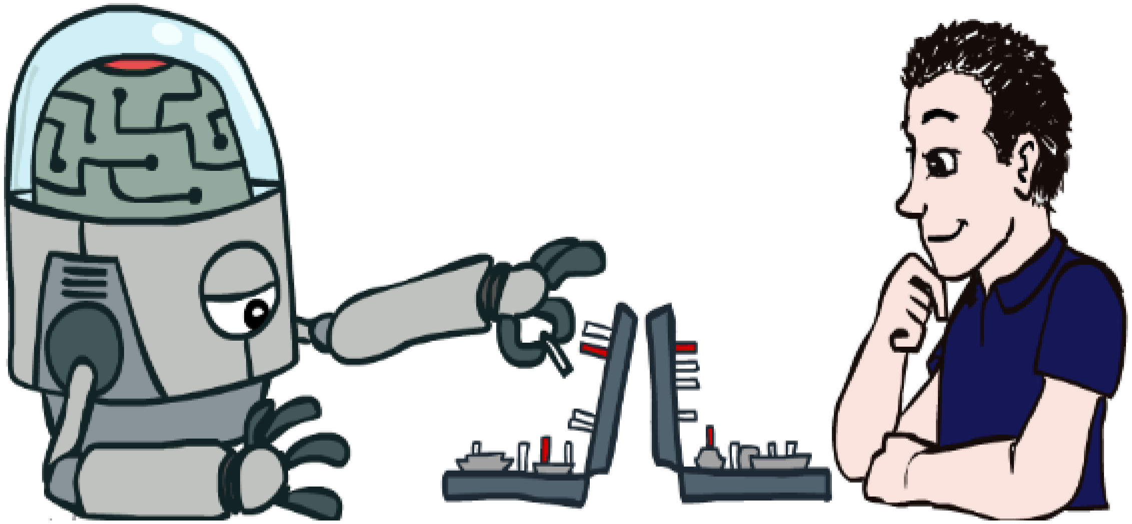
ARTIFICIAL INTELLIGENCE- CS411

Prof. Alaa Sagheer



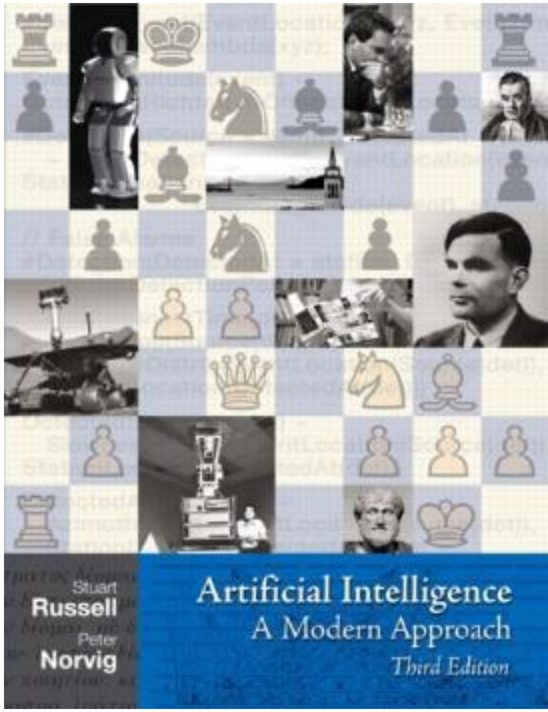
Artificial Intelligence "CS 411"

- **Textbook**:

S. Russell and P. Norvig

Artificial Intelligence: A Modern Approach

Prentice Hall, 2010, *Third Edition*



- **Place:** Online Lectures
- **Grading:**

  Class Activity (5%),
  Project @ Lab (10%),
  Quizzes @ Class (10%),
  Quizzes @ Lab (15 %),
  Mid-term exam (20%),
  Final exam (40%),

Problem-Solving

Adversarial Search

Minimax Strategy

Lecture I

Adversarial Search

- In which we examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.

- We handled the **multi-agent environments**, in which any given agent will need to consider the actions of other agents and how they affect its own welfare.

- The unpredictability of the other agent can introduce many possible emergency into the agent's problem solving process.

- Here, we should distinguish between **cooperative** and **competitive** multi-agent environments. Competitive environments, in which the agents' goals are in conflict, give rise to **adversarial search** problems- known as **<u>Games</u>.**
  - It is this opposition between the agents' utility functions that makes the situation adversarial.

AI Games

- **Game** theory views any multiagent environment as a game provided that the impact of each agent on the others is "significant," regardless of whether the agents are cooperative or competitive.
- **AI Games** are a specialized kind - deterministic, turn taking, two-player (or two-agent), zero sum games of perfect information
  - ❖ A **zero-sum game** is a mathematical representation of a situation in which a participant's gain (or loss) of utility is exactly balanced by the losses (or gains) of the utility of the other participant.
- In our terminology, this means deterministic, fully observable environments with two agents whose actions must alternate and in which the utility values at the end of the game are always equal and opposite. For example, if one player wins a game of chess (+1), the other player necessarily loses (-1).

Why Games?

- Small defined set of rules,
- Well defined knowledge set,
- Easy to evaluate performance,
- Large search spaces:
  - Too large for exhaustive search
- Fame and Fortune:
  - e.g. Chess

Game as a Search Problem

- **Games have a state space search**
  - Each potential board or game position is a state
  - Each possible move is an operation to another state
  - Hard to solve, the state space can be HUGE!!!!!!!
    - Large branching factor, for example, chess has an average branching factor of about 35, and games often go to 50 moves by each player, so the search tree has about 35^{100} nodes.

Game Vs. Search Problem

- **Unpredictable opponent**
- **Solution is a strategy**
  - Specifying a move for every possible opponent reply
- **Time limits**
  - Unlikely to find the goal…agent must approximate

Optimal Decision in Games

- We will consider games with two players, whom we will call **MAX** and **MIN**. **MA X** moves first, and then they take turns moving until the game is over.
- At the end of the game, points are awarded to the winning player and penalties are given to the loser.

❖ **Problem Formulation**: a game can be formally defined as a kind of search problem with the following components:
  - **Initial state**, includes the board position and identifies the player to move.
  - **Successor function**, returns a list of (move, state) pairs, each indicating a legal move and the resulting state.
  - **Goal (terminal) test,** which determines when the game is over.
  - **Utility (objective) function**, which gives a numeric value for the terminal states. In chess, the outcome is a win, loss, or draw, with values +1,-1, or 0 *.

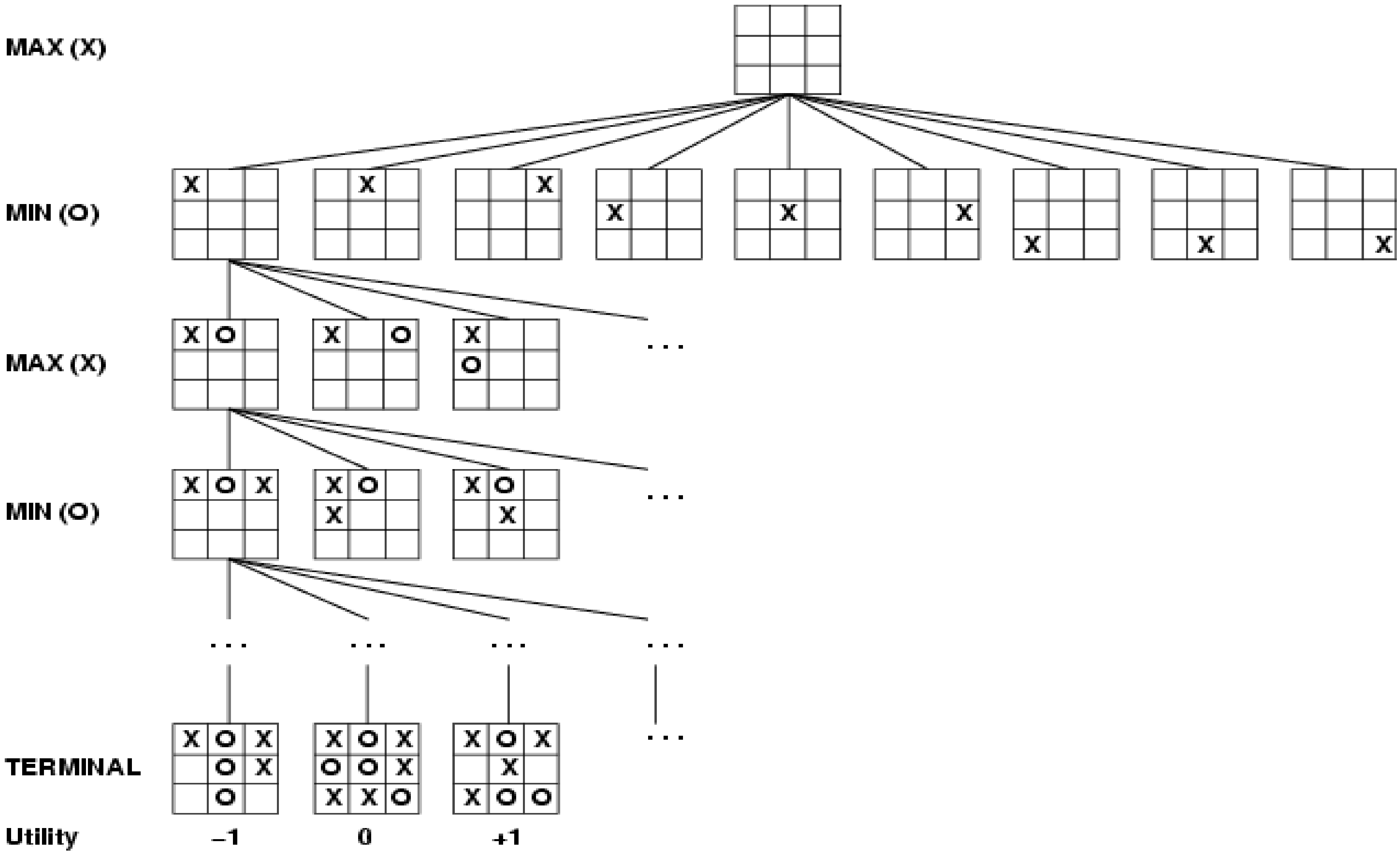
❖ **Search objective:**

- Find the sequence of player's decisions (moves) maximizing its utility (obj.),
- Consider the opponent's moves and their utility

Game Tree (2-player, deterministic, turns)

- The root of the tree is the initial state
  - Next level is all of MAX' s moves
  - Next level is all of MIN' s moves
  - …
- Example: Tic-Tac-Toe
  - Root has 9 blank squares (MAX)
  - Level 1 has 8 blank squares (MIN)
  - Level 2 has 7 blank squares (MAX)
  - …
- Utility function:
  - win for X is +1
  - win for O is -1

Game Tree (2-player, deterministic, turns)

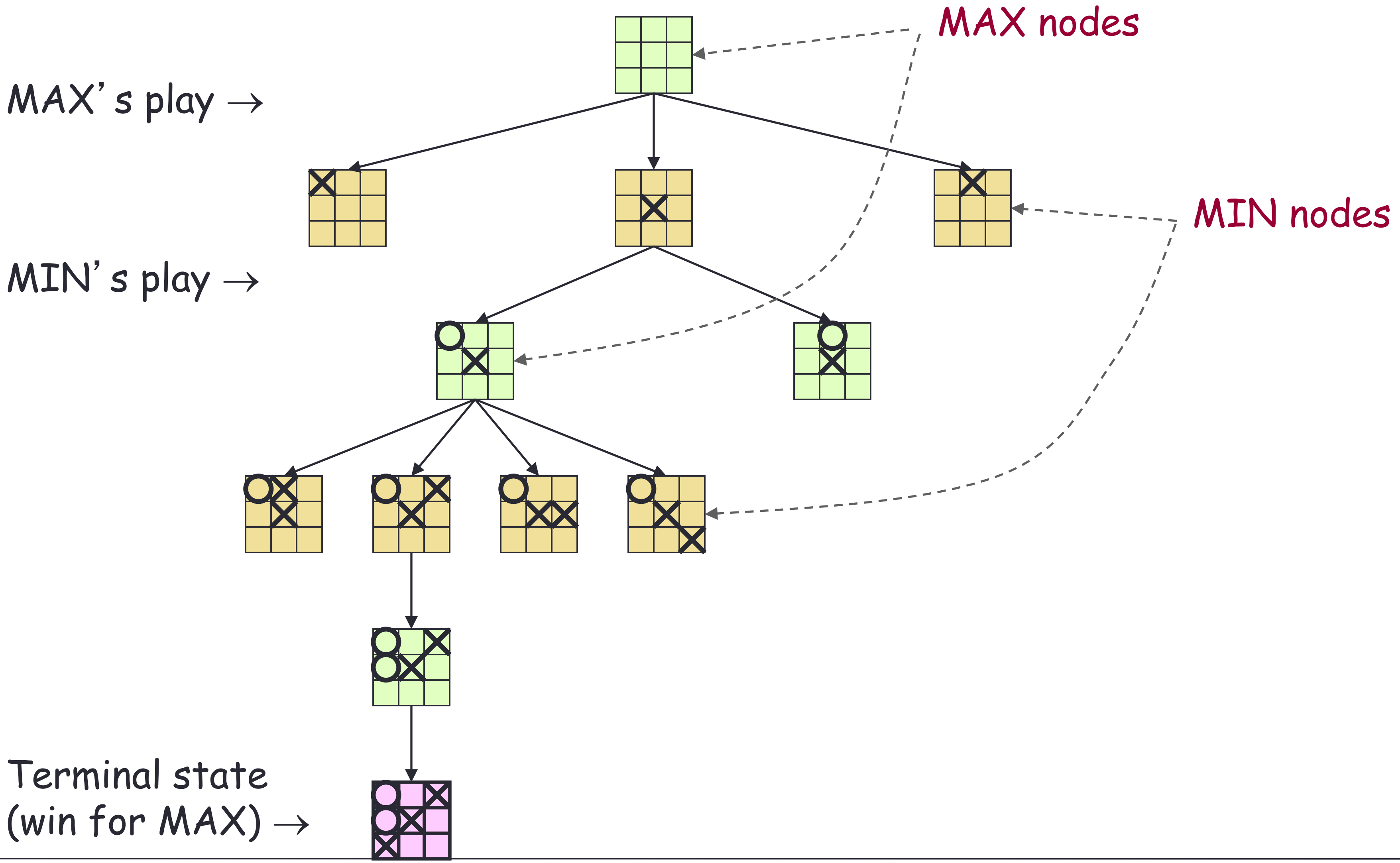


Game Tree: Basic Idea

1) Using the current state as the initial state, build the game tree uniformly to the maximal depth h (called horizon) feasible within the time limit
2) Evaluate the states of the leaf nodes
3) Back up the results from the leaves to the root and pick the best action assuming the worst from MIN

→ Minimax algorithm

Game Tree (2-player, deterministic, turns)

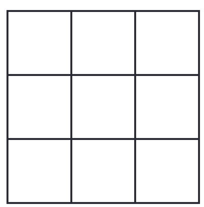


Evaluation Function

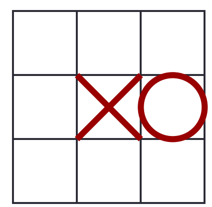
- Function e: state s $\rightarrow$ number e(s)
- e(s) is a heuristic that estimates how favorable s is for MAX
- e(s) $>$ 0 means that s is favorable to MAX (the larger the better)
- e(s) $<$ 0 means that s is favorable to MIN
- e(s) $=$ 0 means that s is neutral

Example: Tic-tac-Toe

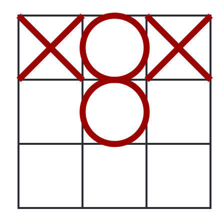
e(s) = number of rows, columns, and diagonals open for MAX
– number of rows, columns, and diagonals open for MIN



8–8 = 0



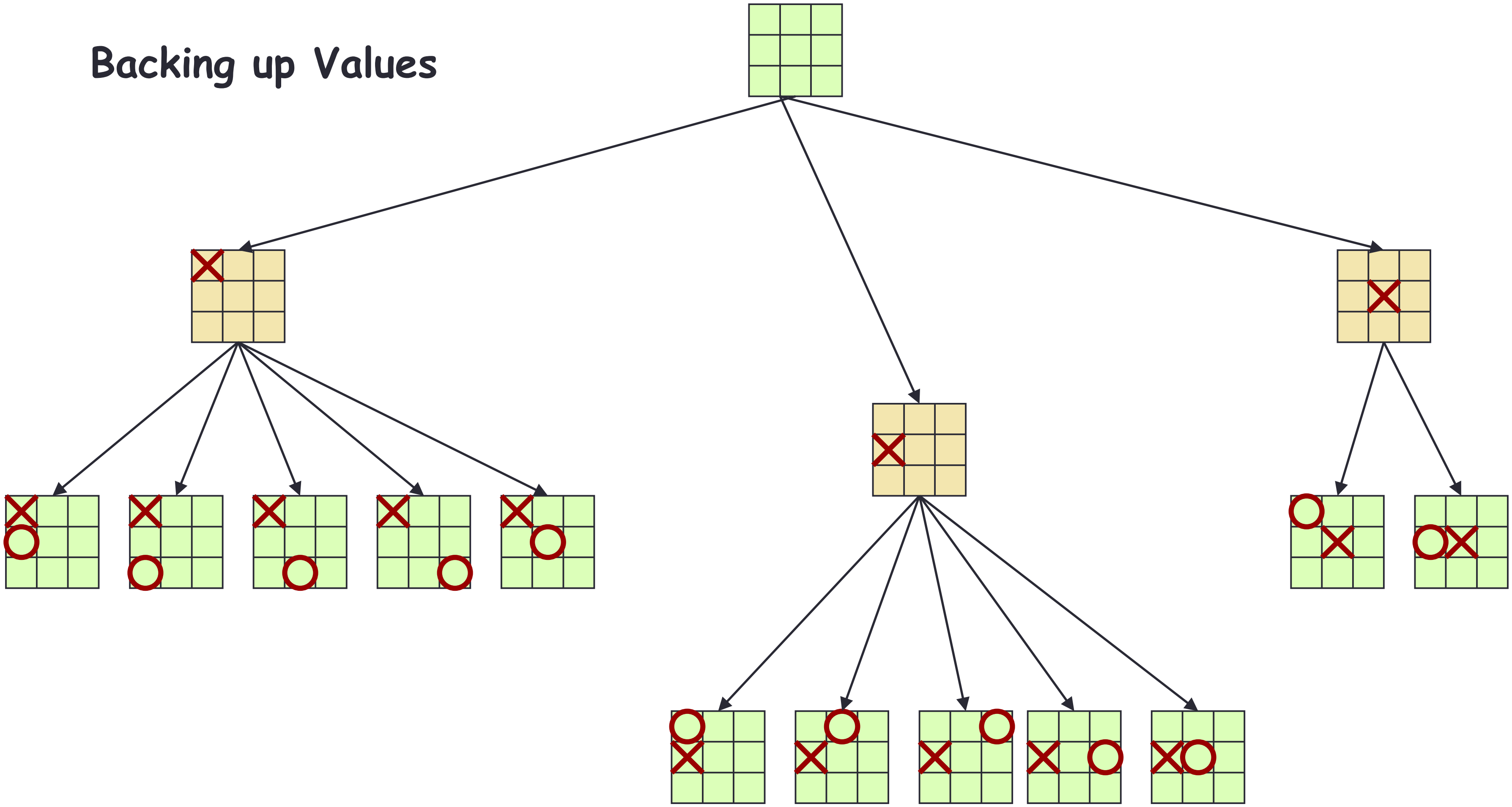
6–4 = 2



3–3 = 0

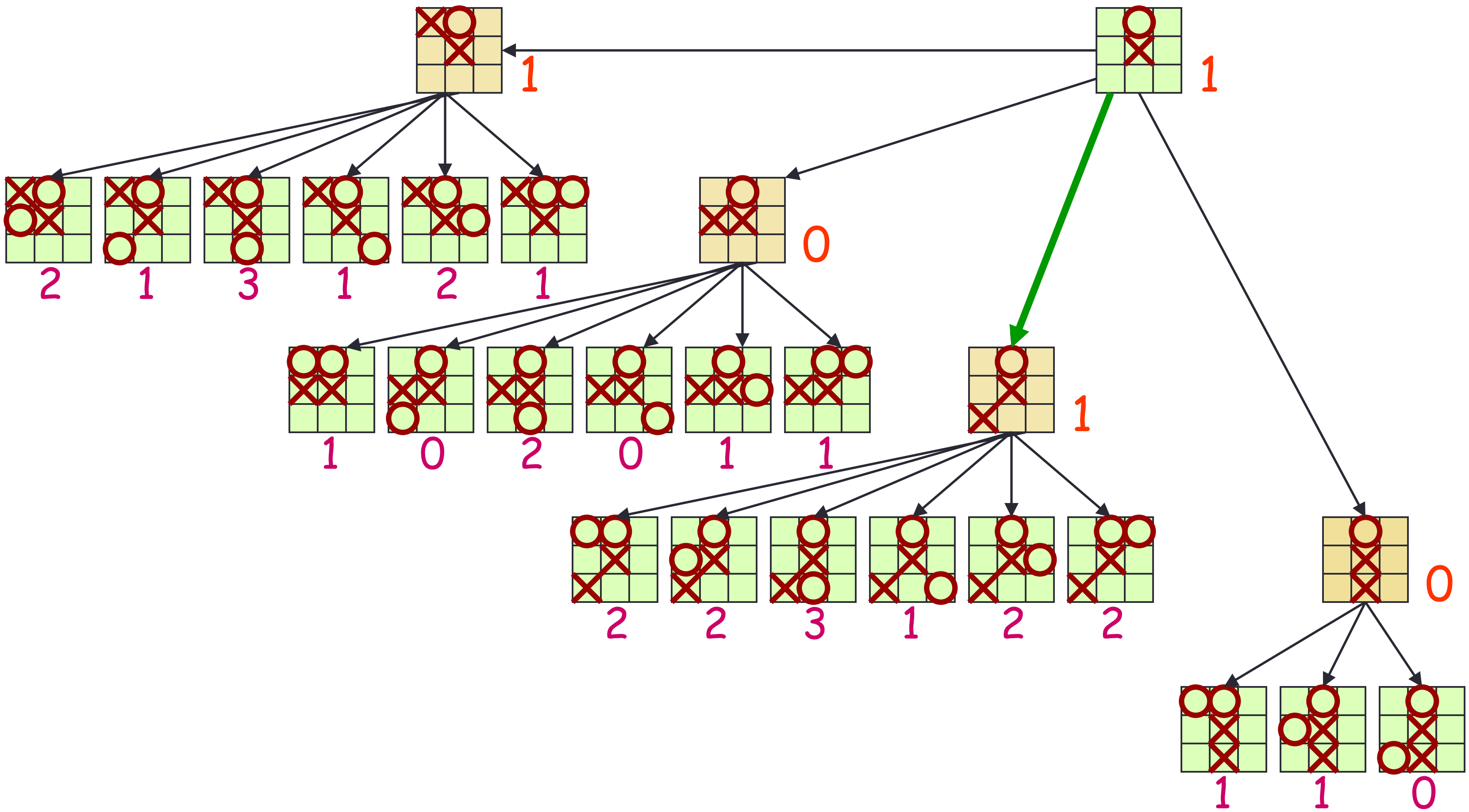
Example: Tic-tac-Toe

Backing up Values



Example: Tic-tac-Toe

Backing up Values



Example: Tic-tac-Toe

O		X
	X	
	O	

O		X
	X	
X	O	

Example: Tic-tac-Toe

O		X
	X	
O		

O		X
X	X	
O		

Example: Tic-tac-Toe

O		X
O		X

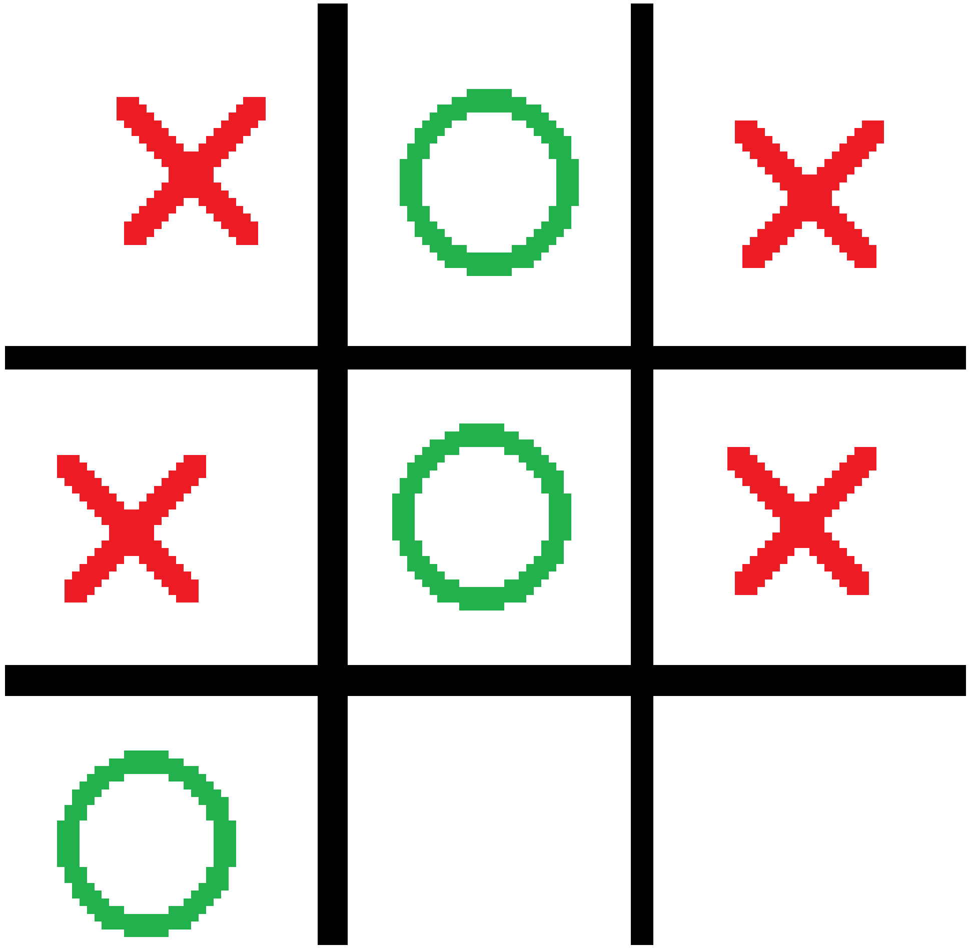
O		X
		X
O		X

Example: Tic-tac-Toe

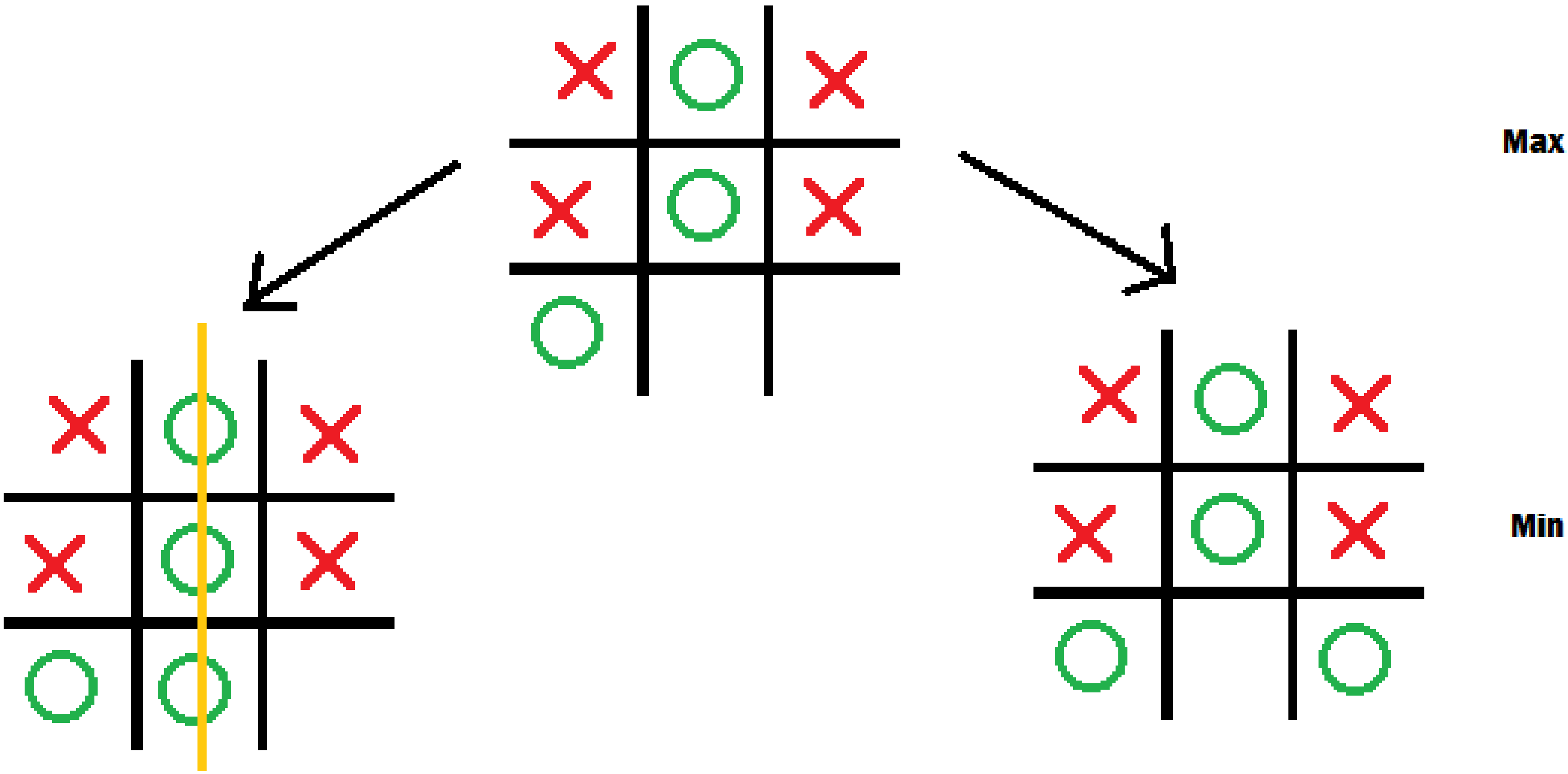
O		
	X	
	O	X

O		X
	X	X
	O	X

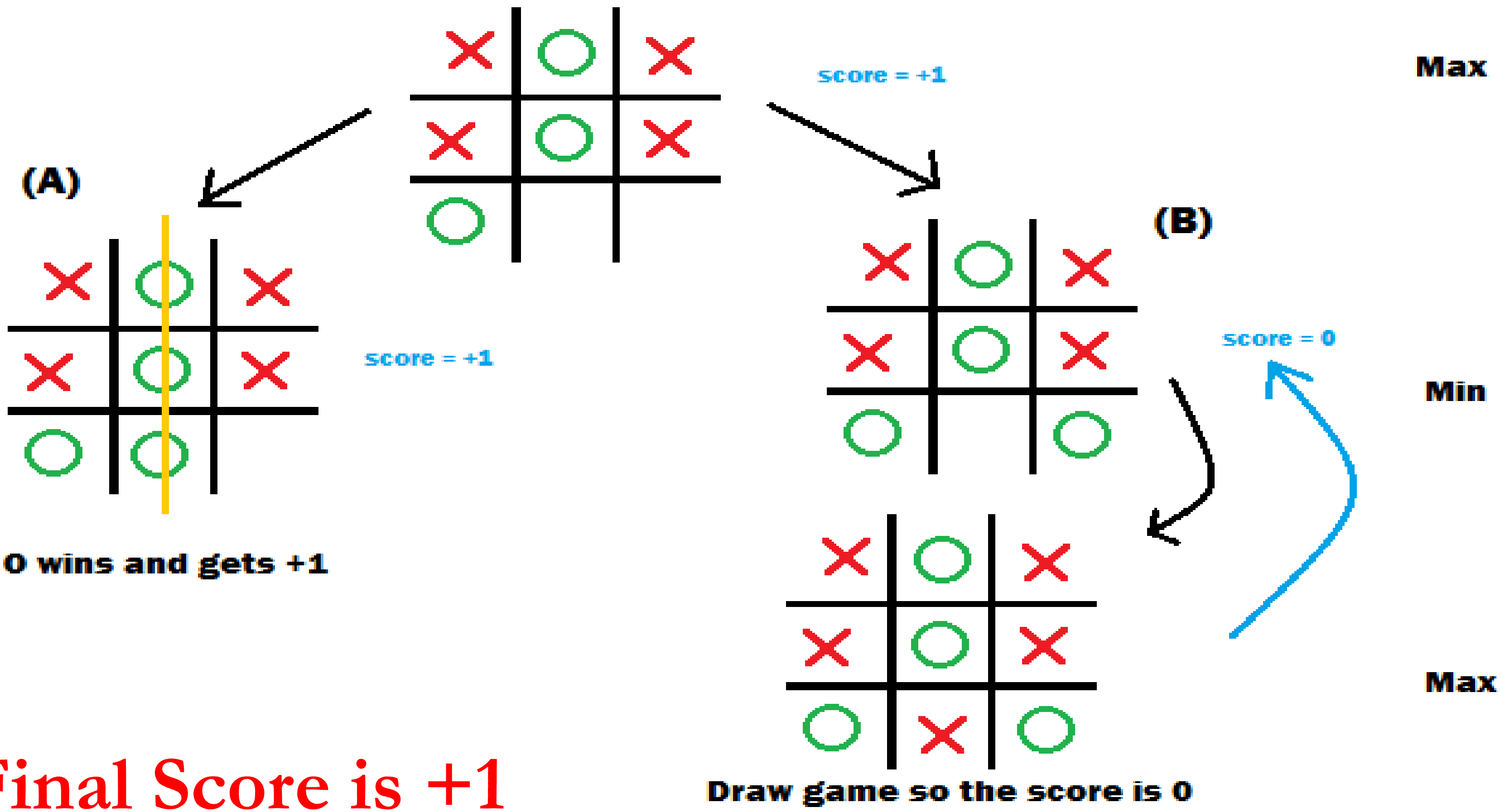
Example: Tic-tac-Toe *



Example: Tic-tac-Toe



Example: Tic-tac-Toe



Final Score is +1

Building an AI algorithm for the Tic-Tac-Toe challenge

https://medium.freecodecamp.org/building-an-ai-algorithm-for-the-tic-tac-toe-challenge-29d4d5adee07

https://medium.freecodecamp.org/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-algorithm-9d690bad4b37

Minimax Algorithm

1. Expand the game tree uniformly from the current state (where it is MAX's turn to play) to depth *h*
2. Compute the evaluation function at every leaf of the tree
3. Back-up the values from the leaves to the root of the tree as follows:
    a. A MAX node gets the <u>maximum</u> of the evaluation of its successors
    b. A MIN node gets the <u>minimum</u> of the evaluation of its successors
4. Select the move toward a leaf node that has the largest backed-up value

Animation!

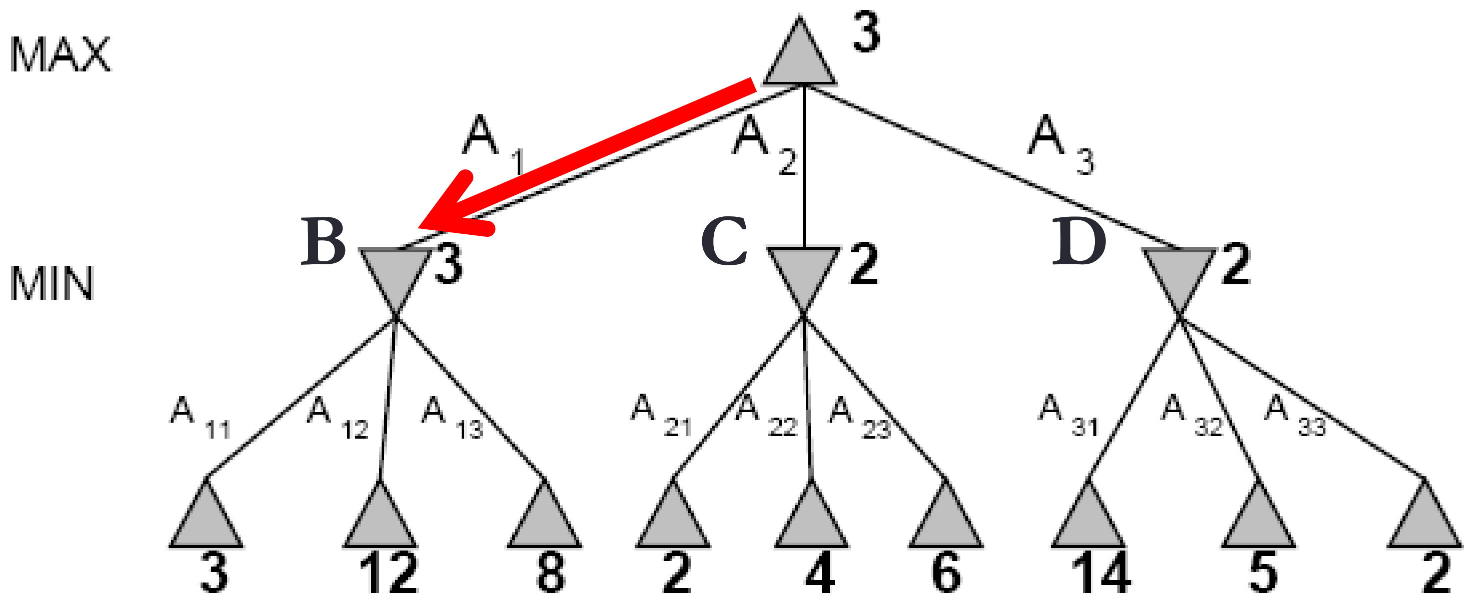
https://www.youtube.com/watch?v=zDskcx8FStA

Minimax Strategy

- The possible moves for MAX at the root node are labeled A_1, A_2, and A_3. The possible replies to A_1 for MIN are A_{11}, A_{12}, A_{13} and so on. The utilities of the terminal states in this game range from 2 lo 14.
- The first MIN node, labeled **B**, has three successors with values 3, 12, and 8, <u>so its minimax value is 3</u> (Now, we can infer that the value of the root is at least 3). Similarly, the other two MIN nodes have minimax value 2. The root node is a MAX node; its successors have minimax values 3, 2, and 2; so it has a minimax value of 3.

Minimax's decision is A_1

<u>Because it leads to highest Minimax value</u>.



Minimax: Recursive Implementation

```
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))
```

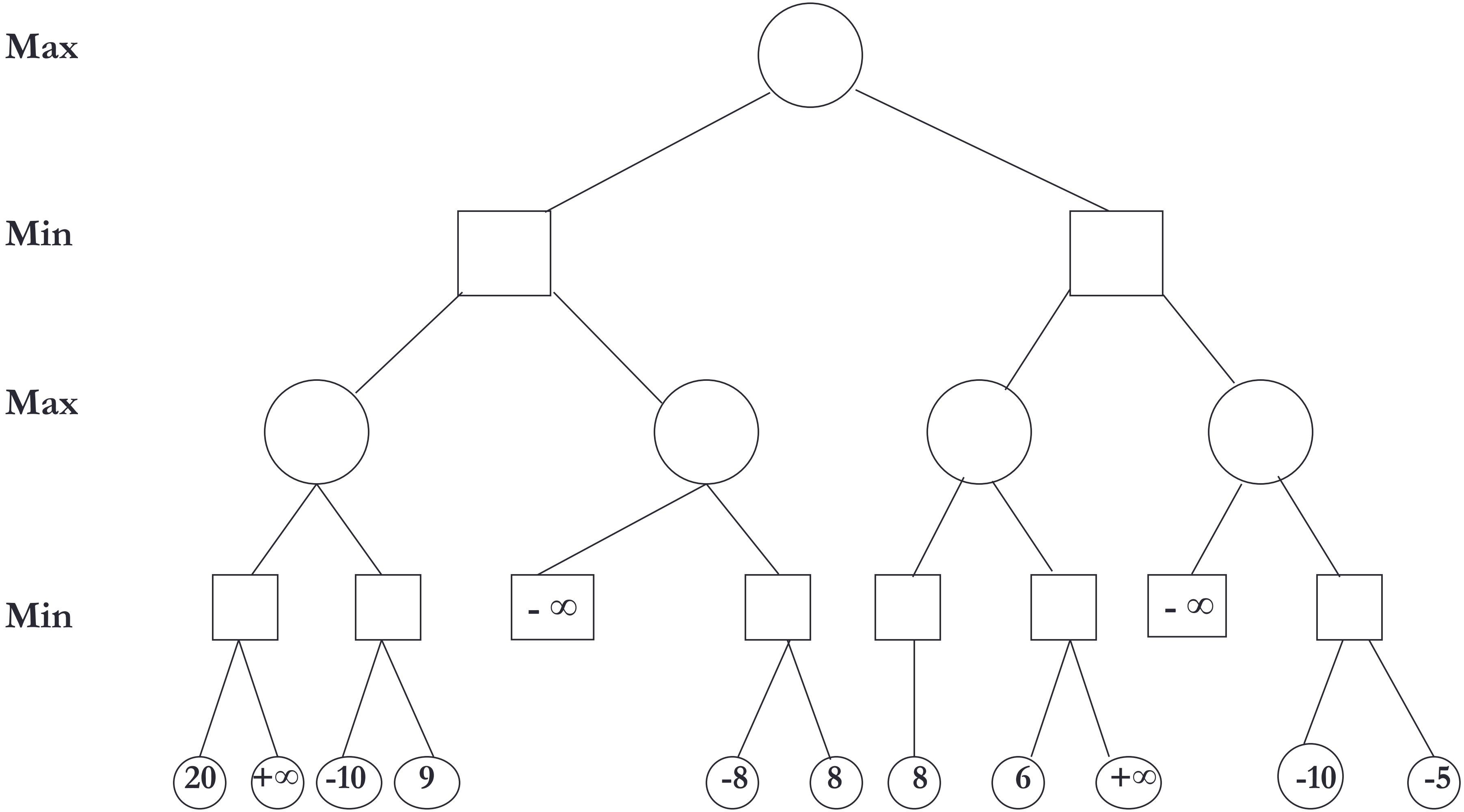
```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v
```

```
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
```

Properties of Minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first exploration)
- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible

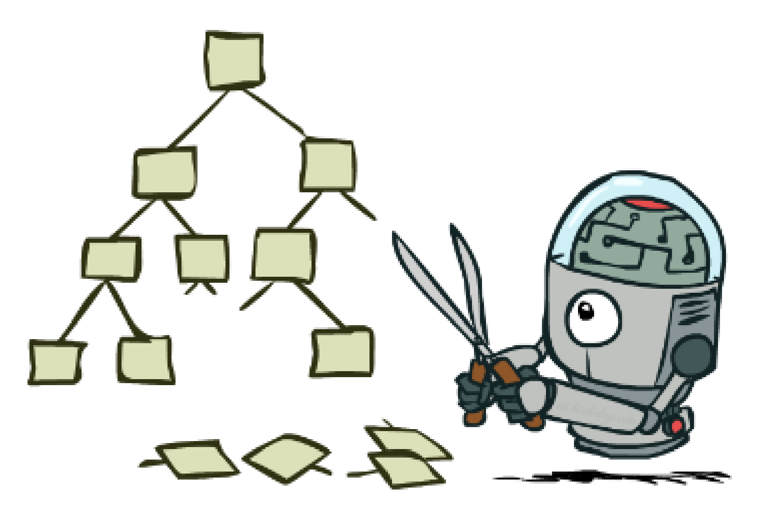
Exercise 1



Problem-Solving

Adversarial Search

Alpha-Beta Pruning



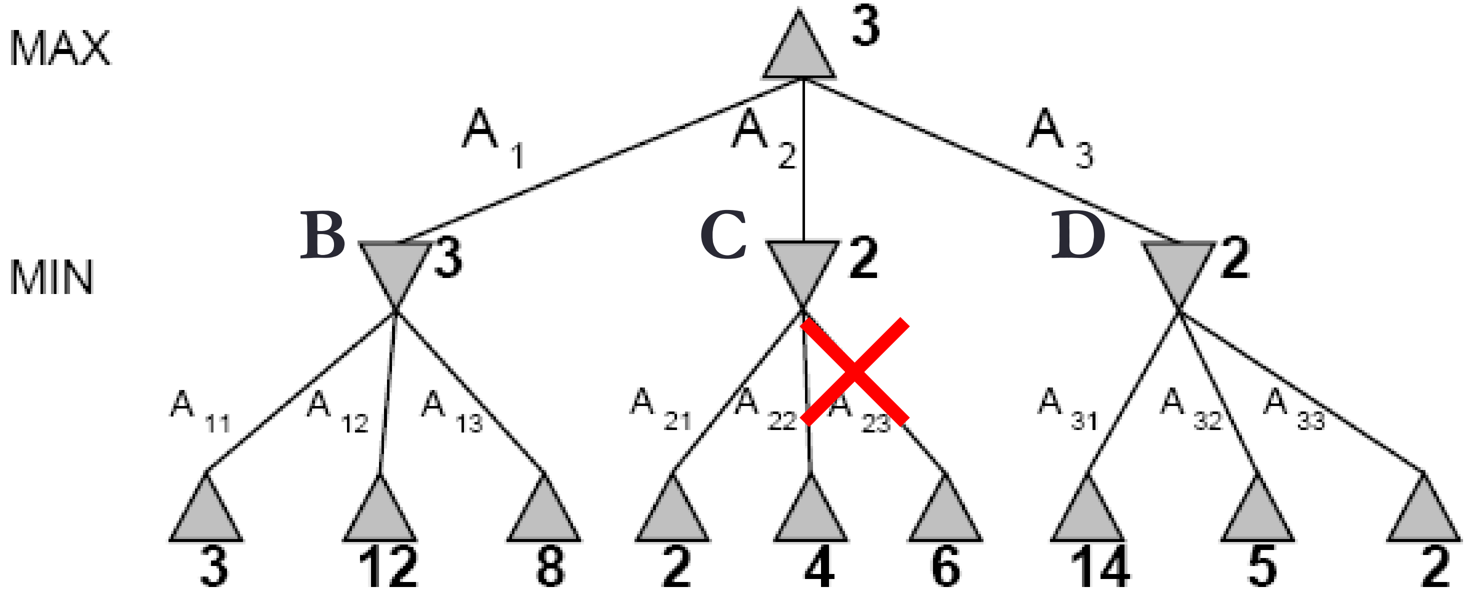
Introduction

- The problem with minimax search is that the number of game states that has to examine is exponential in the number of moves.
- Use pruning to eliminate large parts of the tree from consideration,
- In other words, it is possible to compute the correct minimax decision without looking at every node in the game tree.

Alpha-Beta Pruning

Please pay attention that: It returns the same move as Minimax would, but prunes away branches that cannot possibly influence the final decision.

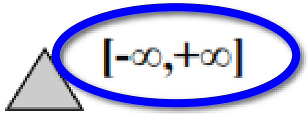
α-β Pruning vs. Minimax



α-β Pruning

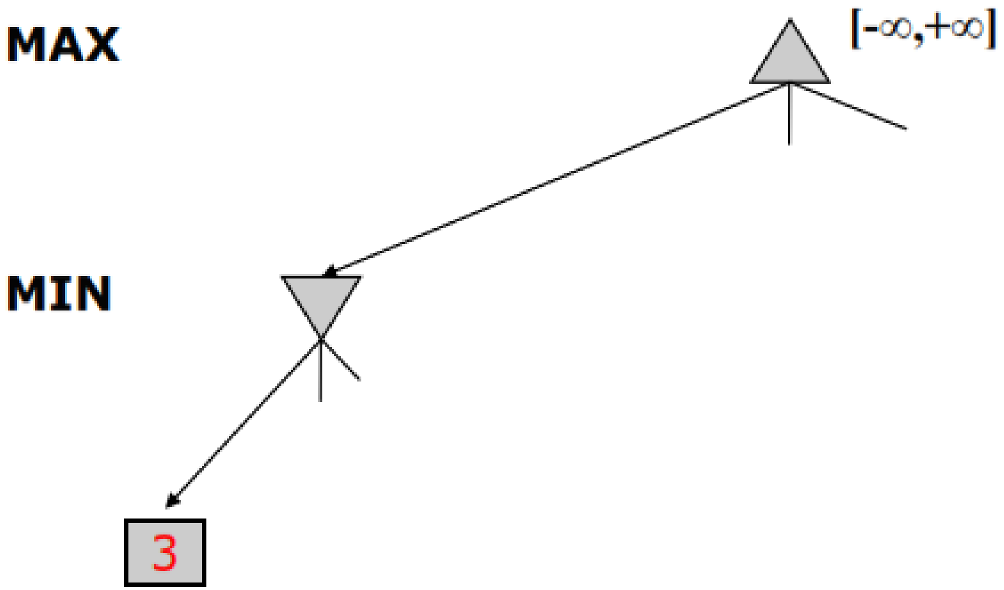
Range of Possible values

MAX

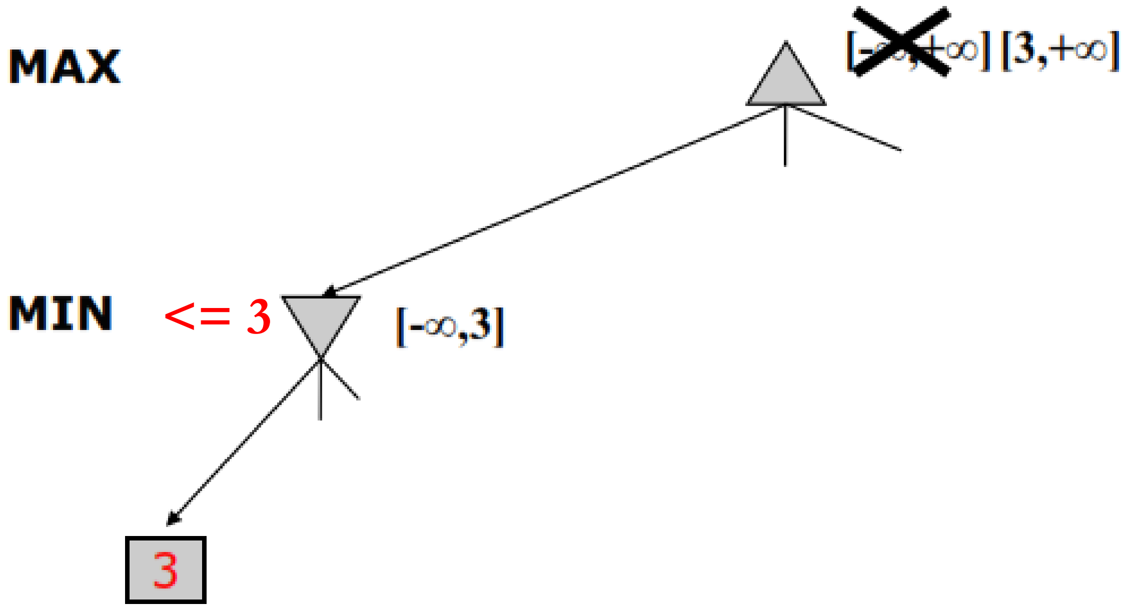


MIN

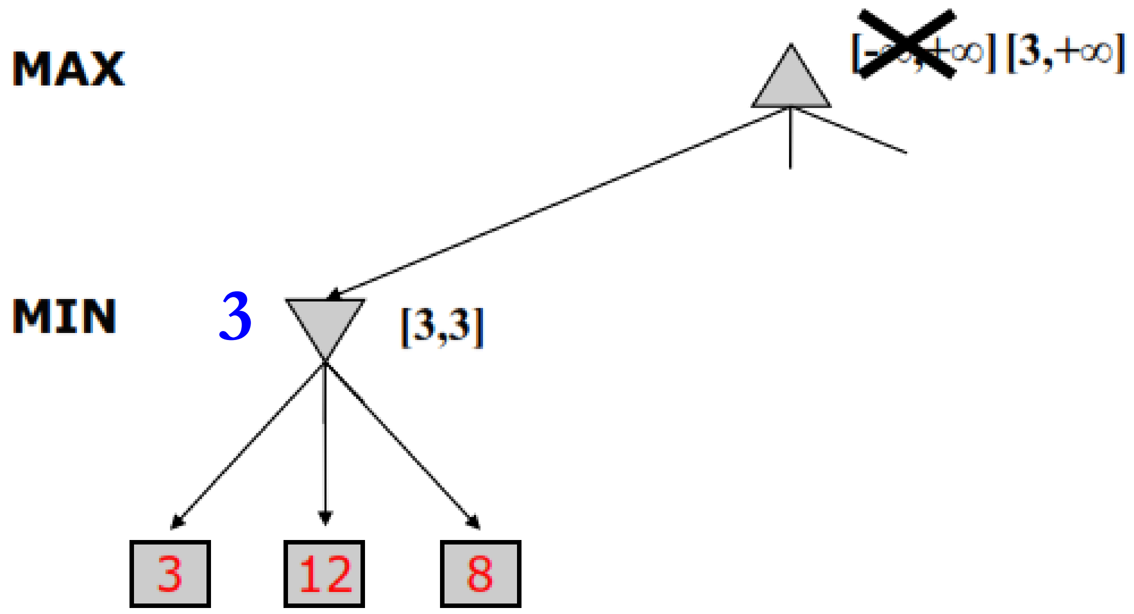
α-β Pruning



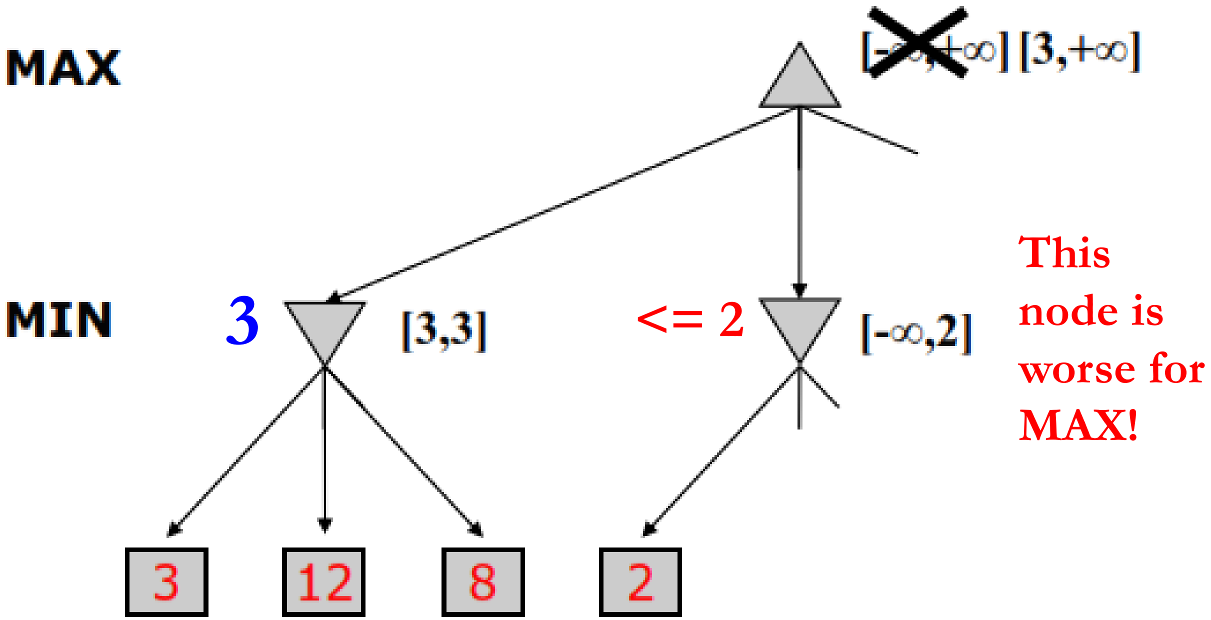
α-β Pruning



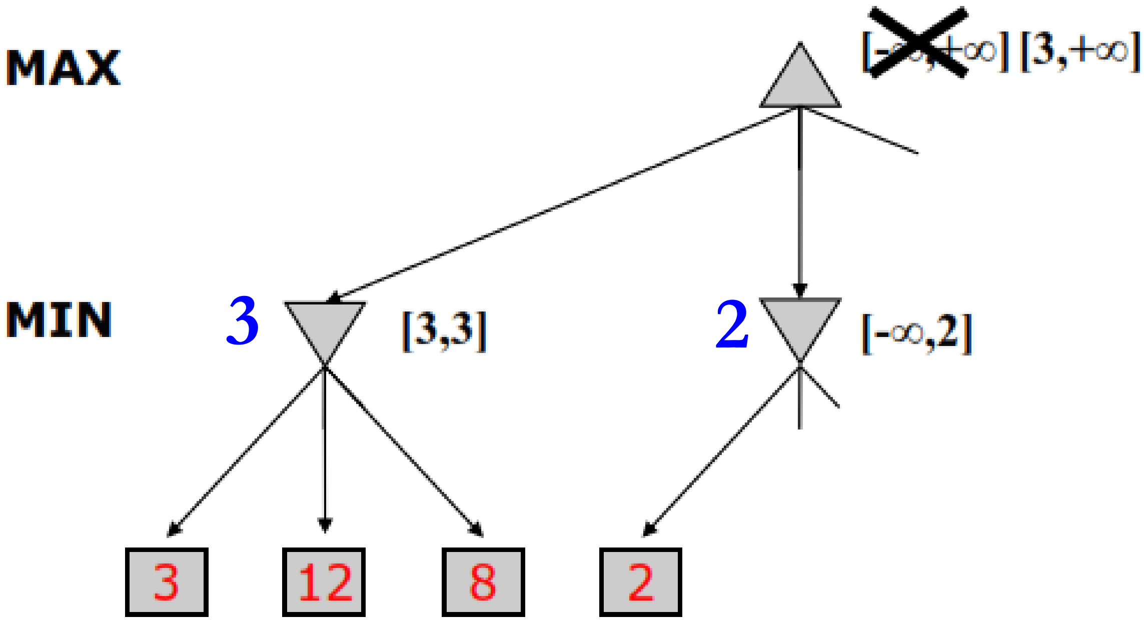
α-β Pruning



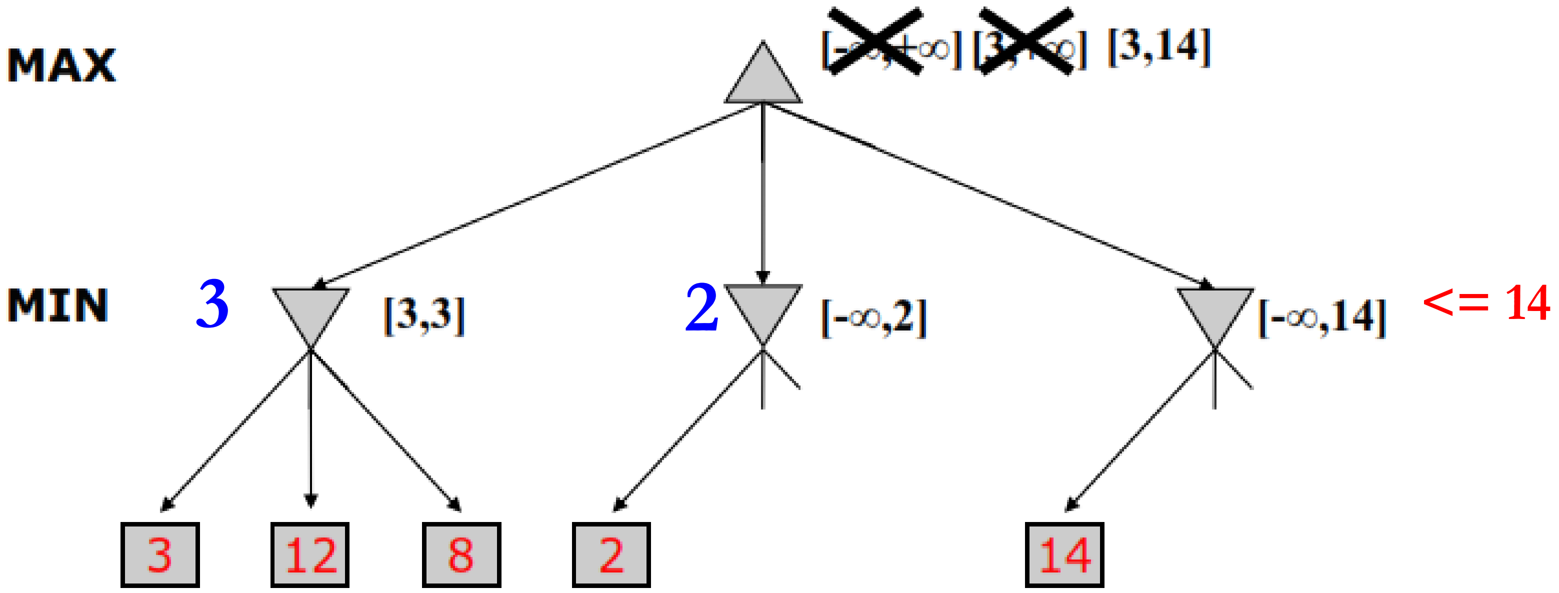
α-β Pruning



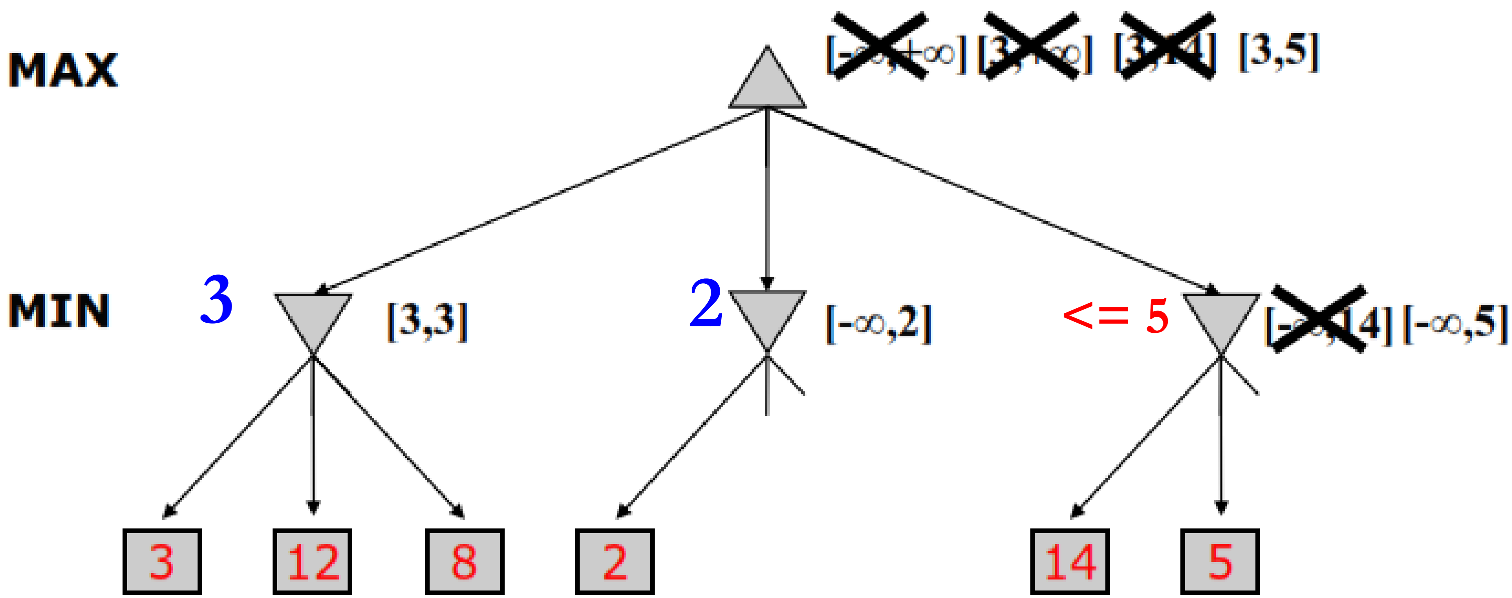
α-β Pruning



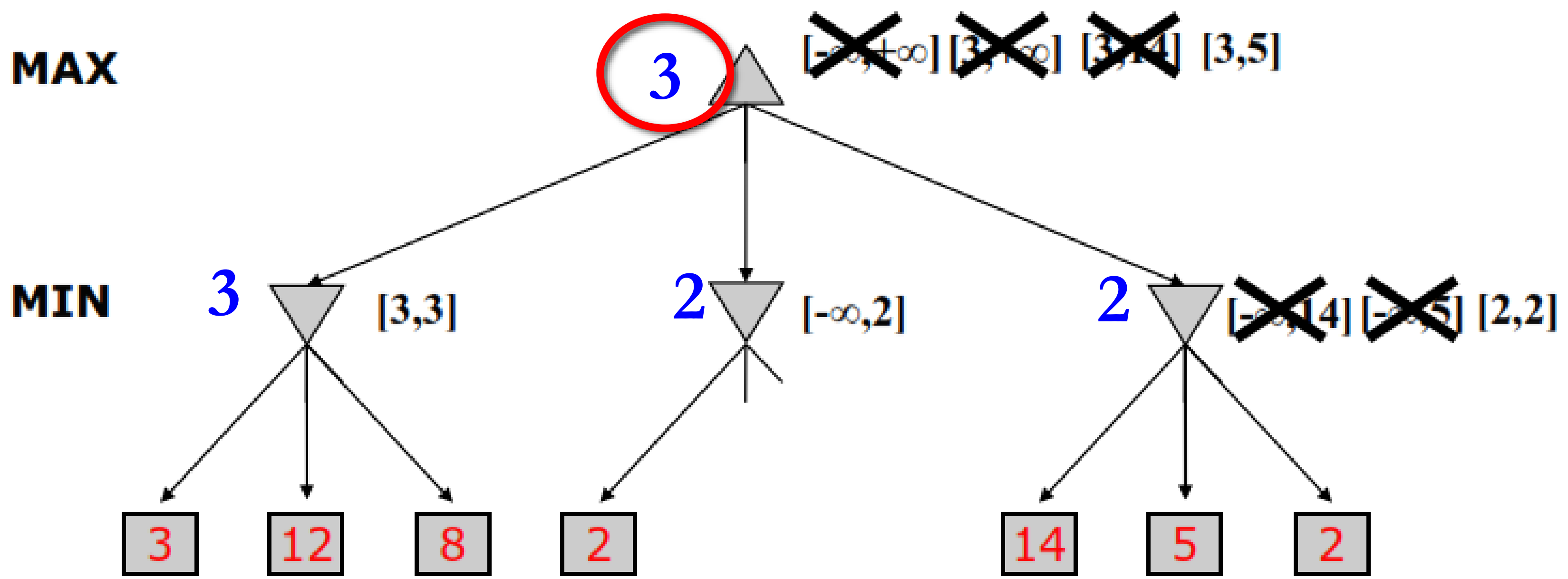
α-β Pruning



α-β Pruning



α-β Pruning



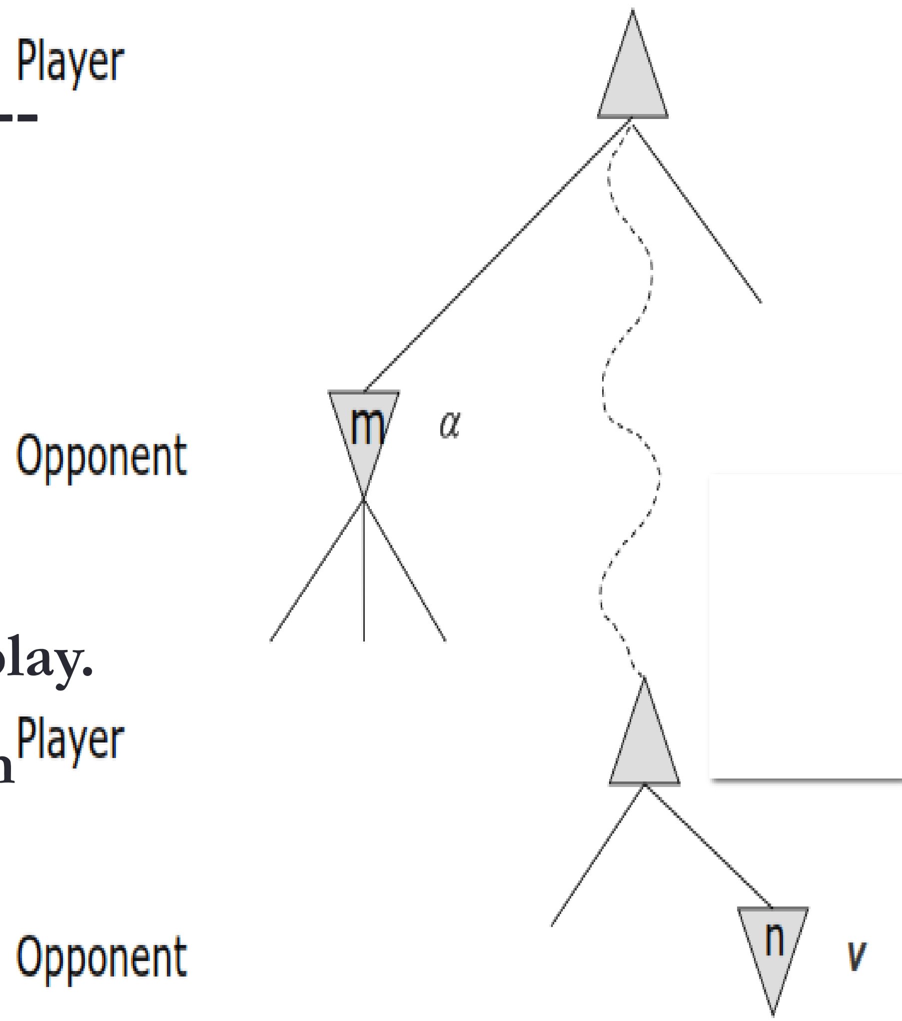
α-β pruning: General Principle

Consider a node *n* in the tree ---

- **If player has a better choice at:**
  - Parent node of n,

  Or

  - Any choice point further up
- **Then *n* will never be reached in play.**
- **Hence, when that much is known about *n*, it can be pruned.**



What is α and β?

α = highest-value choice found so far [1] at any point on the path to the root [2]

for **MAX** (initially, α = − infinity)

= best already explored option along path to the root for maximizer.

β = lowest-value choice found so far [1] at any point on the path to the root [2]

for **MIN** (initially, β = + infinity)

= best already explored option along path to the root for minimizer.

- **Pass current values of α and β down to child nodes during search. Update values of α and β during search:**
  - MAX updates α at MAX nodes
  - MIN updates β at MIN nodes

- **Prune remaining branches at a node when** $\alpha \geq \beta$

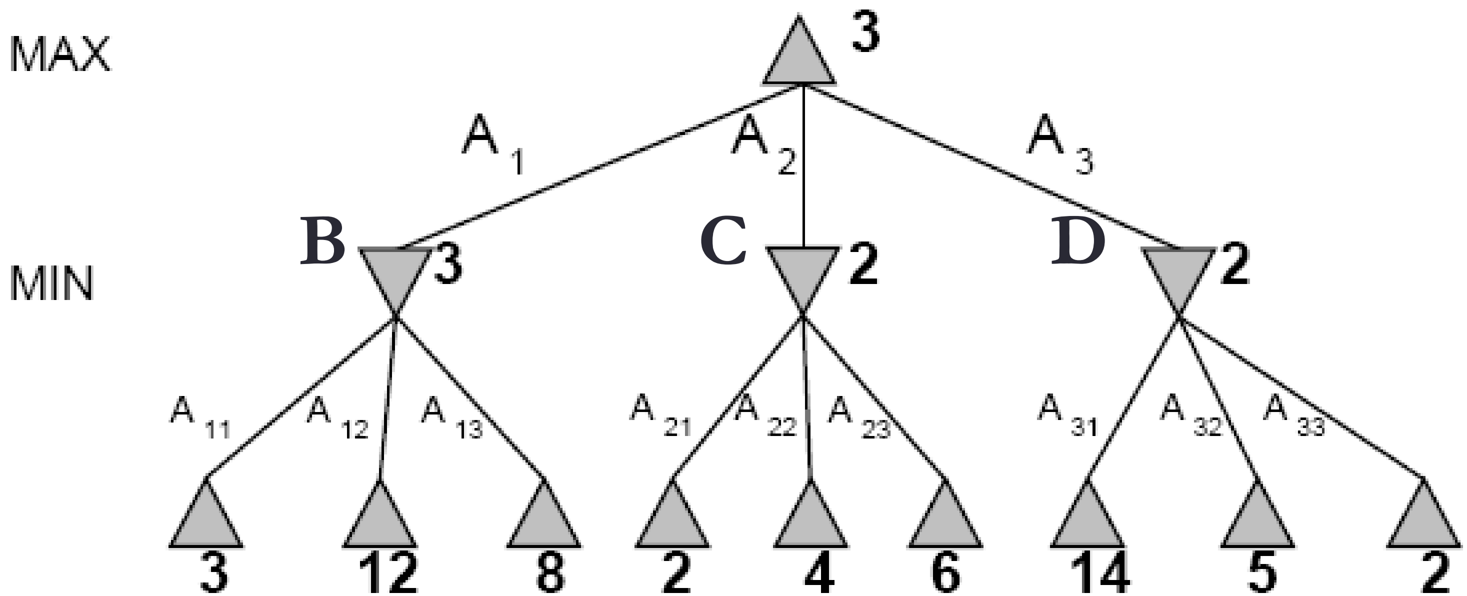
When to Prune ?

❖ **Prune whenever $\alpha \geq \beta$**

Prune below a Max/Min node when the α value becomes greater than or equal to the β value of its ancestors.

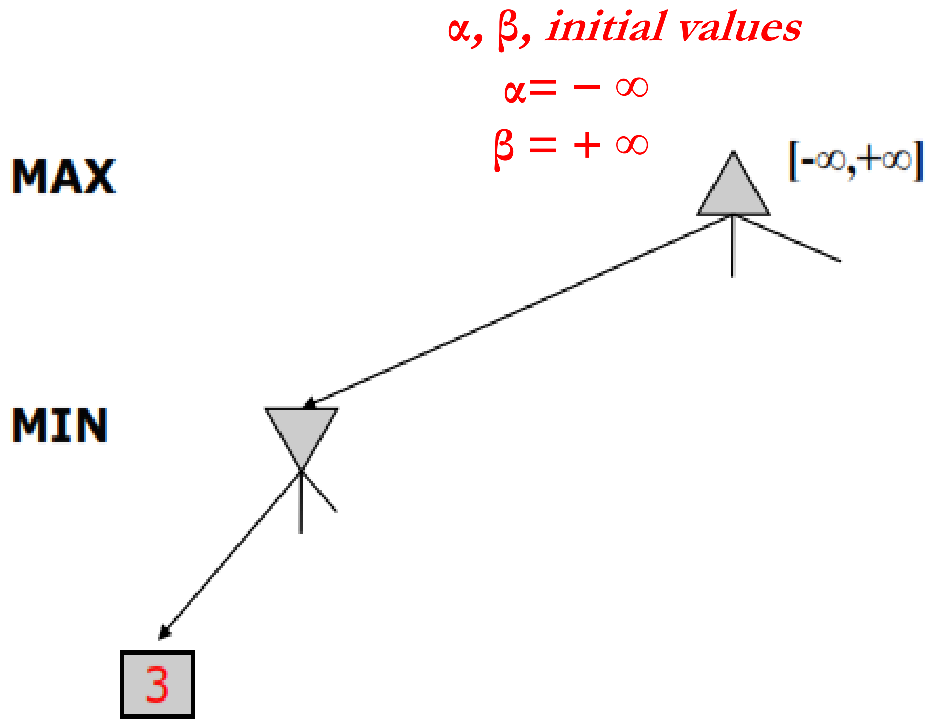
α-β Pruning Example - Revisited

Let the two unevaluated successors of node C have values *x* and *y* and let *z* be the minimum of x and *y*. The value of the root node is given by



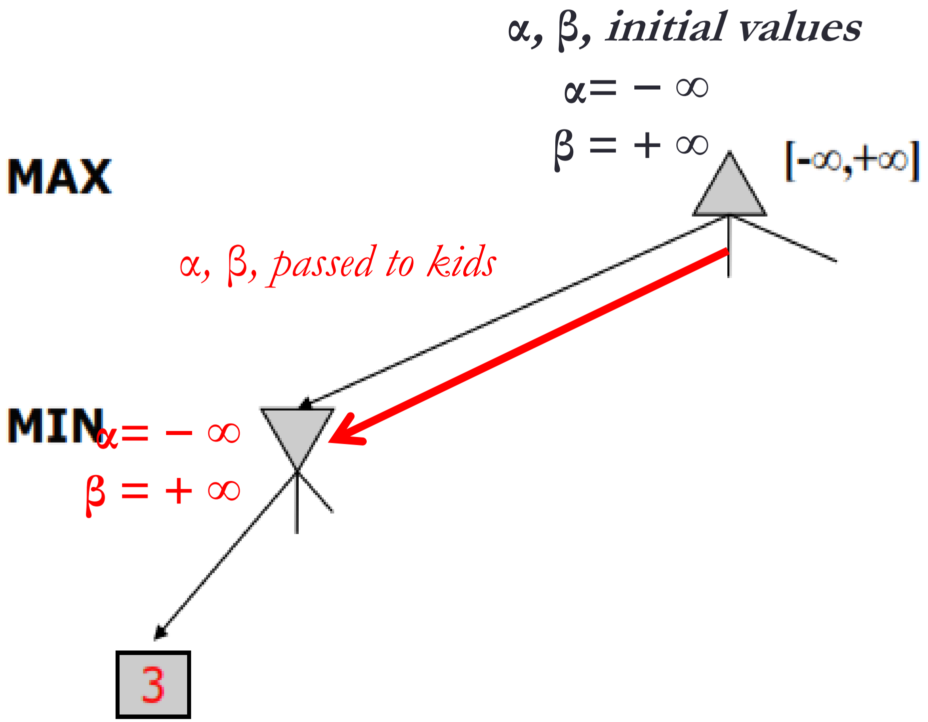
α-β Pruning Example - Revisited

- Do DF-search until first leaf



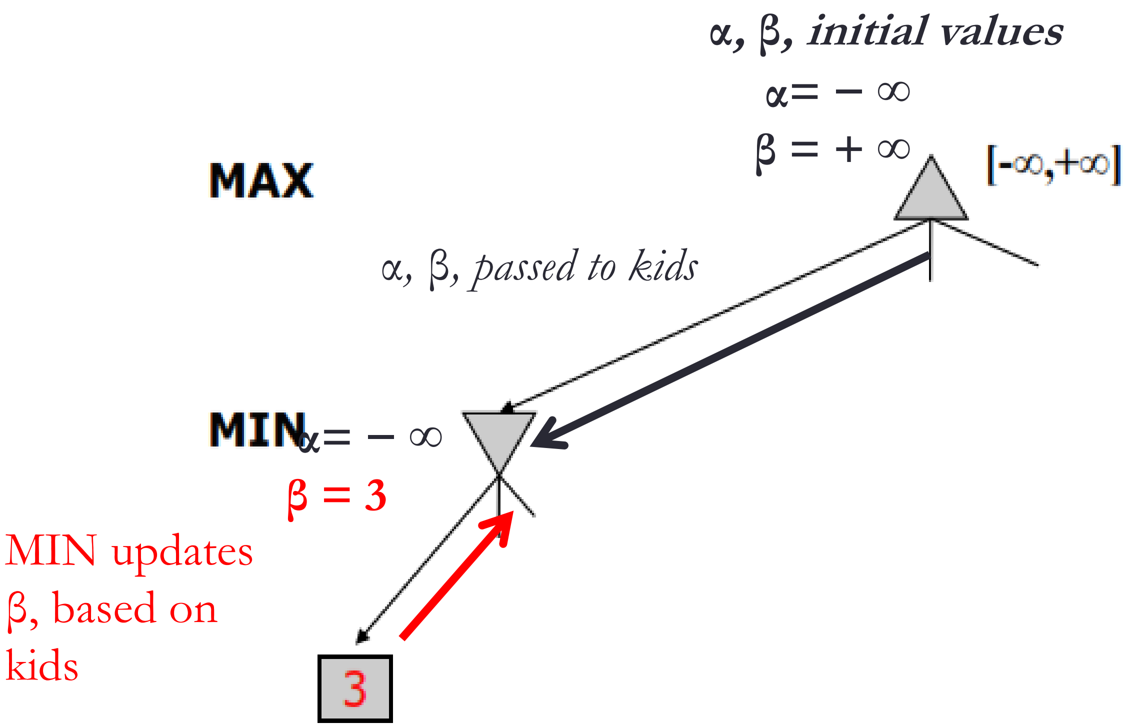
α-β Pruning Example - Revisited

- Do DF-search until first leaf



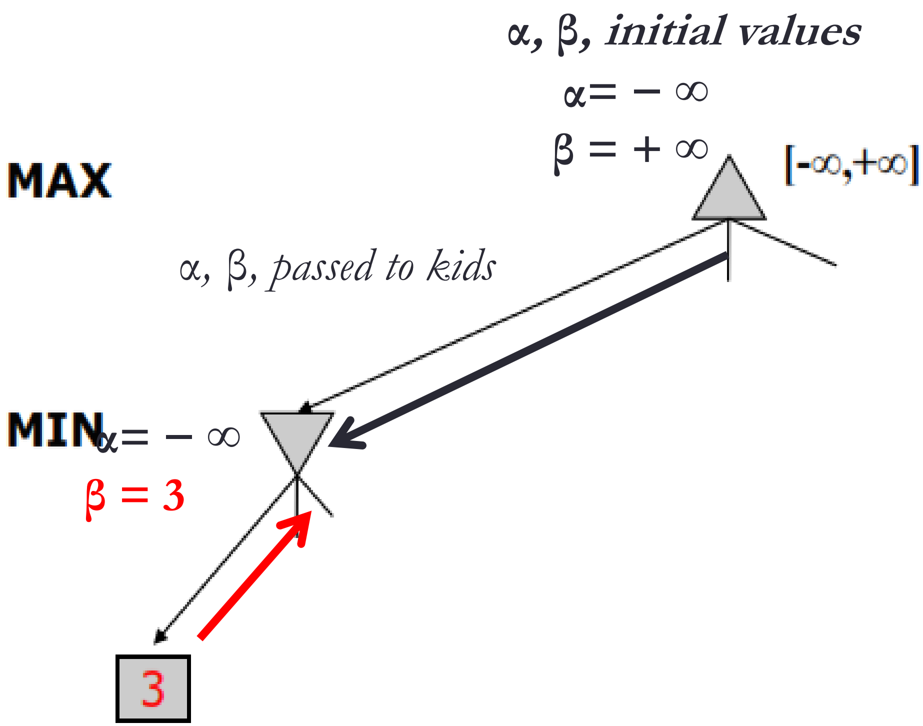
α-β Pruning Example - Revisited

- **Do DF-search until first leaf**

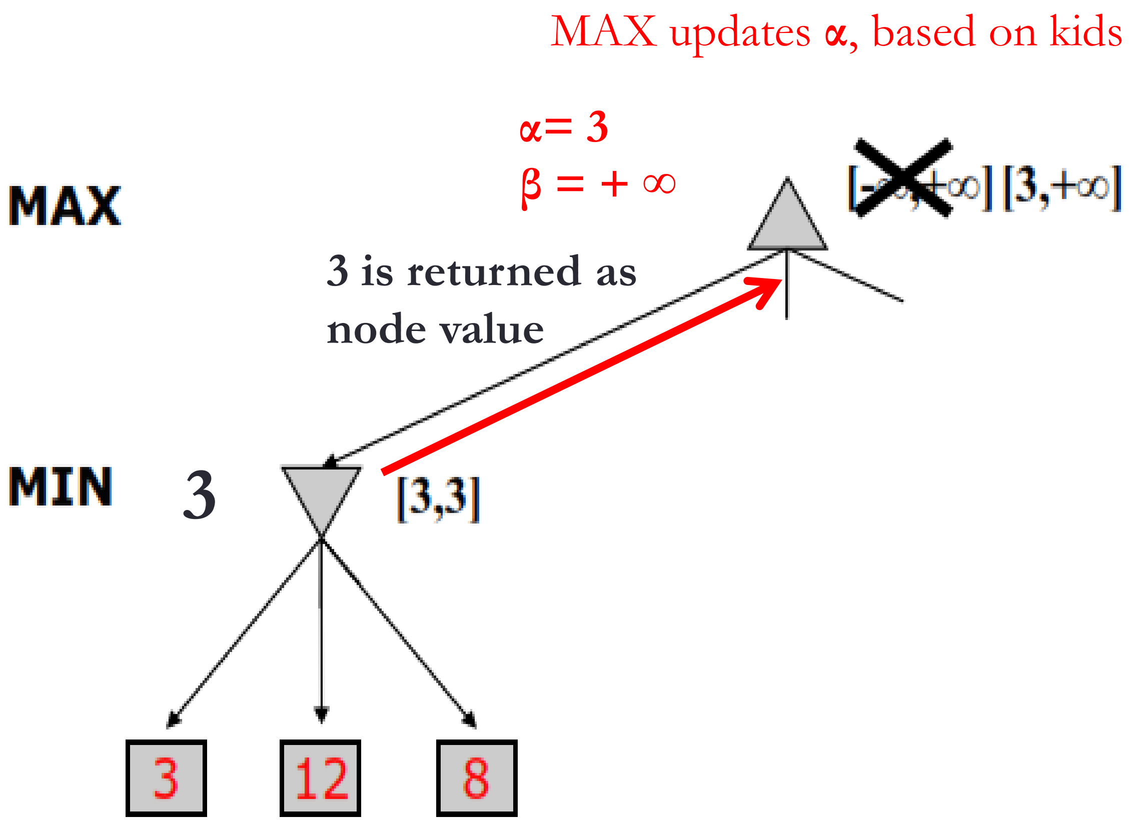


α-β Pruning Example - Revisited

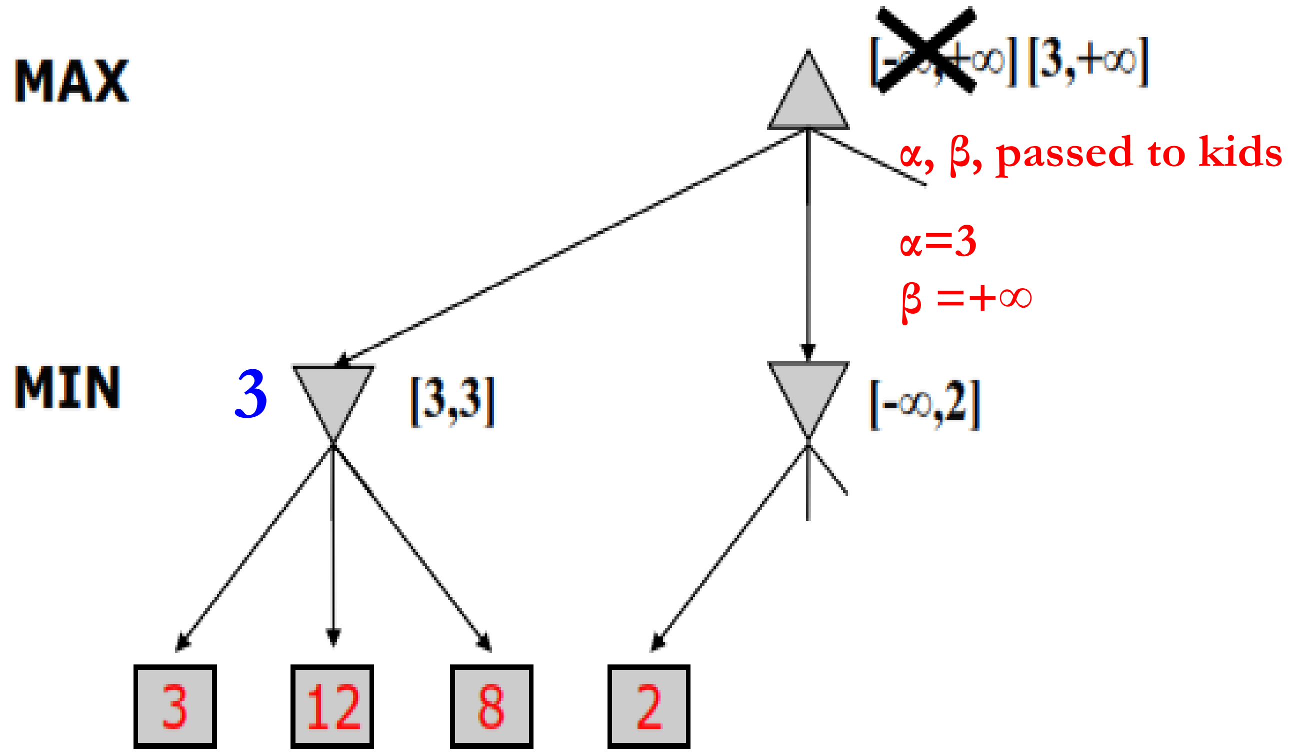
- Do DF-search until first leaf



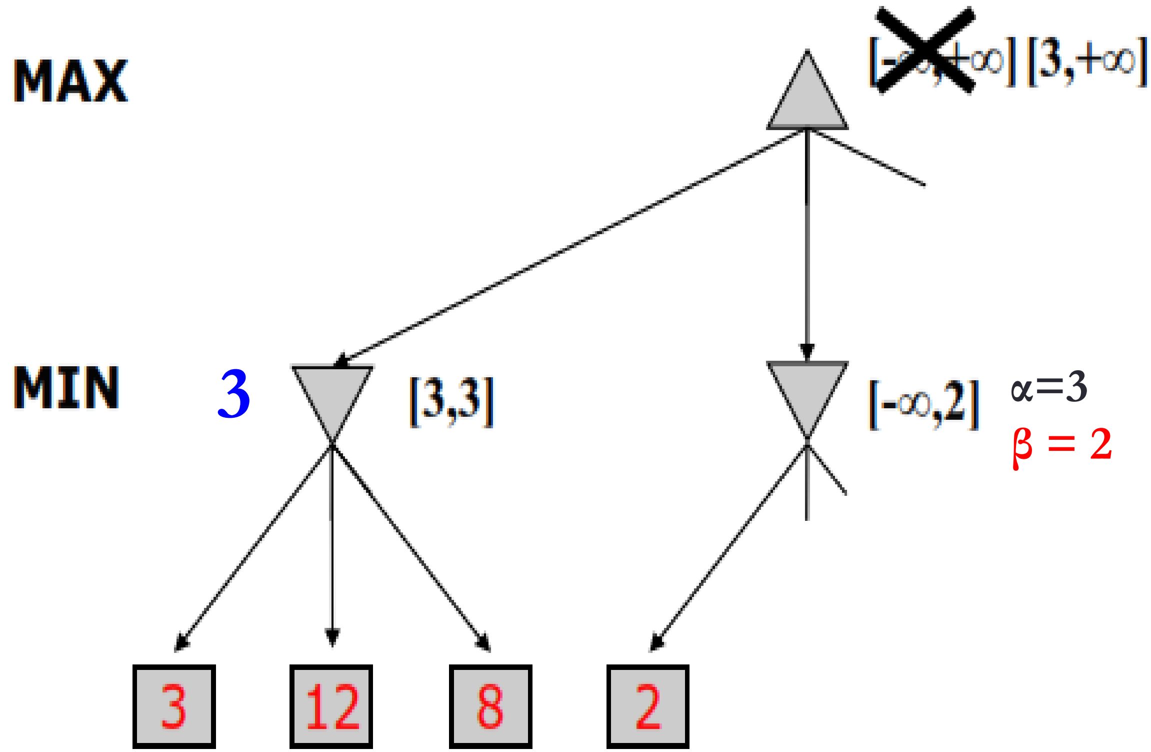
α-β Pruning Example - Revisited



α-β Pruning Example - Revisited

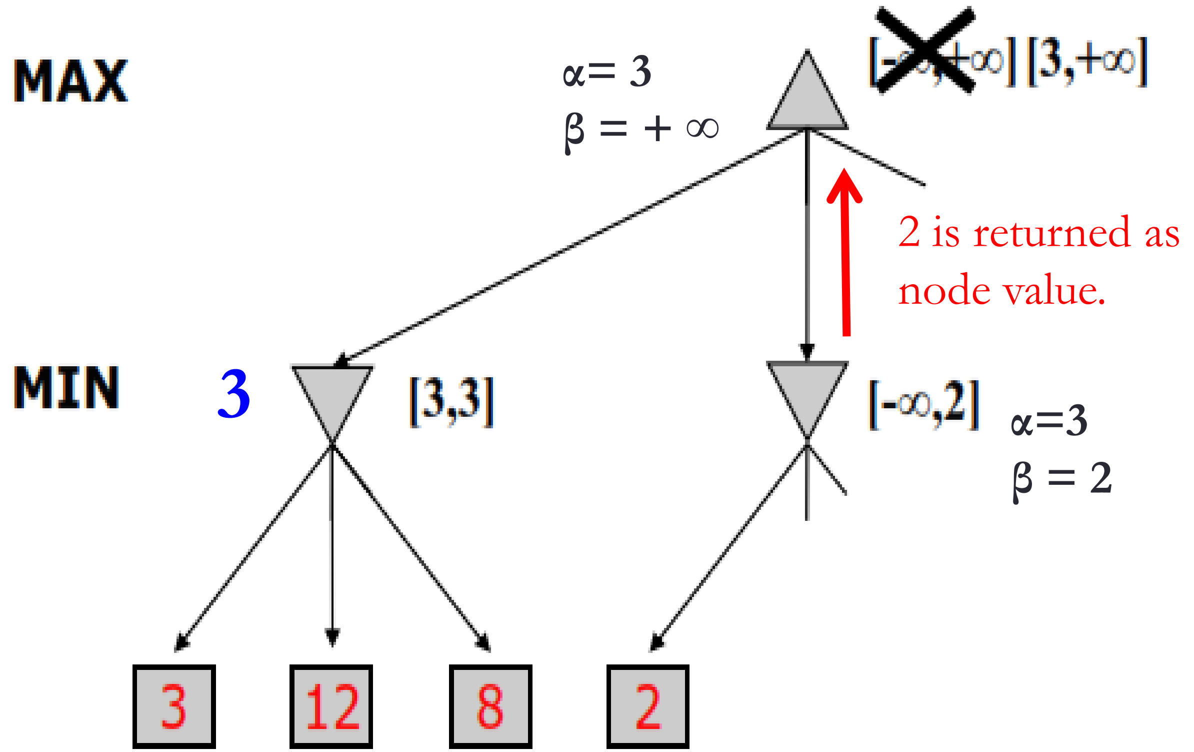


α-β Pruning Example - Revisited

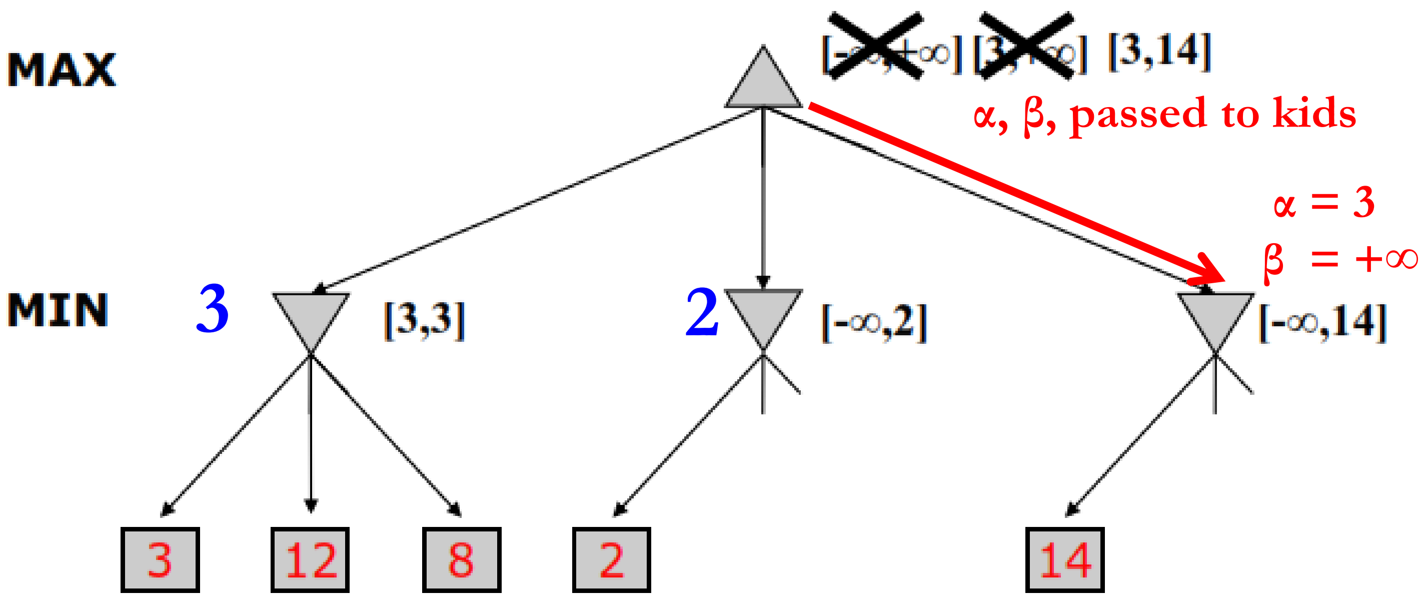


MIN updates β, based on kids

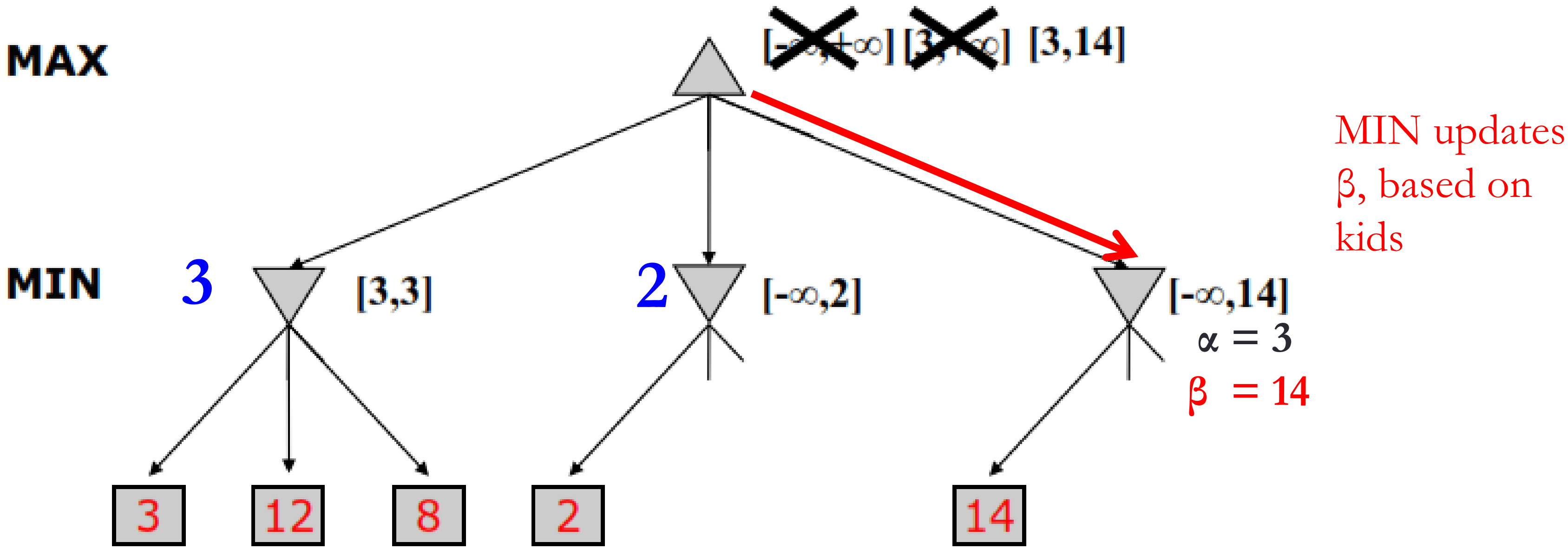
α-β Pruning Example - Revisited



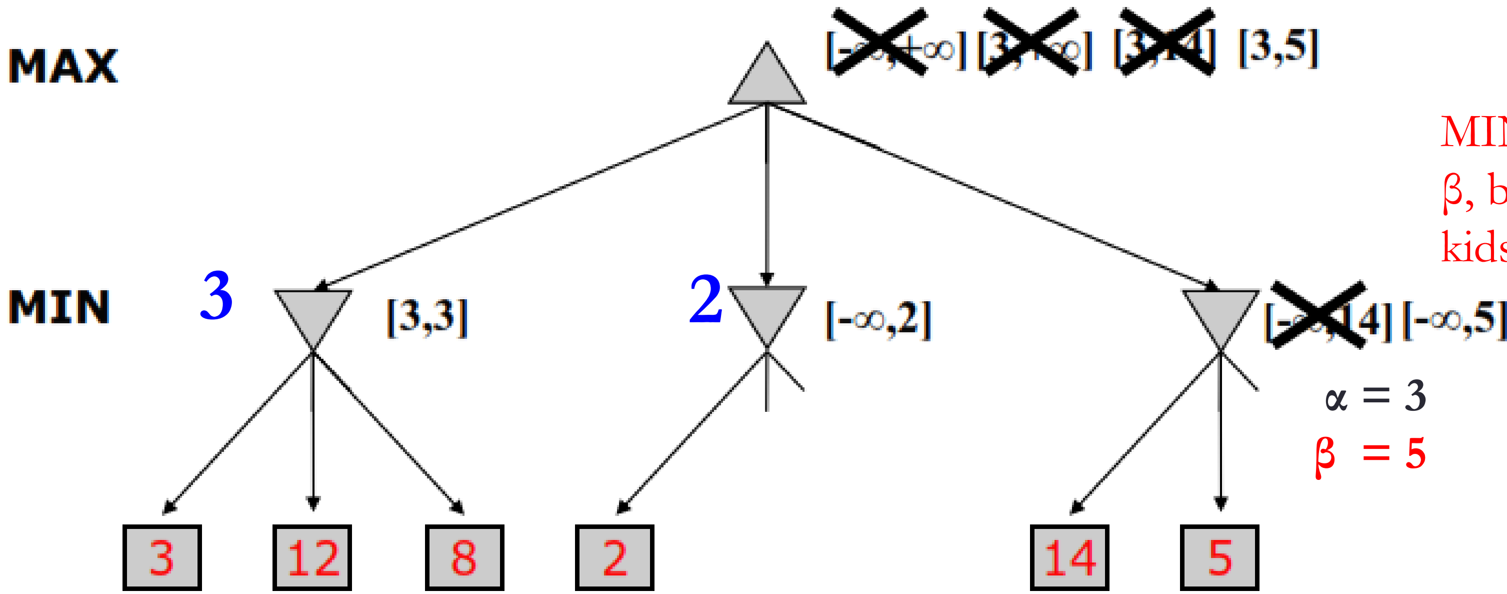
α-β Pruning Example - Revisited



α-β Pruning Example - Revisited

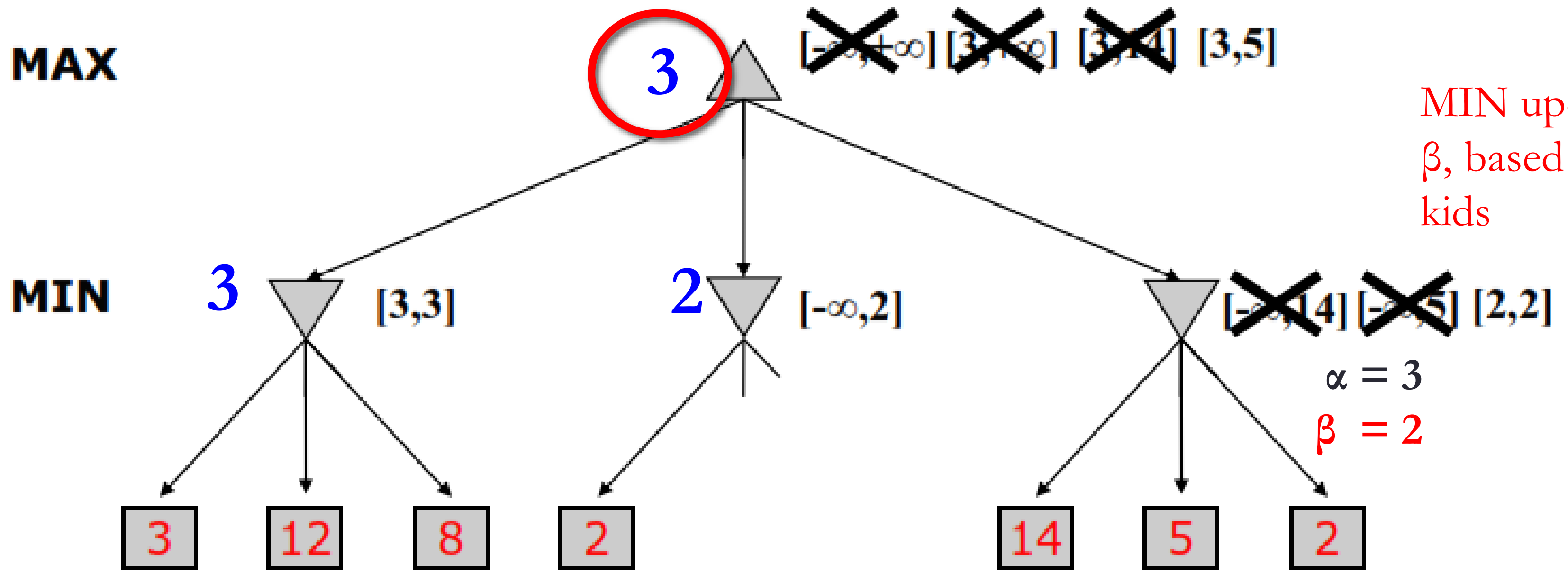


α-β Pruning Example - Revisited



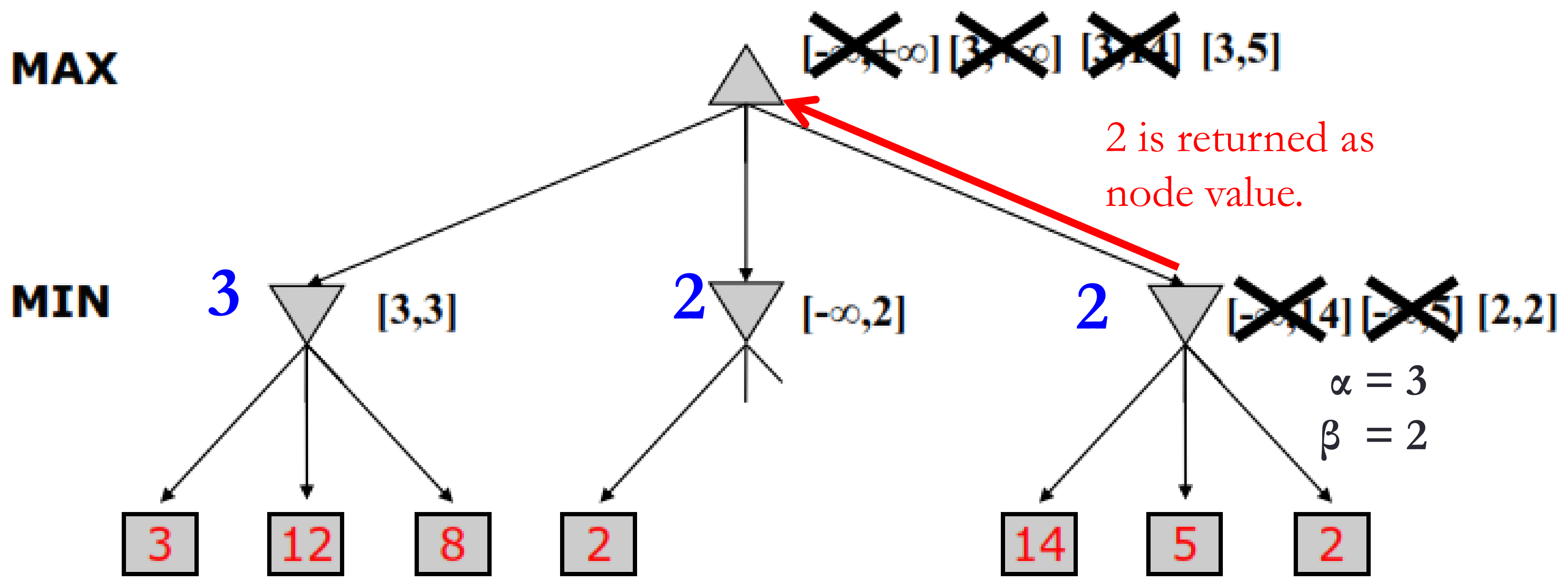
MIN updates β, based on kids

α-β Pruning Example - Revisited

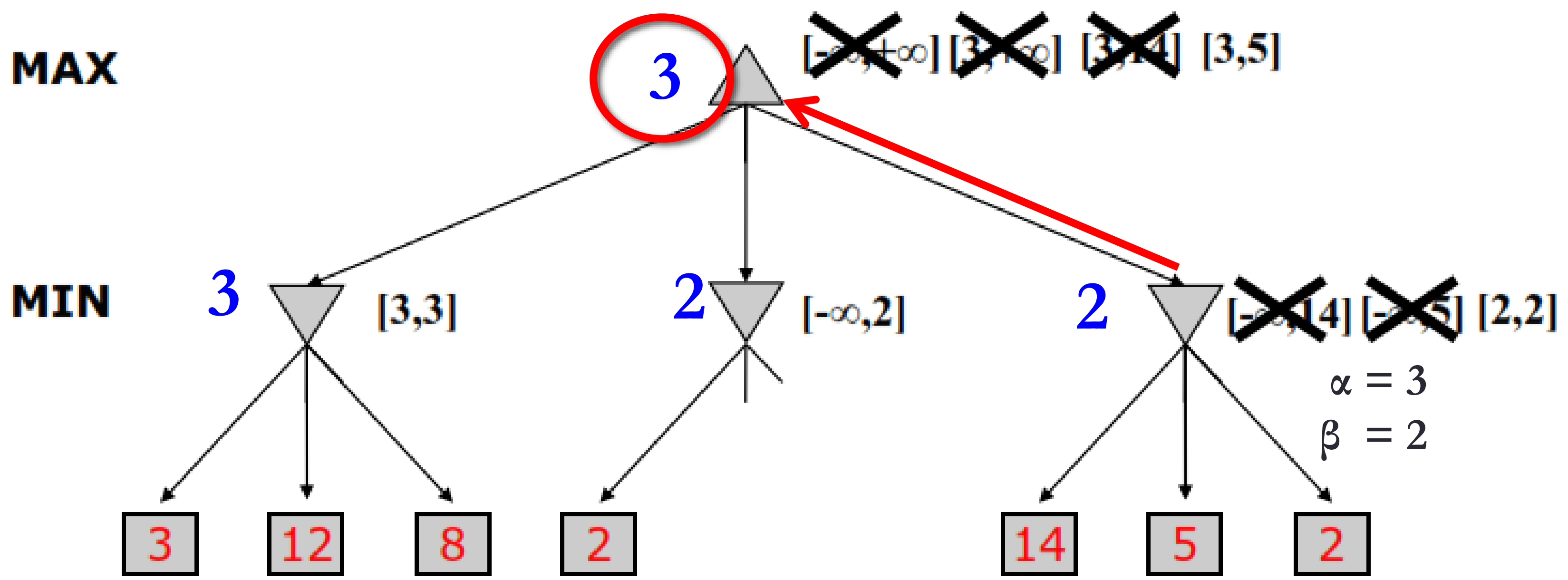


MIN updates β, based on kids

α-β Pruning Example - Revisited



α-β Pruning Example - Revisited



Order of Nodes

- α-β search updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to be **worse** than the current α or β value for MAX or MIN, respectively.

- The effectiveness of alpha-beta pruning is highly **dependent** on the order in which the successors are examined.

Order of Nodes

The effectiveness of alpha-beta pruning is highly dependent on the order in which the successors are examined.

In other words:

If children of a node are visited in the worst possible order, it may be that no pruning occurs.

- **For max nodes, we want to visit the best child first so that time is not wasted in the rest of the children exploring worse scenarios.**
- **For min nodes, we want to visit the worst child first (from our perspective, not the opponent's.)**

The α-β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game

    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for  MAX along the path to state
            β, the value of the best alternative for  MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

The α-β algorithm

```
function MIN-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for  MAX along the path to state
           β, the value of the best alternative for  MIN along the path to state

   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← +∞
   for a, s in SUCCESSORS(state) do
      v ←  MIN(v, MAX-VALUE(s, α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

Exercise 1

Answer the exercise given in AI Exam at Fall 2014?

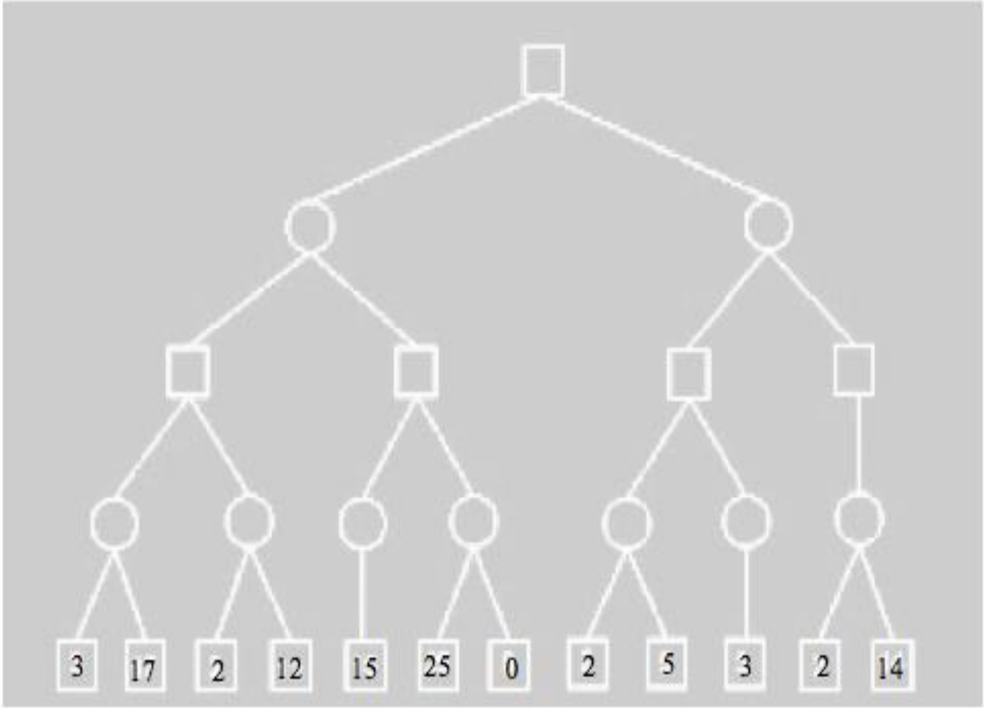
Using the Alpha-beta pruning, **what is the best value** of the Maximizers' node in the following tree?

Enhance your answer with description

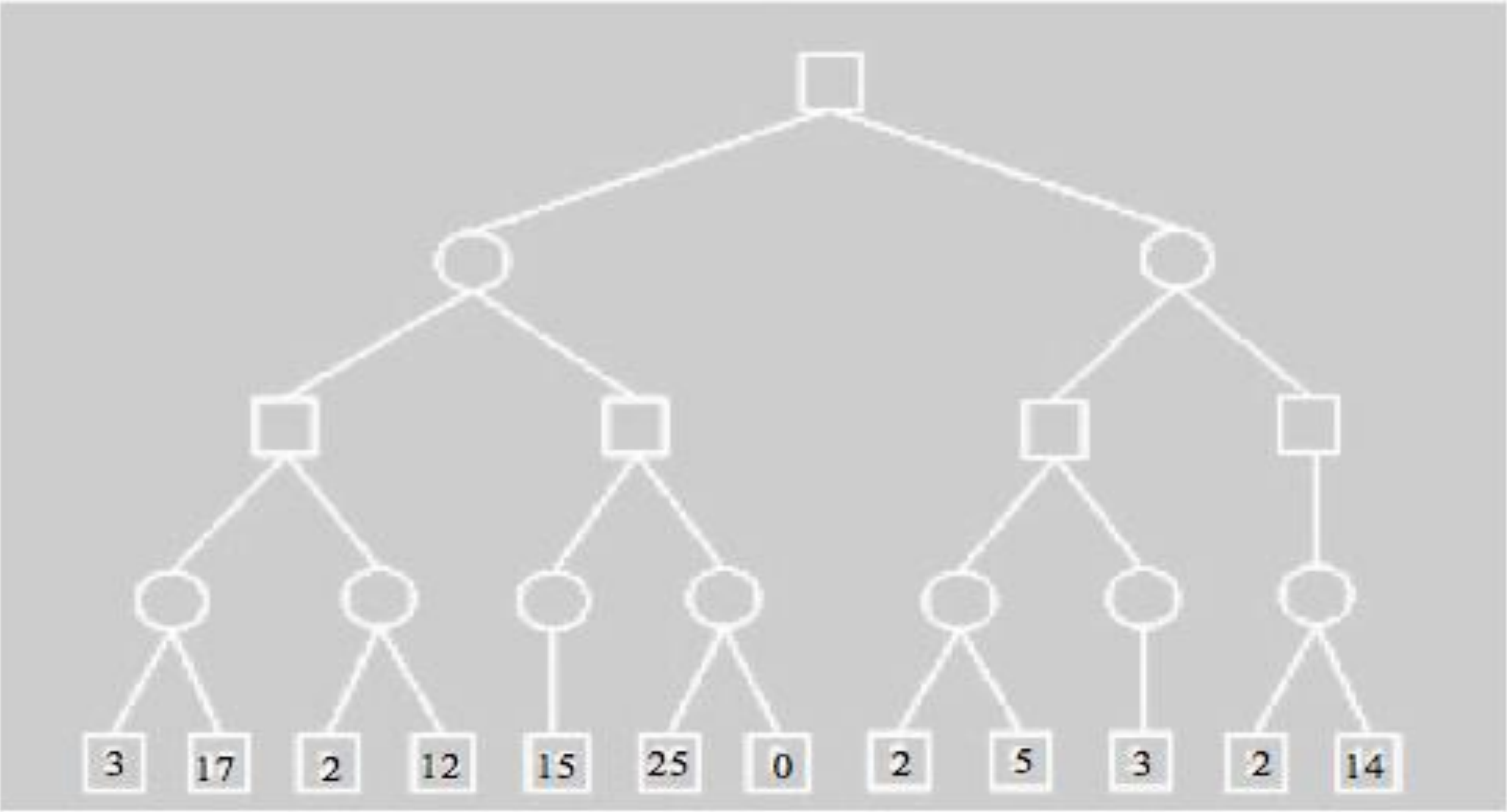
- Where:

  □ A Maximizer's node and
  ○ A Minimaizer's node

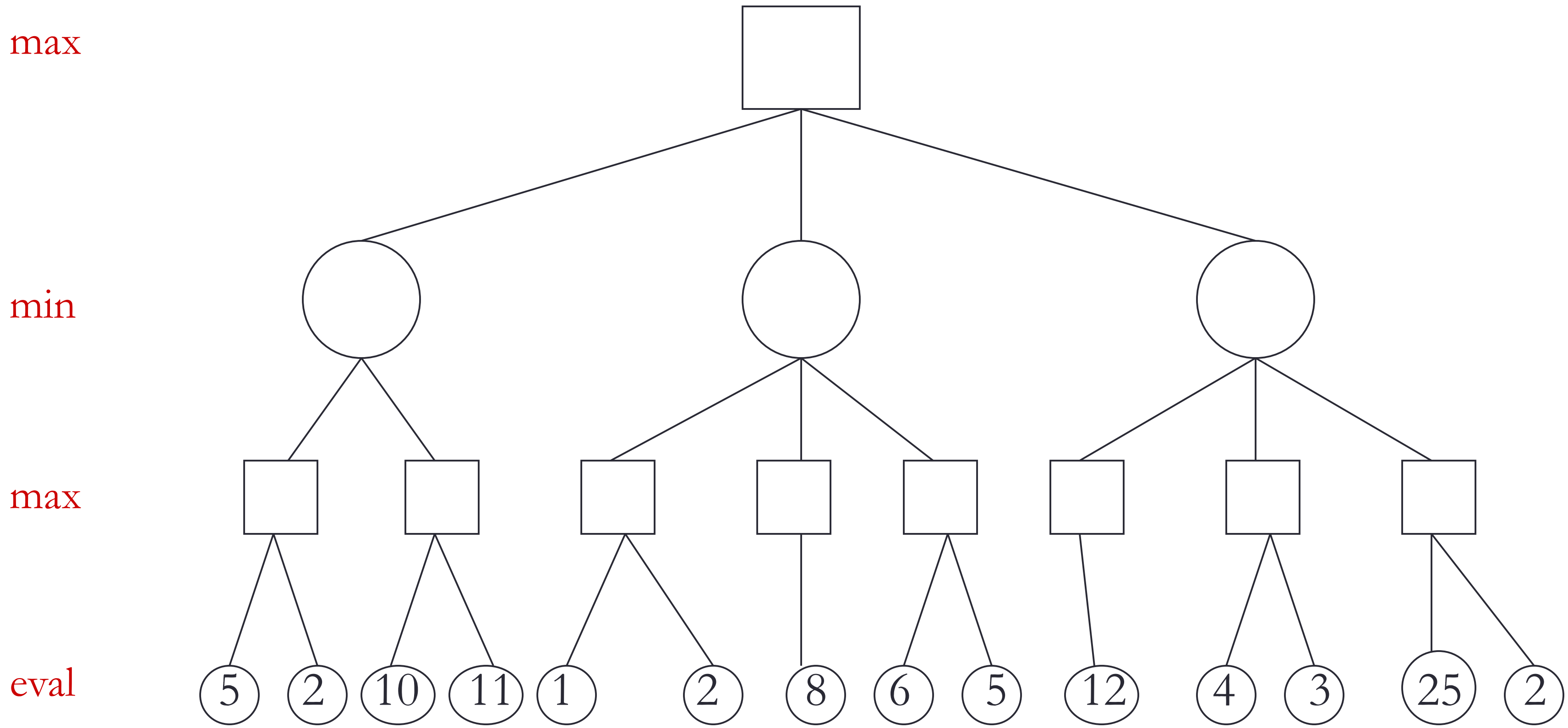
- If the numbers are not clear, they are from **left**:
  3, 17, 2, 12, 15, 25, 0, 2, 5, 3, 2, 14



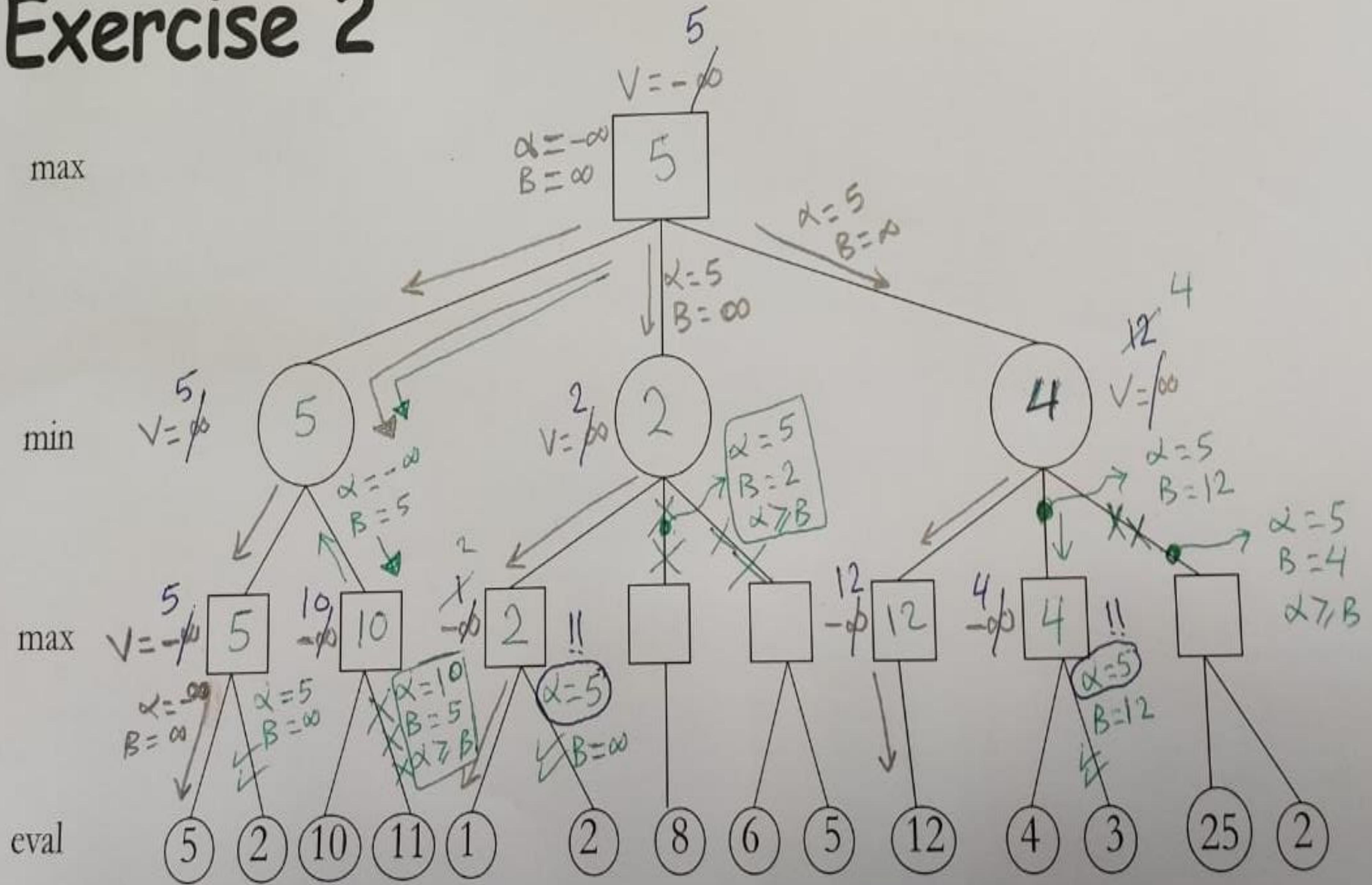
Exercise 1



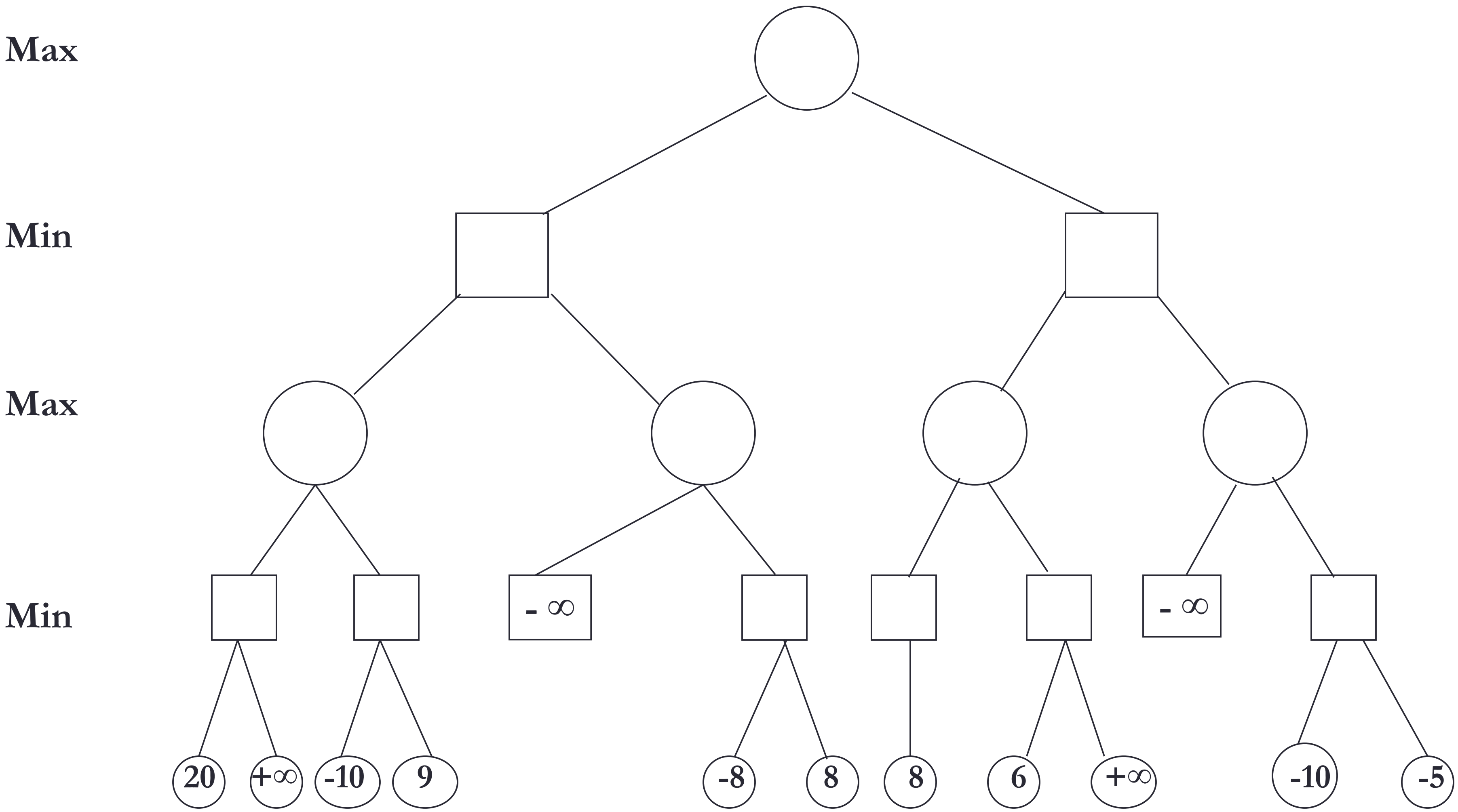
Exercise 2



Exercise 2



Exercise 3



The Chess game sing α-β algorithm

A step-by-step guide to building a simple chess AI

https://medium.freecodecamp.com/simple-chess-ai-step-by-step-1d55a9266977

Problem-Solving

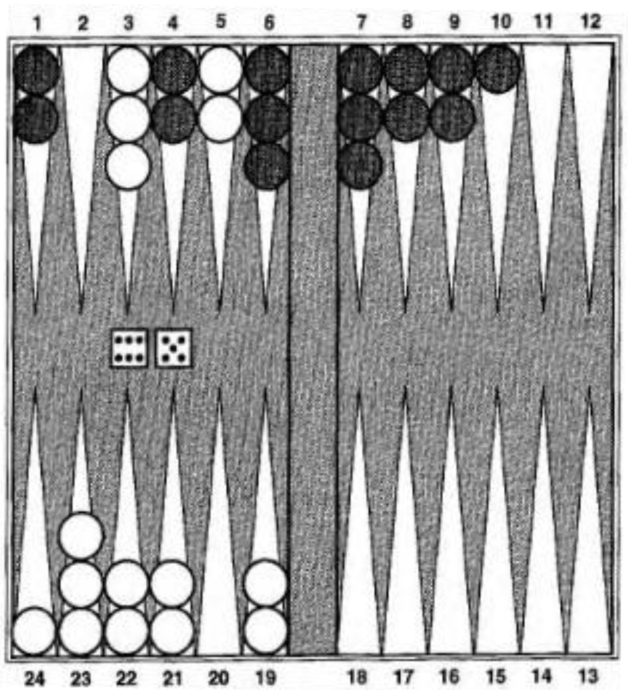
Adversarial Search

State-of-the-Art

Lecture III

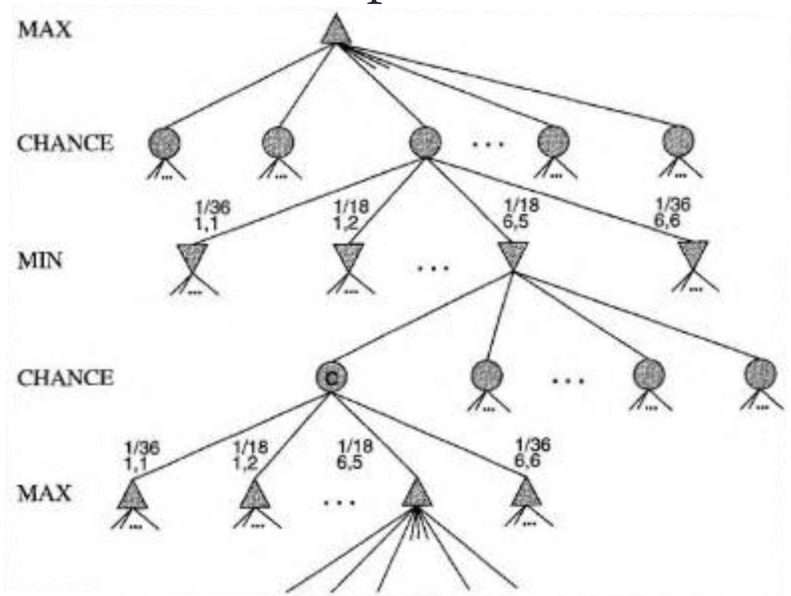
Games that Include Chance

- Many games include a random element, such as "throwing of dice"
- Backgammon is a typical game that combines luck and skill. Dice are rolled at the beginning of a player's turn to determine the legal moves.*
- Although White knows what his or her own legal moves are, White does not know what Black is going to roll and thus does not know what Black's legal moves will be. That means White cannot construct a standard game tree of the sort we saw in chess and tic-tac-toe.
- A game tree in backgammon must include **chance nodes** in addition to **MAX** and **MIN** nodes



Games that Include Chance

- The branches leading from each chance node denote the possible dice rolls, and each is labeled with the roll and the chance that it will occur.
- There are 36 ways to roll two dice, each equally likely; the six doubles (1-1 through 6-6) have a 1/36 chance of coming up, the other rolls have a 1/18 chance each.
- Now, we still want to pick the move that leads to the best position.
- However, the resulting positions do not have definite minimax values. Instead, we can only calculate the **expected value**, where the expectation is taken over all the possible dice rolls that could occur.



Checkers: Tinsley vs. Chinook

Name: Marion Tinsley
Profession: Teach mathematics
Hobby: Checkers
Record: Over 42 years loses only 3 games of checkers

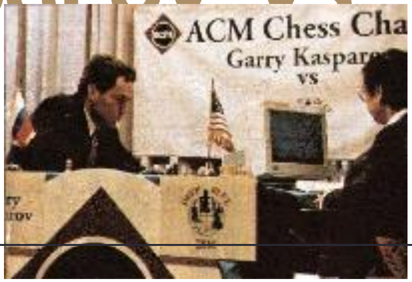
World champion for over 40 years

Mr. Tinsley suffered his 4th and 5th losses against Chinook

Chinook

First computer to become official world champion of Checkers!

Chess: Kasparov vs. Deep Blue



Kasparov		Deep Blue
5'10"	Height	6' 5"
176 lbs	Weight	2,400 lbs
34 years	Age	4 years
50 billion neurons	Computers	32 RISC processors + 256 VLSI chess engines
2 pos/sec	Speed	200,000,000 pos/sec
Extensive	Knowledge	Primitive
Electrical/chemical	Power Source	Electrical
Enormous	Ego	None

1997: Deep Blue wins by 3 wins, 1 loss, and 2 draws

Jonathan Schaeffer

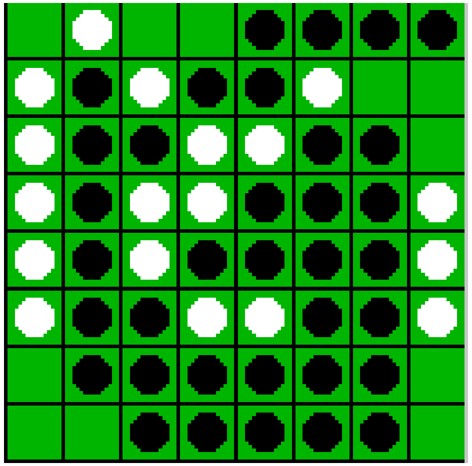
Chess: Kasparov vs. Deep Junior

Deep Junior

8 CPU, 8 GB RAM, Win 2000
2,000,000 pos/sec
Available at $100

August 2, 2003: Match ends in a 3/3 tie!

Othello: Murakami vs. Logistello



Takeshi Murakami
World Othello Champion

1997: The Logistello software crushed Murakami by 6 games to 0

Go: Goemate vs. ??

Name: Chen Zhixing
Profession: Retired
Computer skills:
  self-taught programmer
Author of Goemate (arguably the
  best Go program available today)

Gave Goemate a 9 stone handicap and still easily beat the program, thereby winning $15,000

Jonathan Schaeffer

Go: Goemate vs. ??

Name: Chen Zhixing
Profession: Retired
Computer skills:

Go has too high a branching factor for existing search techniques

Current and future software must rely on huge databases and pattern-recognition techniques

thereby winning $15,000

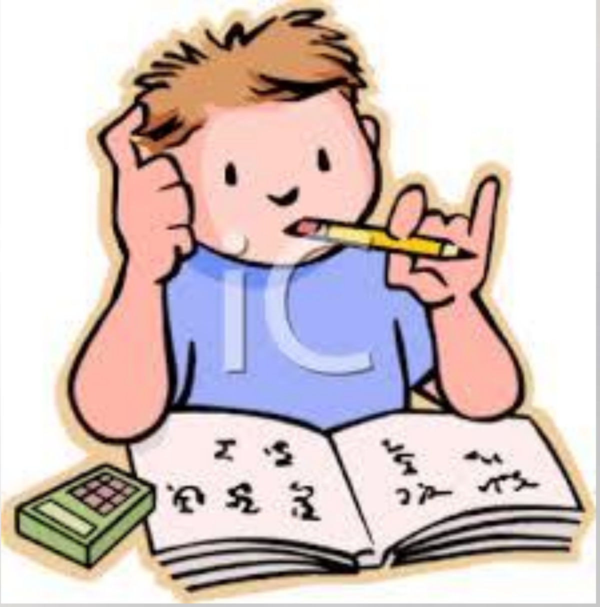
Jonathan Schaeffer

Summary

- Game playing is fun to work on! They illustrate several important points about AI
- Game trees represent alternate computer/opponent moves
- Evaluation functions estimate the quality of a given board configuration for the Max player.
- Minimax is a procedure which chooses moves by assuming that the opponent will always choose the move which is best for them
- Alpha-Beta is a procedure which can prune large parts of the search tree and allow search to go deeper
- For many well-known games, computer algorithms based on heuristic search match or out-perform human world experts.

EXERCISES

Please try to solve the following exercises:




Any questions?