COMP250: Artificial Intelligence

5: Game Tree Search

Minimax search

Minimax

- Terminal game states have a **value**
  - E.g. $+1$ for a win, -1 for a loss, 0 for a draw
- I want to **maximise** the value
- My opponent wants to **minimise** the value
- Therefore I want to **maximise** the **minimum** value my opponent can achieve
- This is generally only true for **two-player zero-sum** games

Minimax search

- Recursively defines a **value** for non-terminal game states
- Consider each possible "next state", i.e. each possible move
- If it's my turn, the value is the **maximum** value over next states
- If it's my opponent's turn, the value is the **minimum** value over next states

Minimax search pseudocode

```
procedure MINIMAX(state)
    if state is terminal then
        return value of state
    else if state.currentPlayer = 1 then
        bestValue = −∞
        for each possible nextState do
            v = MINIMAX(nextState)
            bestValue = MAX(bestValue, v)
        return bestValue
    else if state.currentPlayer = 2 then
        bestValue = +∞
        for each possible nextState do
            v = MINIMAX(nextState)
            bestValue = MIN(bestValue, v)
        return bestValue
```

Stopping early

for each possible nextState **do**
    v = MINIMAX(nextState)
    bestValue = MAX(bestValue, v)

- State values are always between -1 and $+1$
- So if we ever have bestValue $= 1$, we can stop early
- Similarly when minimising if bestValue $= -1$

Using minimax search

- To decide what move to play next...
- Calculate the minimax value for each move
- Choose the move with the maximum score
- If there are several with the same score, choose one at random

Minimax and game theory

- For a **two-player zero-sum** game with **perfect information** and **sequential moves**
- Minimax search will always find a **Nash equilibrium**
- I.e. a minimax player plays **perfectly**
- **But...**

Minimax for larger games

- The game tree for noughts and crosses has only a few thousand states
- Most games are too large to search fully
  - Connect 4 has $\approx 10^{13}$ states
  - Chess has $\approx 10^{47}$ states

Heuristics for search

Depth limiting

- Standard minimax needs to search all the way to **terminal** (game over) states
- **Depth limiting** is a common technique to apply minimax to larger games
- Still evaluate terminal states as $+1$ / 0 / -1
- For nonterminal states at depth d, apply a heuristic evaluation instead of searching deeper
- Evaluation is a number between -1 and $+1$, estimating the probable outcome of the game

1-ply search

- Case $d = 1$
- For each move, evaluate the state resulting from playing that move
- This is computationally fast
- Often easier to design a "which state is better" heuristic than to directly design a "which move to play" heuristic
- This is essentially a **utility-based AI**

Move ordering

- Minimax can **stop early** if it sees a value of $+1$ for maximising player or -1 for minimising player
- Modifications to minimax algorithm (e.g. **alpha-beta pruning**) lead to more of this
- Thus ordering moves from **best to worst** means faster search
- How do we know which moves are "best" and "worst"? Use a heuristic!

Designing heuristics

- The **playing strength** of depth limited minimax depends heavily on the design of the **heuristic**
- Good heuristic design requires **in-depth knowledge** of the tactics and strategy of the game
- What if we don't possess such knowledge?

Monte Carlo evaluation

Monte Carlo methods

- In computing, a **Monte Carlo method** is an algorithm based on **averaging over random samples**
- The **average** over a large number of samples is a good approximation of the **expected value**
- Used for **quickly approximating** quantities over **large domains**
- Generally designed to **converge in the limit**
  - An **infinite** number of samples would give an **exact** answer
  - As the **number of samples** increases, the **accuracy** of the answer improves
- Applications in physics, engineering, finance, weather forecasting, graphics, ...

Aside: “randomness” in computing

- Digital computers are **deterministic**, so there’s no such thing as true randomness
    - Cryptographically secure systems use an external source of randomness e.g. atmospheric noise, radioactive decay
- What we actually have are **pseudo-random number generators (PRNGs)**
- A PRNG is an algorithm which gives an **unpredictable** sequence of numbers based on a **seed**
- Sequence is **uniformly distributed**, i.e. all numbers have equal probability
- Seed is generally based on some source of **entropy**, e.g. system clock, mouse input, electronic noise

Monte Carlo evaluation in games

- Based on **random rollouts**

while *s* is not terminal **do**
  let *m* be a random legal move from *s*
  update *s* by playing *m*

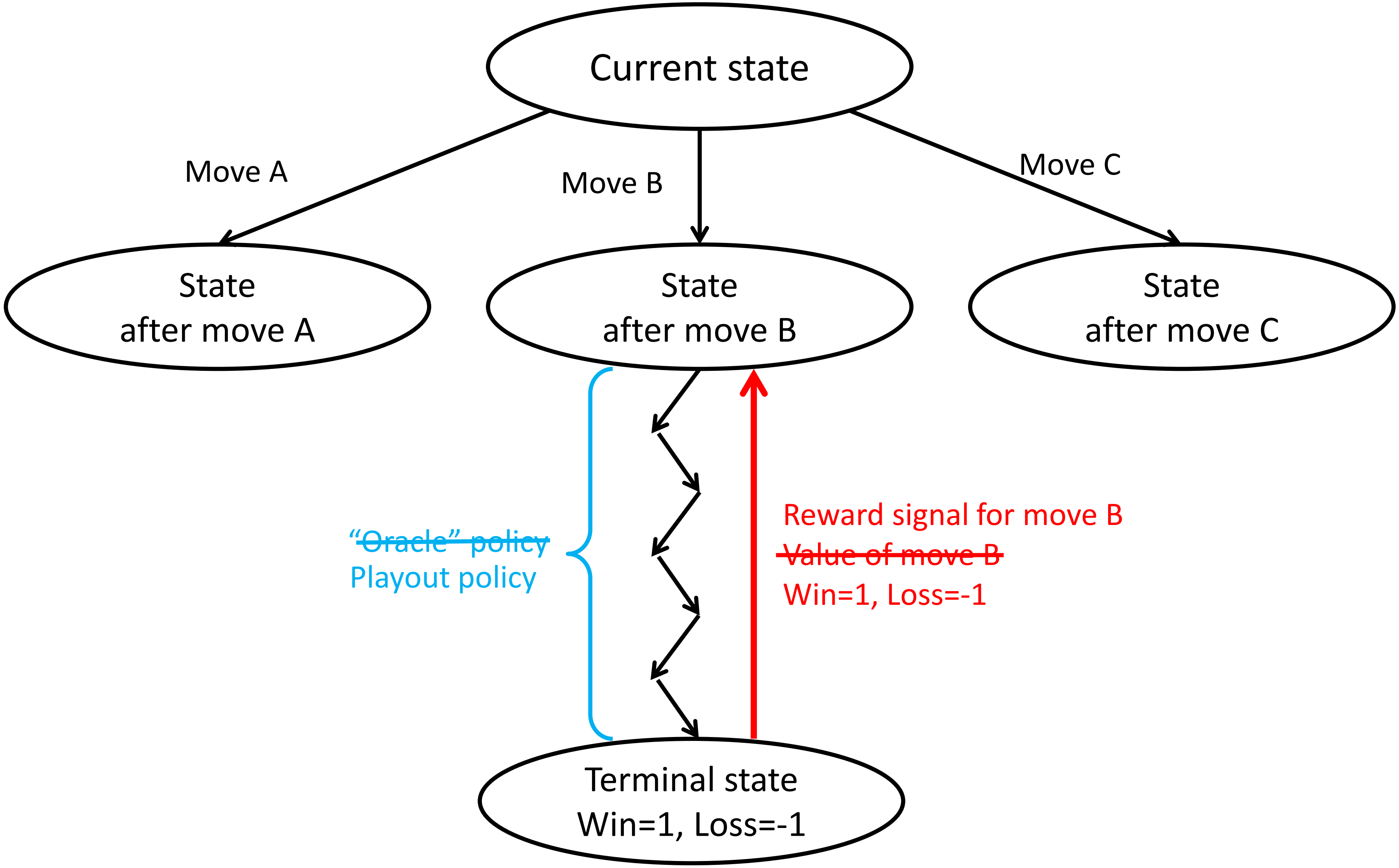
- The **value** of a rollout is the **value** of the terminal state it reaches (i.e. 1 for a win, -1 for a loss, 0 for a draw)
- Averaging gives the **expected value** of the initial state
- Higher expected value $=$ more chance of winning

Monte Carlo search

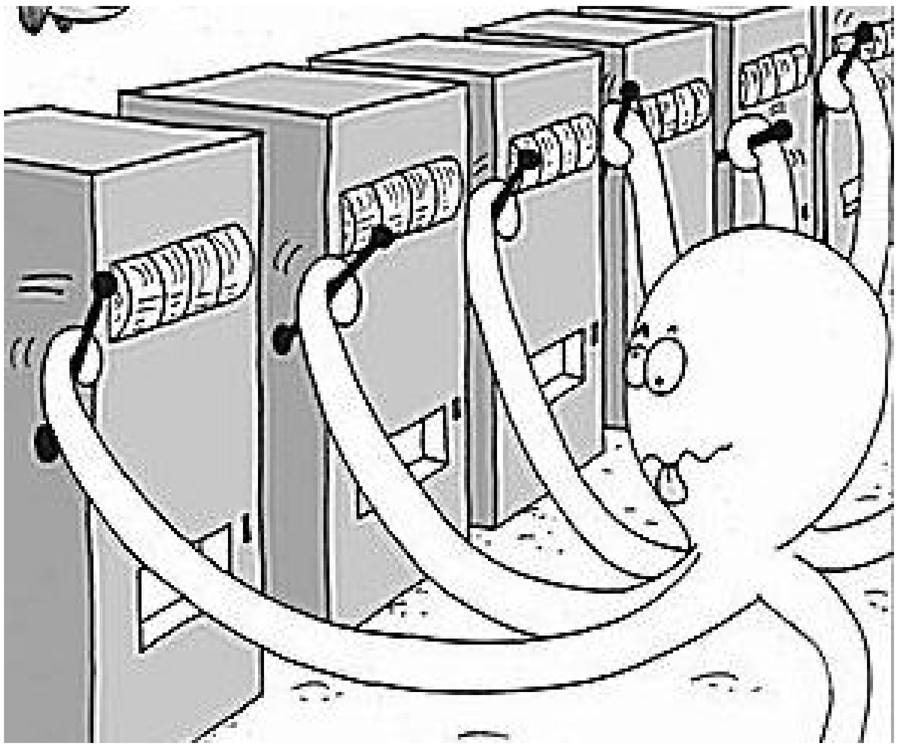
- **Flat Monte Carlo search**: 1-ply search with Monte Carlo evaluation
- How about minimax with $d > 1$ and Monte Carlo evaluation?
  - Minimax assumes the evaluation is **deterministic**, but Monte Carlo is not
  - Not commonly used, mainly because there's something better...

Monte Carlo Tree Search (MCTS)

The perfect 1-ply search



The multi-armed bandit problem



At each step pull one arm

Noisy/random reward signal

In order to:

Minimise regret

Maximise expected return

(Find the best arm)

Upper Confidence Bound (UCB1)

- Balance **exploitation** with **exploration**
- Select the arm that maximises

Total reward from this arm so far

Total number of trials so far

$$\underbrace{\frac{V_a}{n_a}}_{\text{Exploitation}} + k \underbrace{\sqrt{\frac{\log n}{n_a}}}_{\text{Exploration}}$$

Number of times this arm has been selected so far

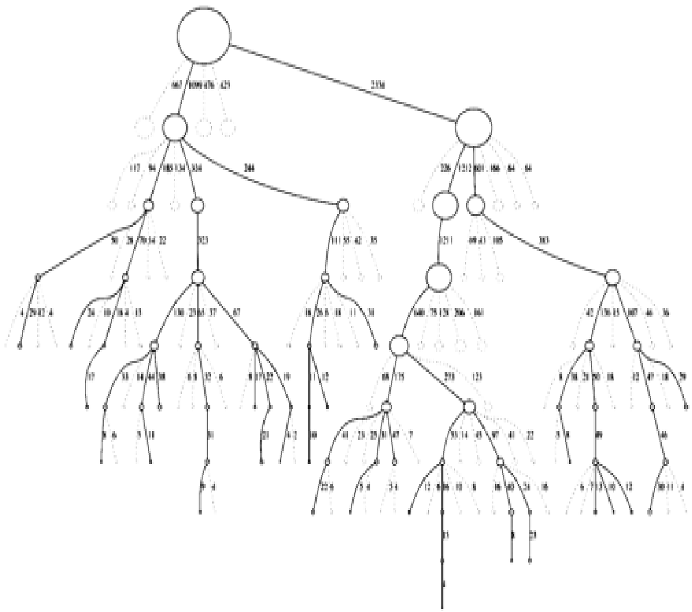
Tuning constant

P Auer, N Cesa-Bianchi, P Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*. 47(2):235-256, 2002.

UCB1 demo

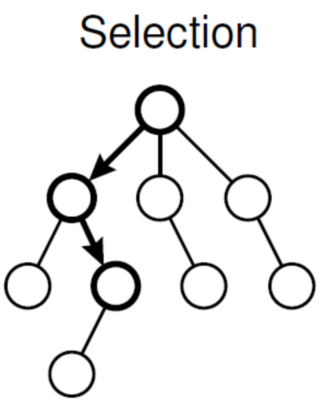
http://orangehelicopter.com/academic/bandits.html?ucb

Monte Carlo Tree Search (MCTS)



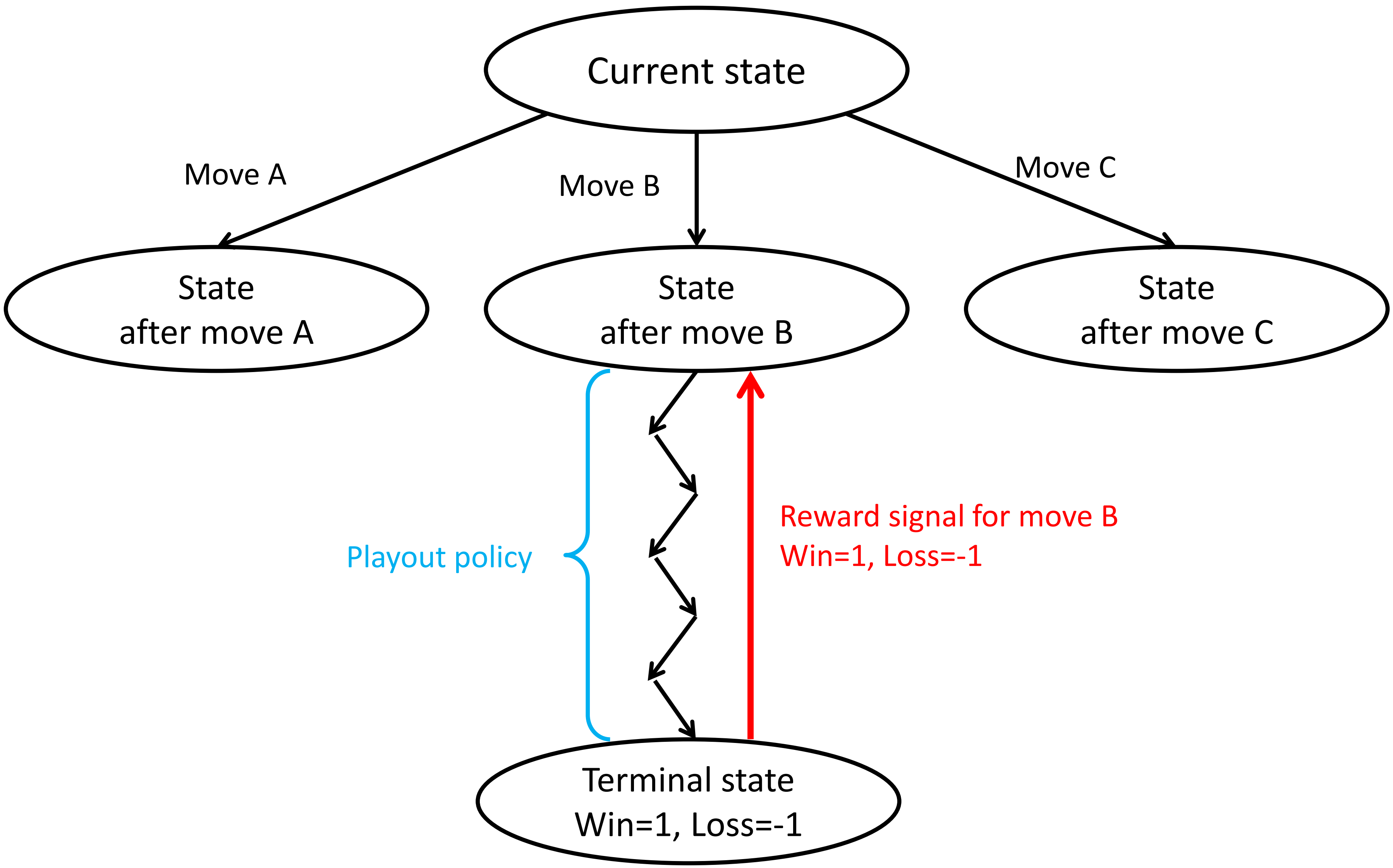
- Iteratively build a partial search tree, one node per iteration
- Monte Carlo evaluation (random playouts) for nonterminal states
- **Asymmetric**
  - balance exploitation with exploration
- **Anytime**
  - can do as many (or as few) iterations as we want
- **Aheuristic**
  - Monte Carlo evaluation requires only a forward model

Monte Carlo Tree Search (MCTS)

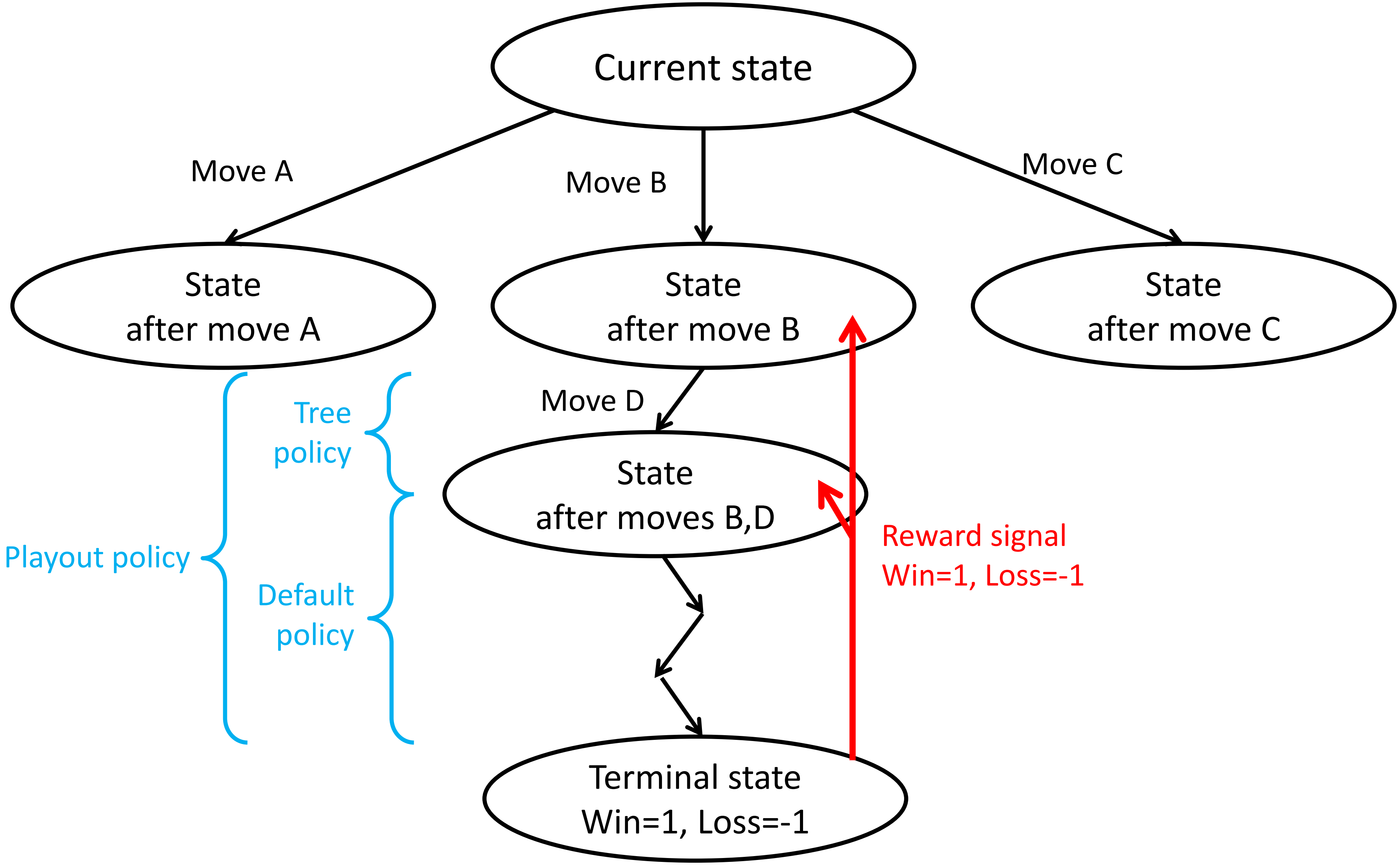


Choose a node with unexpanded children

Game Decisions



Monte Carlo Tree Search (MCTS)



Upper Confidence bound for Trees (UCT)

- Tree policy: UCB1 $\frac{V_a}{n_a} + k\sqrt{\frac{\log n}{n_a}}$

- Default policy: uniform random

L Kocsis, C Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, vol 4212 of *Lecture Notes in Artificial Intelligence,* 282-293, 2006.

Demo


MB
GAMES
Connect 4
The vertical strategy game!
7-Adult
2 players
Drop your counters into the slots. Try to outwit your opponent and be the first to get a row of 4. Not as easy as it looks!

Conclusion

- MCTS is a powerful **general-purpose** AI technique
  - Asymmetric, Anytime, Aheuristic
- MCTS has proven successful in several **challenging classes of games**
  - Games of imperfect information
  - Commercial mobile games
  - Real-time games
- It shows promise in **many other games and non-game applications**