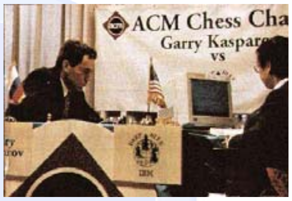
Games

I could feel – I could smell – a new kind of intelligence across the table

- Gary Kasparov



Saying Deep Blue doesn't really think about chess is like saying an airplane doesn't really fly because it doesn't flap its wings.
– Drew McDermott

Deep Blue beats Gary Kasparov - 1997
(3 wins, 1 loss, 2 draws)

- Deep Blue: 32 RISC processors + 256 VLSI chess engines
- 200 million positions per second, 16 plies

Today

- Game tree search (40 min)
  - Minimax
  - Alpha-Beta Pruning
- Games of chance (30 min)

Tonight

- Game tree search (40 min)
- Group exercise: Reversi (50 min)
- Reversi Tournament (20 min)
- Games of chance (30 min)

Games in AI

- In AI, "games" usually refers to deterministic, turn-taking, two-player, zero-sum games of perfect information
  - Deterministic: next state of environment is completely determined by current state and action executed by the agent (not probabilistic)
  - Turn-taking: 2 agents whose actions must alternate
  - Zero-sum games: if one agent wins, the other loses
  - Perfect information: fully observable

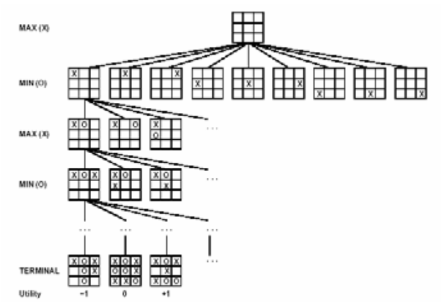
Other Games

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

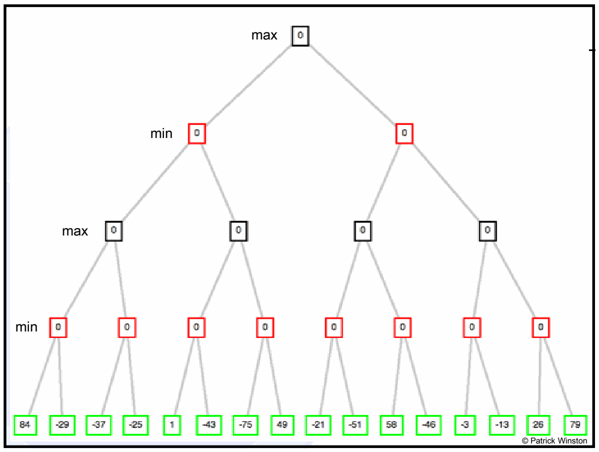
Games as Search

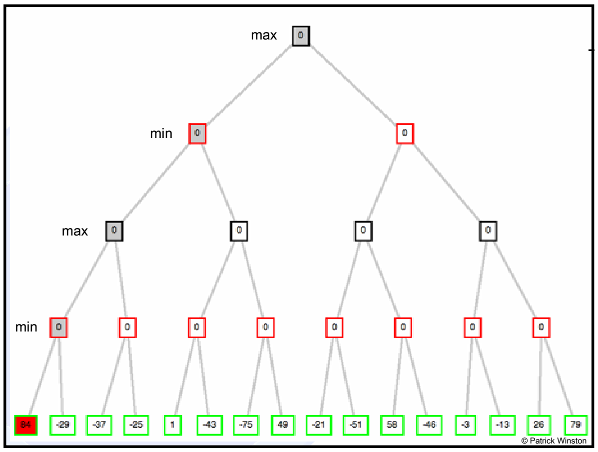
- States:
  - board configurations
- Initial state:
  - the board position and which player will move
- Successor function:
  - returns list of (move, state) pairs, each indicating a legal move and the resulting state
- Terminal test:
  - determines when the game is over
- Utility function:
  - gives a numeric value in terminal states
  (e.g., -1, 0, +1 for loss, tie, win)

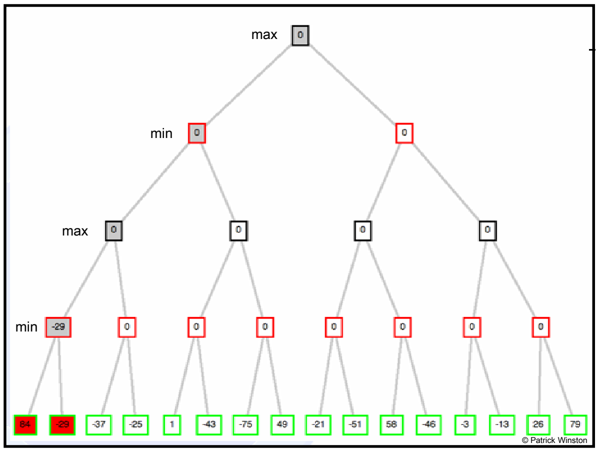
Intuition

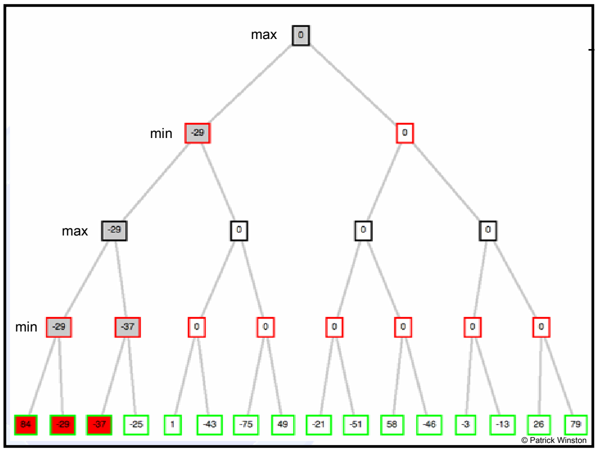


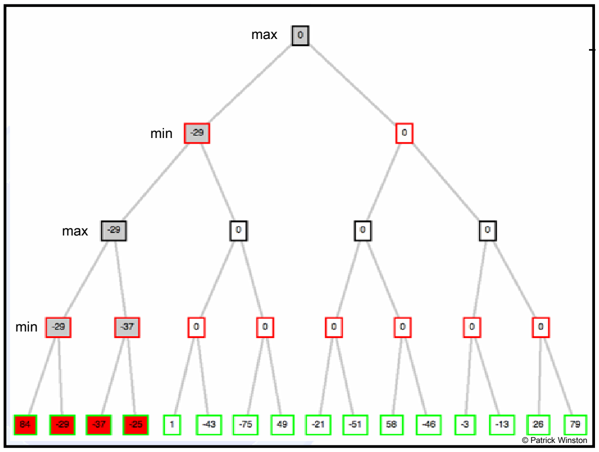
Mini-Max

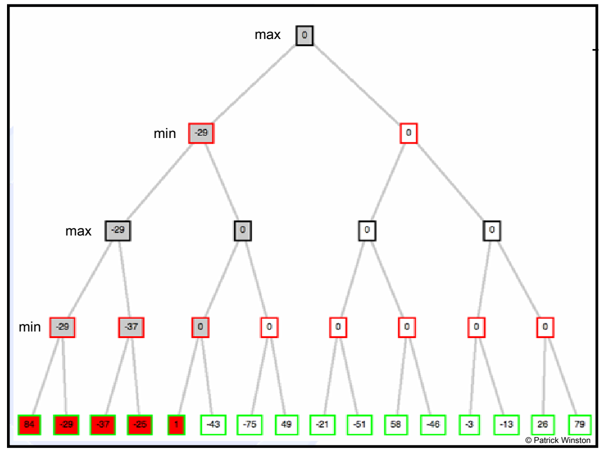


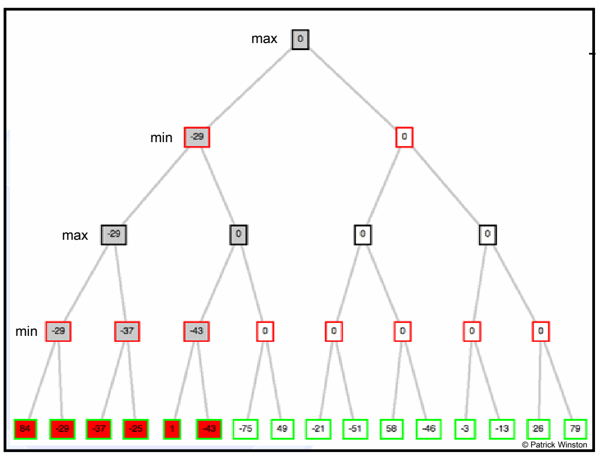


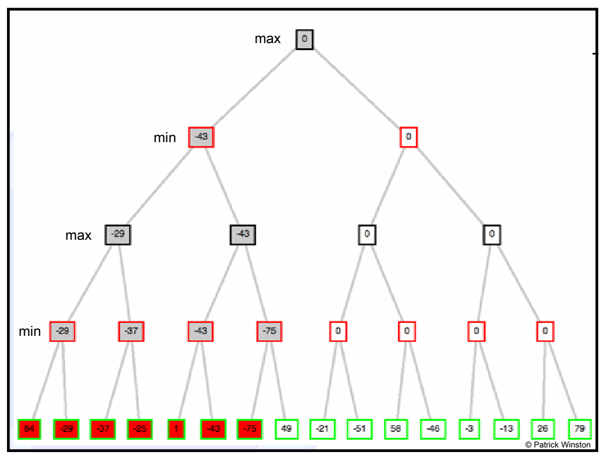


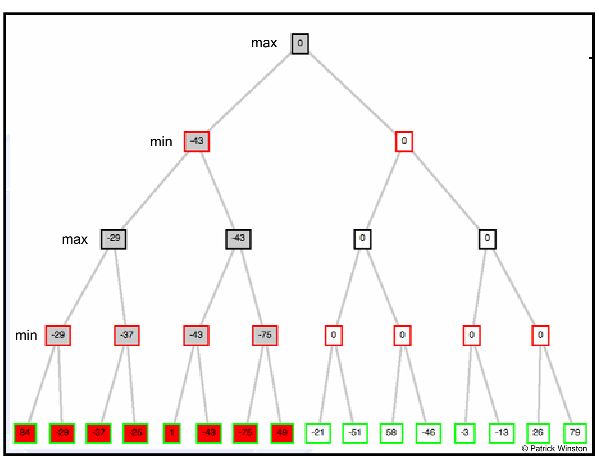


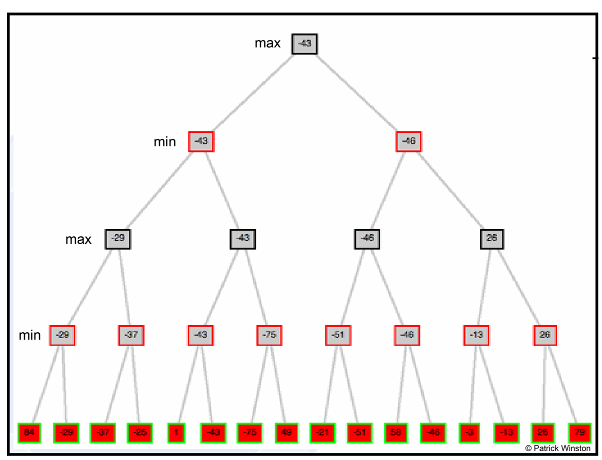

max
min
max
min
© Patrick Winston

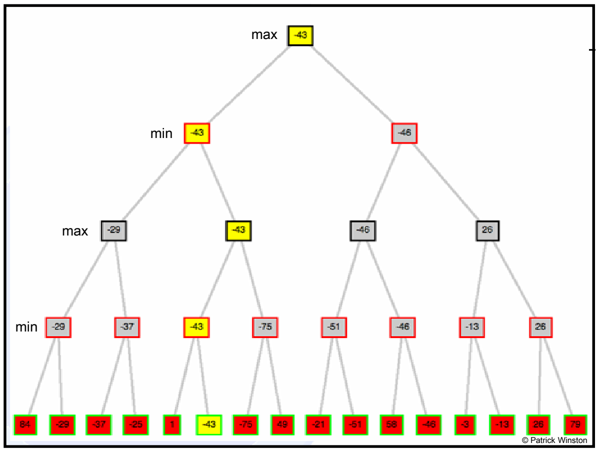

max
min
max
min
© Patrick Winston


max
min
max
min
© Patrick Winston


max
min
max
min
© Patrick Winston


max
min
max
min
© Patrick Winston


max
min
max
min
© Patrick Winston



Mini-Max Properties

- Complete?
- Optimal?
  - Against an optimal opponent?
  - Otherwise?
- Time complexity?
- Space complexity?

Mini-Max Properties

- Complete? Yes, if tree is finite
- Optimal?
  - Against an optimal opponent?
  - Otherwise?
- Time complexity?
- Space complexity?

Mini-Max Properties

- Complete? Yes, if tree is finite
- Optimal?
  - Against an optimal opponent? Yes
  - Otherwise? No: Does at least as well, but may not exploit opponent weakness
- Time complexity?
- Space complexity?

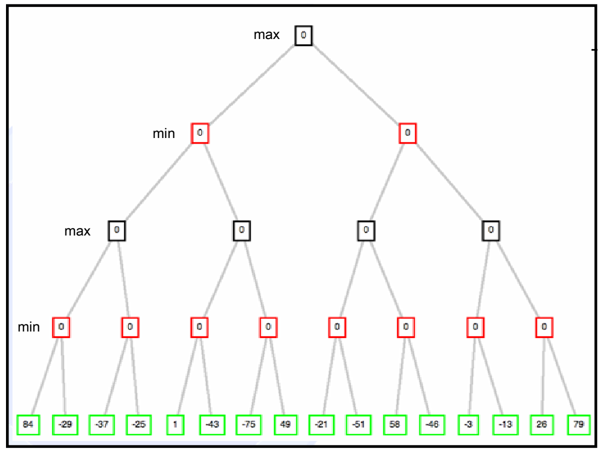
Mini-Max Properties

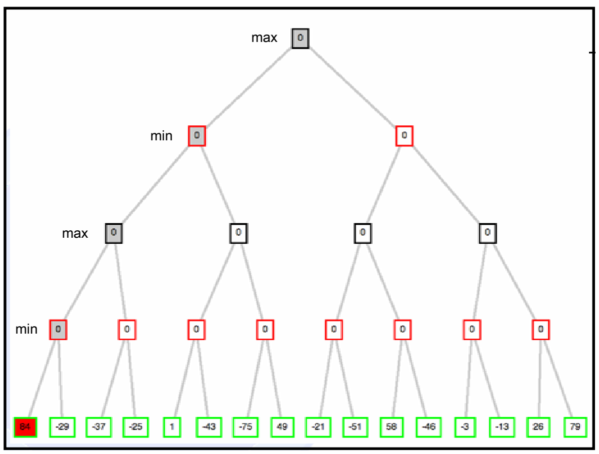
- Complete? Yes, if tree is finite
- Optimal?
  - Against an optimal opponent? Yes
  - Otherwise? No: Does at least as well, but may not exploit opponent weakness
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$

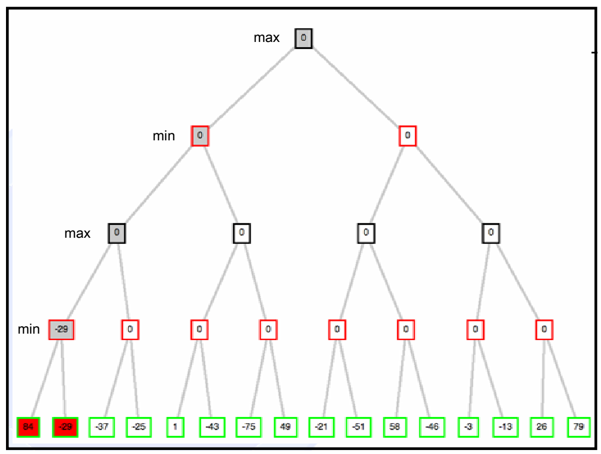
Good Enough?

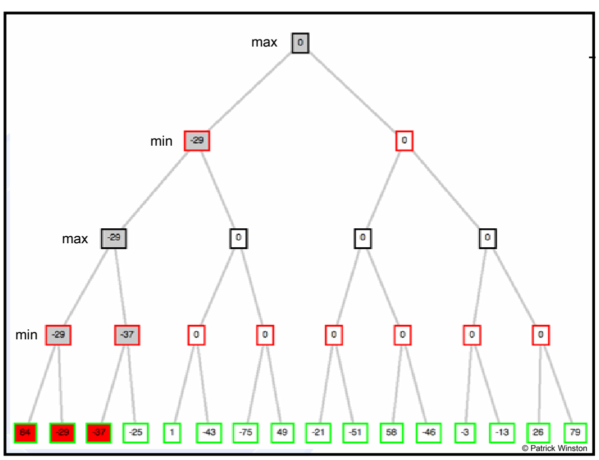
- Chess:
  - branching factor $b \approx 35$
  - game length $m \approx 100$
  - search space $b^m \approx 35^{100} \approx 10^{154}$
- The Universe:
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{18}$ seconds
  - 10^8 moves/sec x 10^{78} x 10^{18} = 10^{104}

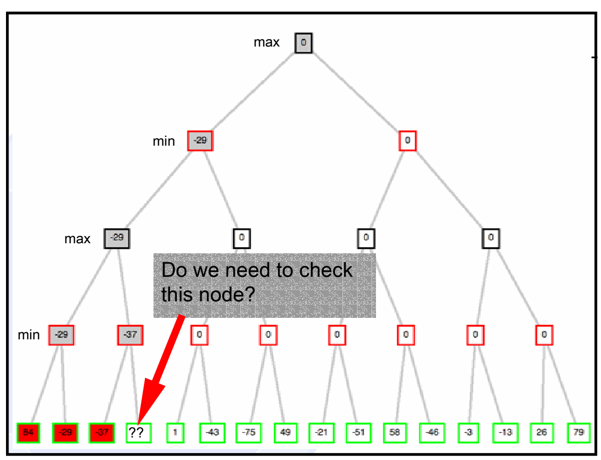
Alpha-Beta Pruning

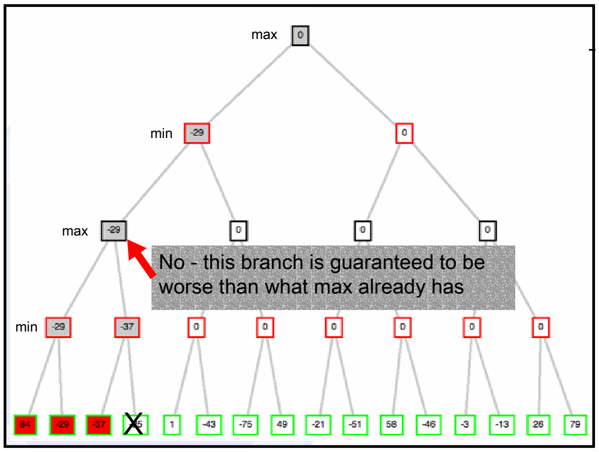












Alpha-Beta

```
MinVal(state, alpha, beta){
 if (terminal(state))
     return utility(state);
 for (s in children(state)){
  child = MaxVal(s,alpha,beta);
  beta = min(beta,child);
  if (alpha>=beta) return child;
 }
 return beta; }
```

alpha = the **highest** value for **MAX** along the path

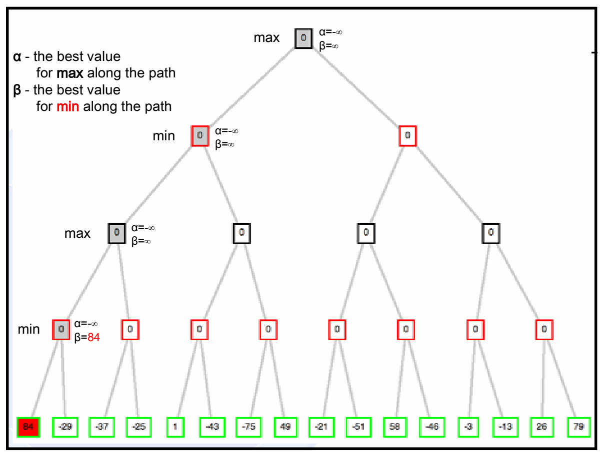
beta = the **lowest** value for **MIN** along the path

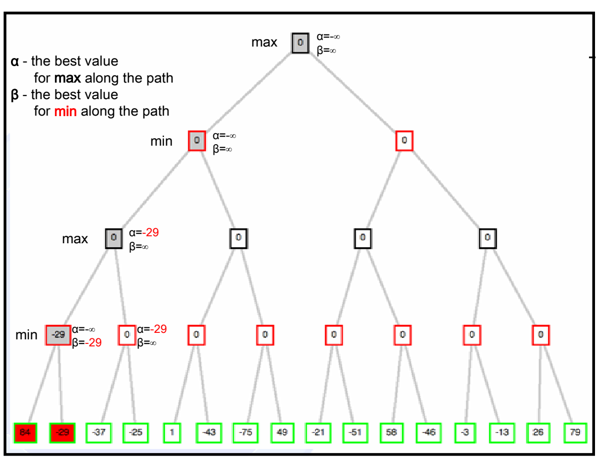
Alpha-Beta

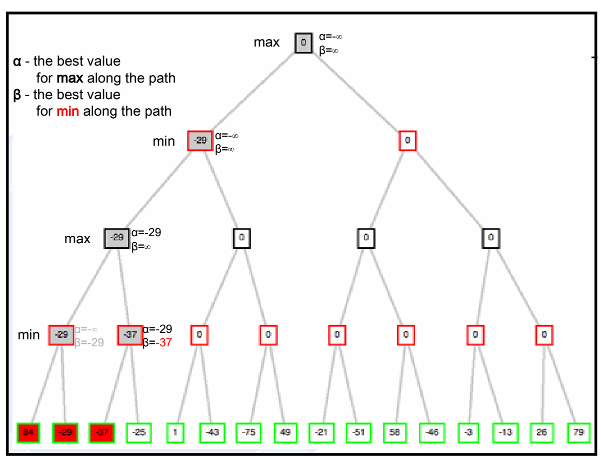
```
MaxVal(state, alpha, beta){
 if (terminal(state))
     return utility(state);
 for (s in children(state)){
  child = MinVal(s,alpha,beta);
  alpha = max(beta,child);
  if (alpha>=beta) return child;
 }
 return beta; }
```

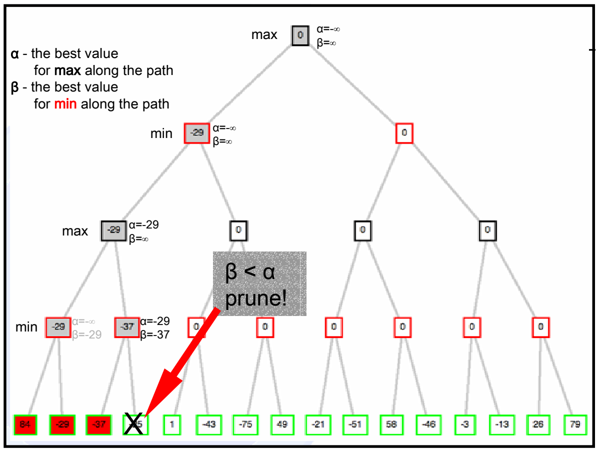
alpha = the **highest** value for **MAX** along the path

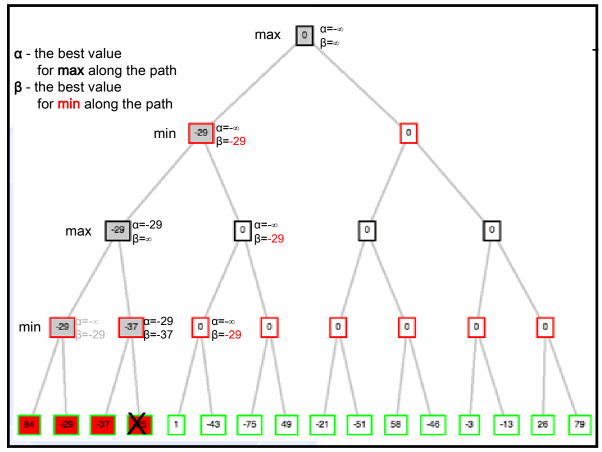
beta = the **lowest** value for **MIN** along the path

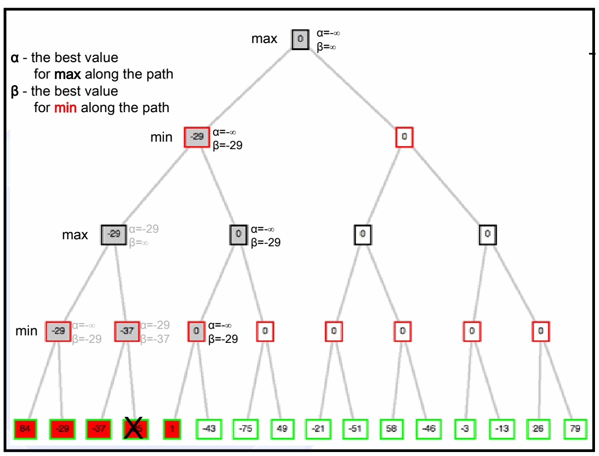


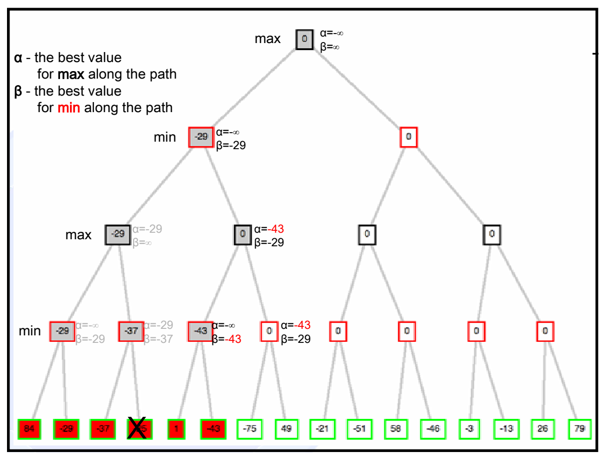


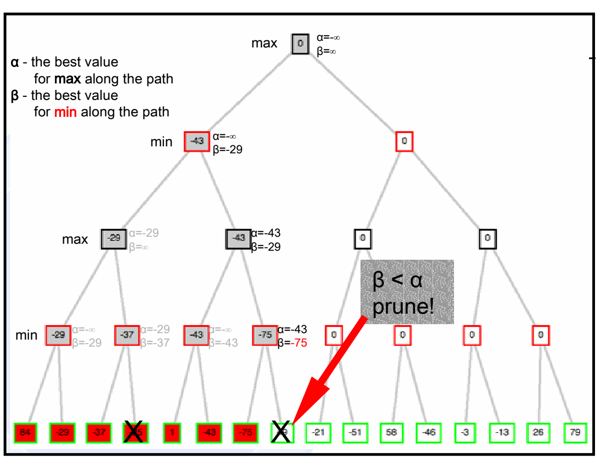


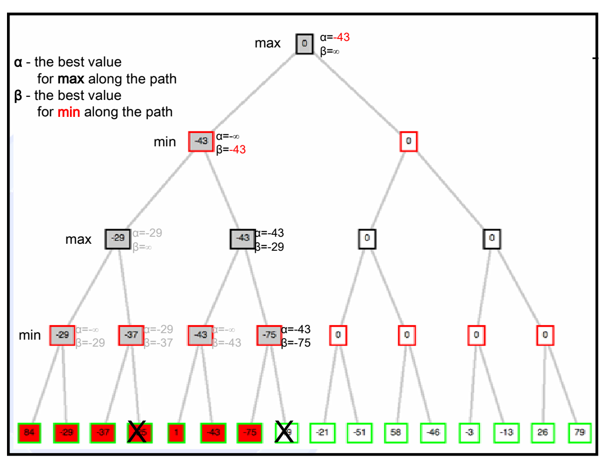

α - the best value
for **max** along the path
β - the best value
for **min** along the path
max
min
max
min
β < α
prune!

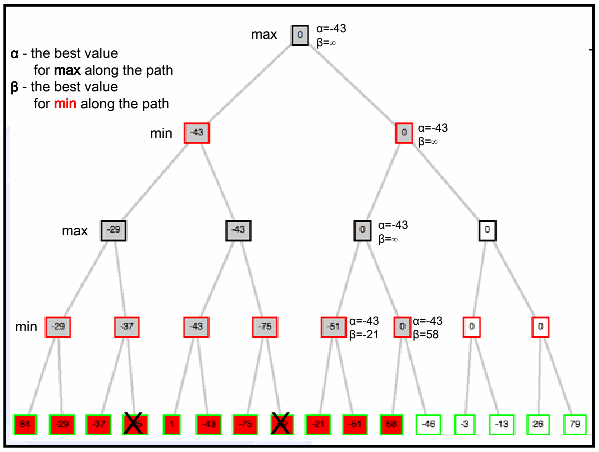

α - the best value
for **max** along the path
β - the best value
for **min** along the path
max
min
max
min

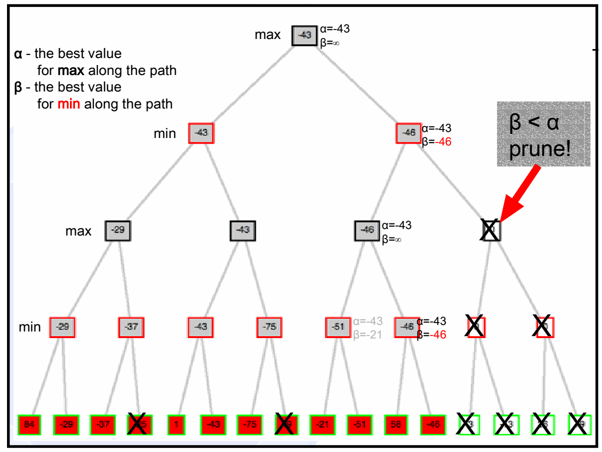

α - the best value
for **max** along the path
β - the best value
for **min** along the path
max
min
max
min


α - the best value
for **max** along the path
β - the best value
for **min** along the path
max
min
max
min


α - the best value
for **max** along the path
β - the best value
for **min** along the path
max
min
max
min
β < α
prune!


α - the best value
for **max** along the path
β - the best value
for **min** along the path
max
min
max
min





Good Enough?

- Chess:
  - branching factor b≈35
  - game length m≈100
  - search space $b^{m/2} \approx 35^{50} \approx 10^{77}$
- The Universe:
  - number of atoms ≈ 10^{78}
  - age ≈ 10^{18} seconds
  - 10^{8} moves/sec x 10^{78} x 10^{18} = 10^{104}

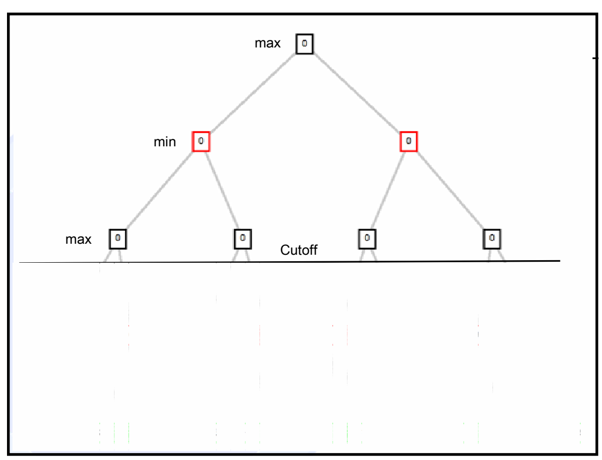
The universe can play chess - can we?

Alpha-Beta Properties

- Still guaranteed to find the best move
  - Best case time complexity: O(bm/2)
- Can double the depth of search!
  - Best case when best moves are tried first
- Good static evaluation function helps!
  - But still too slow for chess...

Partial Space Search

- Strategies:
  - search to a fixed depth
  - iterative deepening (most common)
  - ignore ‘quiescent’ nodes
- Static Evaluation Function assigns a score to a non-terminal state



Evaluation Functions

- Reversi
  - Number squares held?
  - Better: number of squares held that cannot be flipped
  - Prefer valuable squares
    - NxN array w[i,j] of position values
    - Highest value: corners, edges
    - Lowest value: next to corner or edge
    - s[i,j] = +1 player, 0 empty, -1 opponent

$$score = \sum_{i,j} w[i,j]s[i,j]$$

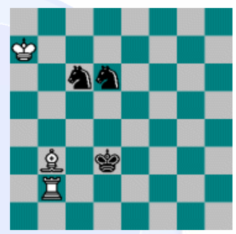
Evaluation Functions

- Chess:
  - eval(s) =
    w1 * material(s) +
    w2 * mobility(s) +
    w3 * king safety(s) +
    w4 * center control(s) + ...
  - In practice MiniMax improves accuracy of heuristic eval function
  - But one can construct pathological games where more search hurts performance!
    (Nau 1981)

End-Game Databases

- Ken Thompson - all 5 piece end-games
- Lewis Stiller - all 6 piece end-games
  - Refuted common chess wisdom: many positions thought to be ties were really forced wins -- 90% for white
  - Is perfect chess a win for white?

The MONSTER



White wins in 255 moves
(Stiller, 1991)

Deterministic Games in Practice

- 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 6 game match in 1997.
- Reversi: human champions refuse to play against computers because software is too good

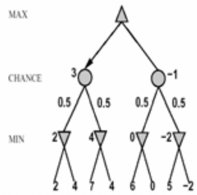
Deterministic Games in Practice

- Go: human champions refuse to compete against computers, because software is too bad.

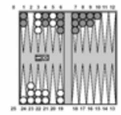
	Chess	Go
Size of board	8 x 8	19 x 19
Average no. of moves per game	100	300
Avg branching factor per turn	35	235
Additional complexity		Players can pass

Nondeterministic Games

- Involve chance: dice, shuffling, etc.
  - Chance nodes: calculate the expected value
  - E.g.: weighted average over all possible dice rolls



In Practice...



- Chance adds dramatically to size of search space
  - Backgammon: number of distinct possible rolls of dice is 21
  - Branching factor b is usually around 20, but can be as high as 4000 (dice rolls that are doubles)
- Alpha-beta pruning is generally less effective
- Best Backgammon programs use other methods

Imperfect Information

- E.g. card games, where opponents' initial cards are unknown
- Idea: For all deals consistent with what you can see
  - compute the minimax value of available actions for each of possible deals
  - compute the expected value over all deals

Probabilistic STRIPS Planning

```
domain: Hungry Monkey
shake:    if (ontable)
        Prob(2/3) -> +1 banana
                Prob(1/3) -> no change
        else
               Prob(1/6) -> +1 banana
               Prob(5/6) -> no change

jump: if (~ontable)
        Prob(2/3) -> ontable
                Prob(1/3) -> ~ontable
        else
            ontable
```

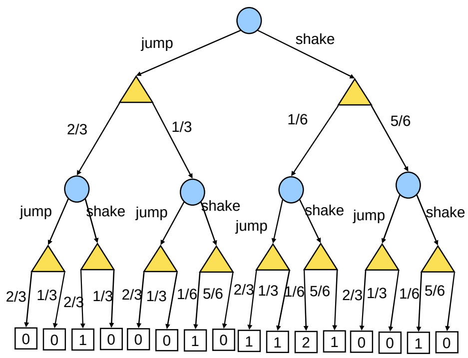
What is the expected reward?

```
[1] shake
[2] jump; shake
[3] jump; shake; shake;
[4] jump; if (~ontable){ jump; shake}
          else { shake; shake }
```

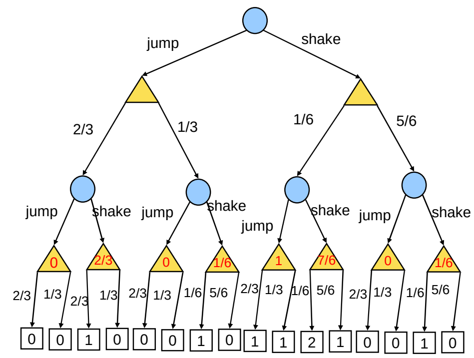
ExpectiMax

$$\text{ExpectiMax}(n) =$$
$$U(n) \text{ if n is a terminal node}$$
$$\max\{\text{ExpectiMax}(s) \mid s \in \text{children}(n)\} \text{ if n is max node}$$
$$\sum_{s \in children(n)} P(s)\text{ExpectiMax}(s) \text{ if n is a chance node}$$

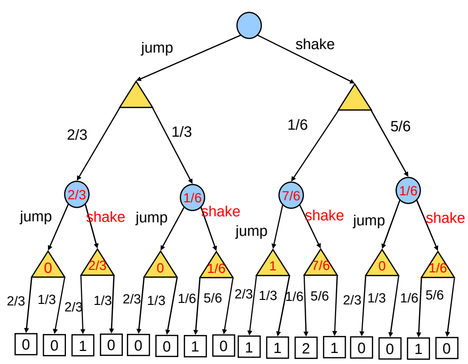
Hungry Monkey: 2-Ply Game Tree



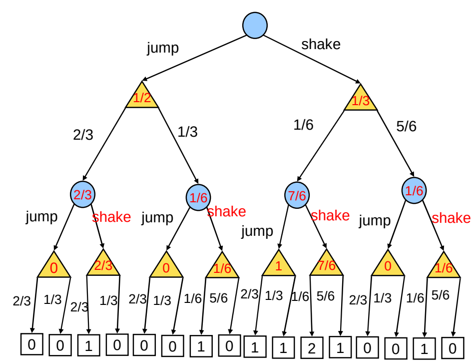
ExpectiMax 1 – Chance Nodes



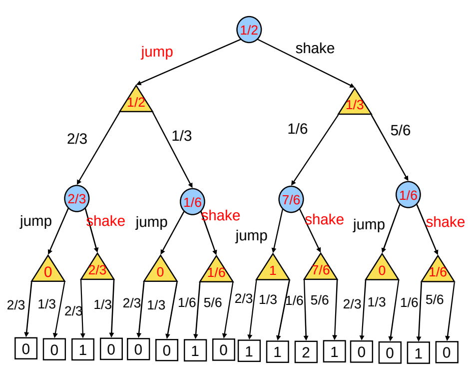
ExpectiMax 2 – Max Nodes



ExpectiMax 3 – Chance Nodes



ExpectiMax 4 – Max Node



Policies

- The result of the ExpectiMax analysis is a conditional plan (also called a policy):
  - Optimal plan for 2 steps: `jump; shake`
  - Optimal plan for 3 steps:
    `jump; if (ontable) {shake; shake}`
    `else {jump; shake}`
- Probabilistic planning can be generalized in many ways, including action costs and hidden state
- The general problem is that of solving a Markov Decision Process (MDP)

Summary

- Deterministic games
  - Minimax search
  - Alpha-Beta pruning
  - Static evaluation functions
- Games of chance
  - Expected value
  - Probabilistic planning
- Strategic games with large branching factors (Go)
  - Relatively little progress