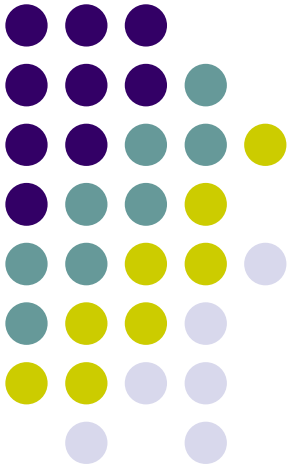
# Introduction to AI

## Chapter05

## Adversarial Search: Game Playing

Pengju Ren@IAIR



# Outline

- Types of Games
  - Formulation of games
- Perfect-Information Games
  - Minimax and Negamax search
  - α-β Pruning
  - Pruning more
  - Imperfect decision
- Stochastic Games
  - EXPECTIMINIMAX
  - Monte Carlo simulation
- Partially Observable Games
  - Nash equilibrium

# Types of Games

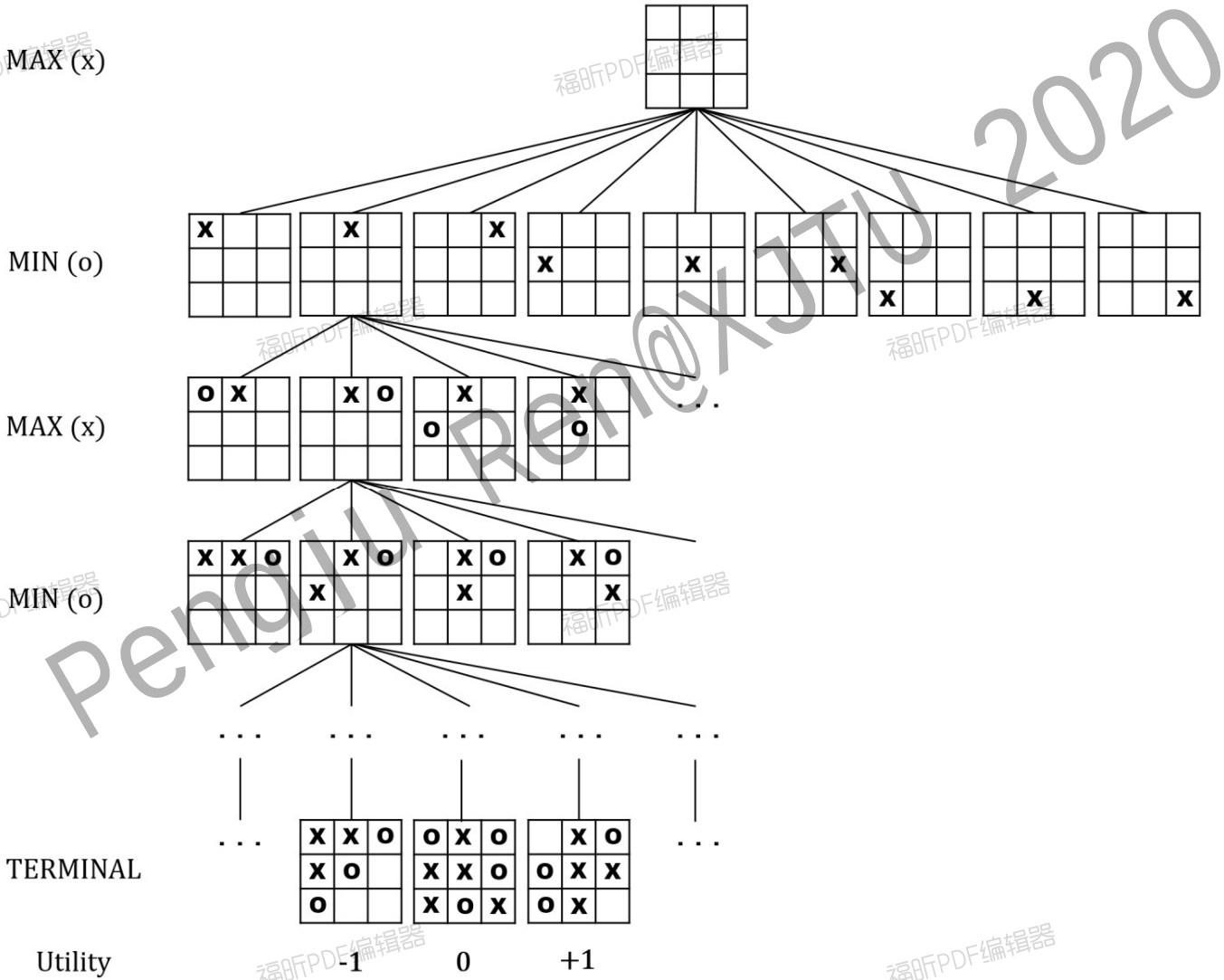
- **Adversarial search** considers **multi-agent** and **competitive** environments.
- **Game theory** consider both **competitive** and **cooperative** environments.
- Most common games are **deterministic**, **turn-taking**, **two-player**, **zero-sum** games with **perfect information**.
  - Let's focus on this type of games for a while until told otherwise.

| | Deterministic | Stochastic |
|---|---|---|
| **Perfect Information** | Chess, Checkers, Go, Othello | Backgammon, Monopoly |
| **Imperfect Information** | Battleships, Bingo | Bridge, Poker |

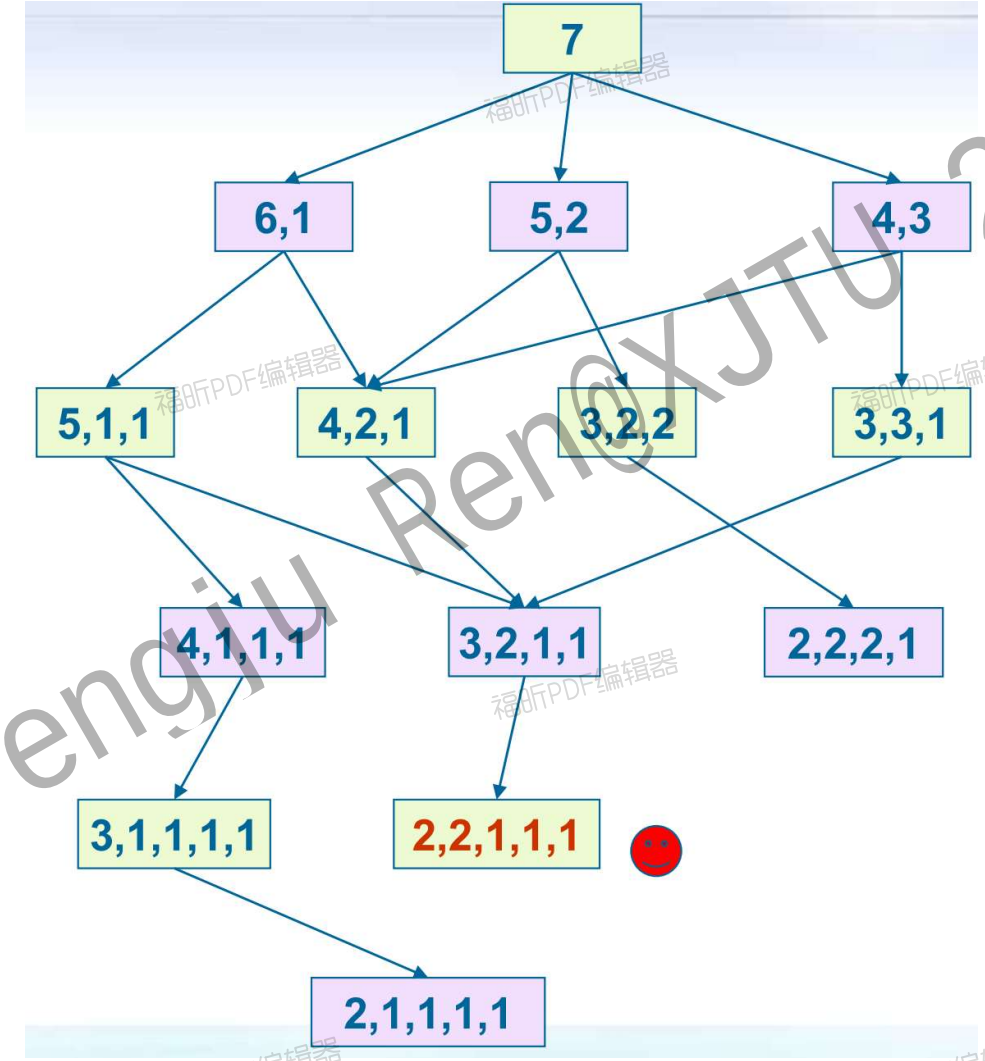
# Types of Games

- $S_0$: Initial state.
- *PLAYER(s)*: The player in state *s*.
- *ACTION(s)*: Returns the set of legal moves in state *s*.
- *RESULT(s,a)*: The transition model, which returns the resulting state of a move *a* in state *s*.
- *TERMINAL-TEST(s)*: TRUE/FALSE. States where the game has terminated are called terminal states.
- *UTILITY(s,p)*: A utility function (also called objective or payoff).
  - UTILITY(*s*) for 2-player, zero-sum games.
  - Reason: *UTILITY*(*s*,*p*1) = -*UTILITY* (*s*,*p*2)

# Game Tree

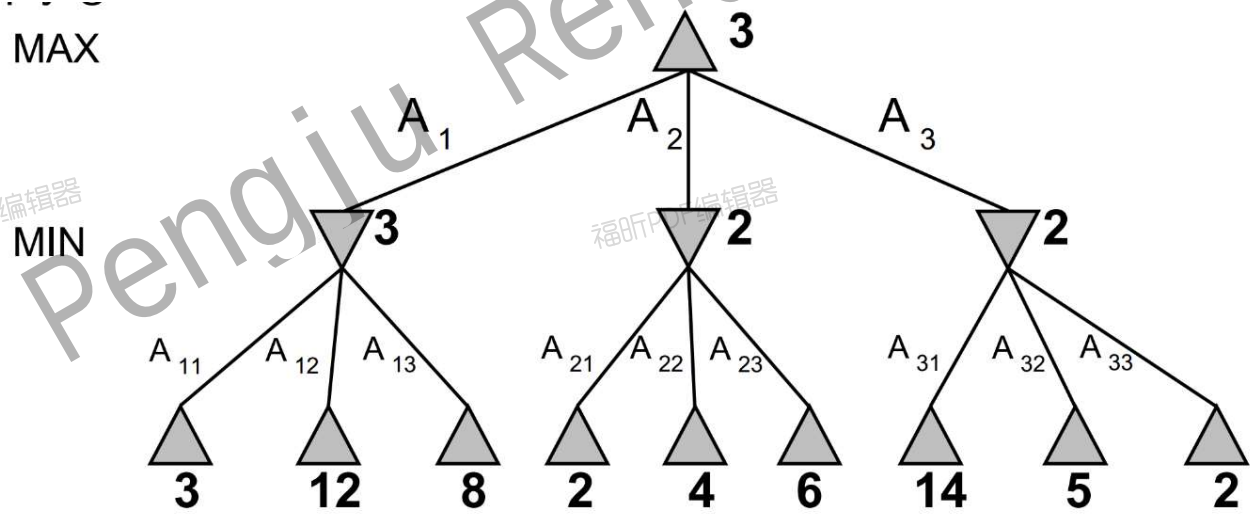


# E.g. Game Tree

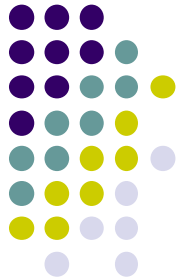


# Optimal Decision

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$



# MinMax Search



```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

---

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v
```

---

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

# Negamax Search

- $\min(a_0, \ldots, a_n) = -\max\{-a_0, \ldots, -a_n\}$

- **Such simplified implementation of *MINIMAX* is called *NEGAMAX*.**
- **Copying the whole state (line 5) is memory consuming. Practical implementation usually adopts *s* = *BACKTRACK*(s', *a*).**

**NEGAMAX(*s*)**

```
1  if TERMINAL-TEST(s)
2        return UTILITY(s, p)
3  result = −∞
4  for each a ∈ ACTION(s)
5        s' = RESULT(s, a)
6        result = max(result, −NEGAMAX(s'))
7  return result
```

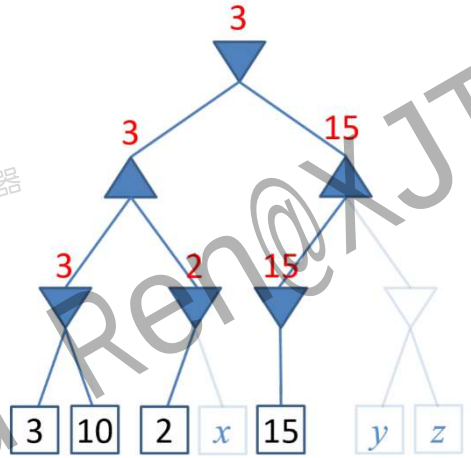
# Properties of Minimax(Negamax) Search

- **Completeness**: Yes, if tree is finite.
- **Optimality**: Yes, against an optimal opponent. Otherwise?
  - Risky moves that leads to complicated variations might be better to revert unfavored situations.
- **Time Complexity**: $O(b^m)$ .
- **Space Complexity**: $O(bm)$ (DFS); or *O(m)* if the algorithm generates actions one at a time.

For chess, $b \approx 35$, $m \approx 100$ => optimal decision is practically intractable.
Do we need to explore every path?

# α-β Pruning

- **Not every node needs to be evaluated.**

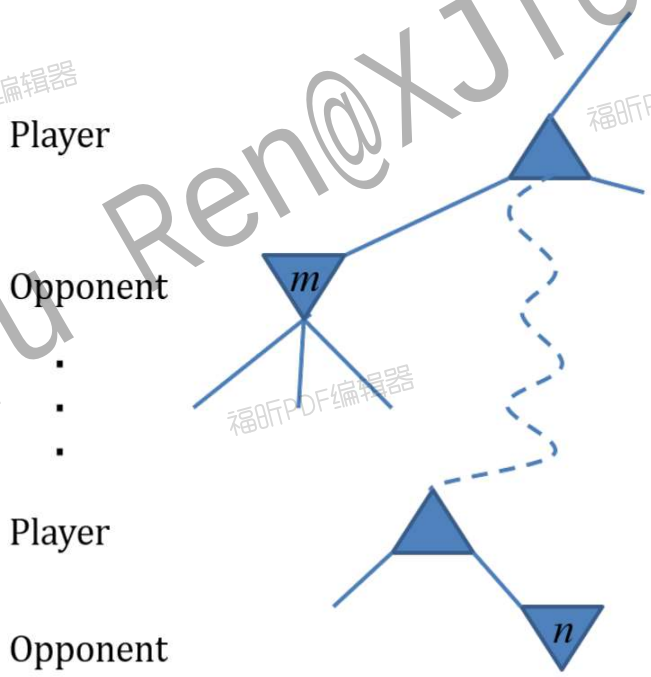



**The value of *root* is independent of two unknown nodes $x$ and $y$**

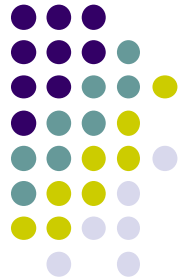
$$\begin{aligned}
\textsc{Minimax}(root) &= \min(\max(\min(3,10),\min(2,x)),\max(15,\min(y,z))) \\
&= \min(\max(3,\min(2,x)),\max(15,\min(y,z))) \\
&= \min(3,\max(15,\min(y,z))) \\
&= 3
\end{aligned}$$

# Main Idea of α-β Pruning

- If *m* is better than *n* for Player, *n* will never be reached in actual play.
- Once we have found enough about *n* to reach this conclusion, we can prune it.



# α-β Pruning



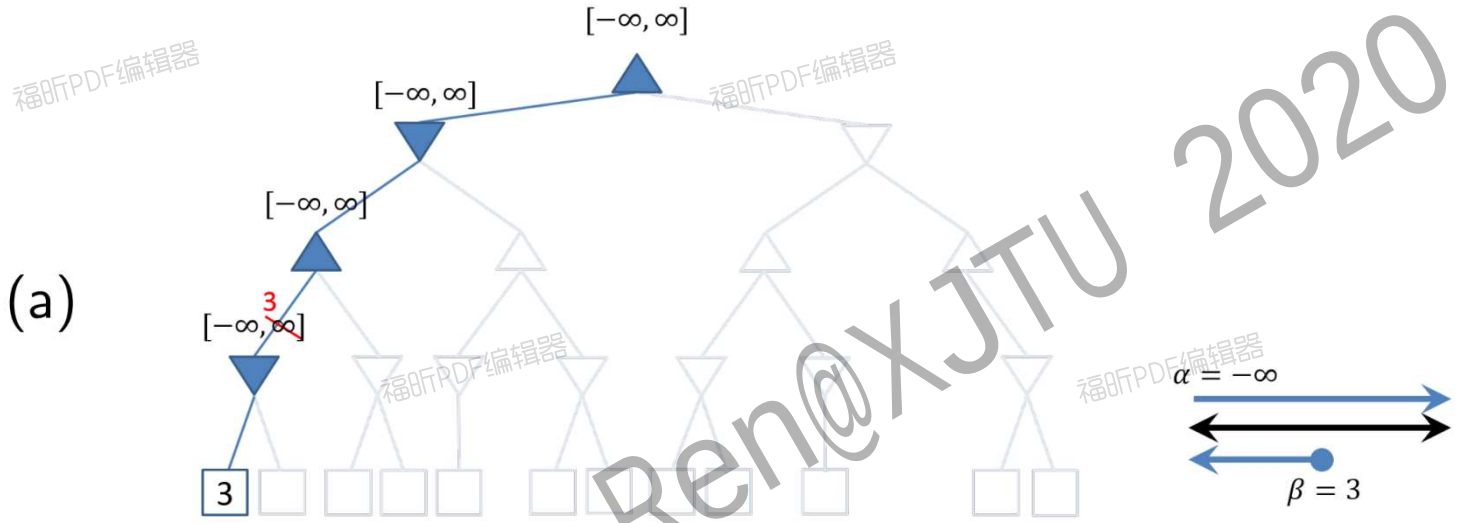
- Keeping α (maximum lower bound) for the maximum utility for player ***MAX***, initialized to -∞ .
- Keeping β (minimum upper bound) for the minimum utility for player ***MIN***, initialized to ∞ .
- Only the moves within the [α, β] window are expanded; otherwise its branches are pruned.
- The pruning does NOT compromise solution quality.

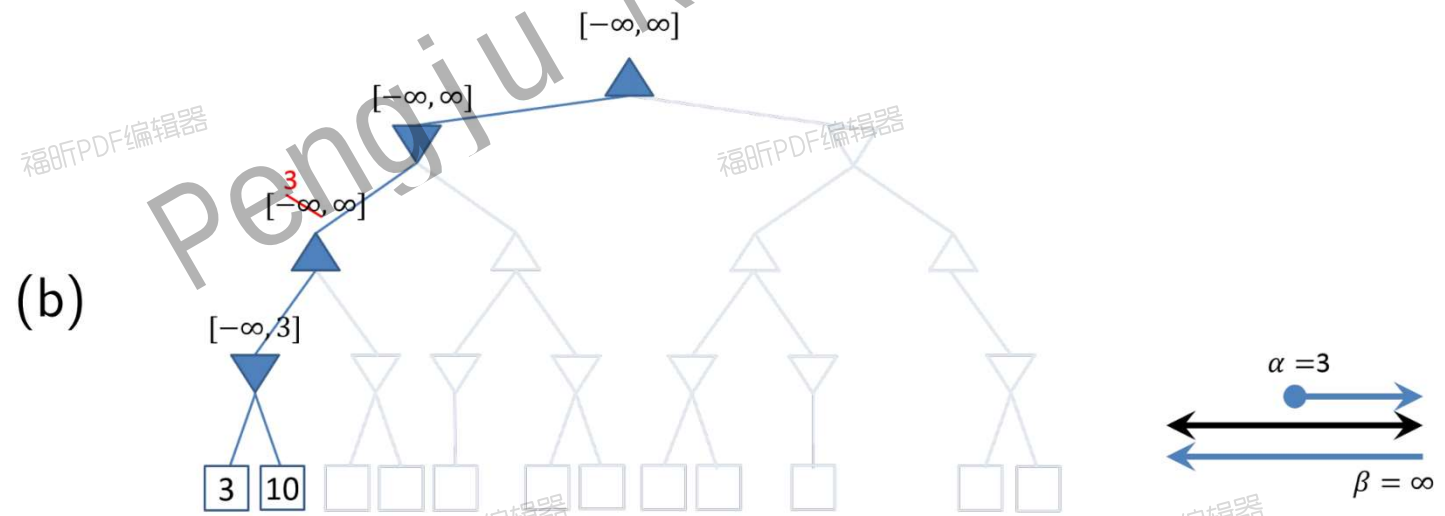
# Implementation of α-β Pruning

**AlphaBeta**$(s, \alpha, \beta)$

```
1  if Terminal-Test(s)
2      return Utility(s)
3  if Player(s) == MAX
4      v = −∞
5      for each a ∈ Action(s)
6          v = max(v, AlphaBeta(Result(s, a), α, β))
7          if v ≥ β return v
8          α = max(α, v)
9      return v
10 else
11     v = ∞
12     for each a ∈ Action(s)
13         v = min(v, AlphaBeta(Result(s, a), α, β))
14         if α ≥ v return v
15         β = min(β, v)
16     return v
```

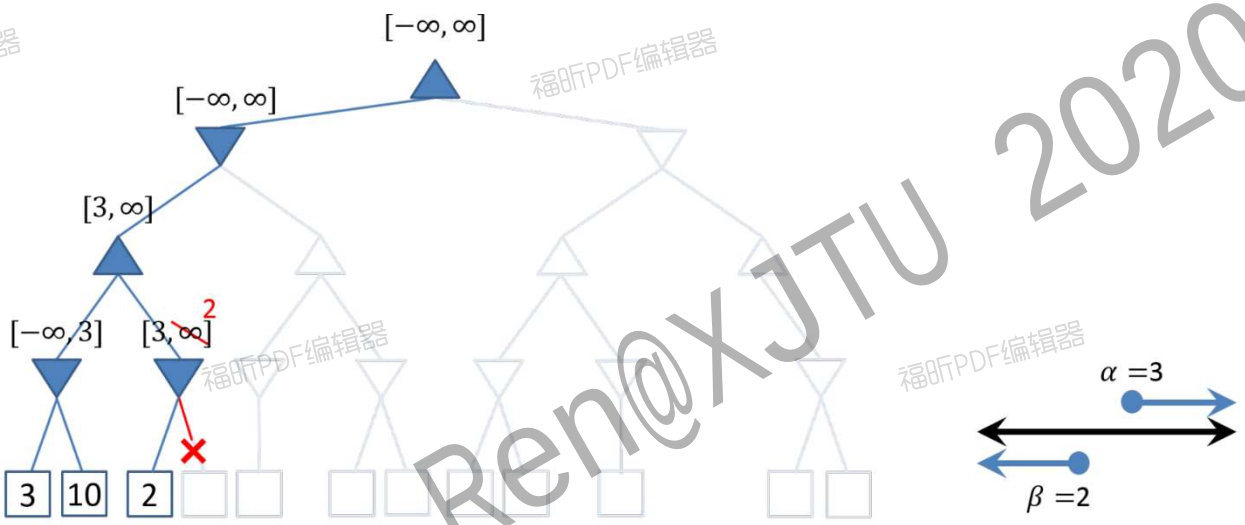
# α-β Pruning



# α-β Pruning

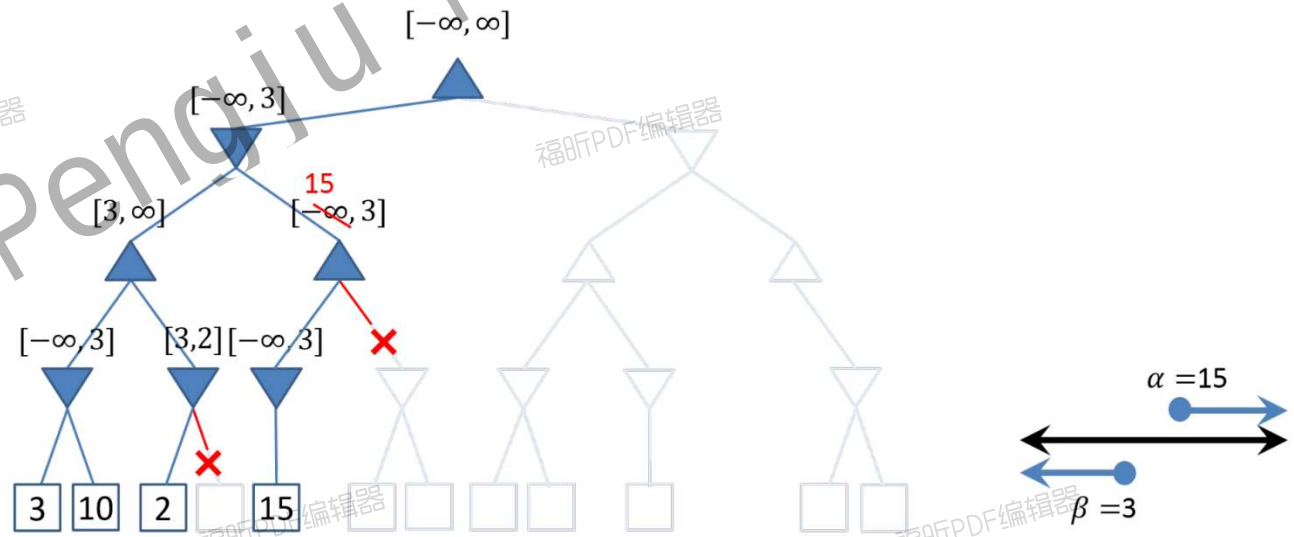
(c)





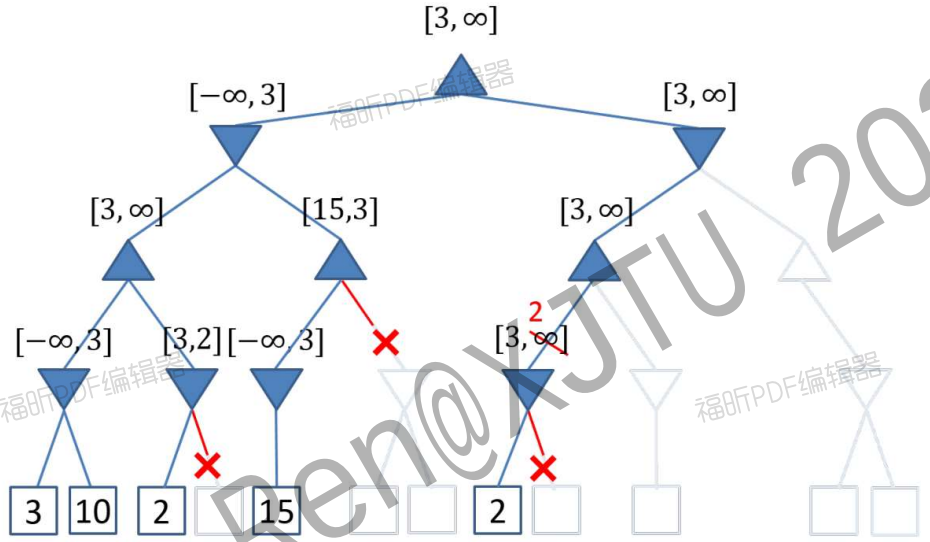


(d)

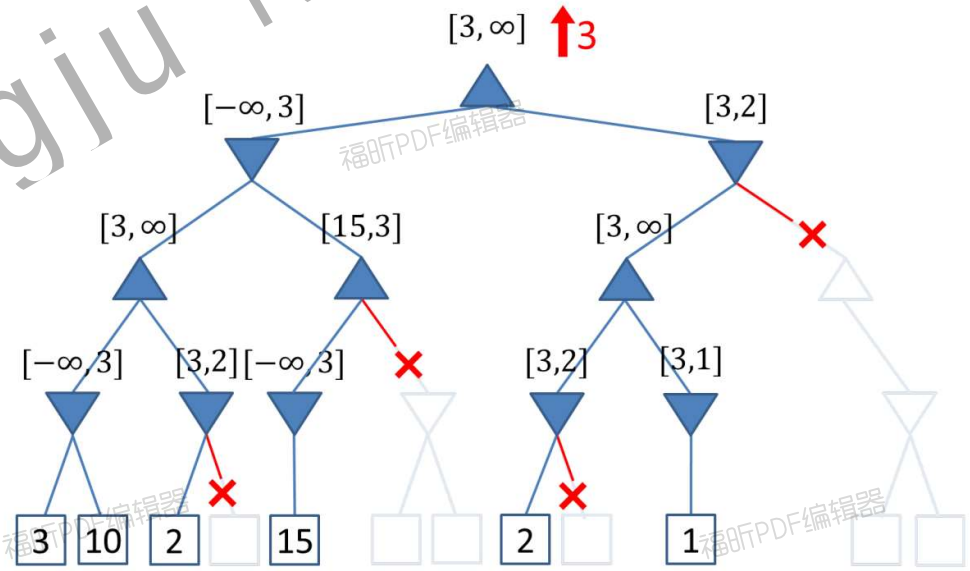


# α-β Pruning

(e)




(f)



# Implementation of α-β Pruning

- ***NEGAMAX + ALPHABATE = AB-NEGAMAX.***

**AB-NegaMax$(s, \alpha, \beta)$**

```
1  if Terminal-Test(s)
2        return Utility(s, p)
3  v = −∞
4  for each a ∈ Action(s)
5        s′ = Result(s, a)
6        v = max(v, −AB-NegaMax(s′, -β, −α))
7        if v ≥ β return v
8        α = max(α, v)
9  return v
```

# Efficiency of α-β Pruning

- Highly depends on the order of moves.
  - Good move ordering improves effectiveness of pruning.
- Worst case: no pruning -> $O(b^m)$.
- Best case: Always check the best move first.
  - Still need to check every move for the first player.
  - Only need to check one move for the second player.
  - $O(b \times 1 \times b \times 1 \ldots) = O(b^{\frac{m}{2}})$.
- Average case: $O(b^{\frac{3m}{4}})$
- Very simple ordering usually achieves $O(b^{\frac{m}{2}})$
  - Another good reason to adopt iterative deepening.
  - Reduce the effective branch factor to $\sqrt{b}$ .
  - Make the search twice as deep as before.

# Imperfect Real-Time Decisions

## Use CUTOFF-TEST instead of TERMINAL-TEST

- Can be as simple as depth limit.
- Can adopt quiescence search to conquer the horizon effect.
- Yet another good reason to adopt iterative deepening. Return the current best move when time's up.

## Use EVAL instead of UTILITY

- Usually maps state $s$ into feature space $f_i$
- Typically use linear combination of features: $\text{EVAL}(s) = \Sigma_i w_i f_i(s)$
- Need to fine tune weights for strong play.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

# Cutting off search

*MinimaxCutoff* is identical to *MinimaxValue* except

1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*

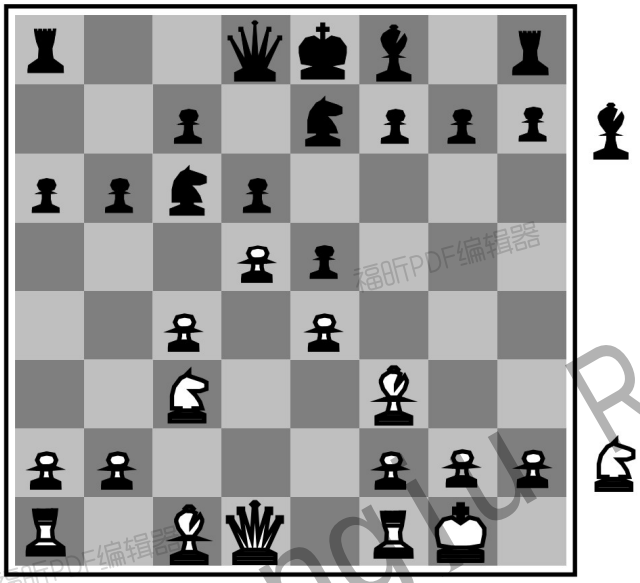
Does it work in practice?

$b^m = 10^6$, b=35 → m=4

4-ply lookahead is a hopeless chess player!

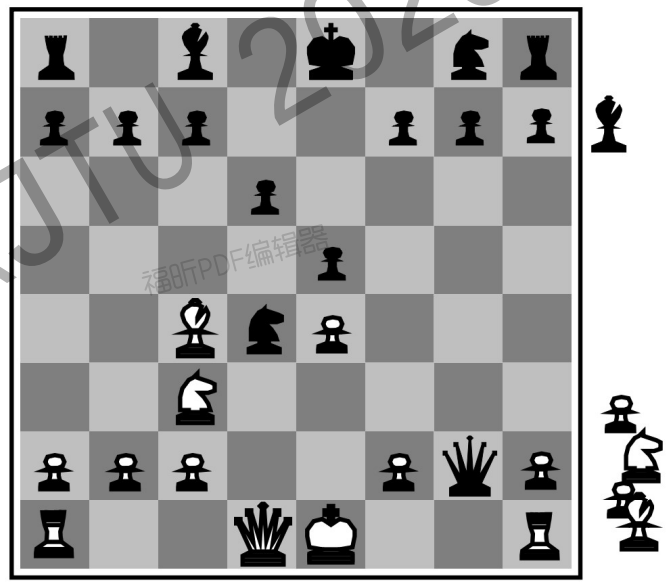
- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 12-ply ≈ Deep Blue, Kasparov

# Evaluation Function



Black to move

White slightly better

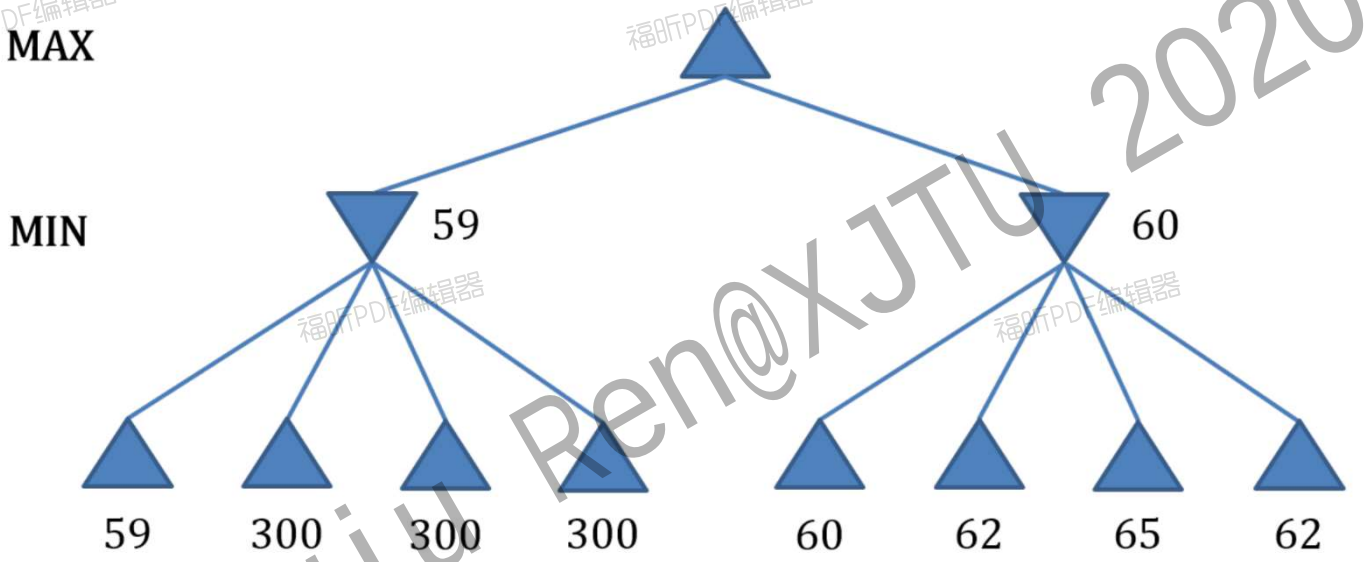


White to move

Black winning

$$EVAL(s) = \sum_i w_i f_i(s)$$

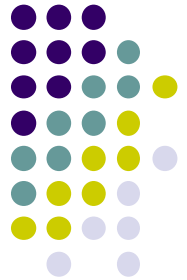
# Heuristic Where Minimax May Go Wrong






- ***MINIMAX*** **chooses the right branch.**
- ***EVAL*** **with an error with zero mean and standard deviation** $\sigma$**.**
- $\sigma = 3$**, the left branch is better 54.6% of the time.**
- $\sigma = 5$**, the left branch is better 64.4% of the time.**

# Forward Pruning



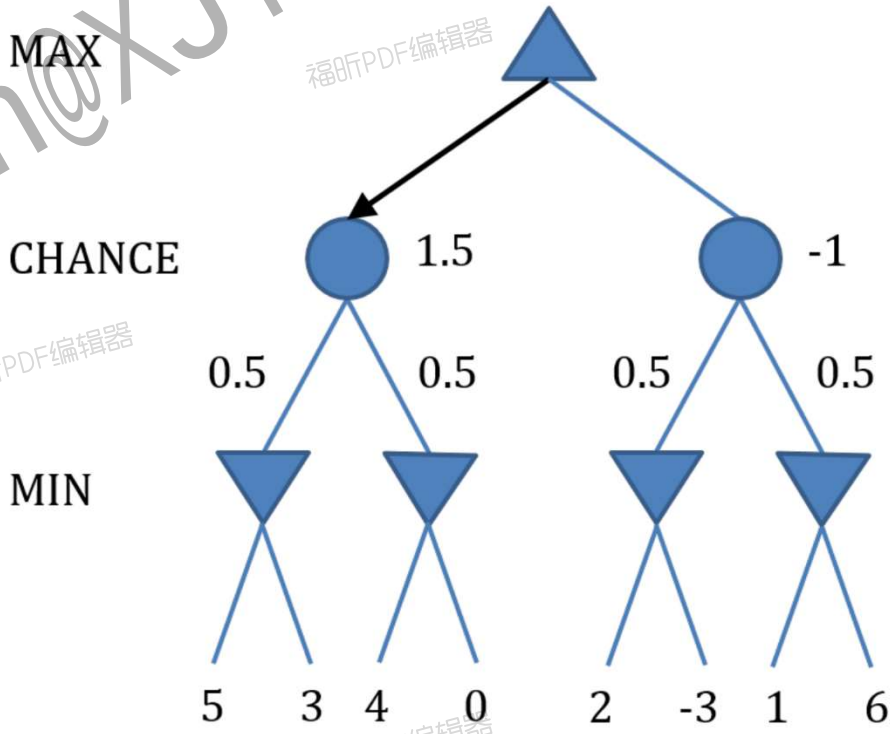
- Forward pruning does compromise solution quality (so is using *EVAL*)
  - Some moves are pruned immediately without further consideration.
- Beam search is one way to forward pruning. Dangerous since the best move might be pruned.
- *PROBCUT* uses the scores from previous searches to estimate the probability that a node is outside the [$\alpha$, $\beta$] window

# Search vs. Table Lookup

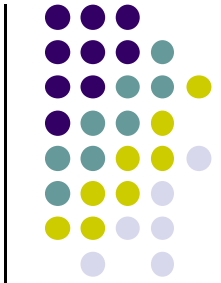
- For many games, deep search usually helps little *at the beginning*.
- Instead, fast table looking up, huge databases, and statistical analysis help more.
- Table lookup also helps a lot toward the *end of games*.
  - The KBNK (king, bishop, knight vs. king) lookup:
    462x62x61x2 = 3, 494, 568 possibilities.
  - Bourzutschky (2006) solved all pawn-less 6-piece endgames and some 7-piece endgames.
    A KQNKRBN endgame requires 517 moves!
- Finally, early exchange favors computers than humans -> deeper search and more probable falls in lookup.

# Stochastic Games

- Introduce *CHANCE* nodes into the minimax tree.
- Instead of searching for maximum/minimum values, we now search for expected maximum/minimum values.

# EXPECTIMINIMAX

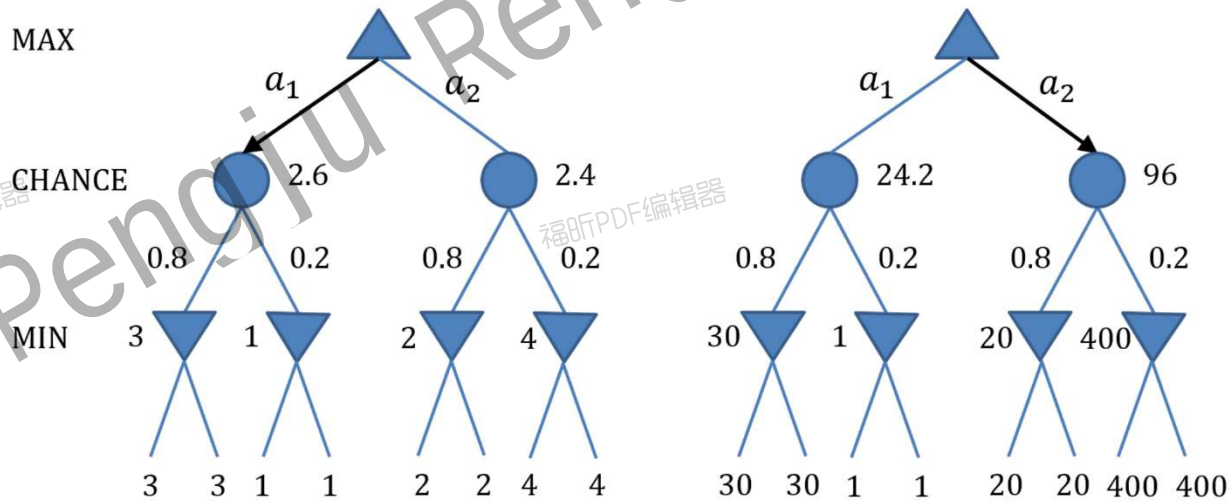


- ***EXPECTIMINIMAX*** **gives perfect play. (in what sense?)**
- **Similar to *MINIMAX*, except we must also handle chance nodes.**

$$\textsc{ExpectiMiniMax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{ExpectiMiniMax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) == \textsc{max} \\ \min_a \textsc{ExpectiMiniMax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) == \textsc{min} \\ \Sigma_a P(a)\textsc{ExpectiMiniMax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) == \textsc{chance} \end{cases}$$

# Sensitivity to Heuristic

- As mentioned before, we rarely can actually use *UTILITY* in *EXPECTMINIMAX*.
- Instead, we use heuristic.
- However, unlike in *MINIMAX*, actual values of heuristic matter now. (In *MINIMAX*, only relative order matters.)




$\{1, 2, 3, 4\} \rightarrow \{1, 20, 30, 400\}$

# Performance of *EXPECTIMINIMAX*

- $\alpha$-$\beta$ pruning now does not apply to *MAX/MIN* nodes (why?).
- $\alpha$-$\beta$ pruning now still applies to *CHANCE* nodes (why?).
- Time Complexity: $O(b^m) \rightarrow O(b^m n^m)$ , where $n$ is the number of distinct dice rolls.
- Causes *EXPECTIMINIMAX* impractical in many cases.
- Solution: Instead of checking every *MAX/MIN* node, adopts Monte Carlo simulation at chance nodes.
- Using random dice rolls to check only a certain number (decided by quality/time limit) of paths.

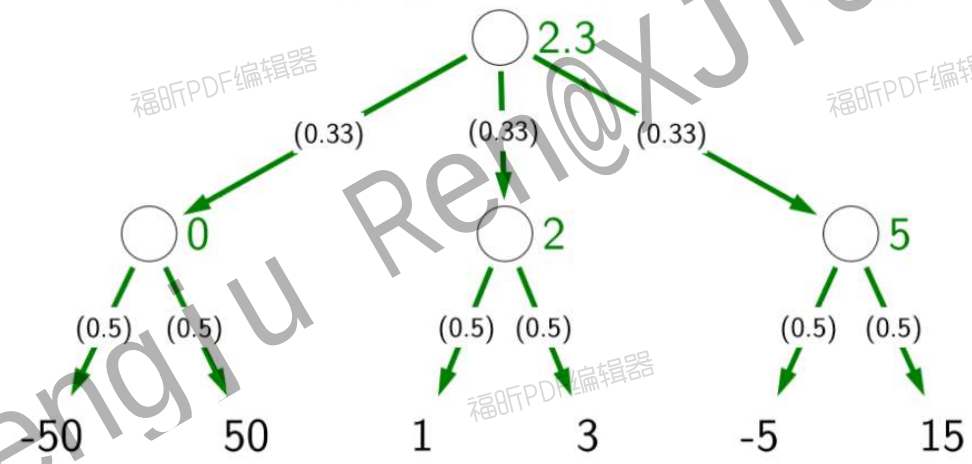
$$\operatorname{argmax}_a \sum_s p(s) MINMAX(RESULT(s,a))$$

Monte Carlo simulation $$\operatorname{argmax}_a \frac{1}{N} \sum_{i=1}^{N} MINIMAX(RESULT(s_i,a))$$

# Monte Carlo method

**Approach:**

- **Simulate n random paths by applying the policies**
- **Average the utilities of the n paths**






Eval = 1/10*[(1) + (3) + (50) + (50) + (50) + (−50) + (−50) + (50) + (15) + (−5)] = 11.4

# Monte Carlo method

$$\operatorname{argmax}_a \sum_s p(s) MINMAX(RESULT(s,a))$$

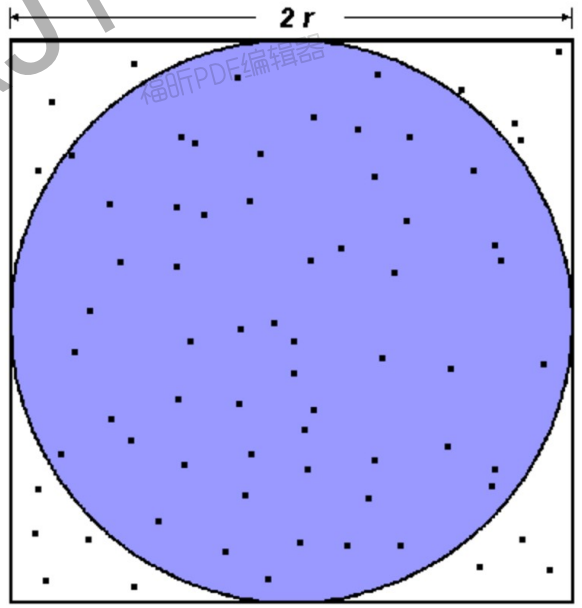
**Monte Carlo simulation**

$$\operatorname{argmax}_a \frac{1}{N} \sum_{i=1}^{N} MINIMAX(RESULT(s_i,a))$$

```
npoints = 10000
circle_count = 0

do j = 1,npoints
  generate 2 random numbers between 0 and 1
  xcoordinate = random1
  ycoordinate = random2
  if (xcoordinate, ycoordinate) inside circle
  then circle_count = circle_count + 1
end do

PI = 4.0*circle_count/npoints
```




$A_S = (2r)^2 = 4r^2$

$A_C = \pi r^2$

$\pi = 4 \times \frac{A_C}{A_S}$

**Monte Carlo methods rely on repeated random sampling to obtain numerical results.**

# Las Vegas Method





**Monte Carlo Method：More sampling, more closer to the optimal solution**
**Las Vegas Method：More sampling, higher opportunity to optimal solution.**

# Partially Observable Games


- Different from stochastic games, unobservable parts are usually controlled by opponents, not probability.
- Examples: Cards held by other player in bridge, folded cards in poker, fogs in star craft.
- Different strategies may be applied and may all considered optimum against different opponents.
- If equilibrium exists, it’s usually considered as optimum strategy.

# Nash Equilibrium

- **By John Nash — check out "A Beautiful Mind (2001)" if you want an informal introduction to him.**

**Information Definition**

A set of strategies is a Nash equilibrium if no player can do better by unilaterally changing his or her strategy.

- **Prisoners' dilemma**

| | B stays silent | B confesses |
|---|---|---|
| A stays silent | Each serves 1 yr | A: 5 yrs; B: free |
| A confesses | A: free; B: 5 yrs | Each serves 3 yrs |

- **Read Chapter 17 if you want to know more.**

# Summary

- ***MINIMAX* search** for zero-sum two-player games.
- **Pruning** techniques enable to search deeper.
- Due to time limit, **heuristics** are used to evaluate the "goodness" for a player.
- For **stochastic games**, we need to introduce **chance nodes** and search for expected maximum/minimum values.
- For stochastic games, $\alpha$-$\beta$ pruning is much less efficient, **Monte Carlo simulations** are often adopted to speed up the search.
- With **limited observation**, optimality is usually not well defined. If **equilibrium** exists, strategies in equilibrium are often considered optimum.