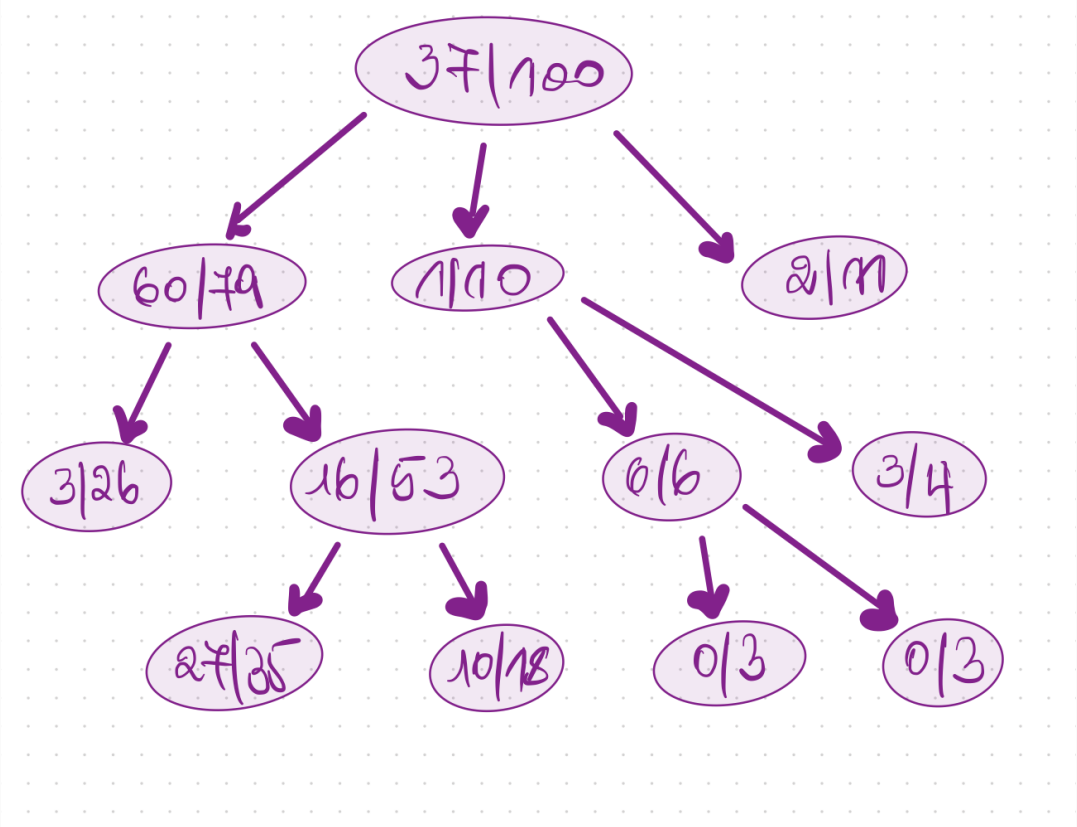
- Update the statistics of nodes along the path from the simulated node to the root.
- Adjust win/loss ratios and visit counts.



All the steps together



All the steps together



Whiteboard time!

Exploration vs Exploitation

- MCTS balances:
  - **Exploration**: Sampling less visited nodes.
  - **Exploitation**: Focusing on nodes with high win rates.
- UCT guides this balance effectively.

Advantages and Limitations of MCTS

- **Advantages**:
  - Effective for large search spaces.
  - Requires no prior knowledge.
  - Adaptable to various domains.
- **Limitations**:
  - Computationally expensive.
  - Results depend on the number of simulations.

Applications of MCTS

- Board games: Go, Chess, ...
- Planning and robotics.
- Real-time strategy games.
- General decision-making problems.

Conclusion

- MCTS is a versatile and powerful search algorithm.
- Balances random sampling with tree-based exploration.
- Widely applicable across domains with complex decision trees.