CSCI 561 - Foundation for Artificial Intelligence

06 -- Game Playing

Professor Wei-Min Shen
University of Southern California

Game Playing

- **Game Playing**
  - The minimax algorithm
  - Resource limitations
  - alpha-beta pruning
  - Elements of chance

What kind of games?

- **Abstraction**: To describe a game we must capture every relevant aspect of the game. Such as:
  - Chess, Tic-Tac-Toe, others
- **Accessible environments:** Such games are characterized by perfect information and completely observable
- **Search:** game-playing then consists of a search through possible game positions
- **Unpredictable opponent:** introduces **uncertainty** thus game-playing must deal with **contingency problems**

2. Searching for the Next Move

- **Complexity:** many games have a huge search space
  - **Chess:** $b = 35, m=100 \Rightarrow nodes = 35^{100}$
    if each node takes about 1 ns to explore
    then each move will take about ***10^{50}* millennia**
    to calculate.

- **Resource (e.g., time, memory) limit:** optimal solution not feasible/possible, thus must approximate

1. **Pruning:** makes the search more efficient by discarding portions of the search tree that cannot improve quality result
2. **Evaluation functions:** heuristics to evaluate utility of a state without exhaustive search

complete

3. Two-Player Games

- A game formulated as a search problem:

• Initial state:	board position and turn
• Operators:	definition of legal moves
• Terminal state:	conditions for when game is over
• Utility function:	a numeric value that describes the outcome of the game. E.g., -1, 0, 1 for loss, draw, win. (AKA **payoff function**)

4. Game vs. Search Problem

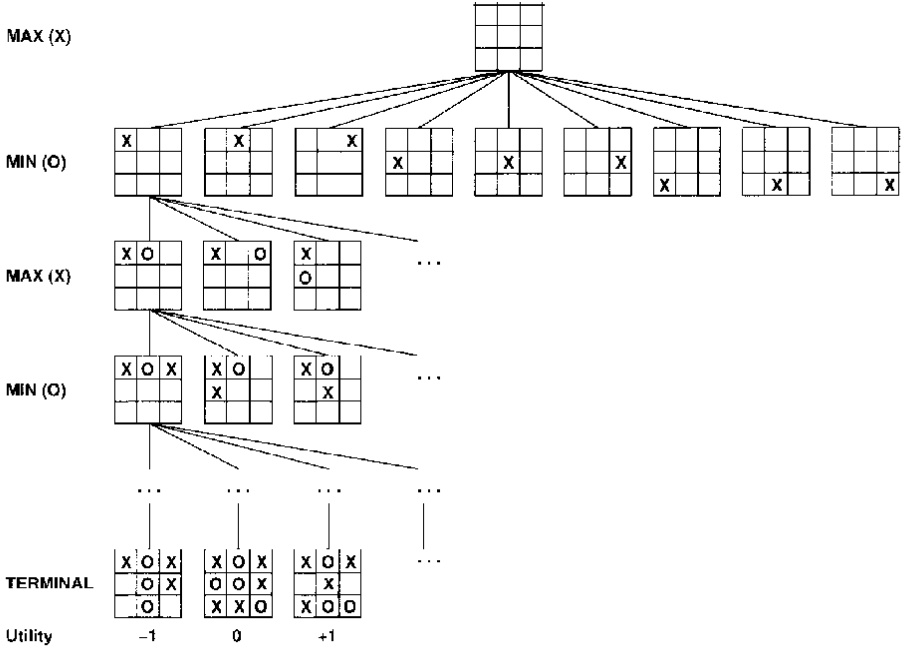
"Unpredictable" opponent ⇒ solution is a contingency plan

Time limits ⇒ unlikely to find goal, must approximate

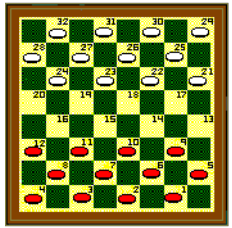
Plan of attack:

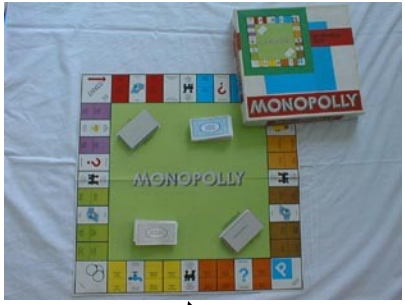
- algorithm for perfect play (Von Neumann, 1944)
- finite horizon, approximate evaluation (Zuse, 1945; Shannon, 1950; Samuel, 1952–57)
- pruning to reduce costs (McCarthy, 1956)

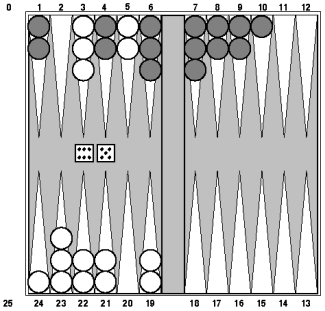
Example: Tic-Tac-Toe



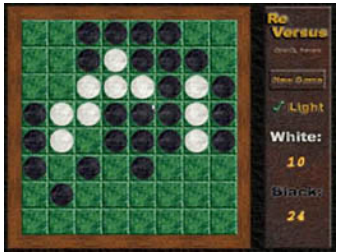
Type of Games

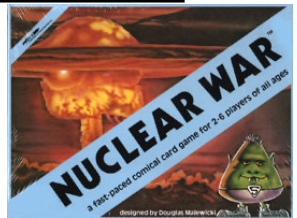






	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war



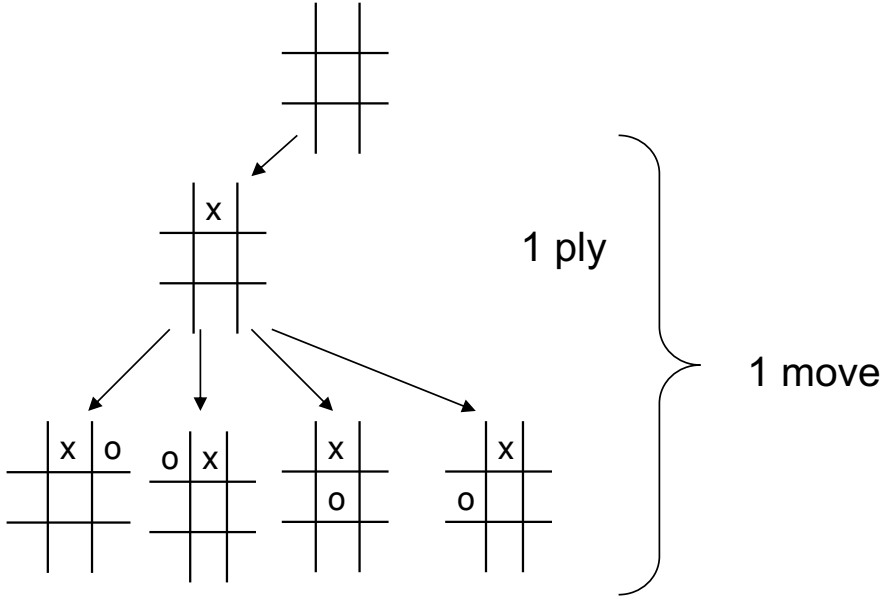


The Minimax Algorithm

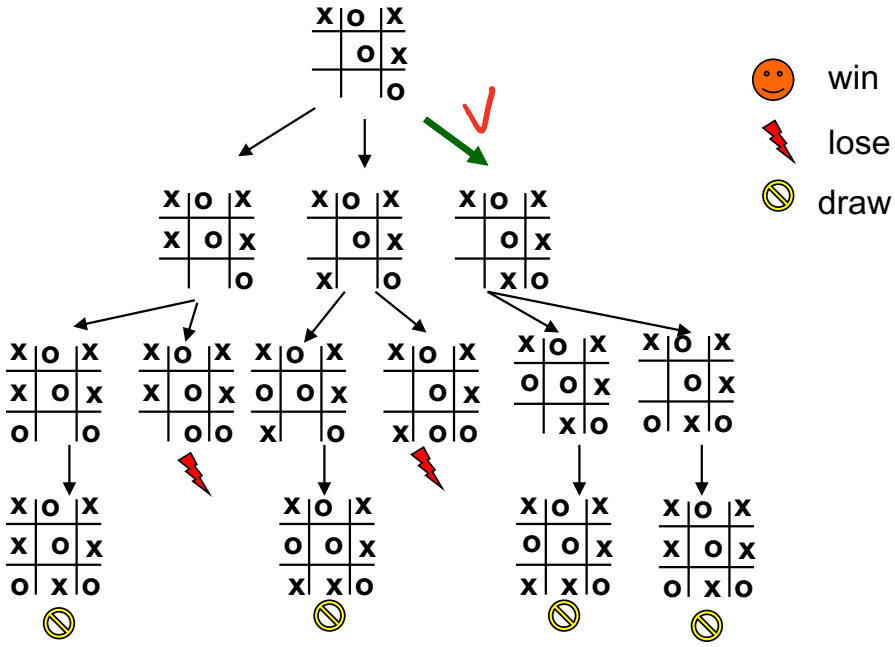
- Perfect play for deterministic environments with perfect information
- **Basic idea:** choose move with highest minimax value
  = best achievable payoff against best play
- **Algorithm:**
  1. Generate game tree completely
  2. Determine utility of each terminal state
  3. Propagate the utility values upward in the three by applying MIN and MAX operators on the nodes in the current level
  4. At the root node use minimax decision to select the move with the max (of the min) utility value

- Steps 2 and 3 in the algorithm assume that the opponent will play perfectly.

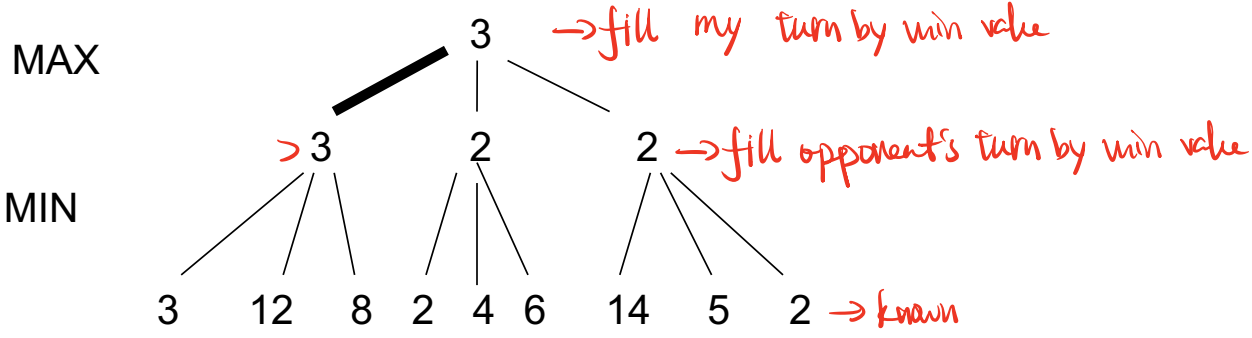
Generate Game Tree



What is a Good Move?

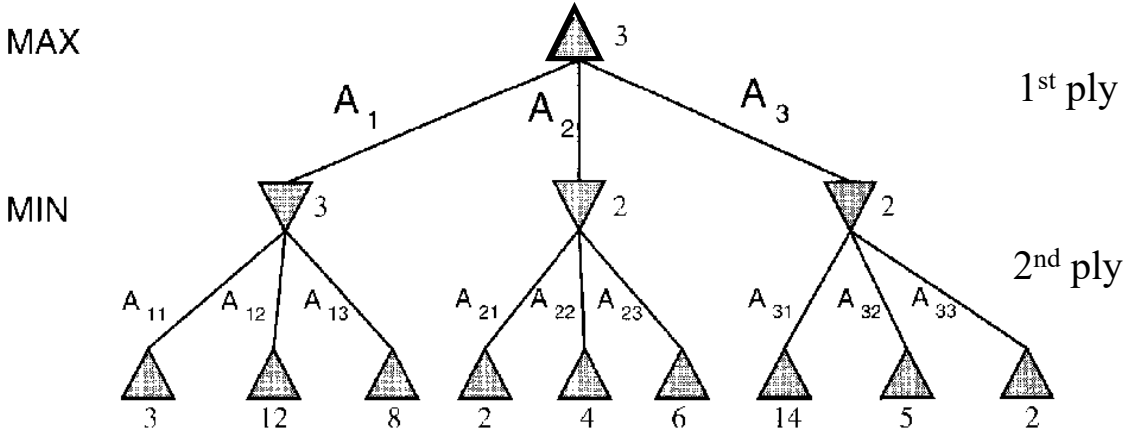


4. Minimax



- Minimize opponent's chance
- Maximize your chance

minimax = maximum of the minimum



Minimax: Recursive implementation

```
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 MIN: return min-value(state)
```

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

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    retur
```

Complete: Yes, for finite state-space
Optimal: Yes

Time complexity: $O(b^m)$
Space complexity: O(bm) (= DFS
Does not keep all nodes in memory.)

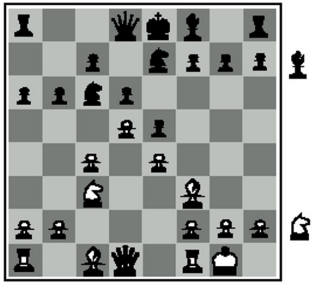
Move Evaluation without Complete Search

- Complete search is too complex and impractical

- **Evaluation function:** evaluates value of state using heuristics and cuts off search

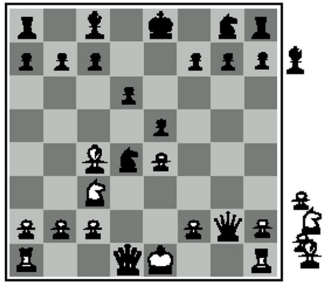
- **New MINIMAX:**
  - CUTOFF-TEST: cutoff test to replace the termination condition (e.g., deadline, depth-limit, etc.)
  - EVAL: evaluation function to replace utility function (e.g., number of chess pieces taken)

Evaluation Functions



Black to move

White slightly better



White to move

Black winning

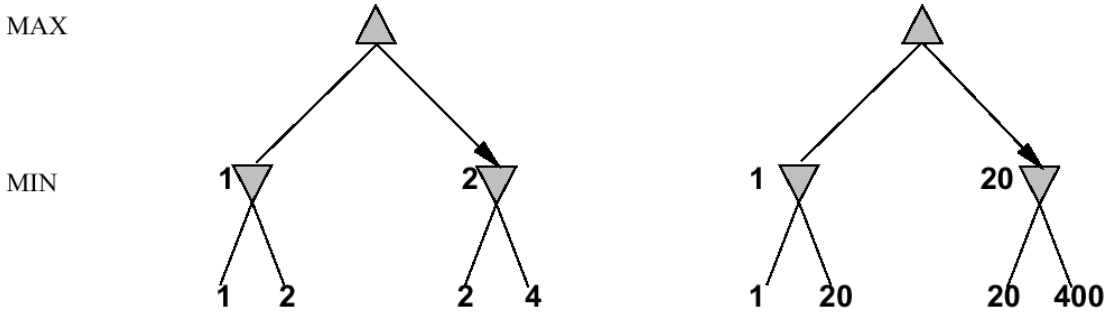
- **Weighted linear evaluation function:** to combine *n* heuristics

$$f = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

E.g, w's could be the values of pieces (1 for prawn, 3 for bishop etc.)
f's could be the number of type of pieces on the board

3. Note: Exact Values do not Matter (Relative Orders are important)

not apply for probability involved



Behaviour is preserved under any *monotonic* transformation of EVAL

Only the order matters:
payoff in deterministic games acts as an *ordinal utility* function

Minimax with Cutoff: Viable Algorithm?

MinimaxCutoff is identical to MinimaxValue except

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

Does it work in practice?

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

4-ply lookahead is a hopeless chess player!

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

Assume we have 100 seconds, evaluate 10^4 nodes/s; can evaluate 10^6 nodes/move

Reduce Search: α-β pruning for search cutoff

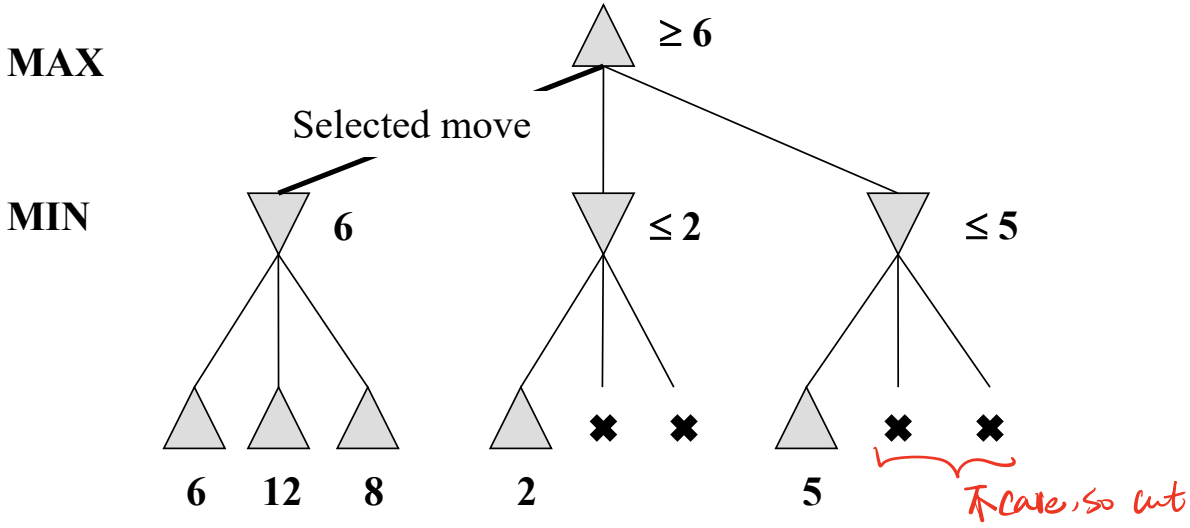
- **Pruning:** eliminating a branch of the search tree from consideration without exhaustive examination of each node

based on minmax

- **α-β pruning:** the basic idea is to prune portions of the search tree that cannot improve the utility value of the max or min node, by just considering the values of nodes seen so far.

- Does it work? Yes, in roughly cuts the branching factor from b to $\sqrt{b}$ resulting in double as far look-ahead than pure minimax

α-β pruning: example

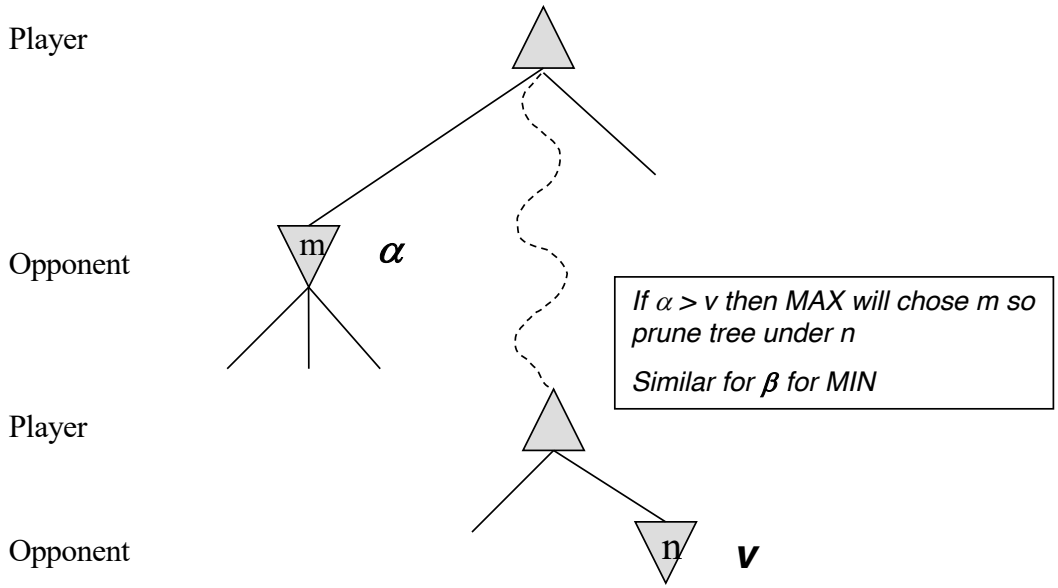


Interactive demo:
https://www.yosenspace.com/posts/computer-science-game-trees.html

α-β pruning: general principle

3.

α, at least I can get
β, at most opponent will give



4.

Properties of α-β

Pruning *does not* affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity = $O(b^{m/2})$

$\Rightarrow$ *doubles* depth of search

$\Rightarrow$ can easily reach depth 8 and play good chess

A simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*)

5.

α: MAX's best option on path to root
β: MIN's best option on path to root

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

cutting

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

cutting

名称来自于计算过程传递的两个边界，根据边界限制不必要的可能的解决方案集。其中，α表示**目前**所有可能解中最大下界，β是**目前**所有可能解中最小上界。

因此，如果搜索树上的一个节点被考虑作为最优解路上的节点（或者说是这个节点被认为是有必要进行搜索的节点），那么它一定满足以下条件（N是当前节点的估价值）：

$$\alpha \leq N \leq \beta$$

在求解过程中α会逐渐逼近β。如果对于某一个节点，出现了$\alpha > \beta$的情况，那么此节点必不会产生最优解，也就不必继续扩展（或生成子节点），从而完成剪枝。

6. More on the α-β algorithm

- Same basic idea as minimax, but prune (cut away) branches of the tree that we know will not contain the solution.

- We know a branch will not contain a solution once we know a better outcome has already been discovered in a previously explored branch.

Remember: Minimax: Recursive implementation

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

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

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

Complete: Yes, for finite state-space
Optimal: Yes

Time complexity: $O(b^m)$
Space complexity: $O(bm)$ (= DFS
Does not keep all nodes in memory.)

The α-β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v          cutting
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
    if v ≤ α then return v          cutting
    β ← MIN(β, v)
  return v
```

More on the α-β algorithm

- Same basic idea as minimax, but prune (cut away) branches of the tree that we know will not contain the solution.

- Because minimax is depth-first, let's consider nodes along a given path in the tree. Then, as we go along this path, we keep track of:
  - α : Best choice so far for MAX
  - β : Best choice so far for MIN

The α-β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
```

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

```
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Note: ***α and β*** are both Local variables. At the Start of the algorithm, We initialize them to ***α = -∞ and β = +∞***

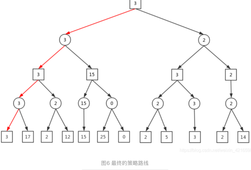


图6 最终的策略路线

(当然，如果后手不是进行最优决策的话，模局的走向就不一定是这样的了，先手也可以尝试走有风险、但是收益更高的局面，这就不在我们的讨论范围之内了，参考强化学习部分)

可以看到，如果按照MinMax算法来进行决策的话，需要的计算量是随着向后看的步数的增加而呈指数级增长的，但是，这些状态中其实是包含很多不必要的状态的，所以我们可以进行剪枝，

$2^{\alpha-\beta}$剪枝

名称来自于计算过程传递的两个边界，根据边界限制不必要的可能的解决方案集，其中，α表示目前所有可能解中最大下界，β是目前所有可能解中最小上界，

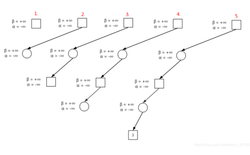
因此，如果搜索树上的一个节点被考虑作为最优解路上的节点（或者说是这个节点被认为是有必要进行搜索的节点），那么它一定满足以下条件（N是当前节点的估价值）：

$\alpha \leq N \leq \beta$

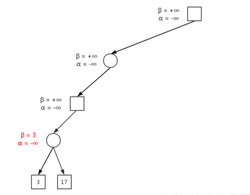
在求解过程中α会逐渐逼近β，如果对于某一个节点，出现了$\alpha > \beta$的情况，那么此节点必不会产生最优解，也就不必继续扩展（或生成子节点），从而完成剪枝，

以上图的情况进行说明：

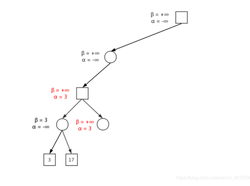
（1）从根节点开始向下搜索，到第四步，红色序号代表顺序，



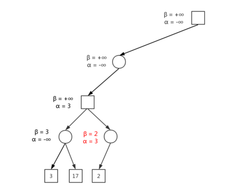
当搜索到了第一个叶子节点的时候，发现它的权值是3，且父节点是Min节点，又因为父节点的最小上界β > 3，所以更新父节点，令β = 3，（因为父节点要取最小值，这个最小值不会比3更大，所以我们更新其上界）然后继续向下搜索，



因为17比3大，所以这个节点我们可以无视掉（没啥用），我们已经搜索完了当前这个Min节点的所有孩子，所以我们返回它的节点值给它的父节点（Max节点），尝试更新父节点的α 值，（因为这是一个Max节点，他的孩子的估价值和α 、β 值已经确定了，所以父节点取值范围的下界也需要被更新），此处更新父节点，令α = 3，然后继续进行搜索，注意新生成的子节点的α 、β 值继承自父节点，

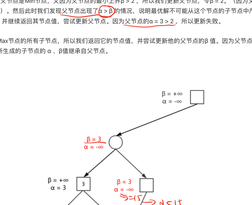


继续搜索，至叶子节点：

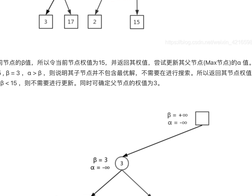


我们发现它的权值是2，并且它的父节点是Min节点，又因为父节点的最小上界β > 2，所以我们更新父节点，令β = 2，（因为父节点要取最小值，这个最小值不会比2更大，所以我们更新其上界），然后此时我们发现父节点出现了β > α的情况，说明最优解不可能从这个节点的子节点中产生，所以我们不再继续搜索它的其他子节点，（这就是β 剪枝）并继续返回其节点值，尝试更新父节点，因为父节点的α = 3 > 2，所以更新失败，

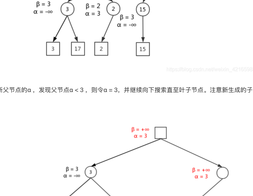
然后我们已经搜索完了当前这个Max节点的所有子节点，所以我们返回它的节点值，并尝试更新他的父节点的β 值，因为父节点的β > 3，所以我们令β = 3，并继续向下搜索至叶子节点，注意新生成的子节点的α 、β值继承自父节点，



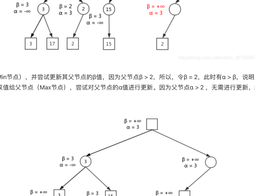
然后，我们发现15并不能更新当前节点的β值，所以令当前节点权值为15，并返回其权值，尝试更新其父节点(Max节点)的α 值，因为其父节点的α < 15，所以我们令α = 15 ，此时，该节点α = 15，β = 3，α > β，则说明其子节点并不包含最优解，不需要在进行搜索，所以返回其节点权值给父节点（Min节点），尝试对父节点的β值进行更新，父节点的β < 15，则不需要进行更新，同时可确定父节点的权值为3，



继续返回权值给父节点，尝试更新父节点的α，发现父节点α < 3，则令α = 3，并继续向下搜索直至叶子节点，注意新生成的子节点的α 、β 值继承自父节点，



从叶子节点返回权值给父节点（Min节点），并尝试更新其父节点的β值，因为父节点β > 2，所以，令β = 2，此时有α > β，说明其子节点并不包含最优解，不需要再进行搜索，所以返回其节点权值给父节点（Max节点），尝试对父节点的α值进行更新，因为父节点α > 2，无需进行更新，继续搜索其子节点至叶子节点，



从叶子节点返回权值给父节点（Min节点），并尝试更新其父节点的β值，因为父节点β > 3，所以，令β = 3，同时确认父节点权值为3，继续返回权值给父节点，并尝试更新其父节点的α 值，因为父节点α = 3，所以无需进行更新，同时确定该节点权值为3，

因为该节点的所有子节点全部搜索完毕，所以返回该点权值给父节点，并尝试更新其父节点的β值，因为父节点β > 3，所以，令β = 3，同时确认父节点权值为3. 因为此时有α = β = 3，所以无需再搜索其子节点，直接返回权值给根节点，并尝试更新根节点的α 值，因为根节点α = 3，所以无需进行更新，

根节点的所有子节点搜索完毕，则得出最优解为3，

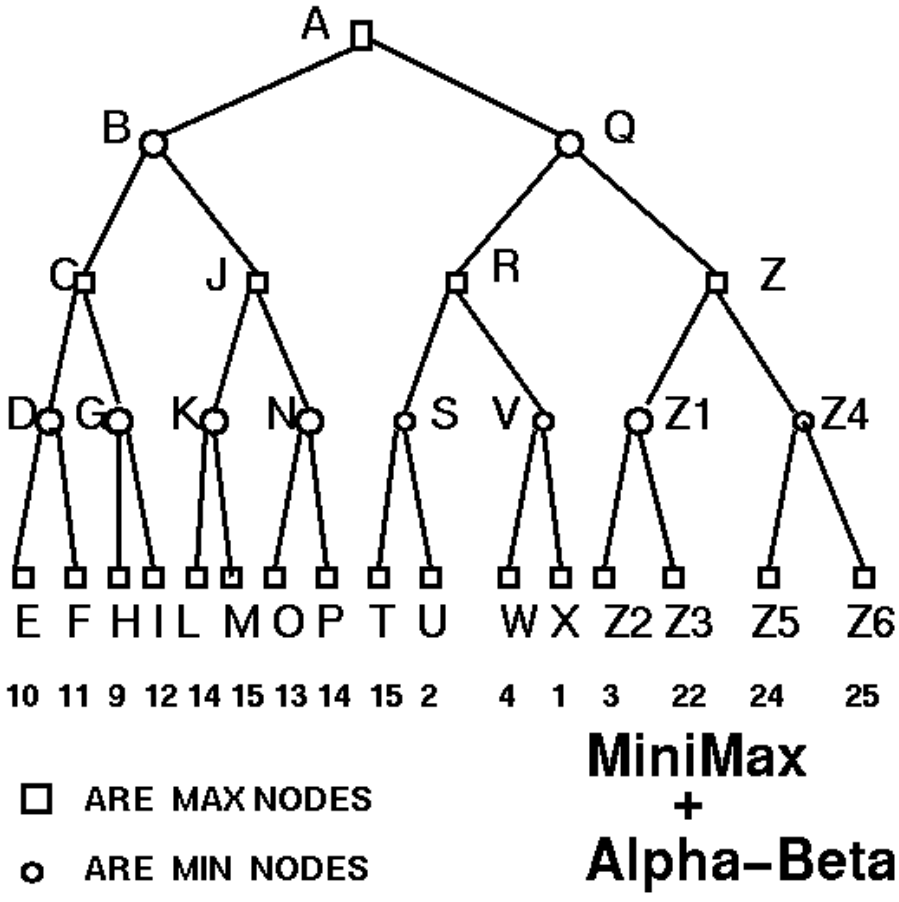
可以对比一下不加剪枝与加了剪枝之后的搜索过程，可以发现Alpha-Beta剪枝对算法的性能有了很大的提升，

7. Another way to understand the algorithm

- For a given node N,

  α is the value of N to MAX
  β is the value of N to MIN

Example


A
B
Q
C
J
R
Z
D
G
K
N
S
V
Z1
Z4
E F H I L M O P T U W X Z2 Z3 Z5 Z6
10 11 9 12 14 15 13 14 15 2 4 1 3 22 24 25
ARE MAX NODES
ARE MIN NODES
MiniMax
+
Alpha–Beta

α-β algorithm: slight variant (from earlier version of textbook)

Basically MINIMAX + keep track of α, β + prune

function MAX-VALUE($state, game, \alpha, \beta$) **returns** the minimax value of *state*
  inputs: *state*, current state in game
    game, game description
    α, the best score for MAX along the path to *state*
    β, the best score for MIN along the path to *state*
  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each s **in** SUCCESSORS(*state*) **do**
    $\alpha \leftarrow$ MAX(α, MIN-VALUE($s, game, \alpha, \beta$))
    if $\alpha \geq \beta$ **then return** β
  end
  return α

Is this wrong compared to latest version of textbook?

---

function MIN-VALUE($state, game, \alpha, \beta$) **returns** the minimax value of *state*
  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each s **in** SUCCESSORS(*state*) **do**
    $\beta \leftarrow$ MIN(β, MAX-VALUE($s, game, \alpha, \beta$))
    if $\beta \leq \alpha$ **then return** α
  end
  return β

Please always use latest version of the algorithm as in 3rd edition of textbook.

Solution

NODE	TYPE	ALPHA	BETA	SCORE
A	MAX	-Inf	Inf	
B	MIN	-Inf	Inf	
C	MAX	-Inf	Inf	
D	MIN	-Inf	Inf	
E	MAX	10	10	10
D	MIN	-Inf	10	
F	MAX	11	11	11
D	MIN	-Inf	10	10
C	MAX	10	Inf	
G	MIN	10	Inf	
H	MAX	9	9	9
G	MIN	10	9	9
C	MAX	10	Inf	10
B	MIN	-Inf	10	
J	MAX	-Inf	10	
K	MIN	-Inf	10	
L	MAX	14	14	14
K	MIN	-Inf	10	
M	MAX	15	15	15
K	MIN	-Inf	10	10
…				

NODE	TYPE	ALPHA	BETA	SCORE
…				
J	MAX	10	10	10
B	MIN	-Inf	10	10
A	MAX	10	Inf	
Q	MIN	10	Inf	
R	MAX	10	Inf	
S	MIN	10	Inf	
T	MAX	15	15	15
S	MIN	10	15	
U	MAX	2	2	2
S	MIN	10	2	2
R	MAX	10	Inf	
V	MIN	10	Inf	
W	MAX	4	4	4
V	MIN	10	4	4
R	MAX	10	Inf	10
Q	MIN	10	10	10
A	MAX	10	Inf	10

State-of-the-art for deterministic games

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

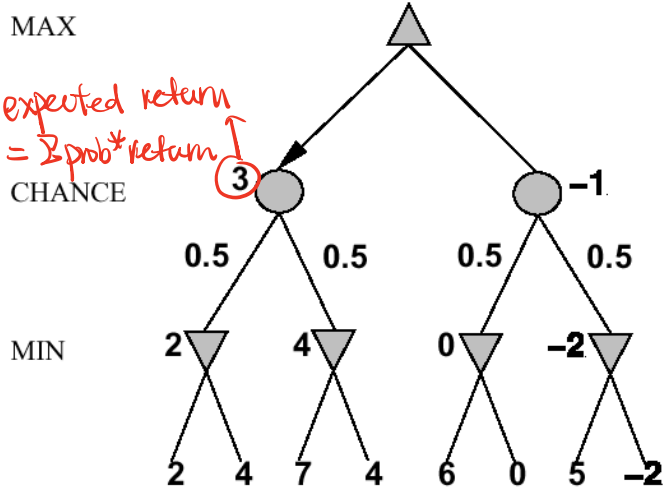
Before 2020

After 2020: Alpha-GO Win !!

Nondeterministic games

E..g, in backgammon, the dice rolls determine the legal moves

Simplified example with coin-flipping instead of dice-rolling:



Algorithm for nondeterministic games

EXPECTIMINIMAX gives perfect play

Just like MINIMAX, except we must also handle chance nodes:

```
...
if state is a chance node then
        return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
...
```

A version of α–β pruning is possible
but only if the leaf values are bounded. Why??

Remember: Minimax algorithm

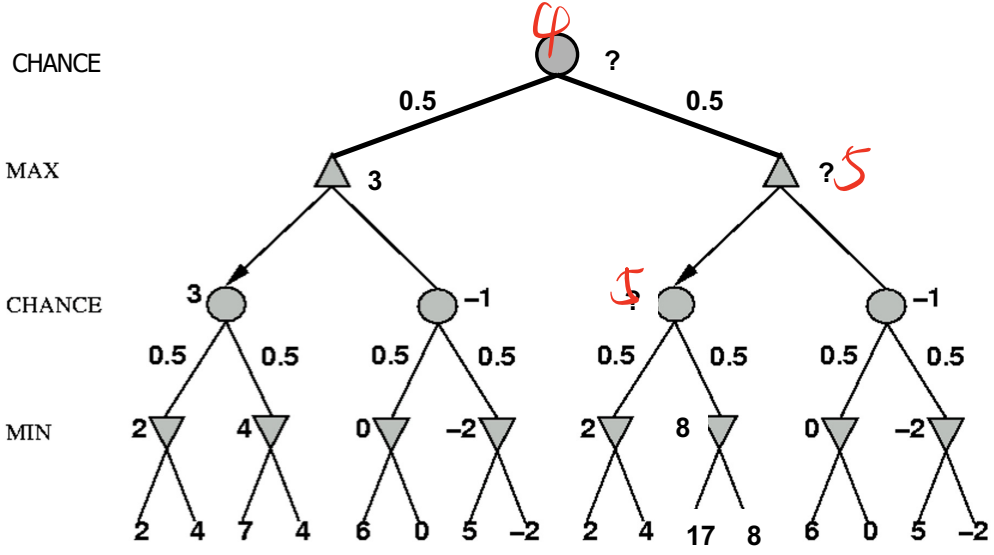
```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

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

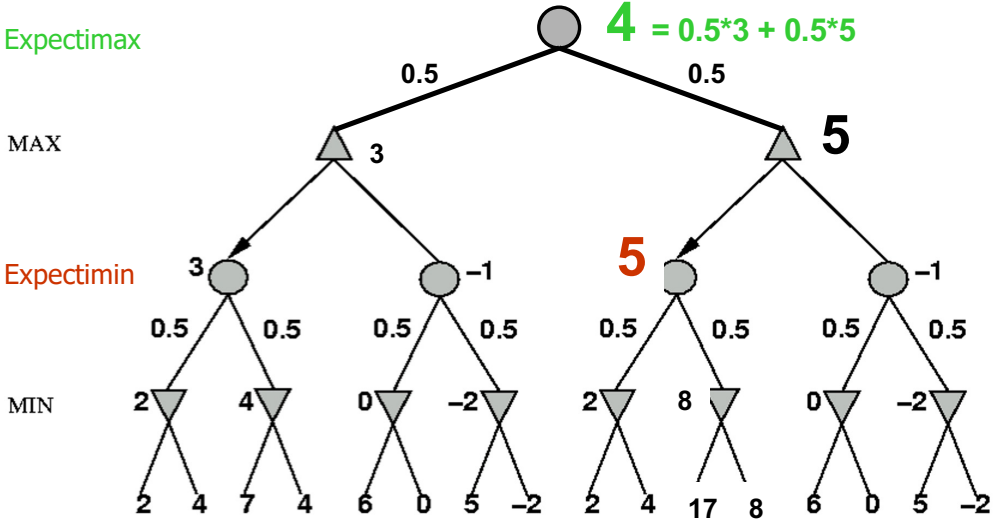
```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Nondeterministic games: the element of chance

expectimax and **expectimin**, expected values over all possible outcomes

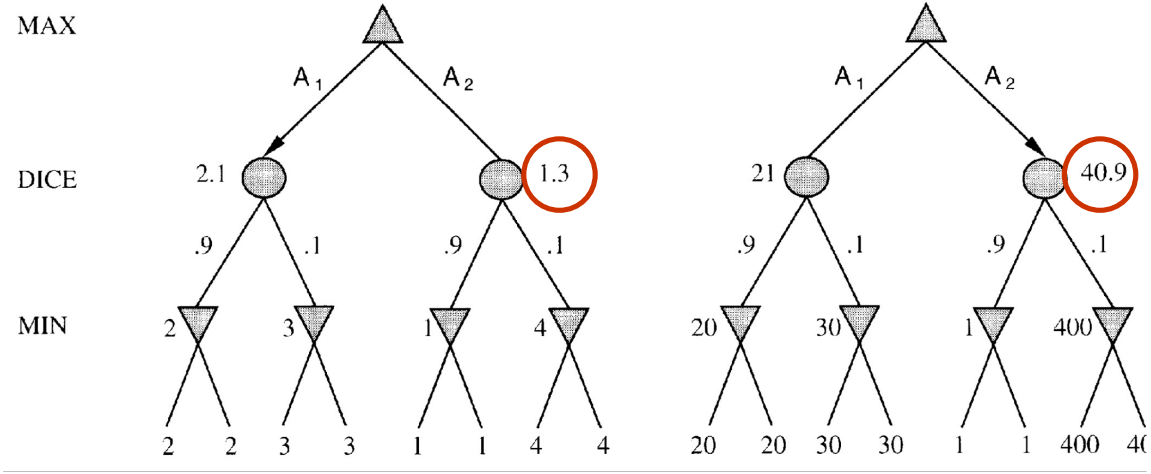


Nondeterministic games: the element of chance



Evaluation functions: Exact values DO matter

Order-preserving transformation do not necessarily behave the same!



State-of-the-art for nondeterministic games

Dice rolls increase b: 21 possible rolls with 2 dice
Backgammon $\approx$ 20 legal moves (can be 6,000 with 1-1 roll)

$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

As depth increases, probability of reaching a given node shrinks
$\Rightarrow$ value of lookahead is diminished

α–β pruning is much less effective

Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- ♢ perfection is unattainable ⇒ must approximate
- ♢ good idea to think about what to think about
- ♢ uncertainty constrains the assignment of values to states

Games are to AI as grand prix racing is to automobile design

Exercise: Game Playing

Consider the following game tree in which the evaluation function values are shown below each leaf node. Assume that the root node corresponds to the maximizing player. Assume the search always visits children left-to-right.

(a) Compute the backed-up values computed by the minimax algorithm. Show your answer by writing values at the appropriate nodes in the above tree.

(b) Compute the backed-up values computed by the alpha-beta algorithm. What nodes will not be examined by the alpha-beta pruning algorithm?

(c) What move should Max choose once the values have been backed-up all the way?