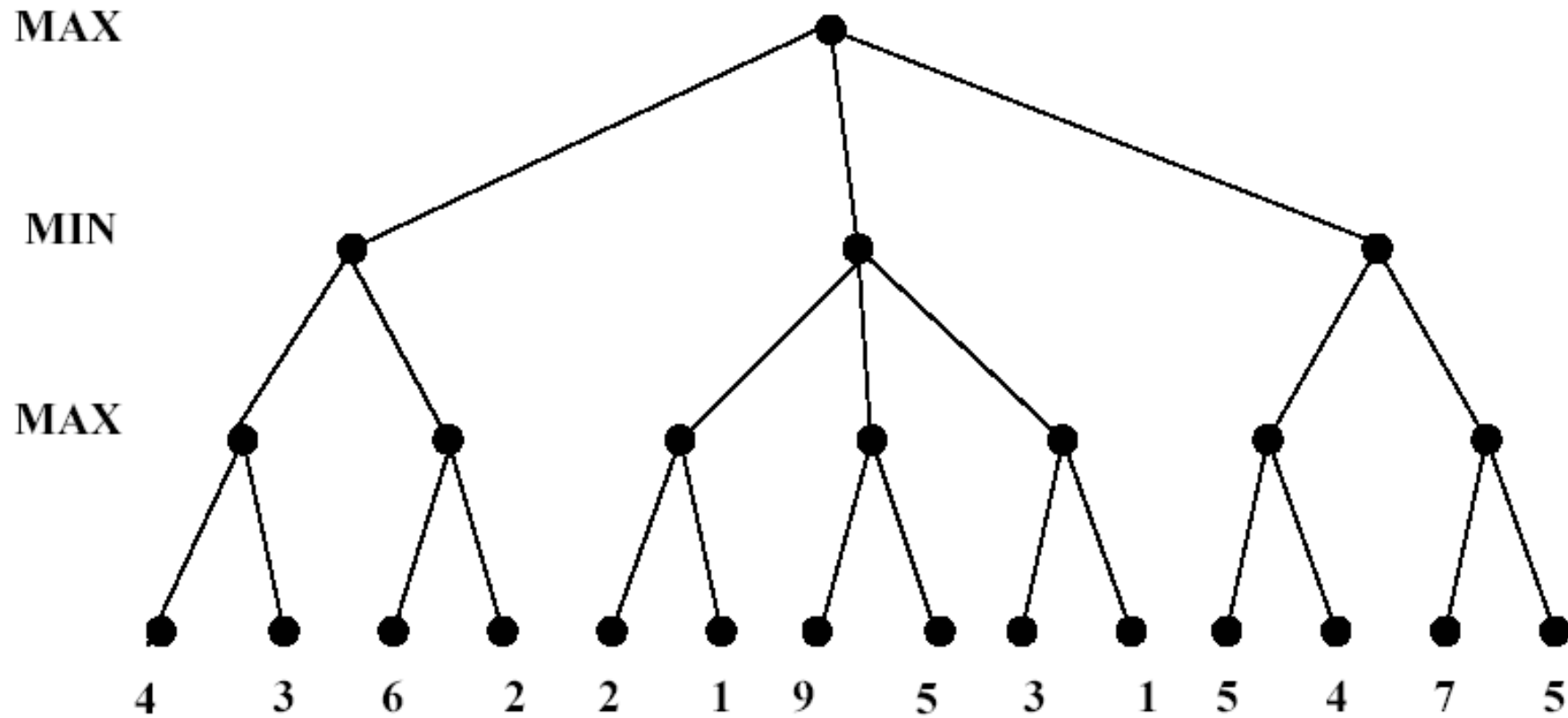


# Convergence Properties of GA's

Proof of convergence for SA and GA!

A Global Convergence Proof for a Class of Genetic Algorithms

# Minimax Example Zero Sum Game



Effectively depth first search, time complexity  $O(b^d)$

# Properties of Minimax Search

Complete?? Yes, if tree is finite (chess has specific rules for this)

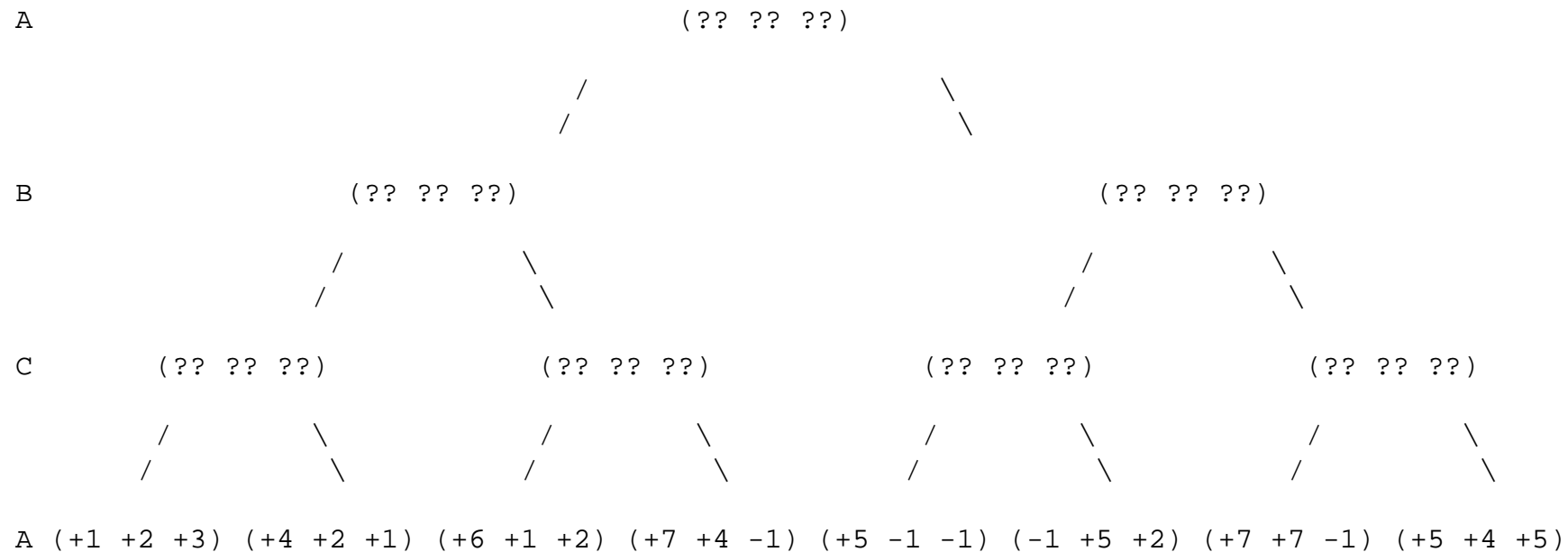
Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity??  $O(b^m)$

Space complexity??  $O(bm)$  (depth-first exploration)

For chess,  $b \approx 35$ ,  $m \approx 100$  for “reasonable” games  
⇒ exact solution completely infeasible

# Three Player and Non-Zero Sum Games

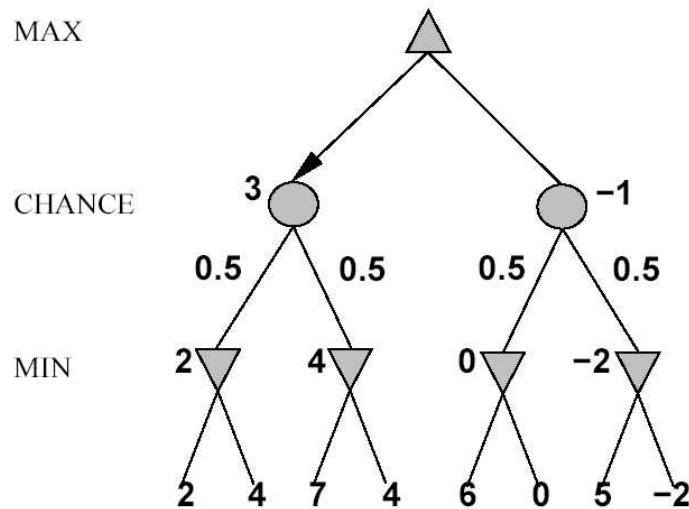


How do we create game trees that incorporate chance?

# Chance as a Bipartisan Third Player

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



Before each link implicitly  
Had  $Pr=1$  on it.

EXPECTIMINIMAX gives perfect play

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

...

if *state* is a MAX node then

return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

if *state* is a MIN node then

return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

if *state* is a chance node then

return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

...

# Depth Limited Search

MINIMAXCUTOFF is identical to MINIMAXVALUE except

1. TERMINAL? is replaced by CUTOFF?
2. UTILITY is replaced by EVAL

Evaluation function is just a heuristic (not an admissible heuristic)

Break game state into  $m$  components

$$\text{Eval}(n) = w_1c_1(n) + w_2c_2(n) \dots w_m c_m(n)$$

Note multiple states have the exact same component values

Eval function some estimate of the the expectation of

#wins/#games from this point (as described by the components)

State space, evaluation function for a single player  
checkers program?

# Checkers

	32		31		30		29
28		27		26		25	
	24		23		22		21
20		19		18		17	
	16		15		14		13
12		11		10		9	
	8		7		6		5
4		3		2		1	

## Evaluation function features

Samuel experimented with Over 50.  
Recent work looks at learning  
the feature weights.

Simplest evaluation function

$$F(n) = 3 \cdot c_3 + 2 \cdot c_2 + 1 \cdot c_1 + 30 \cdot a_3 + 20 \cdot a_2 + 10 \cdot a_1$$

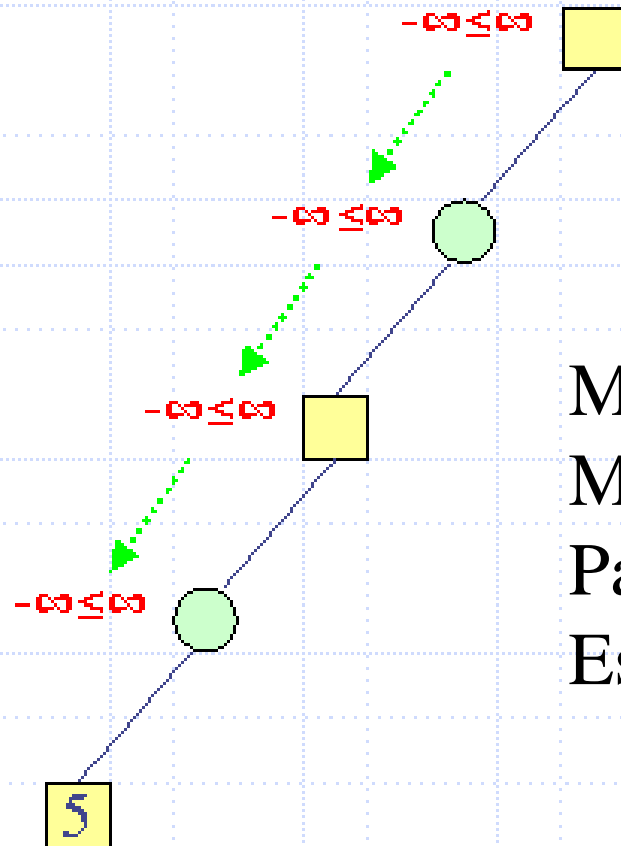
# Alpha Beta Pruning

- Produces exact same results as minimax algorithm
- Reduces search by not expanding sub-optimal paths
- Alpha and Beta terms are the floor and ceiling of the range of values, the player is interested in.

# Alpha-beta pruning example: Step 1

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

$\beta$  is the minimum upper bound of possible solutions (MIN plays)

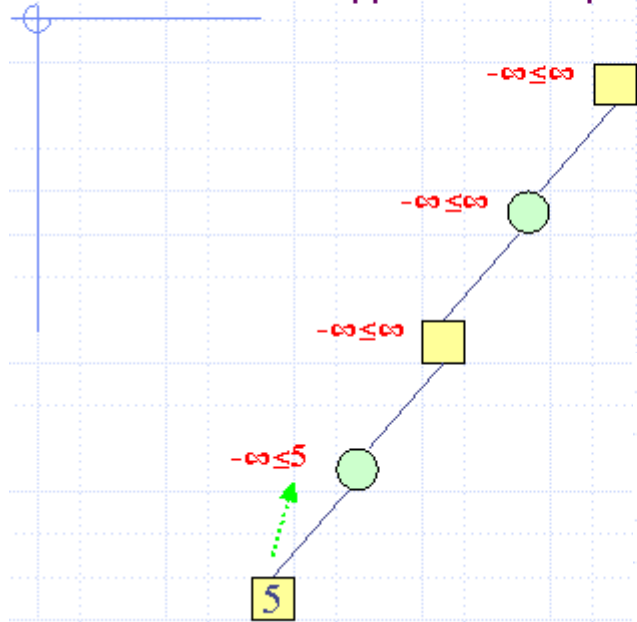


Max fills left hand alpha value  
Min fills in right beta value  
Parent can set current best  
Estimates for alpha and beta

### Alpha-beta pruning example: Step 2

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

$\beta$  is the minimum upper bound of possible solutions (MIN plays)

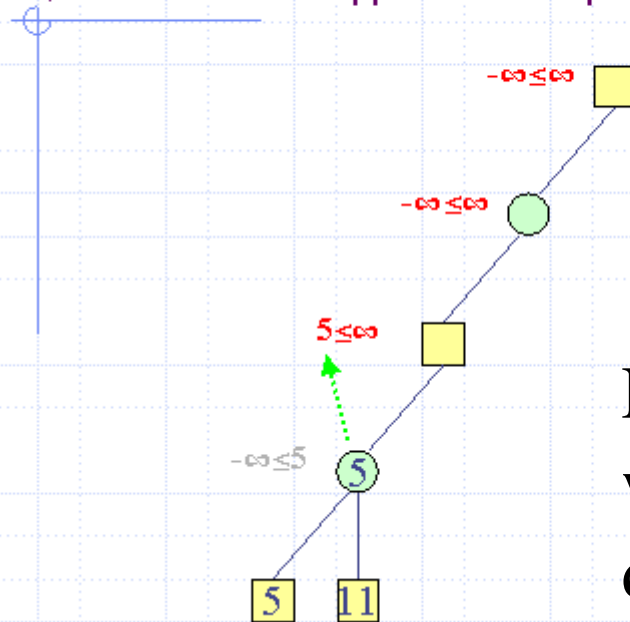


For this path,  
worst min can  
do is 5

### Alpha-beta pruning example: Step 3

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

$\beta$  is the minimum upper bound of possible solutions (MIN plays)

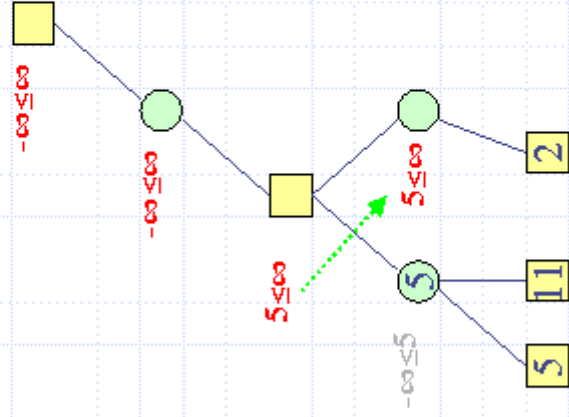


For this path,  
worst max can  
do is 5

## Alpha-beta pruning example: Step 4

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

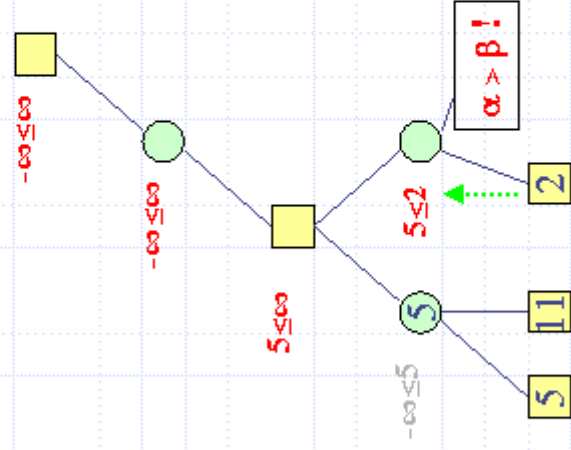
$\beta$  is the minimum upper bound of possible solutions (MIN plays)



## Alpha-beta pruning example: Step 5

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

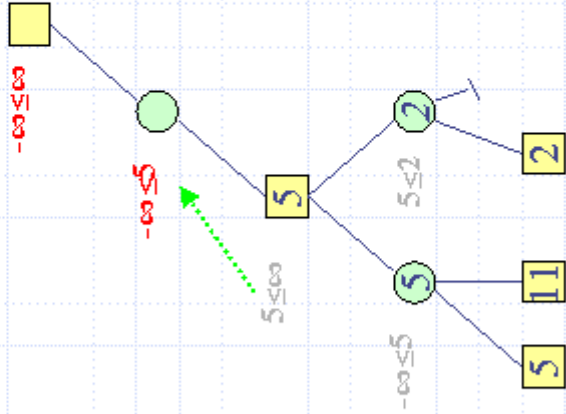
$\beta$  is the minimum upper bound of possible solutions (MIN plays)



## Alpha-beta pruning example: Step 6

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

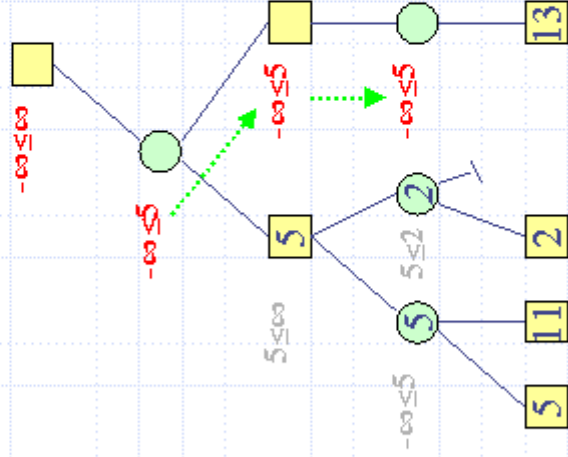
$\beta$  is the minimum upper bound of possible solutions (MIN plays)



## Alpha-beta pruning example: Step 7

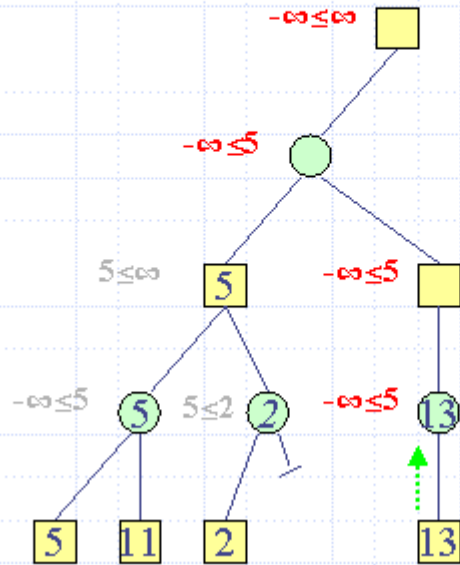
$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

$\beta$  is the minimum upper bound of possible solutions (MIN plays)



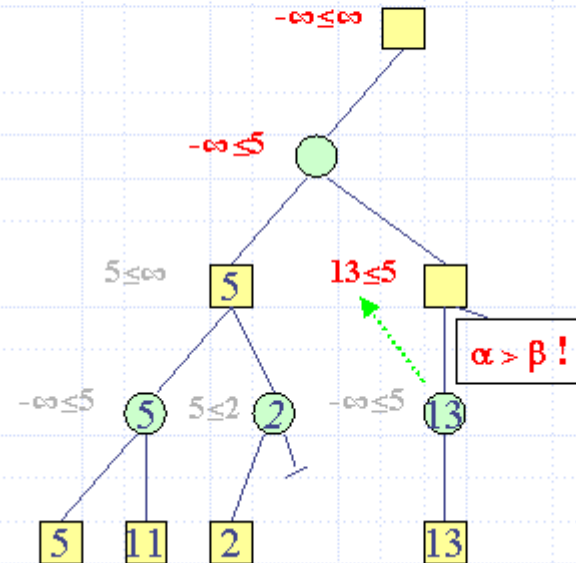
# Alpha-beta pruning example: Step 8

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)  
 $\beta$  is the minimum upper bound of possible solutions (MIN plays)



# Alpha-beta pruning example: Step 9

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)  
 $\beta$  is the minimum upper bound of possible solutions (MIN plays)

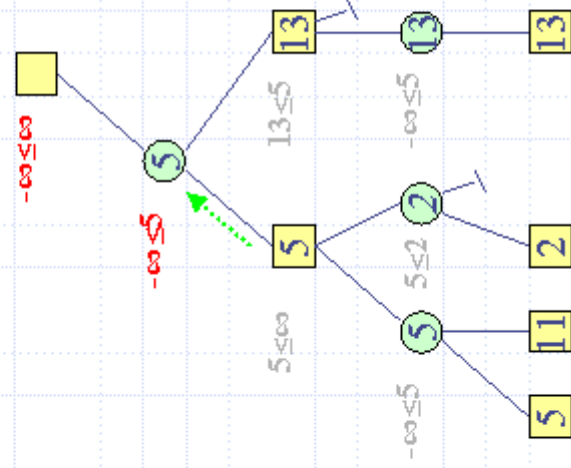


All branches “complete”

## Alpha-beta pruning example: Step 10

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

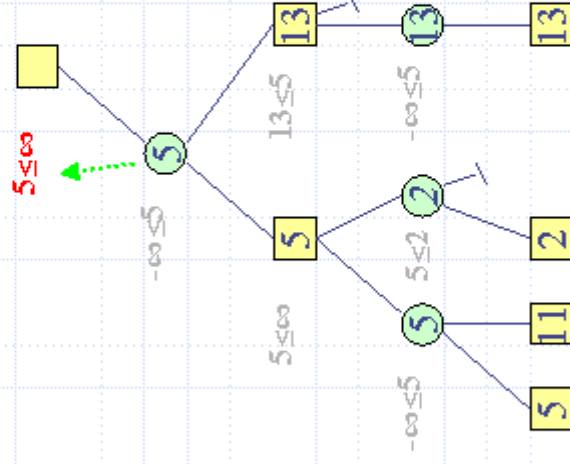
$\beta$  is the minimum upper bound of possible solutions (MIN plays)



## Alpha-beta pruning example: Step 11

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

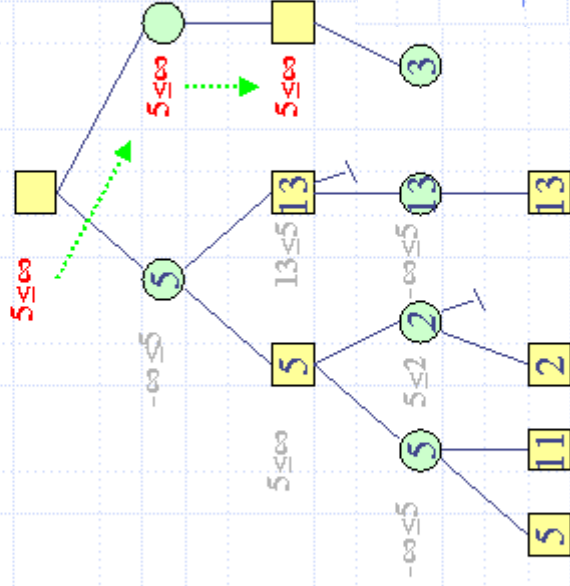
$\beta$  is the minimum upper bound of possible solutions (MIN plays)



## Alpha-beta pruning example: Step 12

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

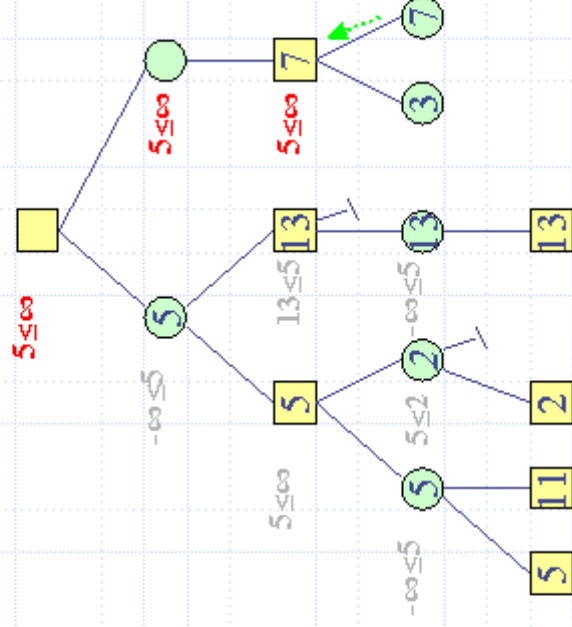
$\beta$  is the minimum upper bound of possible solutions (MIN plays)



## Alpha-beta pruning example: Step 13

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

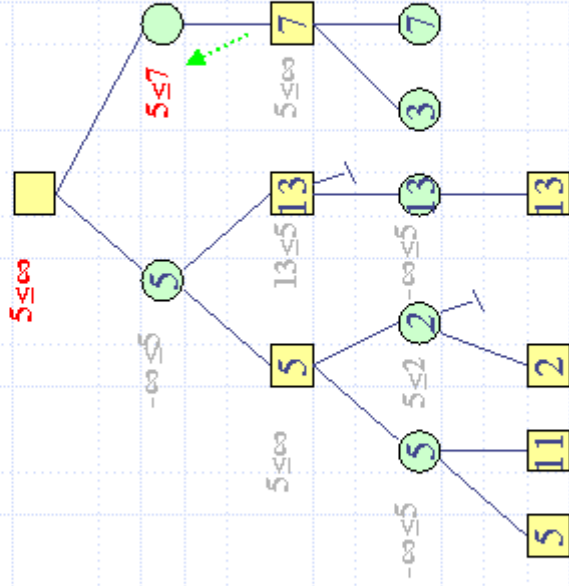
$\beta$  is the minimum upper bound of possible solutions (MIN plays)



## Alpha-beta pruning example: Step 14

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

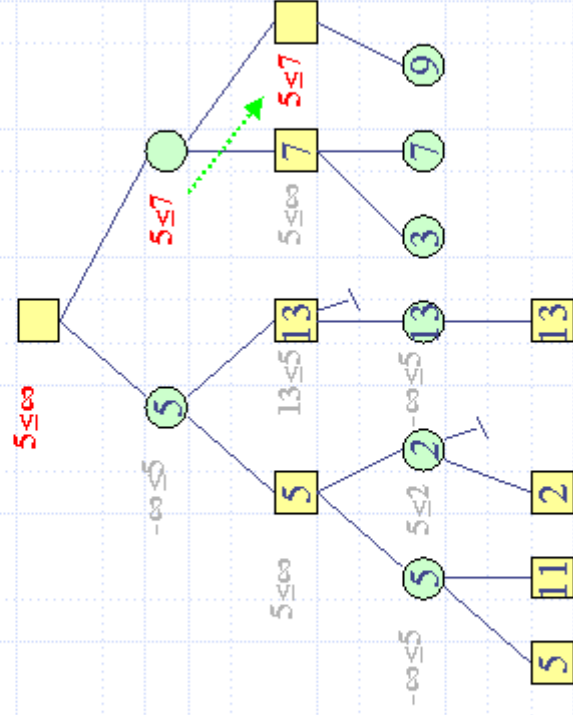
$\beta$  is the minimum upper bound of possible solutions (MIN plays)



## Alpha-beta pruning example: Step 15

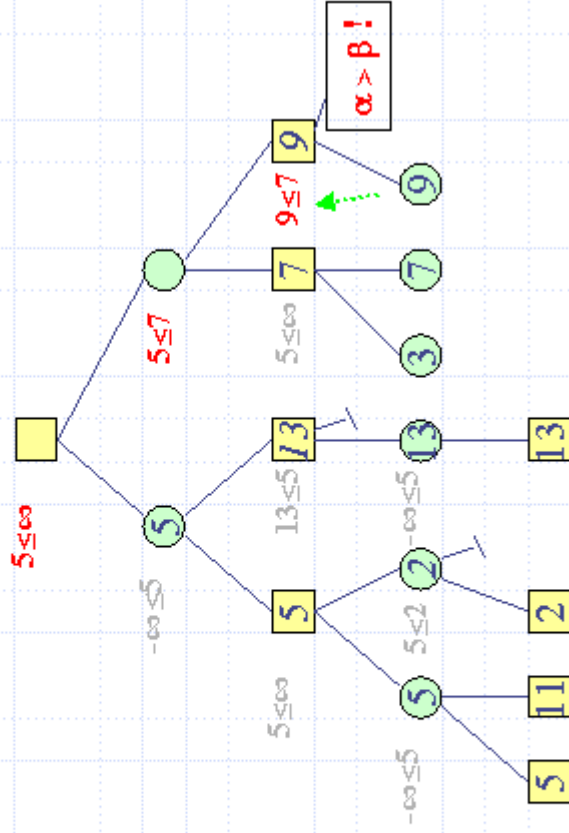
$\alpha$  is the maximum lower bound of possible solutions (MAX plays)

$\beta$  is the minimum upper bound of possible solutions (MIN plays)



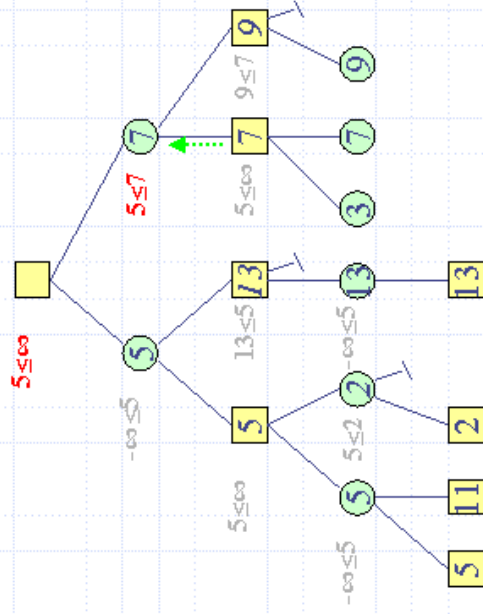
### Alpha-beta pruning example: Step 16

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)  
 $\beta$  is the minimum upper bound of possible solutions (MIN plays)



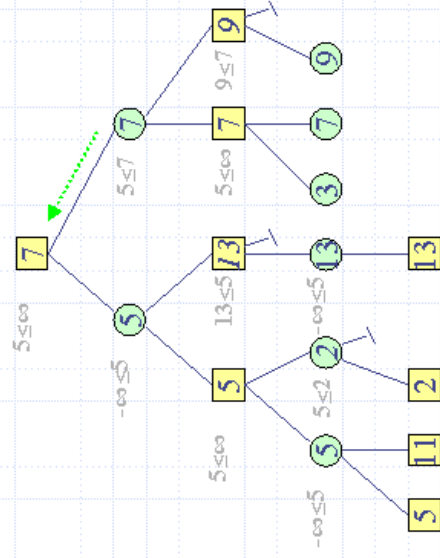
### Alpha-beta pruning example: Step 17

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)  
 $\beta$  is the minimum upper bound of possible solutions (MIN plays)



### Alpha-beta pruning example: Step 18

$\alpha$  is the maximum lower bound of possible solutions (MAX plays)  
 $\beta$  is the minimum upper bound of possible solutions (MIN plays)



Initial call is to MAX\_VALUE(root\_node,  $-\infty$ ,  $\infty$ )  
Alpha  $\leq$  Beta, otherwise cut-off search

## Alpha-Beta pruning

**function** MAX-VALUE(*state*, *game*,  $\alpha$ ,  $\beta$ ) **returns** the minimax value of *state*

**inputs:** *state*, current state in game

*game*, game description

$\alpha$ , the best score for MAX along the path to *state*

$\beta$ , the best score for MIN along the path to *state*

**if** GOAL-TEST(*state*) **then return** EVAL(*state*)

**for each** *s* **in** SUCCESSORS(*state*) **do**

$\alpha \leftarrow$  MAX( $\alpha$ , MIN-VALUE(*s*, *game*,  $\alpha$ ,  $\beta$ ))

**if**  $\alpha \geq \beta$  **then return**  $\beta$

**end**

**return**  $\alpha$

*Max of min values*

---

**function** MIN-VALUE(*state*, *game*,  $\alpha$ ,  $\beta$ ) **returns** the minimax value of *state*

**if** GOAL-TEST(*state*) **then return** EVAL(*state*)

**for each** *s* **in** SUCCESSORS(*state*) **do**

$\beta \leftarrow$  MIN( $\beta$ , MAX-VALUE(*s*, *game*,  $\alpha$ ,  $\beta$ ))

**if**  $\beta \leq \alpha$  **then return**  $\alpha$

**end**

**return**  $\beta$

*Min of max values*

# Analysis of Alpha Beta Pruning

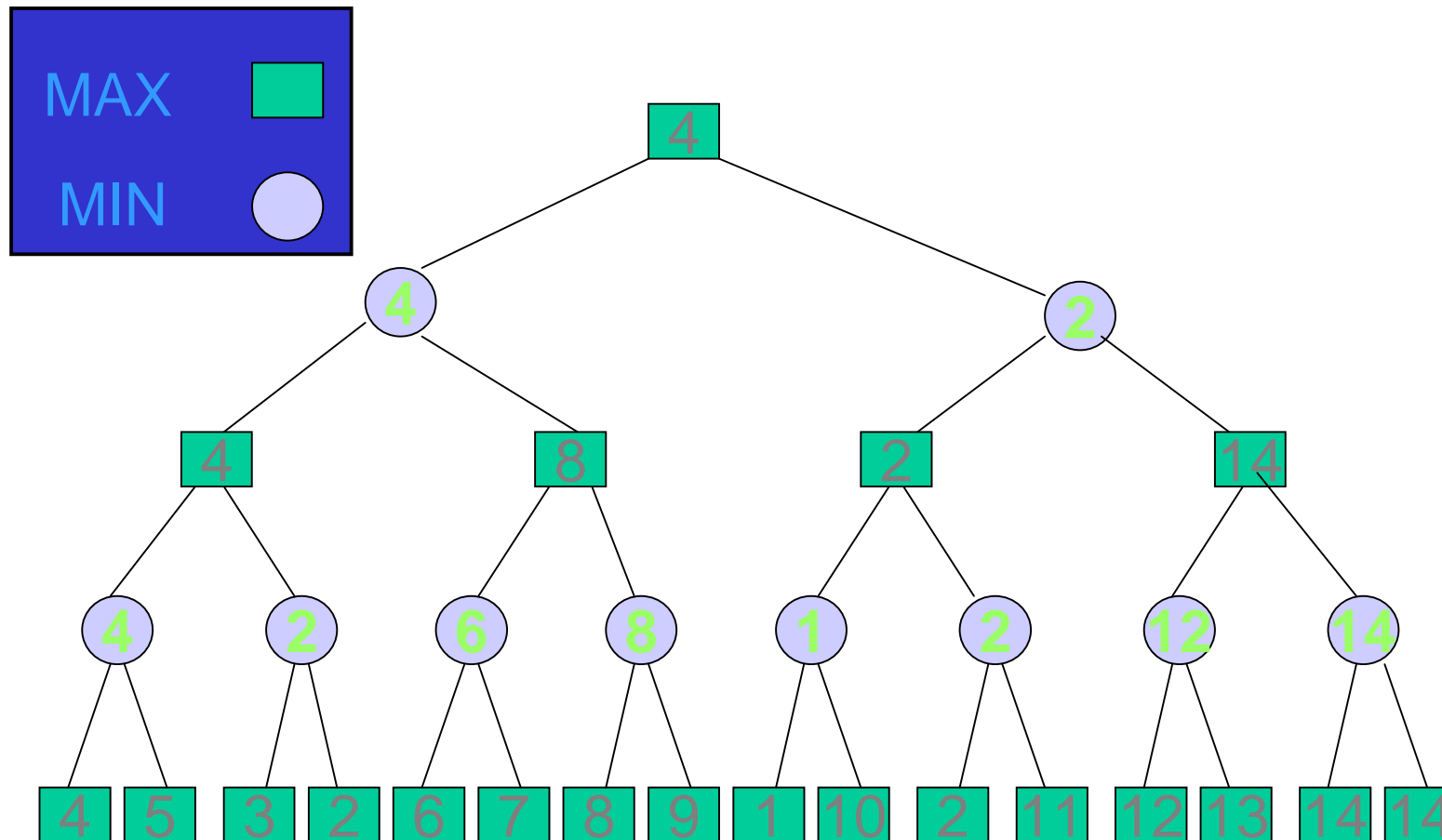
- Knuth, D. E., and Moore, R. W. "An analysis of Alpha-Beta pruning," *Artificial Intelligence*, 6:293-326, 1975.
  - Most results for perfectly ordered trees
  - Some for randomly ordered trees

# Performance analysis of Alpha-Beta Pruning

- Since alpha-beta pruning performs a minimax search while pruning much of the tree, its effect is to allow a deeper search with the same amount of computation.
- *The question:* how much does alpha-beta improve performance?

# Example of alpha-beta worst case

- Evaluation from left to right



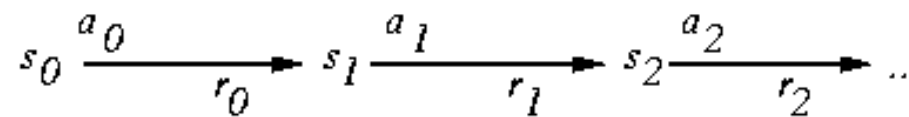
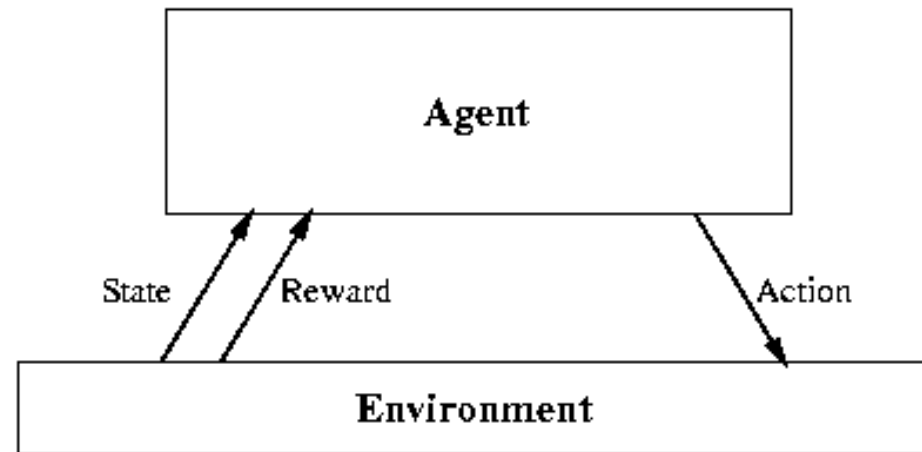
# Minimax value of game trees

- The most natural definition for the average case is that the leaf nodes are randomly ordered.
- Heuristic node ordering would violate this assumption.
- Average case performance is not a prediction of its performance in practice

# Next Class

- Reinforcement learning – Chapter 21

# Agent / Environment Interaction



Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots, \text{ where } 0 \leq \gamma < 1$$

*Different rewards  
Emphasize  
different behavior*

# Examples and Properties (1)

Consider learning to choose actions, e.g.,

- Robot learning to dock on battery charger
- Learning to choose actions to optimize factory output
- Learning to play Backgammon

Note several problem characteristics:

- Delayed reward
- Opportunity for active exploration
- Possibility that state only partially observable
- Possible need to learn multiple tasks with same sensors/actuators

# Examples and Properties (2)

- Each unique location is NOT a state
- Can be applied to any autonomous agent.
  - Software agents, web-crawlers
- Not unlike Pavlovian conditioning
  - Condition the learner to optimal behavior
- Agent receives indirect & delayed rewards, no guidance or corrections given.
  - Different than supervised learning
- Agent does not necessarily know how their actions change their state.
- Recent move towards competing agents.

# A Successful Example

[Tesauro, 1995]

Learn to play Backgammon

Immediate reward

- +100 if win
- -100 if lose
- 0 for all other states

Trained by playing 1.5 million games against itself

Now approximately equal to best human player

# Different Environments and Agents

- Environments
  - Direct or indirect rewards/training
- Agents
  - Deterministic versus non-deterministic actions.
  - Incomplete knowledge of effect of actions
  - Incomplete knowledge of the state
  - Agent memory or model of the environment
- Regardless of differences always have:
  - $\delta(s_t, a_t)$  chooses the action to perform
  - $r(s_t, a_t)$  provides the reward given the action
  - Agent may not know either function