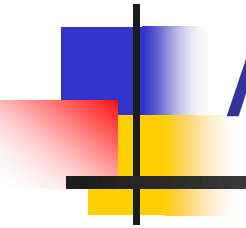


Artificial Intelligence

5. Elements of Game Theory (I)

Florin Leon

"Gheorghe Asachi" Technical University of Iași
Faculty of Automatic Control and Computer Engineering

"Al. I. Cuza" University of Iași
Faculty of Computer Science

Elements of Game Theory (I)

1. Sequential games
   1.1. Introduction and formalization
   1.2. The minimax algorithm
   1.3. Alpha-beta pruning
   1.4. Monte Carlo Tree Search
2. Strategic games
   2.1. Domination
   2.2. Pure Nash equilibrium
3. Conclusions

Elements of Game Theory (I)

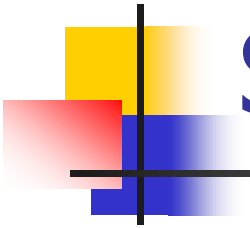
1. Sequential games
   1.1. Introduction and formalization
   1.2. The minimax algorithm
   1.3. Alpha-beta pruning
   1.4. Monte Carlo Tree Search
2. Strategic games
   2.1. Domination
   2.2. Pure Nash equilibrium
3. Conclusions

Game theory

- Game theory studies strategic interactions between rational players/agents that choose different actions in order to maximize their payoff
- More formally, it represents the study of mathematical models of conflict and cooperation between intelligent, rational decision makers
- Game theory is an interdisciplinary approach to study human behavior
- John von Neumann, Oskar Morgenstern: *Theory of Games and Economic Behavior* (1944)

Payoff maximization

- Each agent/player must maximize its own rewards in an environment influenced by the strategies of other agents
  - The choices that an agent makes depend upon those made by all the other agents
  - Each agent tries to maximize its payoff regardless of the other agents' actions
  - Conflicts, cooperation
  - Social decisions: with whom to cooperate and how
- Making rational strategy choices may also involve maximizing the rewards for the entire group of agents



Sequential games

- Games which need the exploration of alternatives have always been considered as challenges for human intelligence
  - Checkers (Mesopotamia, ca. 3000 B.C.)
  - Go (China, ca. 600 B.C.)
  - Chess (India, ca. 600 A.D.)

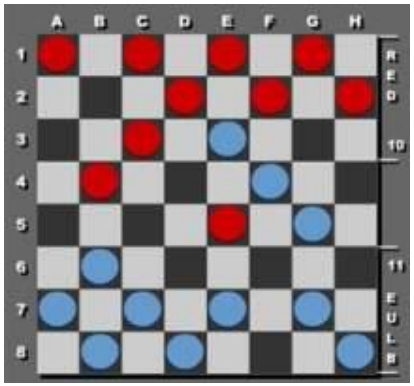
- AI often uses games to design and test algorithms

Sequential games

- One of the oldest subfields of AI (Zuse, Shannon, Wiener, Turing, 1950s)
- Abstract, pure form of competition that requires intelligence
- Difficult problems with a minimal initial structure
  - States and actions easy to represent
  - Very little knowledge required about the environment
- Games often have very large search spaces
- They are fun ☺

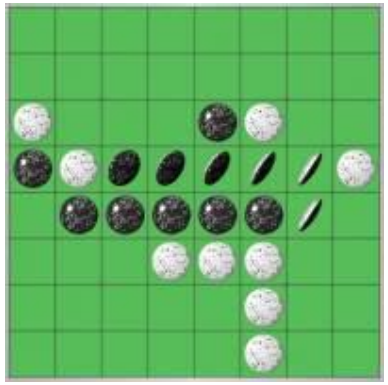
Deterministic, perfect information

Chess

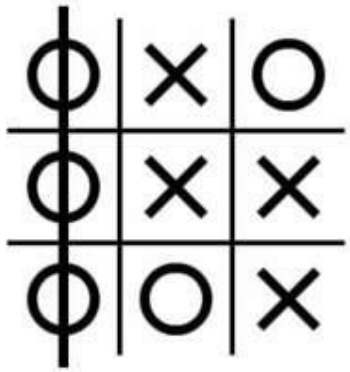


Checkers

Go



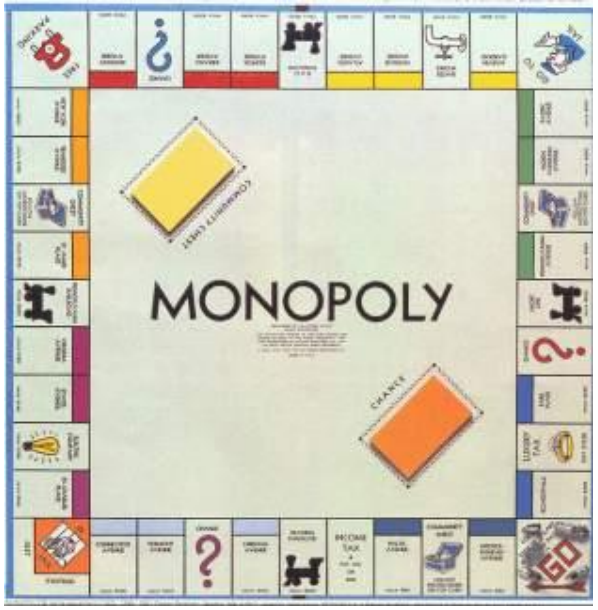
Othello



Tic-tac-toe

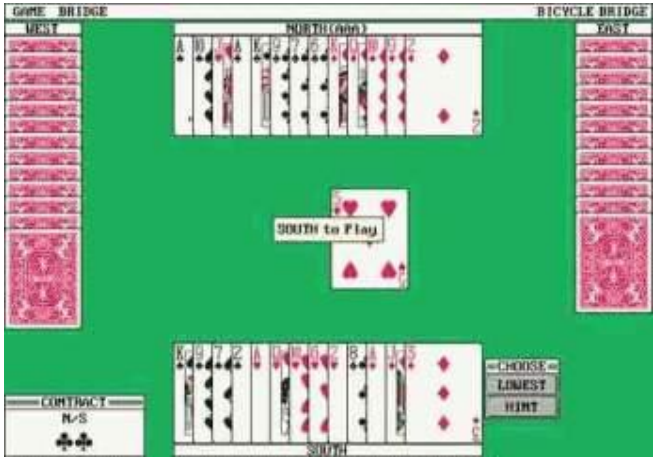
Nondeterministic, perfect information

Backgammon



Monopoly

Nondeterministic, imperfect information



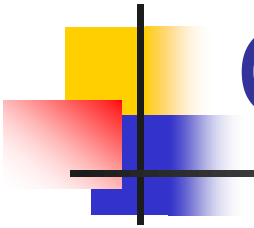
Bridge

Poker

Scrabble

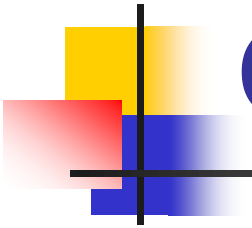
The size of the search space

- Chess ("the Drosophila of AI")
  - Branching factor about 35
  - About 50 moves per player
  - About 35^{100} (10^{154}) nodes
  - About 10^{40} distinct states (the size of the search graph)
- Go
  - Branching factor begins from 361 (a 19 x 19 board)
  - About 200 moves per state, 300 levels ⇒ 200^{300} (10^{690}) nodes



Games and search

- Classic search: a single agent that tries unobstructed to reach its goal
- Games: search in the presence of an opponent
- Representing board games as search problems
  - States: board configurations
  - Operators: legal moves
  - Initial state: current board configuration
  - Goal state: winning/terminal board configuration



Game playing

- The steps of playing a typical game:
  - Consider all the legal moves you can make
  - Each move leads to a new board configuration (position)
  - Evaluate each resulting position and determine which is best
  - Make that move
  - Wait for your opponent to move and repeat
- Key problems are:
  - Representing the “board”
  - Generating all legal next boards
  - Evaluating a position
  - Look ahead

The evaluation function

- It evaluates the "goodness" of a game position
  - Considering the *g* and *h* functions from classic search, here only *h* matters; *g* is irrelevant
- It is a heuristic function and contains the domain expert knowledge
- The intelligence of the computer greatly depends on the evaluation function

The evaluation function

- We assume that the game is zero-sum
- We can use a single evaluation function to describe the goodness of a board with respect to both players
- $f(n) > 0$: position n is good for the computer and bad for the (human) opponent
- $f(n) < 0$: position n is bad for the computer and good for the opponent

Examples: evaluation functions

- Tic-Tac-Toe
  - $f(n)$ = [number of 3-lengths open for the computer] – [number of 3-lengths open for the opponent]
  - A 3-length is a complete row, column or diagonal
- Chess (Alan Turing's function)
  - $f(n) = w(n) / b(n)$
  - $w(n)$ = sum of the point value of white pieces
  - $b(n)$ = sum of the point value of black pieces
  - Pawn = 1 point, knight/bishop = 3 points, rook = 5 points, queen = 9 points

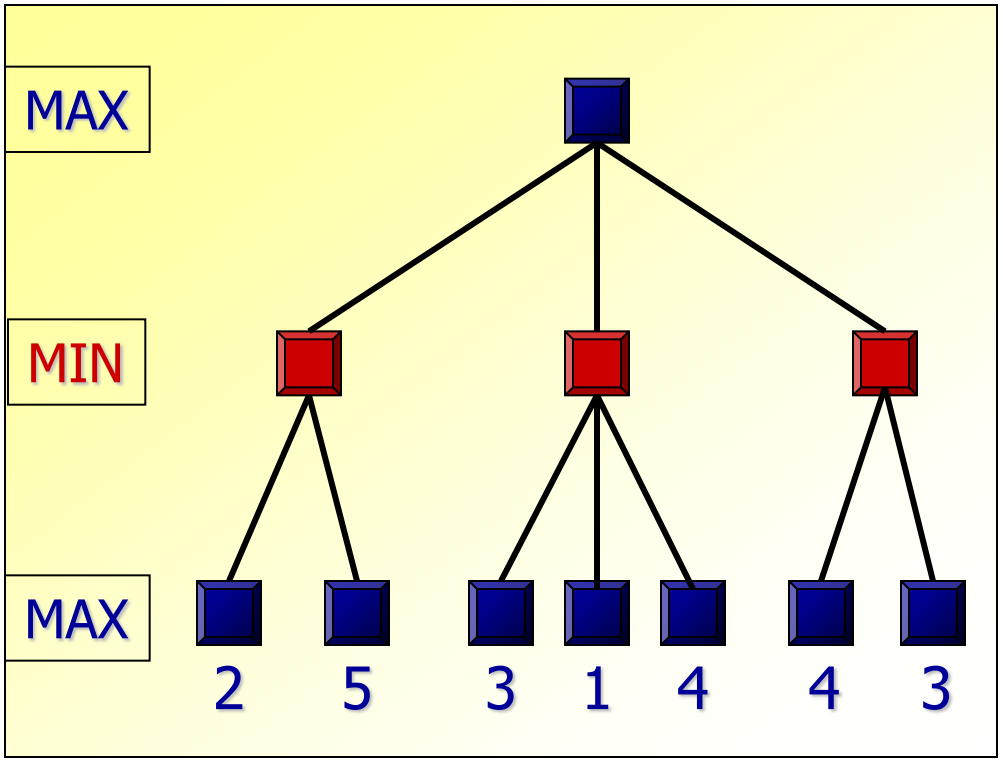
Examples: evaluation functions

- Most evaluation functions are specified as a weighted sum of position features:
  - $f(n) = w_1 \cdot t_1(n) + w_2 \cdot t_2(n) + \ldots + w_k \cdot t_k(n)$
  - Example features for chess are piece count, piece placement, squares controlled etc.
- Deep Blue had about 8000 features in its evaluation function
- There can be nonlinear combinations of features
  - A bishop values more by the end of the game than at the beginning
  - 2 bishops value more than twice the value of a single one

Elements of Game Theory (I)

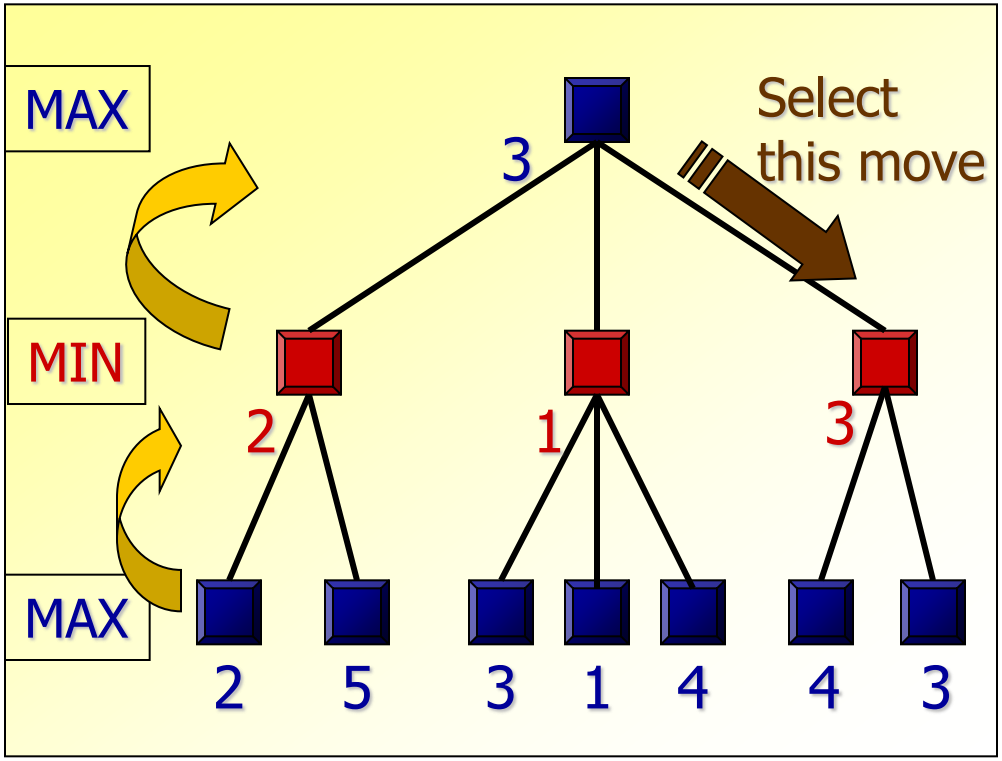
1. Sequential games
   - 1.1. Introduction and formalization
   - 1.2. The minimax algorithm
   - 1.3. Alpha-beta pruning
   - 1.4. Monte Carlo Tree Search
2. Strategic games
   - 2.1. Domination
   - 2.2. Pure Nash equilibrium
3. Conclusions

Minimax



- Restrictions:
  - 2 players: MAX (computer) and MIN (opponent)
  - Deterministic, perfect information
- Select a depth-bound (e.g. 2) and evaluation function

How it works



- Construct the tree up to the depth-bound
- Compute the evaluation function for the leaves
- Propagate the evaluation function upwards:
  - taking minima in MIN
  - taking maxima in MAX

Pseudocode

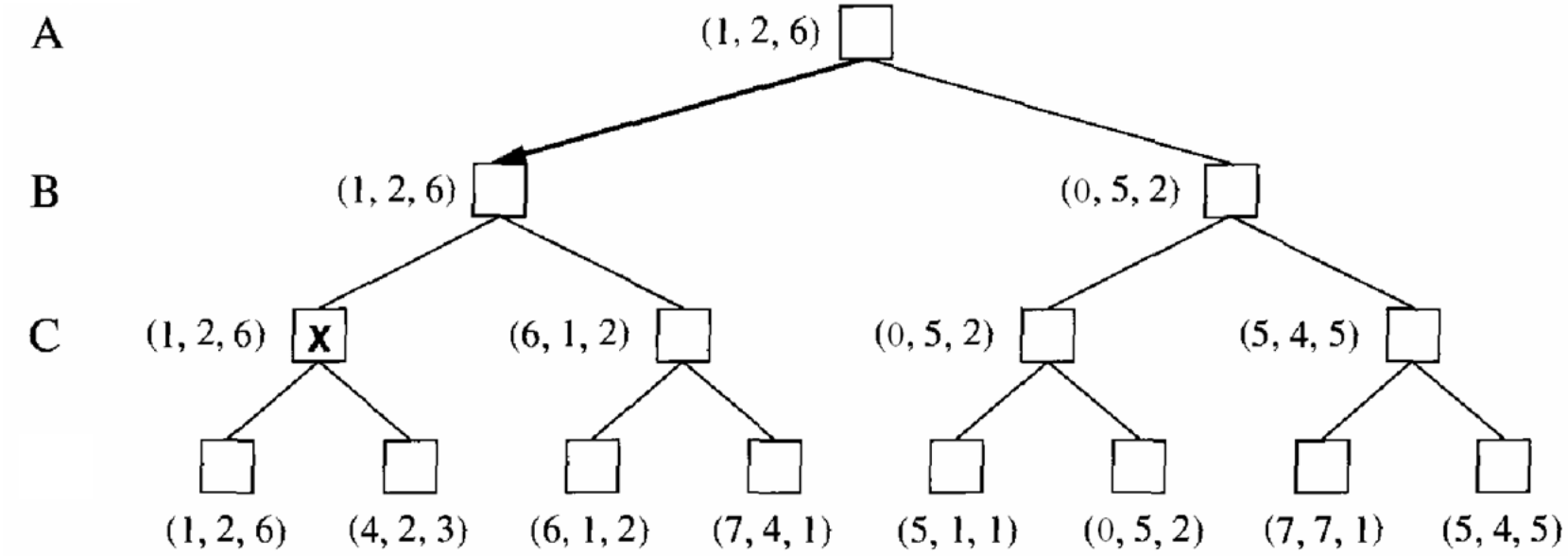
```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

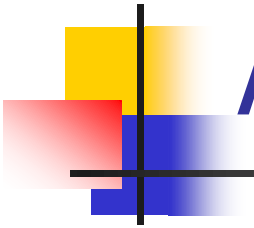
```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v
```

```
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
```

Multiplayer games

- The evaluation function returns a vector with the utilities of all players



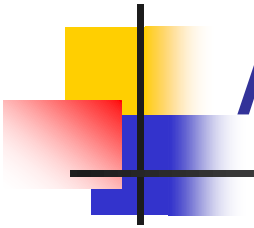


Alliances

- Alliances may result from the application of individual optimal strategies
- Cooperation may results from following the (selfish) individually rational behavior

Elements of Game Theory (I)

1. Sequential games
   - 1.1. Introduction and formalization
   - 1.2. The minimax algorithm
   - 1.3. Alpha-beta pruning
   - 1.4. Monte Carlo Tree Search
2. Strategic games
   - 2.1. Domination
   - 2.2. Pure Nash equilibrium
3. Conclusions

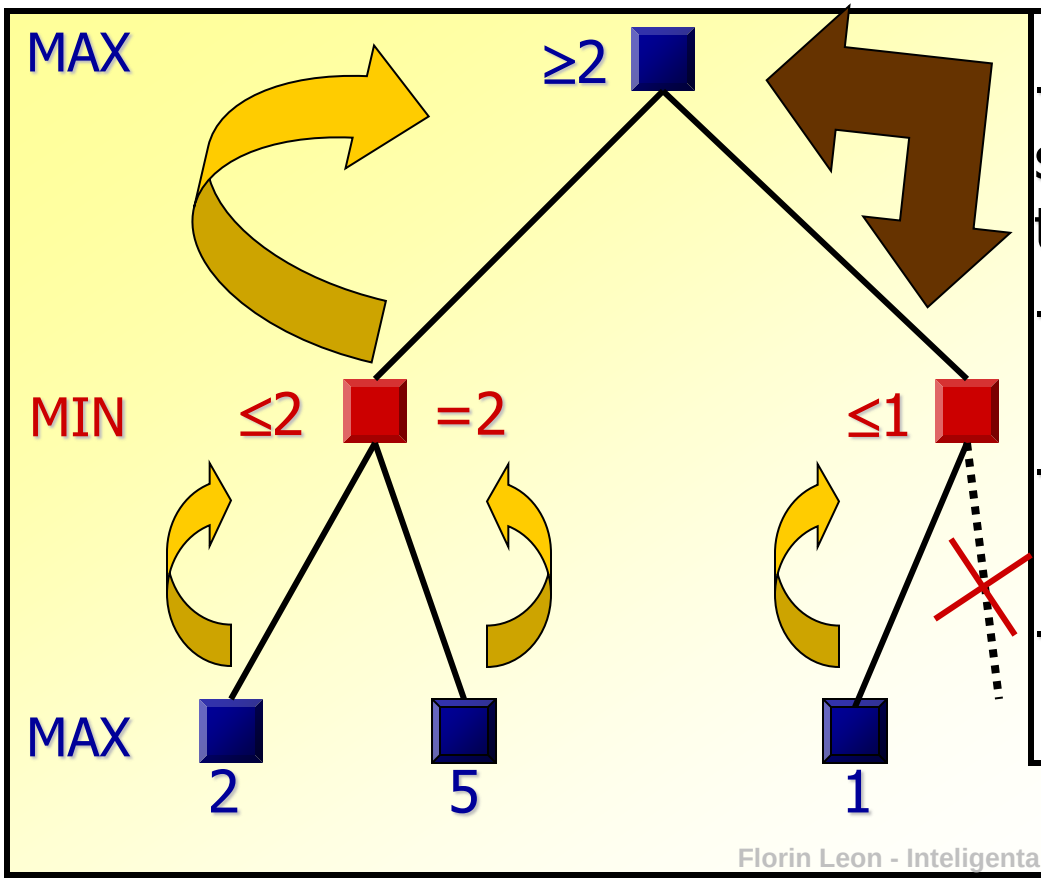


Alpha-beta pruning

- Generally applied optimization of minimax
- Minimax
  - First creates the entire tree (up to the depth-level)
  - Then does all propagation
- Alpha-beta pruning
  - Interleaves the generation of the tree and the propagation of values
  - Some of the obtained values in the tree will provide information that other parts not yet generated are redundant and do not need to be generated at all

Alpha-beta idea

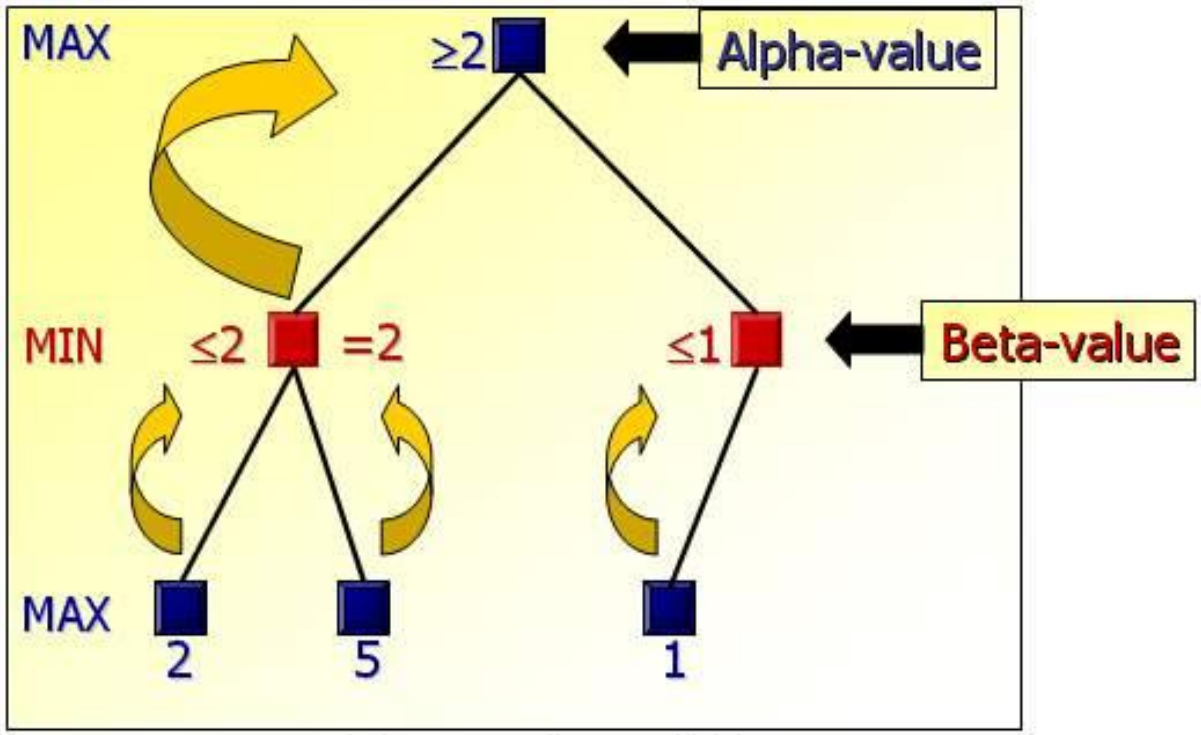
- Generate the tree depth-first, left-to-right
- Propagate the final values of the nodes as initial estimates for their parent node



- The MIN-value (1) is already smaller than the MAX-value of the parent (2)
- The MIN-value can only decrease further,
- The MAX-value is only allowed to increase,
- No point in computing further below this node

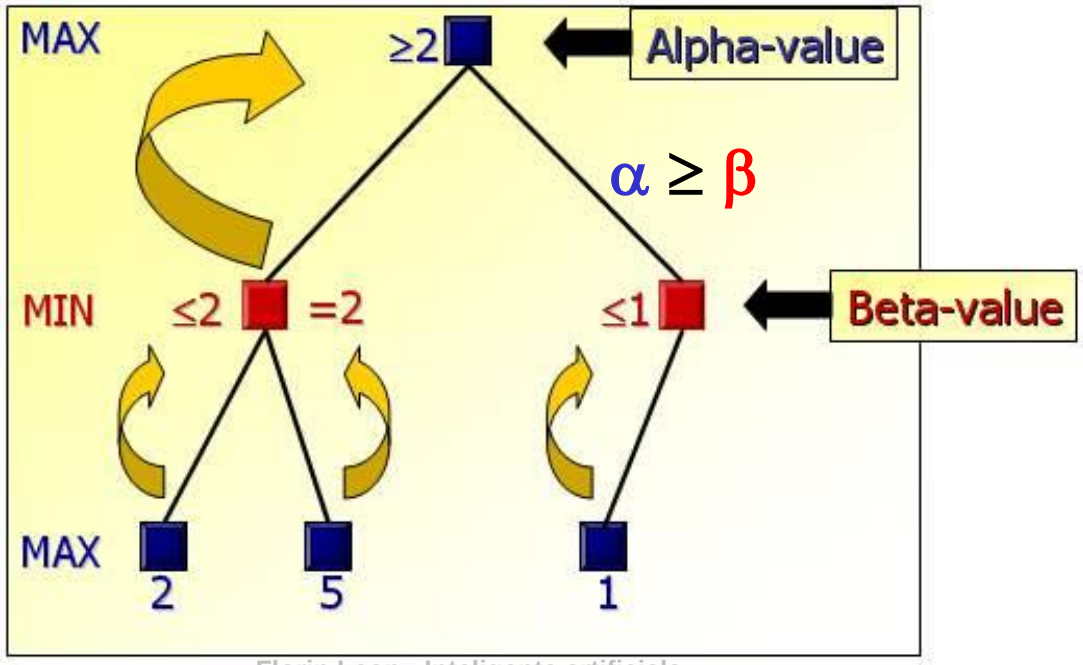
Terminology

- The (temporary) values at MAX-nodes are ALPHA-values
- The (temporary) values at MIN-nodes are BETA-values



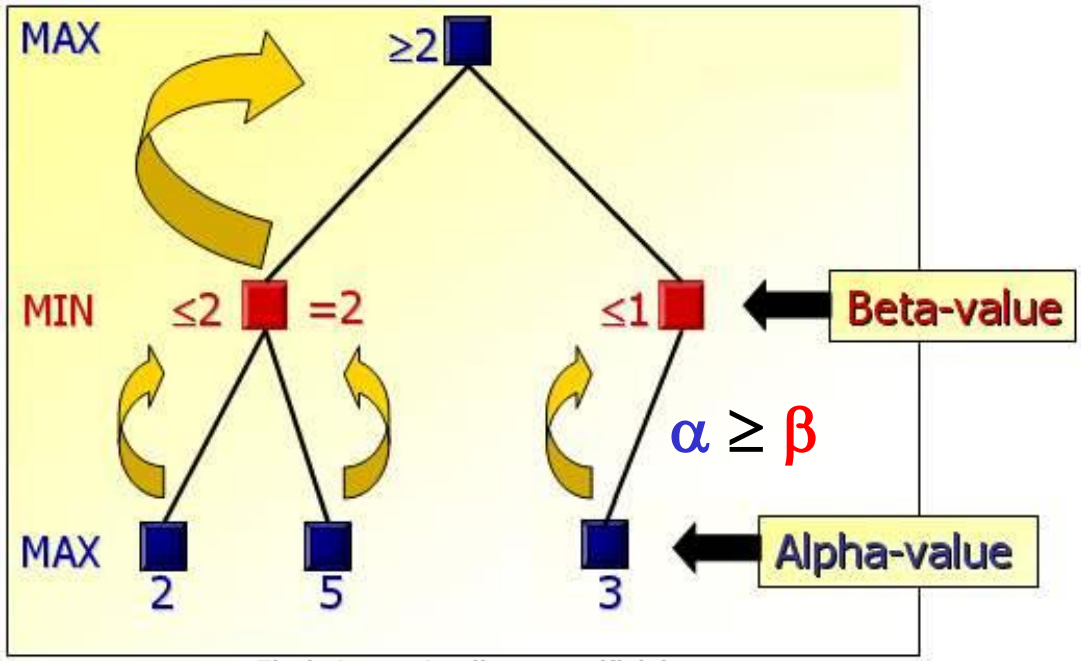
Alpha-beta principles

- **If** an alpha-value is larger or equal than the beta-value of a descendant node
- **Then** stop the generation of the children of the descendant

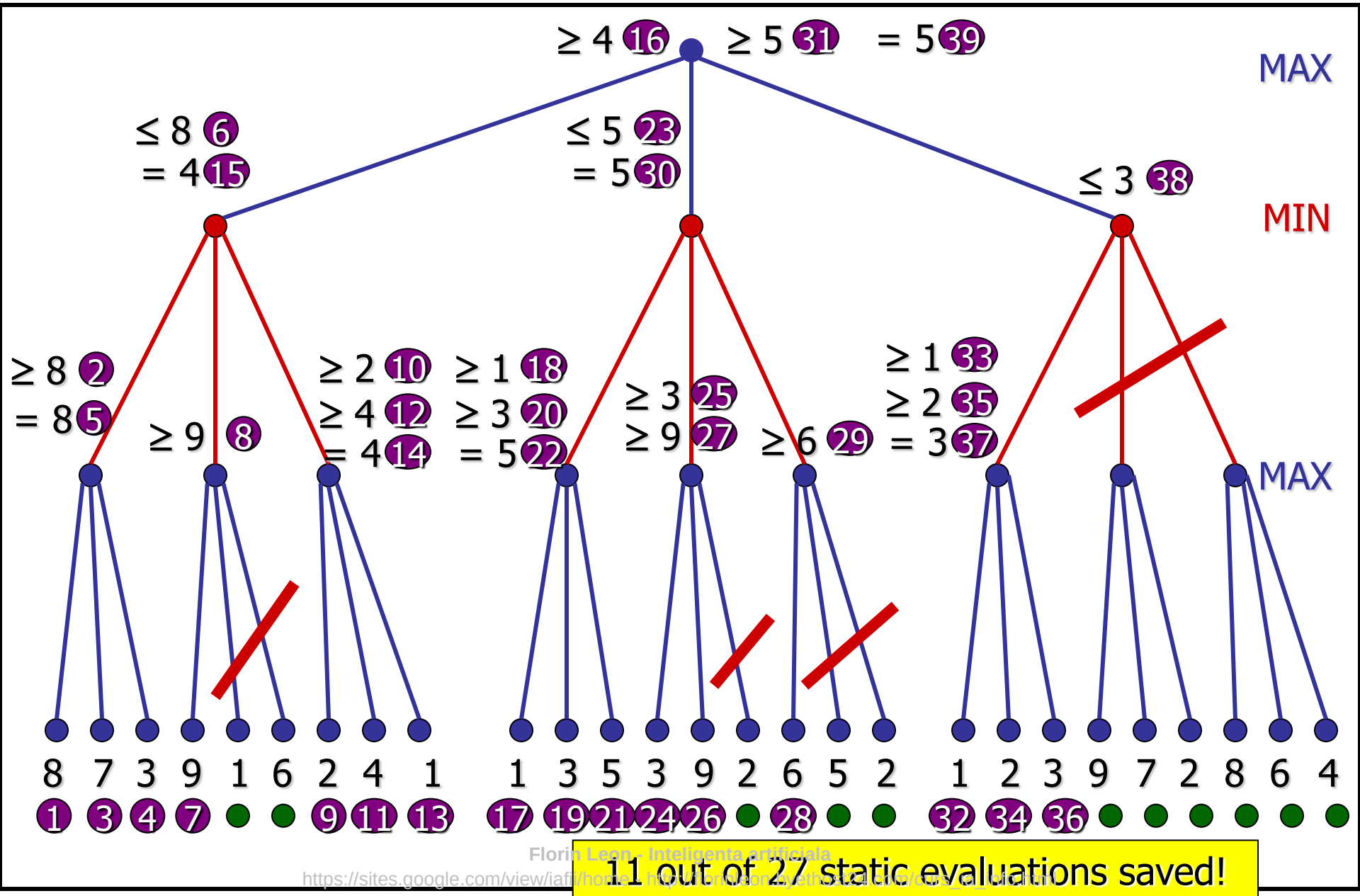


Alpha-beta principles

- **If** a beta-value is smaller or equal than the alpha-value of a descendant node
- **Then** stop the generation of the children of the descendant



Example: minimax with α - β



11 out of 27 static evaluations saved!

Pseudocode

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

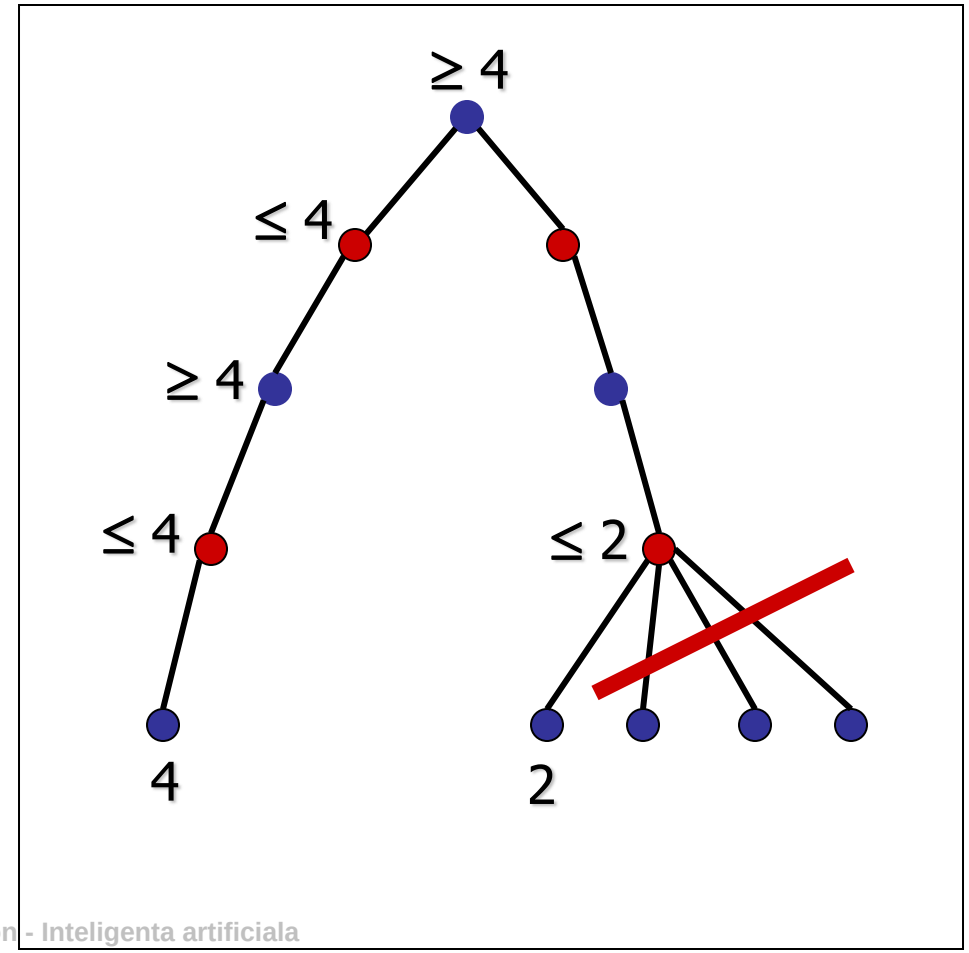
function MAX-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

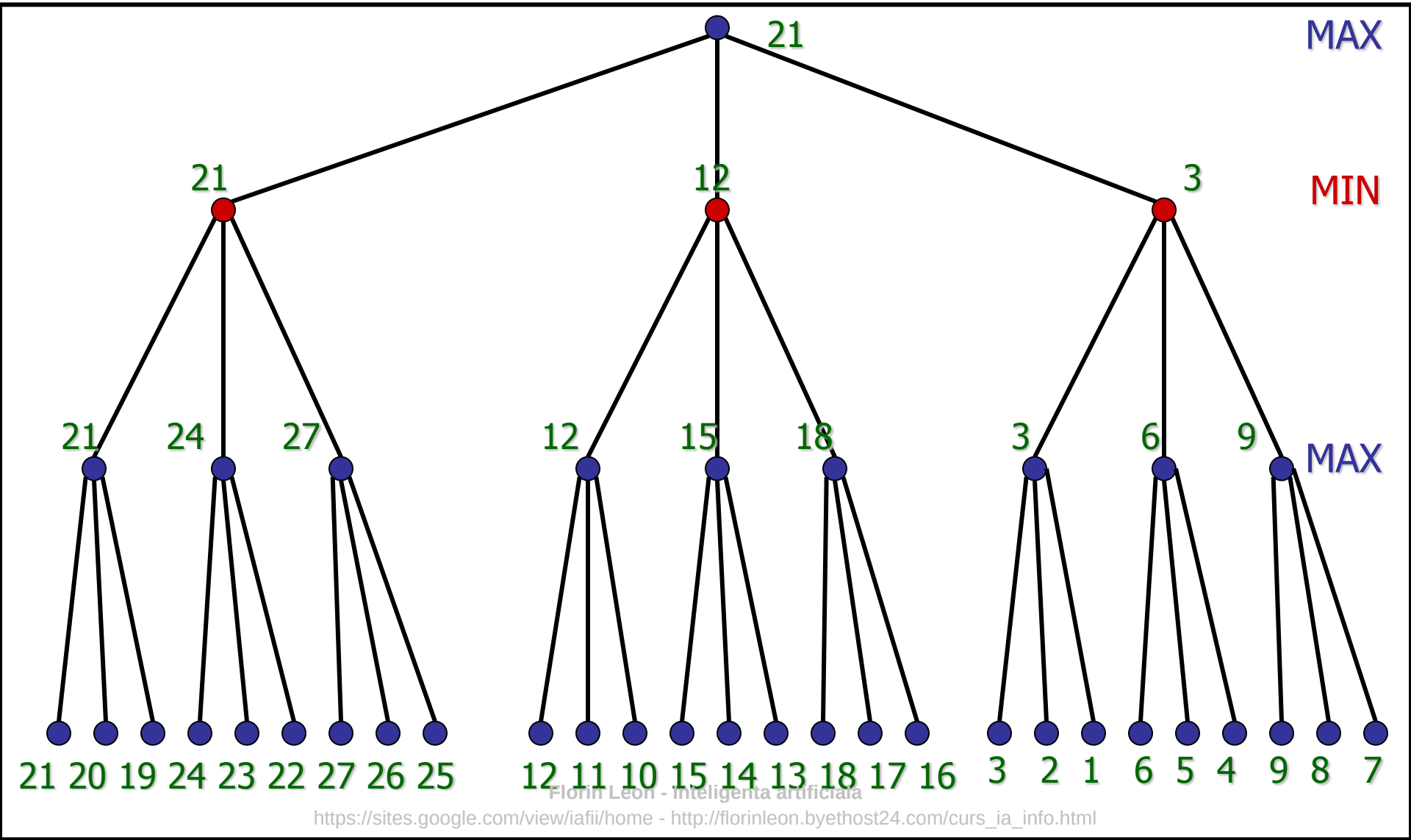
function MIN-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Deep pruning

- For game trees with at least 4 min/max layers the alpha - beta rules also apply to deeper levels

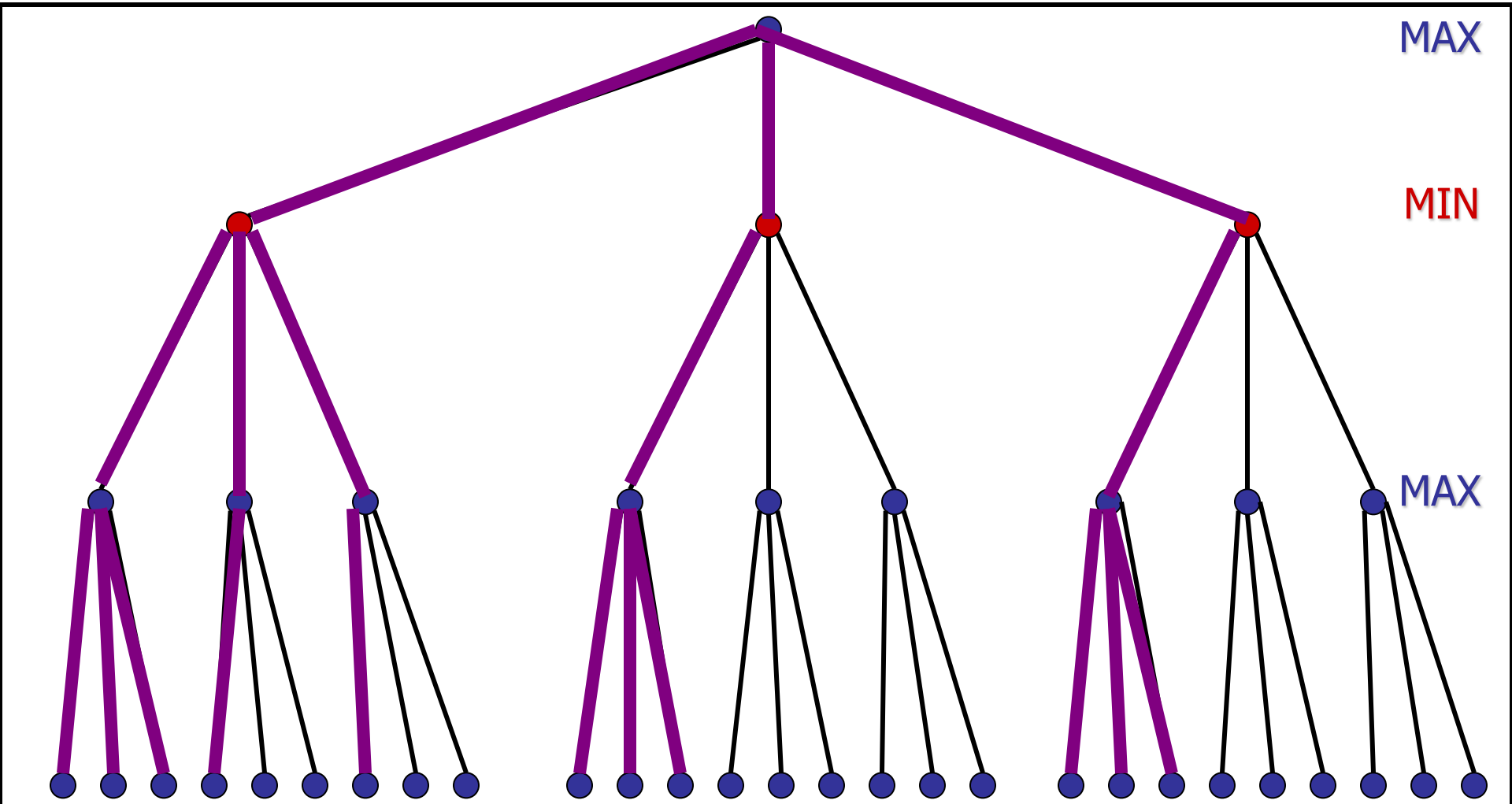


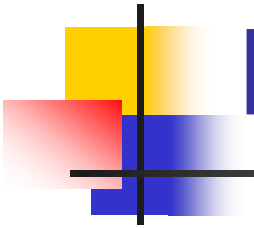
Best case:
a perfectly ordered tree



Best case

- If at every layer the best node is the left-most one





Reordering

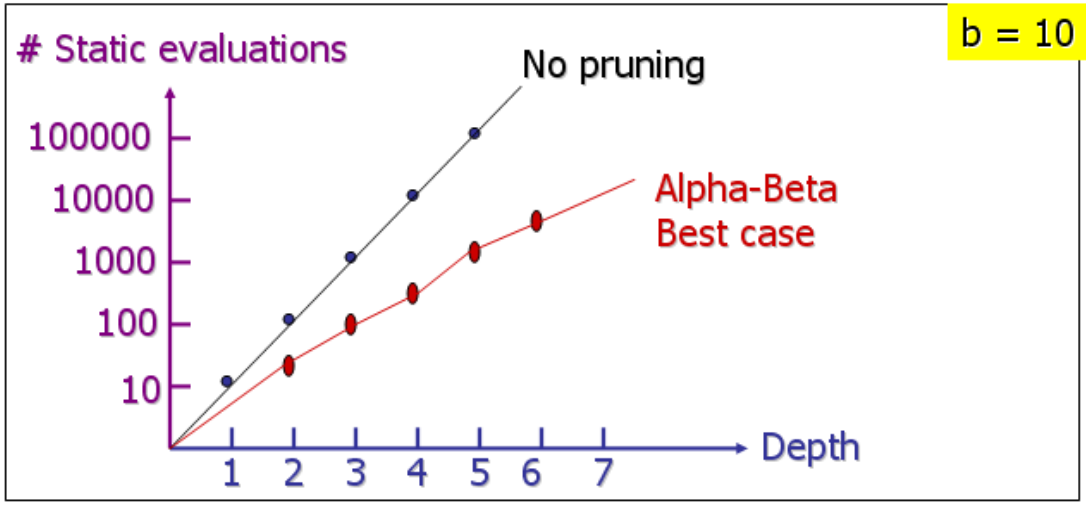
- For example, in chess:
  - The moves found to be the best in the previous search
  - Captures
  - Threats
  - Forward moves
  - Backward moves

Best case

- The number of static evaluations:
  - $n_{se} = 2 \cdot b^{d/2} - 1$, if d is even
  - $n_{se} = b^{(d+1)/2} + b^{(d-1)/2} - 1$, if d is odd
- In the previous example:
  - $d = 3, b = 3 \Rightarrow n_{se} = 9 + 3 - 1 = 11$
- where:
  - b is the branching factor
  - d is the depth of the tree

Comparison: minimax vs. alpha-beta

- Best case

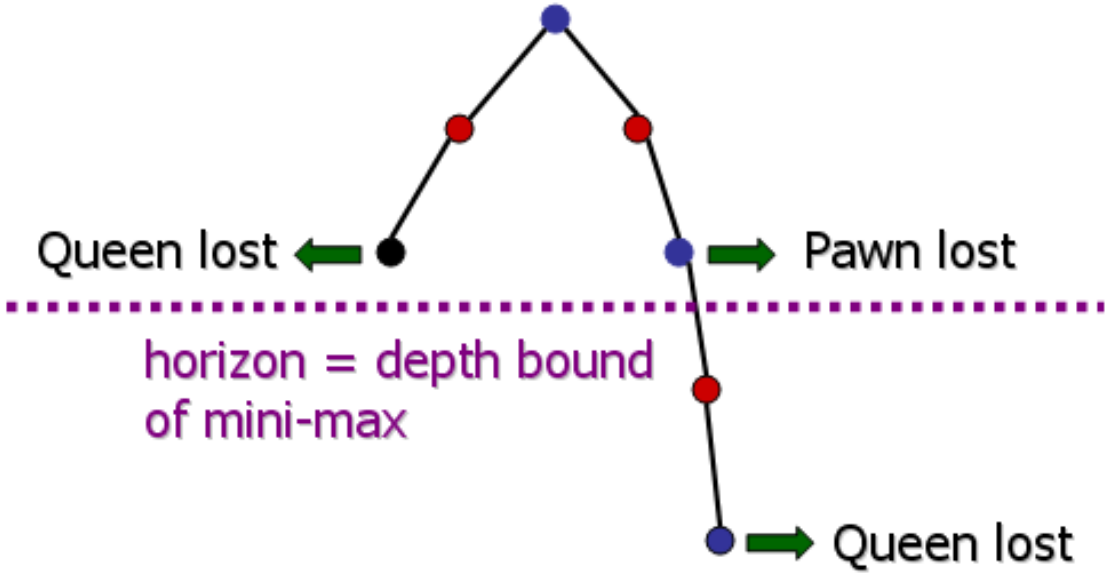


- The graph has logarithmic scale
  - Alpha-beta still has exponential complexity
- Worst case
  - For some trees, alpha-beta has no effect
  - For some trees, it is impossible to reorder to allow pruning

Effectiveness of alpha-beta

- Alpha-beta is guaranteed to compute the same value for the root node as computed by minimax, with less or equal computation
- Worst case: no pruning, examines $O(b^d)$ leaf nodes, where each node has b children and a d-ply search is performed
- Average case: $O\left(\left(\frac{b}{\log b}\right)^d\right)$
- Best case: examines only $O(b^{d/2})$ leaf nodes
  - It can search twice as deep as minimax
  - Best case is when each player's best move is the first alternative generated
- For Deep Blue, it was empirically discovered that alpha-beta pruning reduced the average branching factor from 35 to 6

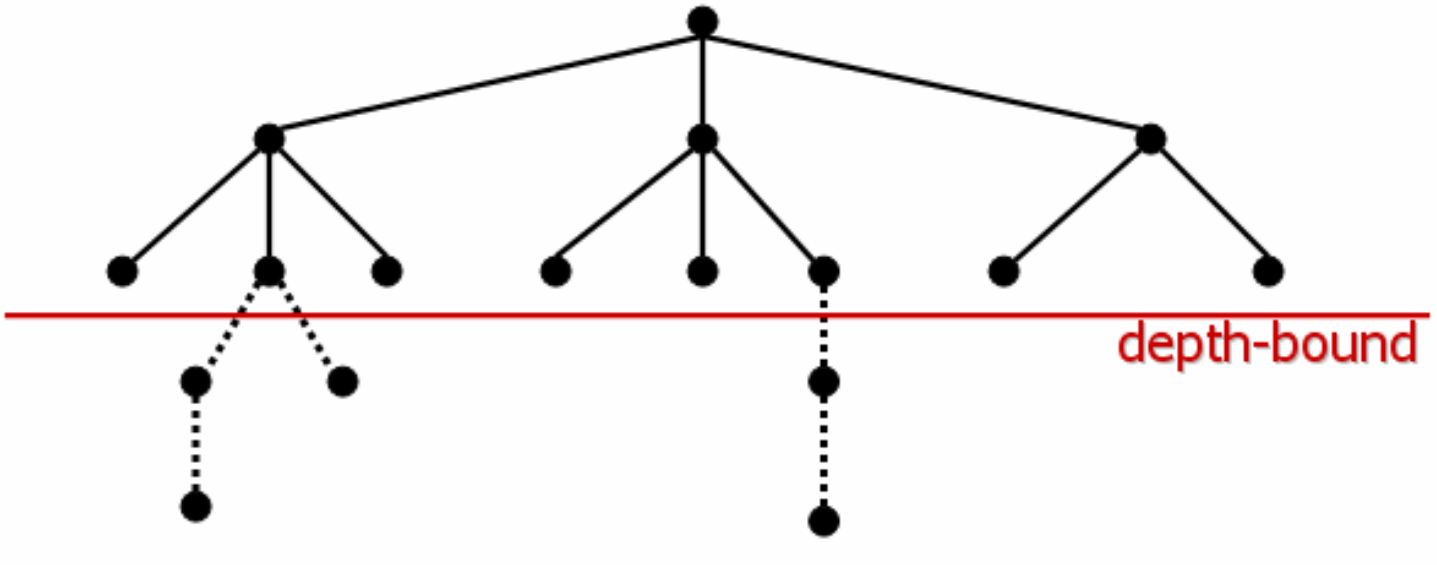
The horizon effect



- The right choice seems better than the left one, but it is not. Other moves may be better (e.g., losing a knight)
- Solution: heuristic continuation

Heuristic continuation

- In situations that are identified as strategically crucial e.g. king in danger, imminent piece loss, pawn to become queen etc., extend the search beyond the depth-bound

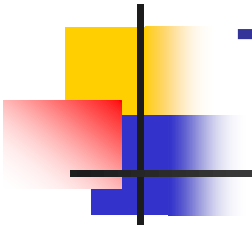


Secondary search

- Sometimes, it may be beneficial to further verify the goodness of a move
- For example, if we search for 6 plies and we found the best move, we can expand that position only for 2 more plies and see if the move remains good enough

Forward pruning

- A human player doesn't consider all the possible moves, only those which seem useful
- A sub-tree is pruned
  - When there are more symmetrical or equivalent moves
  - For moves that seem irrational (that lead to apparently unfavorable situations)
  - Only for deep levels in the tree
  - Not recommended near the root



Time bounds

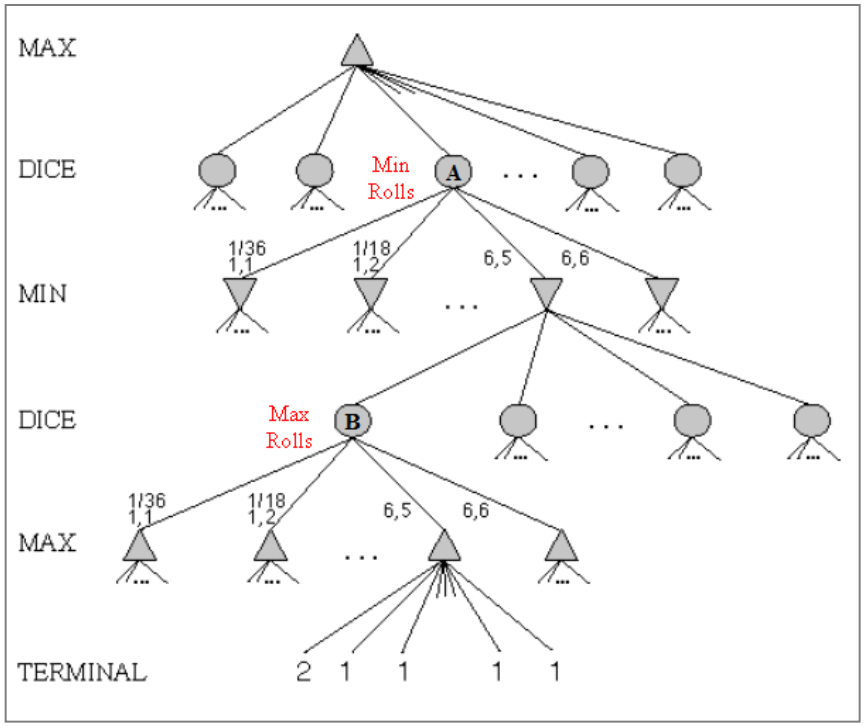
- Even with fixed depth-bound, times can vary strongly
- Solution: iterative deepening
  - A move is available at any time
  - The quality of the move increases with time

The effect of heuristics: chess

- With minimax we can search e.g. about 5 plies
- An average player analyses 6-8 plies
- With the alpha-beta pruning we can search about 10 plies (alpha-beta pruning makes the difference)
- Recent heuristic methods can decrease the branching factor from 35 to about 3
  - E.g. "null move" – the opponent moves twice in the beginning
- Deep Blue
  - Average search on 14 plies, maximum 40
  - Evaluation of 30 billion positions per move
  - Database with 700,000 grandmaster games
  - 4000 opening strategies
  - All the solutions for the positions with 5 pieces and many of the positions with 6 pieces

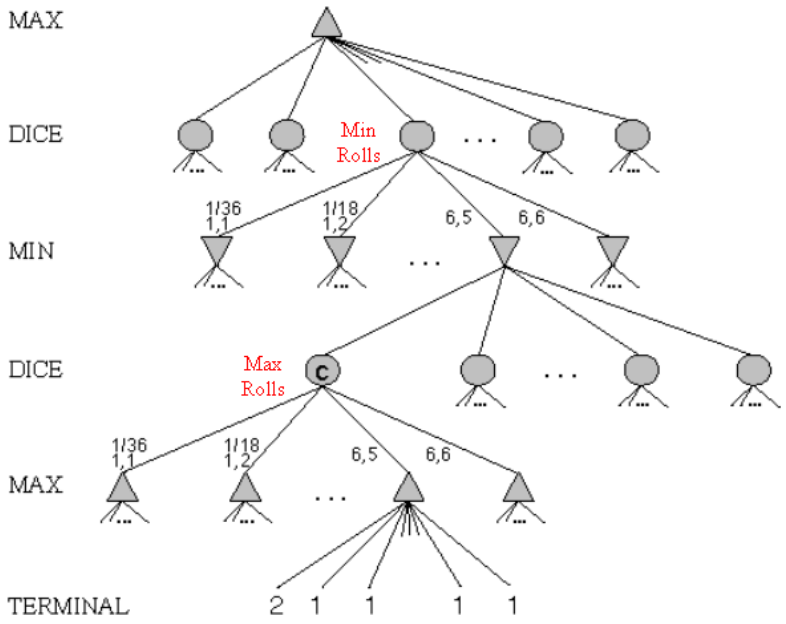
Nondeterministic games

- The trees include chance nodes (the circles in the figure), which represent random events
- For a random event with *n* possible outcomes, each chance node has *n* distinct children, and each child has an associated probability
- For example, for 2 dice, 21 results are possible

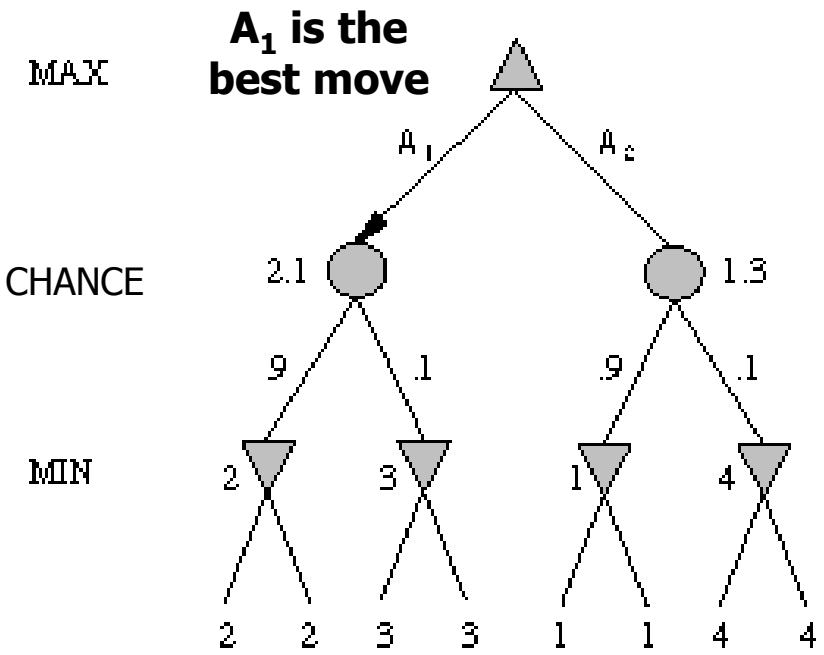


Game trees with chance nodes

- Minimax is used to compute values for MAX and MIN nodes
- Expected values are used for chance nodes
- For chance nodes over a MIN node
  - $expectimin(A) = \Sigma_i(P(d_i) \cdot minvalue(i))$
- For chance nodes over a MAX node
  - $expectimax(B) = \Sigma_i(P(d_i) \cdot maxvalue(i))$



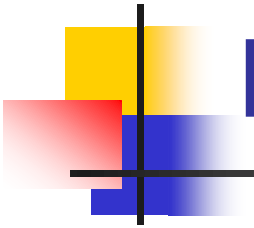
Example



2 outcomes with probabilities {0.9, 0.1}

Elements of Game Theory (I)

1. Sequential games
   - 1.1. Introduction and formalization
   - 1.2. The minimax algorithm
   - 1.3. Alpha-beta pruning
   - 1.4. Monte Carlo Tree Search
2. Strategic games
   - 2.1. Domination
   - 2.2. Pure Nash equilibrium
3. Conclusions

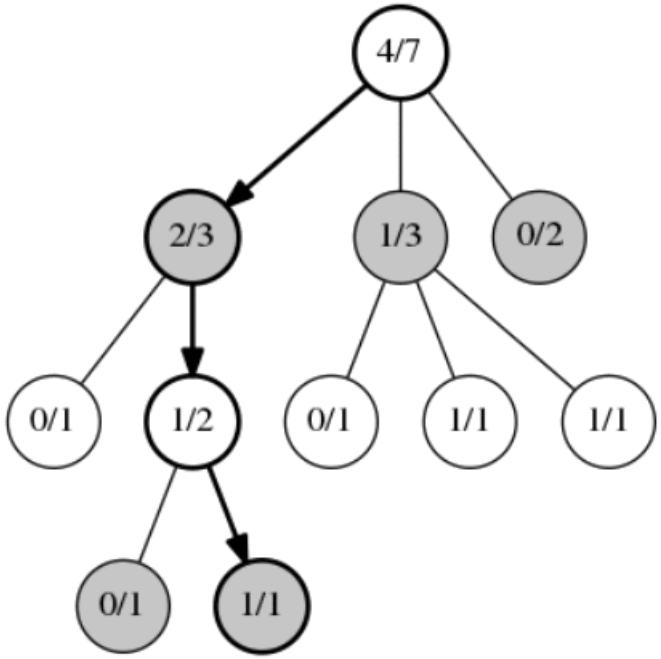


Monte Carlo Tree Search

- Monte Carlo Tree Search is a stochastic method for finding solutions in games with high branching factors
- It has been successfully used by Google DeepMind for AlphaGo, in combination with deep neural networks and reinforcement learning methods

Phase 1. Selection

- The colors of the nodes in the figure represent the two players
- It is assumed that the algorithm has already been applied several times
- Each node contains the number of wins / number of games in which it has been visited
- The moves selected by the UCB1 algorithm (see next slide) are marked with thick lines
- Selection is applied until it reaches a node with no statistics for all its children



The UCB1 selection

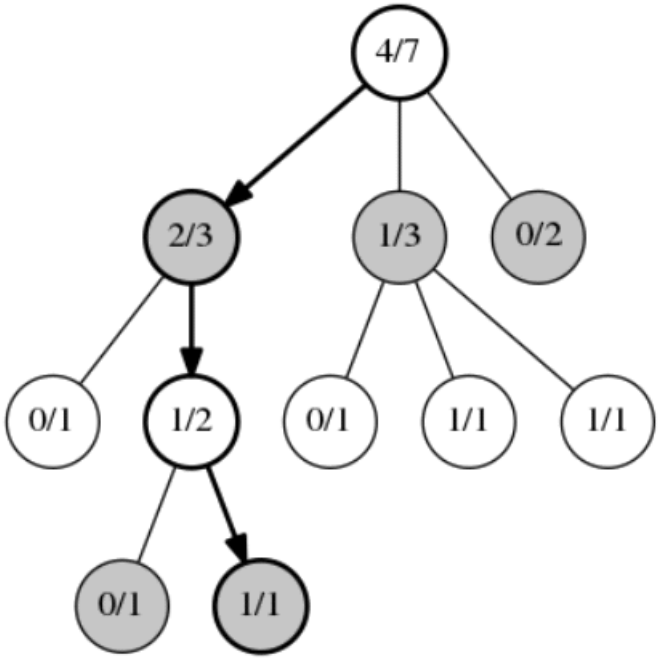
- *Upper Confidence Bound*
- From the current node, the action (child node) *i* is selected which maximizes the following expression:

$$\frac{w_i}{n_i} + \sqrt{\frac{2 \ln n}{n_i}}$$

- where: w_i is the number of wins in child *i*, n_i is the number of simulations in child *i*, and *n* is the number of simulations in the current node
- The first term represents exploitation, the second represents exploration
- MCTS with the UCB1 action selection discovers the optimal sequence of moves
- Instead of 2, there may be another value, in order to weight exploration and exploitation

Example: UCB1 selection

- Node 2/3: $\frac{2}{3} + \sqrt{\frac{2 \cdot \ln 7}{3}} = 1.806$
- Node 1/3: $\frac{1}{3} + \sqrt{\frac{2 \cdot \ln 7}{3}} = 1.472$
- Node 0/2: $\frac{0}{2} + \sqrt{\frac{2 \cdot \ln 7}{2}} = 1.395$

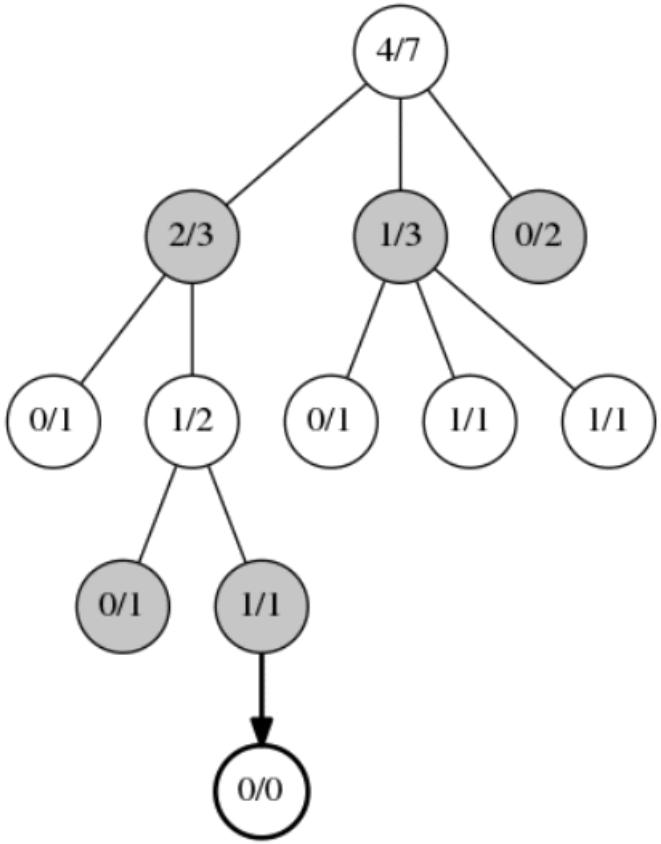


Terminal nodes

- Selection can choose a node that already represents a terminal state
- If the state represents a defeat, the node can be given a very small value (or -∞), so that it will not be chosen next time
- If the state represents a victory, the node can be given a very high value (or +∞), and its immediate parent a very small value (or -∞)

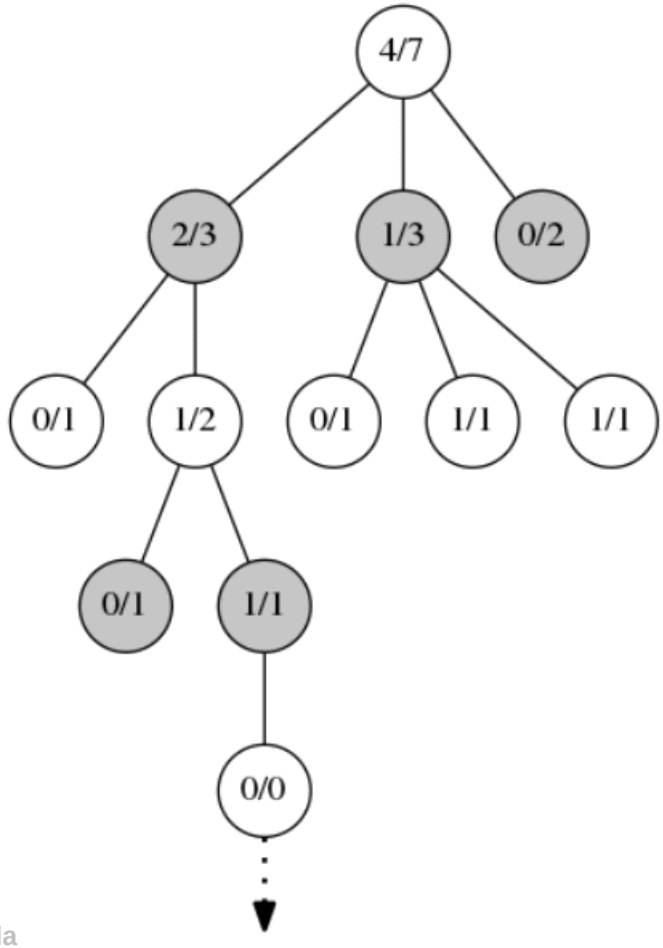
Phase 2. Expansion

- Expansion is applied when selection can no longer be applied
- A leaf node that is still unvisited is randomly selected and a new statistics record (0/0) is added for each child
- Nodes with 0/0 will be selected when they are reached in the future, as they are considered to have infinite UCB1 value



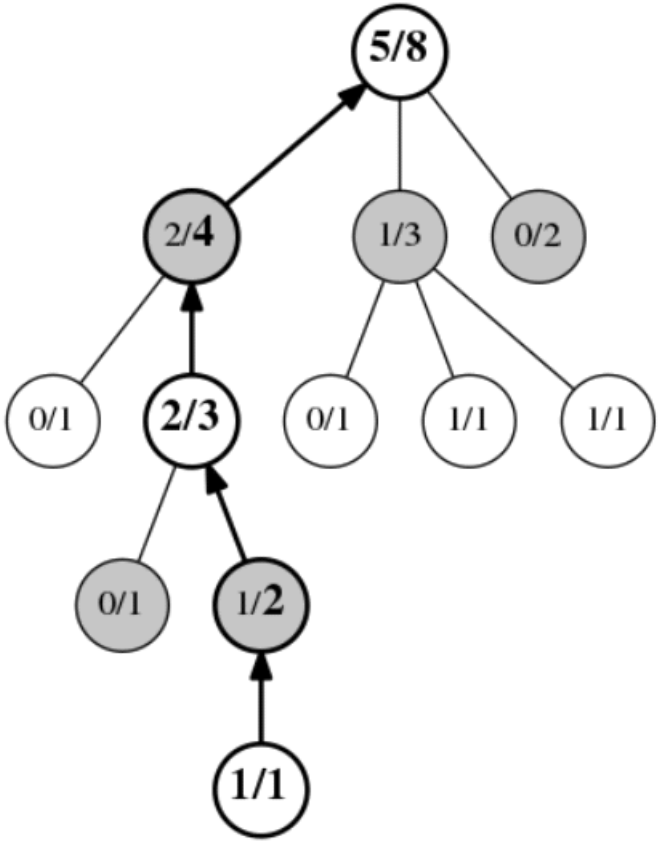
Phase 3. Simulation

- After expansion, the Monte Carlo simulation begins, in which random moves are chosen until a terminal state is reach (for example, victory or defeat)
- Such a simulation is also called *rollout* or *playout*
- Sometimes, instead of purely random search, weighting heuristics can be used to choose moves considered to be better

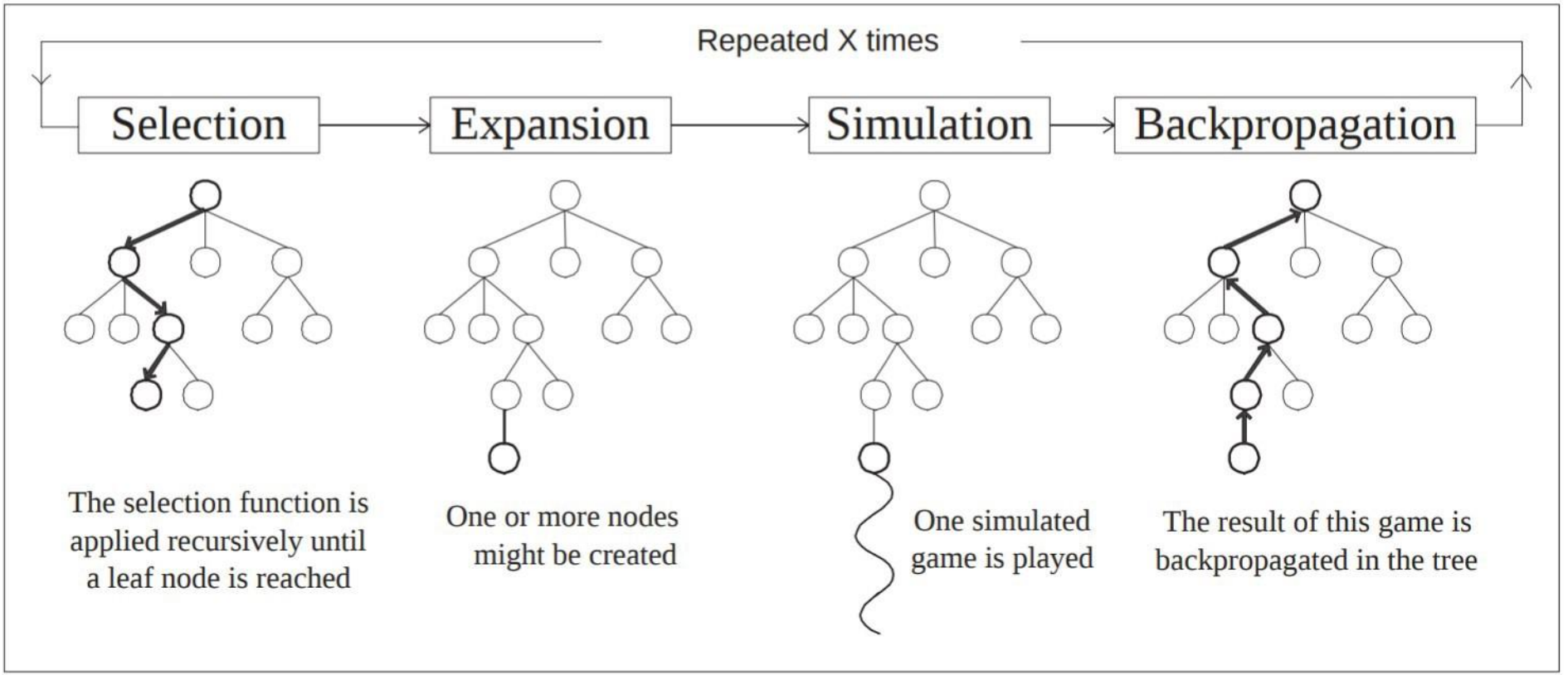


Phase 4. Backpropagation

- After the simulation is completed, the number of games and, if applicable, the number of wins is incremented for all the positions visited
- The number of wins is incremented only for the nodes on the winning player's levels
- The nodes visited down in the simulation are not updated, but only those starting up from the selected node (in the considered example, the new expanded node and all its predecessors)



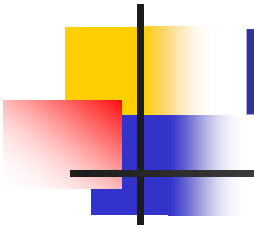
The MCTS algorithm



Choosing a move

- After applying the algorithm (repeatedly), the move with the highest number of visits (n_i in w_i / n_i) is chosen because its value is best estimated
- Since it has been explored the most, also its actual value (w_i / n_i) should be high

- After the computer and the (human) opponent make a move, the next search can reuse the values in the corresponding sub-tree as initial values



MCTS vs. minimax

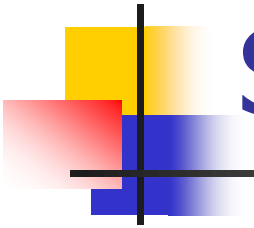
- MCTS does not need heuristics
- MCTS search is asymmetric: tree exploration converges to better moves
- MCTS is an *anytime* algorithm: at any time, it can produce an estimate of the optimal move

Current state of the art

- Backgammon (TD-Gammon): reinforcement learning with neural networks, world top 3 (1992)
- Checkers (Chinook, 1995), Othello (Logistello, 1997): computers are better than humans
- Bridge (GIB): world champion (1998)
- Go (AlphaGo): defeated world champion Ke Jie (2017) using deep reinforcement learning, variants: AlphaGo Zero, Alpha Zero
- Poker (Libratus): defeated four of the world's best players (2017)
- StarCraft (AlphaStar): defeated one of the best professional players (2019)
- Chess (Stockfish 12): 3665 Elo rating (2020), Magnus Carlsen, highest recorded: 2882

Elements of Game Theory (I)

1. Sequential games
   - 1.1. Introduction and formalization
   - 1.2. The minimax algorithm
   - 1.3. Alpha-beta pruning
   - 1.4. Monte Carlo Tree Search
2. Strategic games
   - 2.1. Domination
   - 2.2. Pure Nash equilibrium
3. Conclusions



Strategic games

- They are different from sequential games solvable for example by minimax
- They involve strategic interactions among rational agents/players, who *simultaneously* choose different actions in order to maximize their payoff

Elements of a game

- A strategic game is a situation where:
  - There are at least two players/agents
  - Each agent has a number of available strategies
  - The chosen strategies determine the outcome of the game
  - For each outcome, there is a numeric payoff for each player

The prisoner's dilemma

- Players
  - 2 prisoners
- Actions
  - Prisoner 1: Confess, Deny
  - Prisoner 2: Confess, Deny
- Strategies
  - Choose actions simultaneously, without knowing each other's actions
- Outcomes
  - Quantified in prison years
- Payoff
  - Fewer years $\Rightarrow$ better payoff

Representation of games in normal (strategic) form

- A matrix which shows the players, strategies and payoffs
- It is presumed that the players act simultaneously
- For the prisoner's dilemma:

		Agent 2: *Denies*	Agent 2: *Confesses*
Agent 1	*Denies*	-1, -1	-5, 0
	Confesses	0, -5	-3, -3

		Agent 2: *Denies*	Agent 2: *Confesses*
Agent 1	*Denies*	win-win	lose much-win much
	Confesses	win much-lose much	lose-lose

Applications of strategic games

- Wherever strategic interactions exist between rational agents
  - Economy
  - Geopolitical strategies
  - Psychology
  - Sociology
  - Computer networks: routing, sharing in peer-to-peer networks, etc.

Zero-sum games

- For any outcome of the game, the players' payoffs have the sum equal to 0

$$\begin{bmatrix} 2 & -1 \\ 1 & -2 \end{bmatrix} \qquad \begin{bmatrix} 2 & -3 \\ 0 & 2 \\ -5 & 10 \end{bmatrix}$$

The payoff of Rose ("rows") = – the payoff of Colin ("columns")

For a general sum game, pairs are necessary

For a zero-sum game, (2) is equivalent with (2, –2)

Elements of Game Theory (I)

1. Sequential games
   - 1.1. Introduction and formalization
   - 1.2. The minimax algorithm
   - 1.3. Alpha-beta pruning
   - 1.4. Monte Carlo Tree Search
2. Strategic games
   - 2.1. Domination
   - 2.2. Pure Nash equilibrium
3. Conclusions

Dominance. Definitions

- A strategy *S* dominates a strategy *T* (*T* is dominated by *S*) if every outcome of *S* is at least as good as the corresponding outcome of *T*
- A rational agent should never play a dominated strategy
- If each agent has a dominant strategy and each agent plays it, then the combination of dominant strategies and the corresponding payoffs represent the dominant strategy equilibrium of the game

Examples

Example 1. Consider the game with payoff table:

	C	D
A	(1,3)	(4,2)
B	(2,4)	(7,1)

Clearly B dominates A for Player 1 and C dominates D for Player 2. Thus (B, C) will be the dominant strategy equilibrium.

Examples

Example 2. Consider the game with payoff table:

	C	D	E
~~A~~	~~(1,1)~~	~~(2,0)~~	~~(3,−1)~~
B	(2,1)	(4,3)	(2,0)

At first sight, it seems that the row player's strategies A and B do not dominate each other, so there will be no dominant strategy equilibrium. However, the game still has it. What we need is *the Principle of Higher Order Dominance*: we first cross out any dominated strategies for the players. In the resulting smaller game, some strategies may become dominated, even though they weren't in the original game. For this example, we first cross out strategy E because it is dominated by D. Once we did that, we find out that B will dominate A, so we cross out A. Then finally, we see that D dominates C, and we arrive at our dominant strategy equilibrium (B, D).

Examples

Example 3. Consider the following 2-person zero-sum game:

	E	F	G	H
A	1	1	1	2
B	2	1	1	2
C	2	2	1	1
D	2	2	2	3

Can you find its dominant strategy equilibrium? The process of elimination should be like E, F, A, B, C, and H. Then finally we get the solution (D, G).

Dominant strategy equilibrium

$$\begin{bmatrix} 2 & -1 \\ 1 & -2 \end{bmatrix}$$

The entry -1 is the payoff to Rose if Rose plays his first strategy R_1 and Colin plays his second strategy C_2. In this case, the payoff to Colin is $-(-1) = 1$. More precisely, in this case, Rose will have to pay \$1 to Colin.

Notice that the game in Example 1 has a dominant strategy equilibrium. In fact, R_1 dominates R_2 while C_2 dominates C_1. Thus the dominant strategy equilibrium is (R_1, C_2). However, not all 2-by-2 zero-sum games have dominant strategy equilibrium

Elements of Game Theory (I)

1. Sequential games
   - 1.1. Introduction and formalization
   - 1.2. The minimax algorithm
   - 1.3. Alpha-beta pruning
   - 1.4. Monte Carlo Tree Search
2. Strategic games
   - 2.1. Domination
   - 2.2. Pure Nash equilibrium
3. Conclusions

Nash equilibrium

- A strategy profile (i.e. a combination of strategies) is a Nash equilibrium if every player maximizes his/her payoff, given the strategies used by the other players
- The Nash equilibrium identifies the strategy profiles that are stable, in the sense that each player is content with his/her chosen action, given the actions of the other players
- The behavior generated by a Nash equilibrium is expected to remain stable over time

Nash equilibrium

- Nash equilibrium for a pure strategy

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*)$$

Payoff (utility) of player *i*

Strategy in Nash equilibrium

Strategy of player *i*

Strategies of other players except for *i*

Deterministic, not involving probabilities

Nash equilibrium

- Pure strategy Nash equilibrium

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*)$$

- Pure strategy strict Nash equilibrium

$$u_i(s_i^*, s_{-i}^*) > u_i(s_i, s_{-i}^*)$$

- The states from which no player can increase his/her expected payoff by unilaterally changing his/her strategy

Computing pure NEs

- We highlight the maximum values on the rows for the first player with {
- We highlight the maximum values on the columns for the second player with }
- The states enclosed by { } are pure Nash equilibria

		Prisoner 2	
		Denies	Confesses
Prisoner 1	Denies	-1, -1	-5, 0 }
	Confesses	{ 0, -5	{ -3, -3 }

Examples

The prisoner's dilemma

		Agent 2: Denies	Agent 2: Confesses
Agent 1	Denies	-1, -1	-5, 0
	Confesses	0, -5	-3, -3

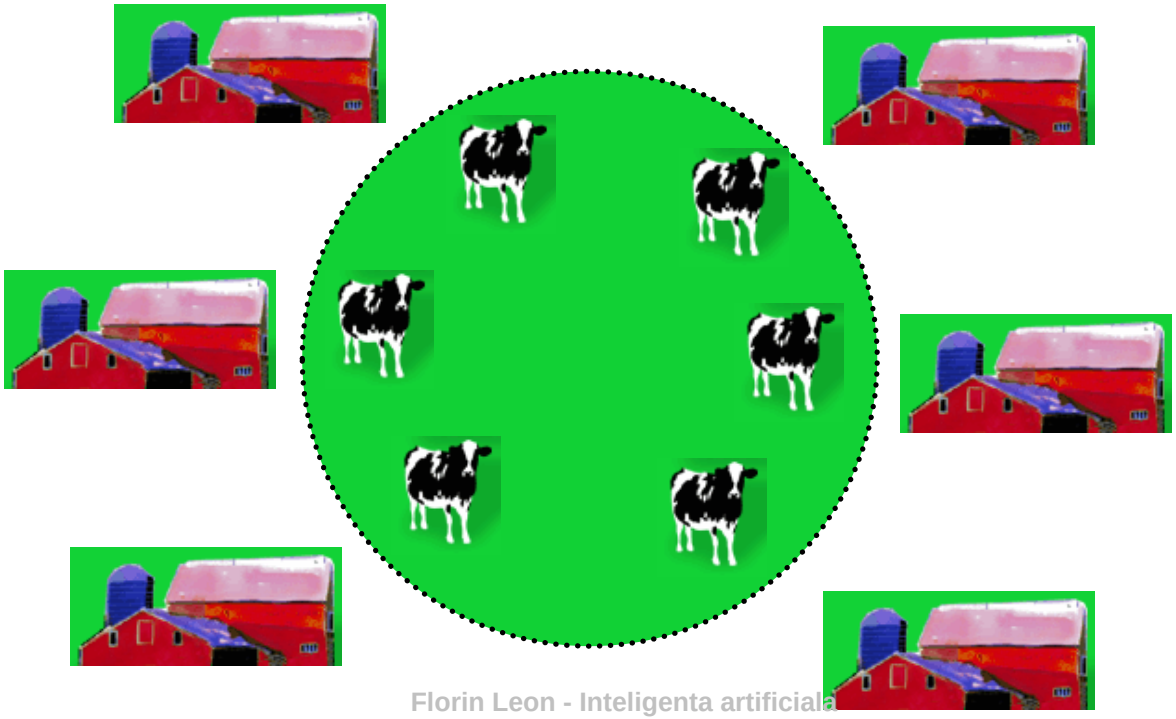
NE

The battle of the sexes

		Mary: Football (A)	Mary: Opera (B)
John	Football (A)	2, 1	0, 0
	Opera (B)	0, 0	1, 2

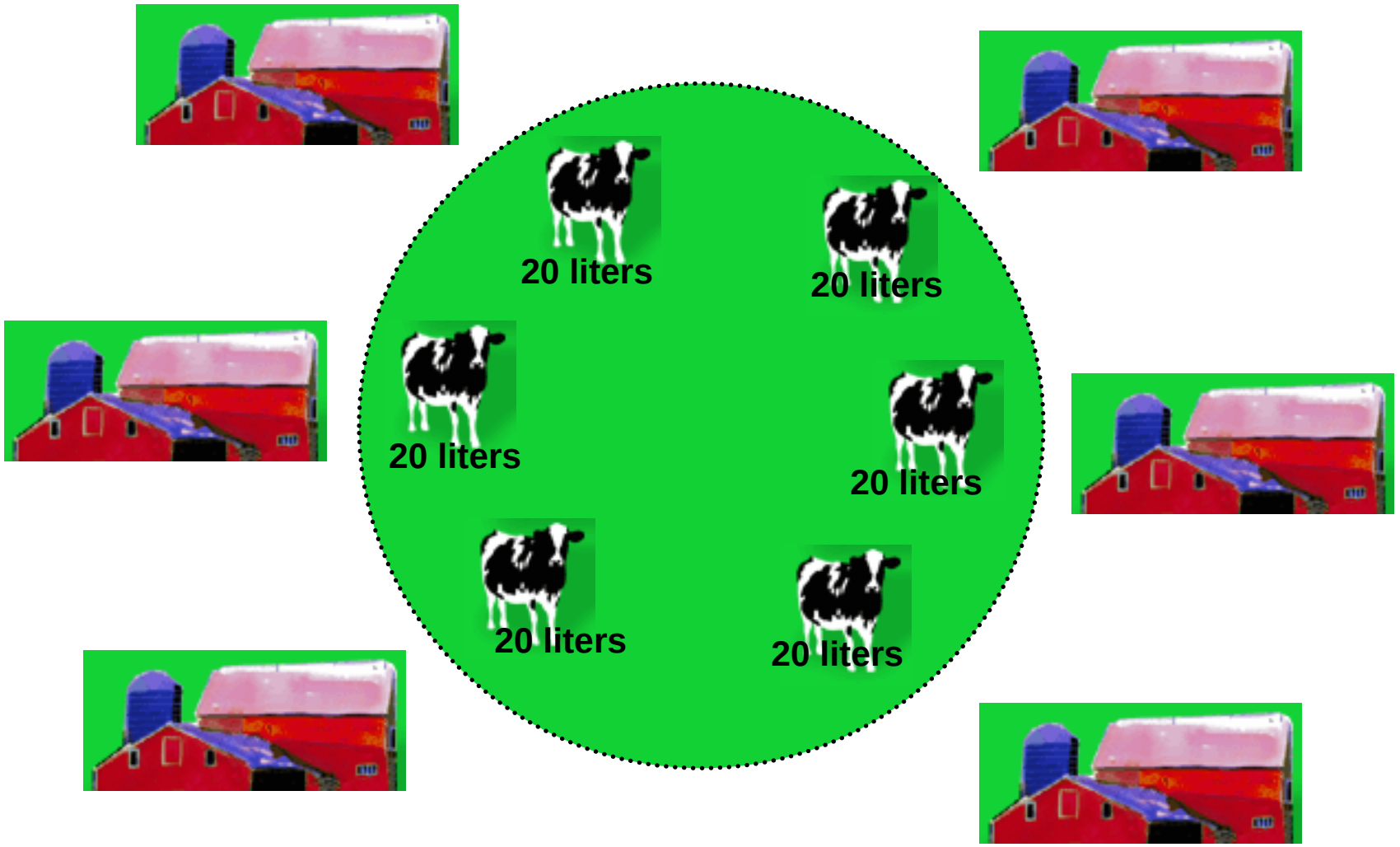
The tragedy of the commons

- Imagine a field of grass shared by 6 farmers, each with one cow



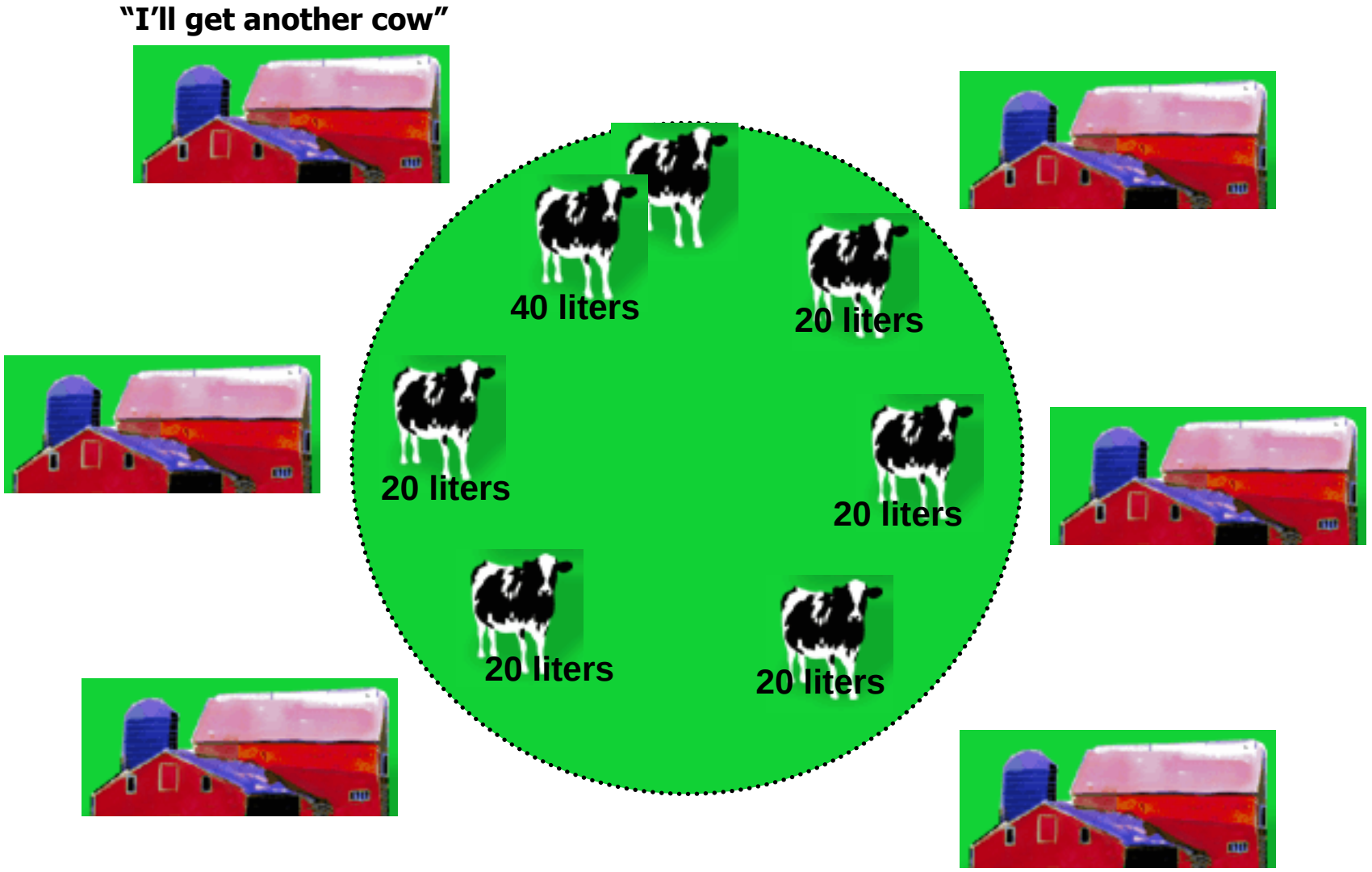
The tragedy of the commons

- Each cow currently produces 20 liters of milk per day
- The carrying capacity of the commons is 8 cows
- For each cow above 8, the milk production declines by 2 liters
  - Due to overgrazing, there is less grass for each cow: less grass, less milk



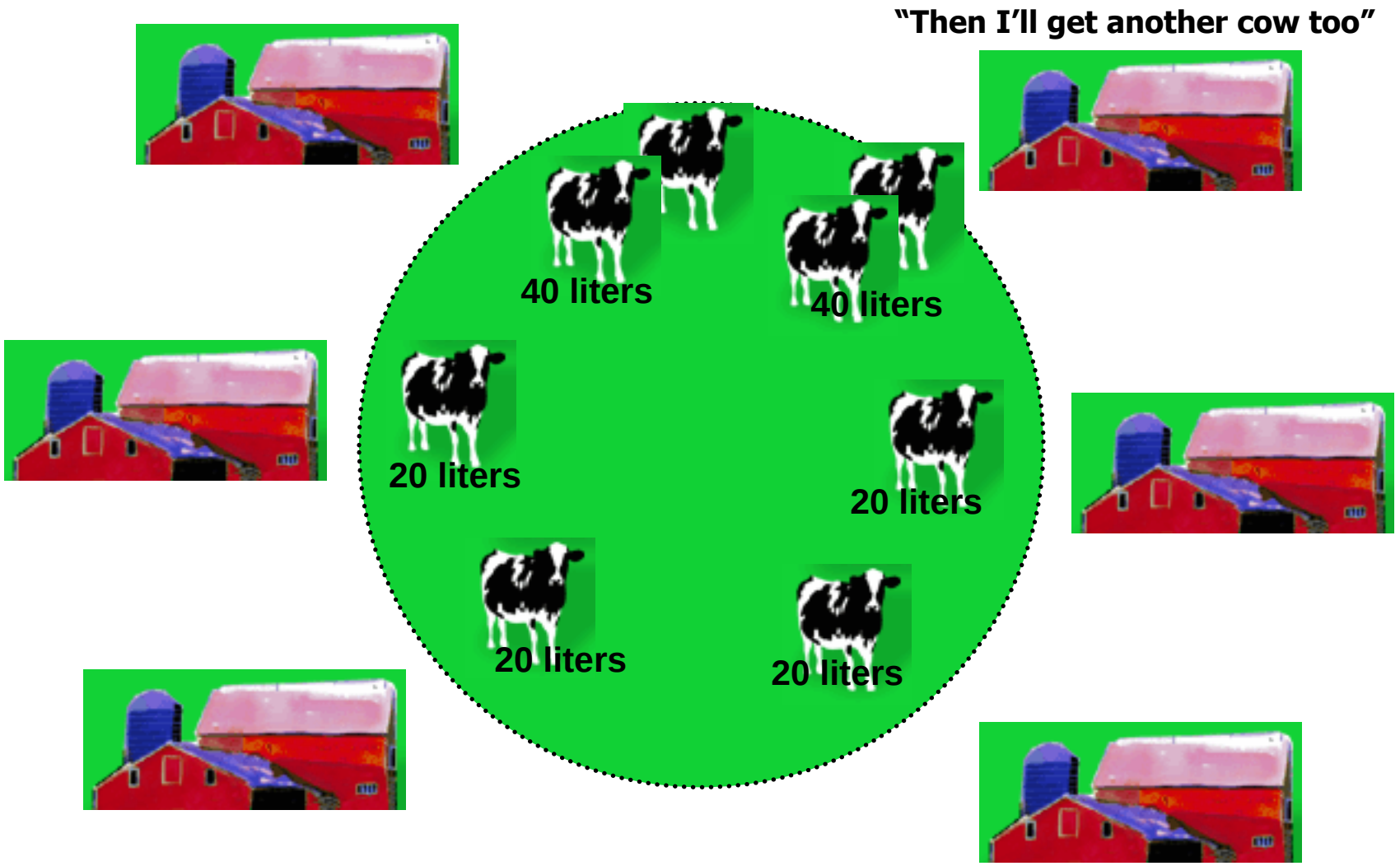
Total daily milk production for the commons: 120 liters

Farmers want to maximize their milk production

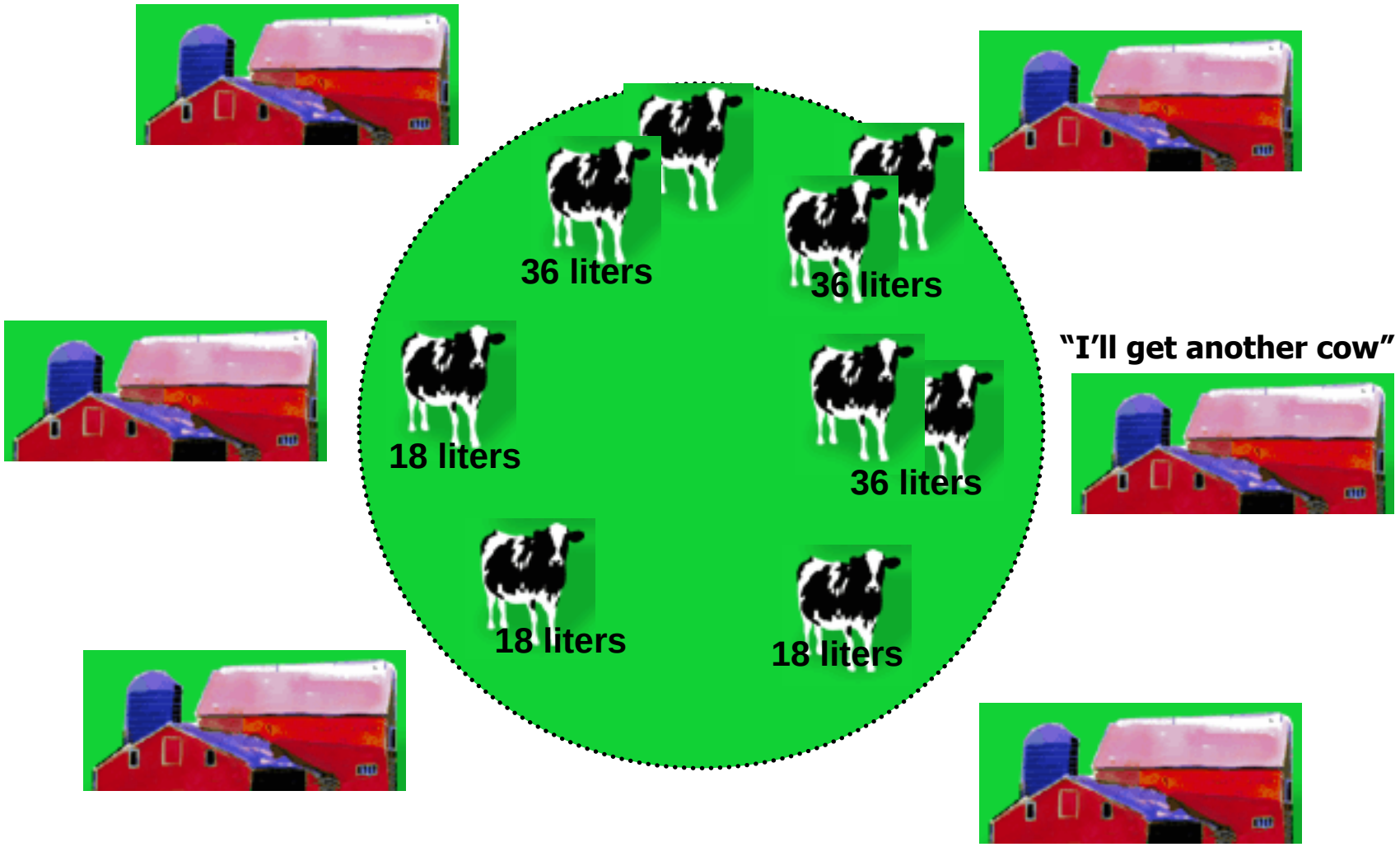


Total daily milk production for the commons: 140 liters (7 cows)

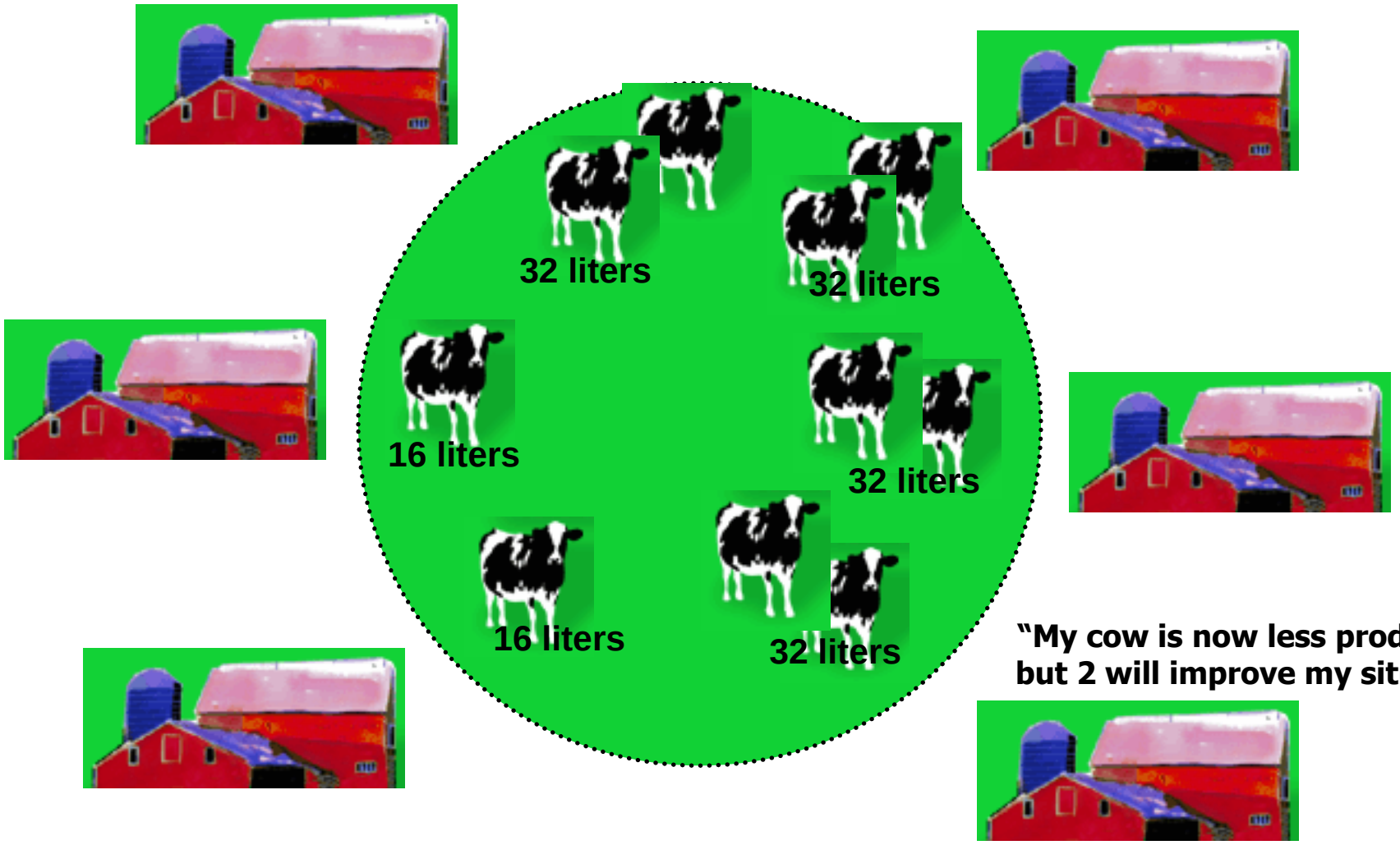
They are now at the carrying capacity -- do they stop? No.



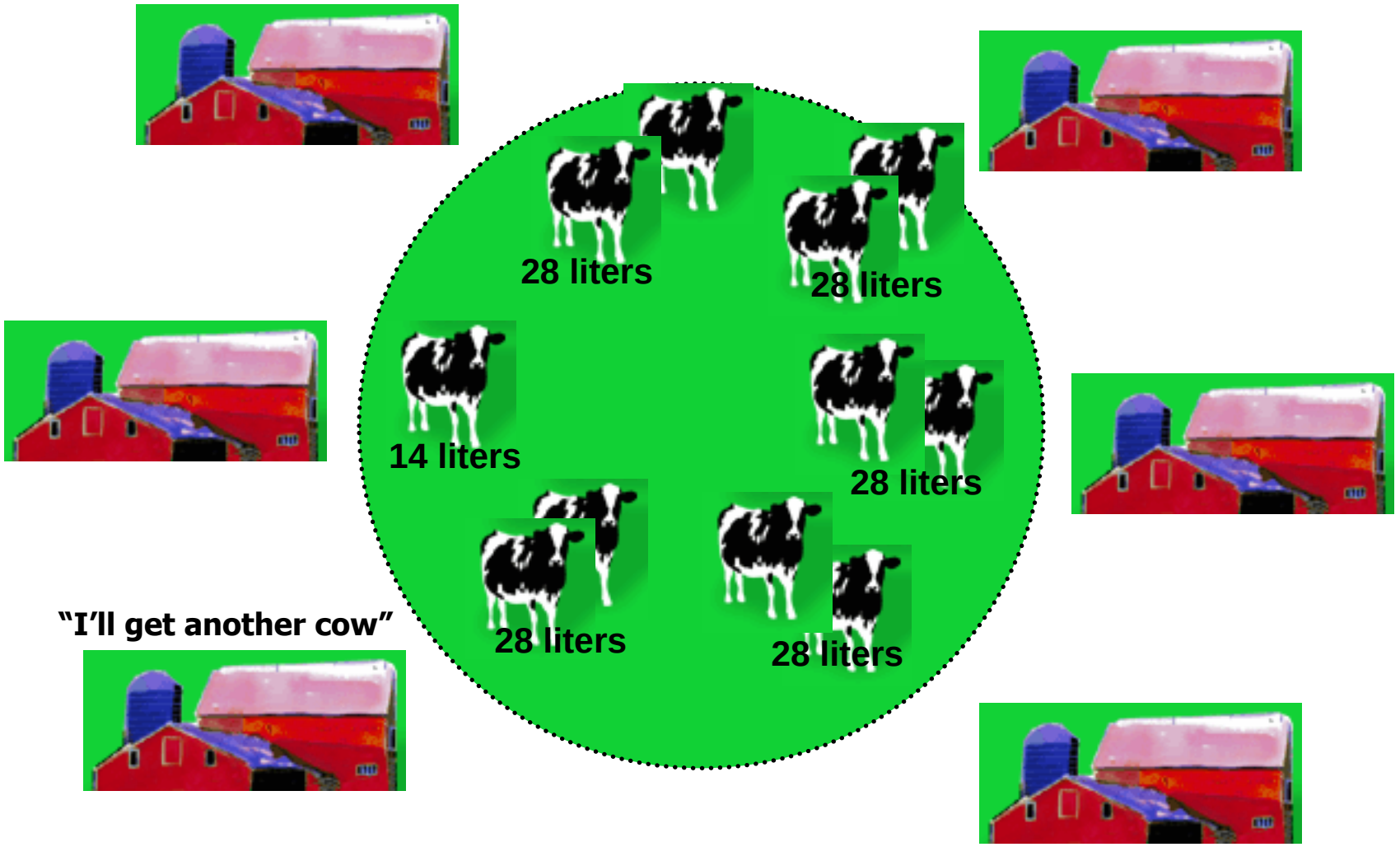
Total daily milk production for the commons: 160 liters (8 cows)



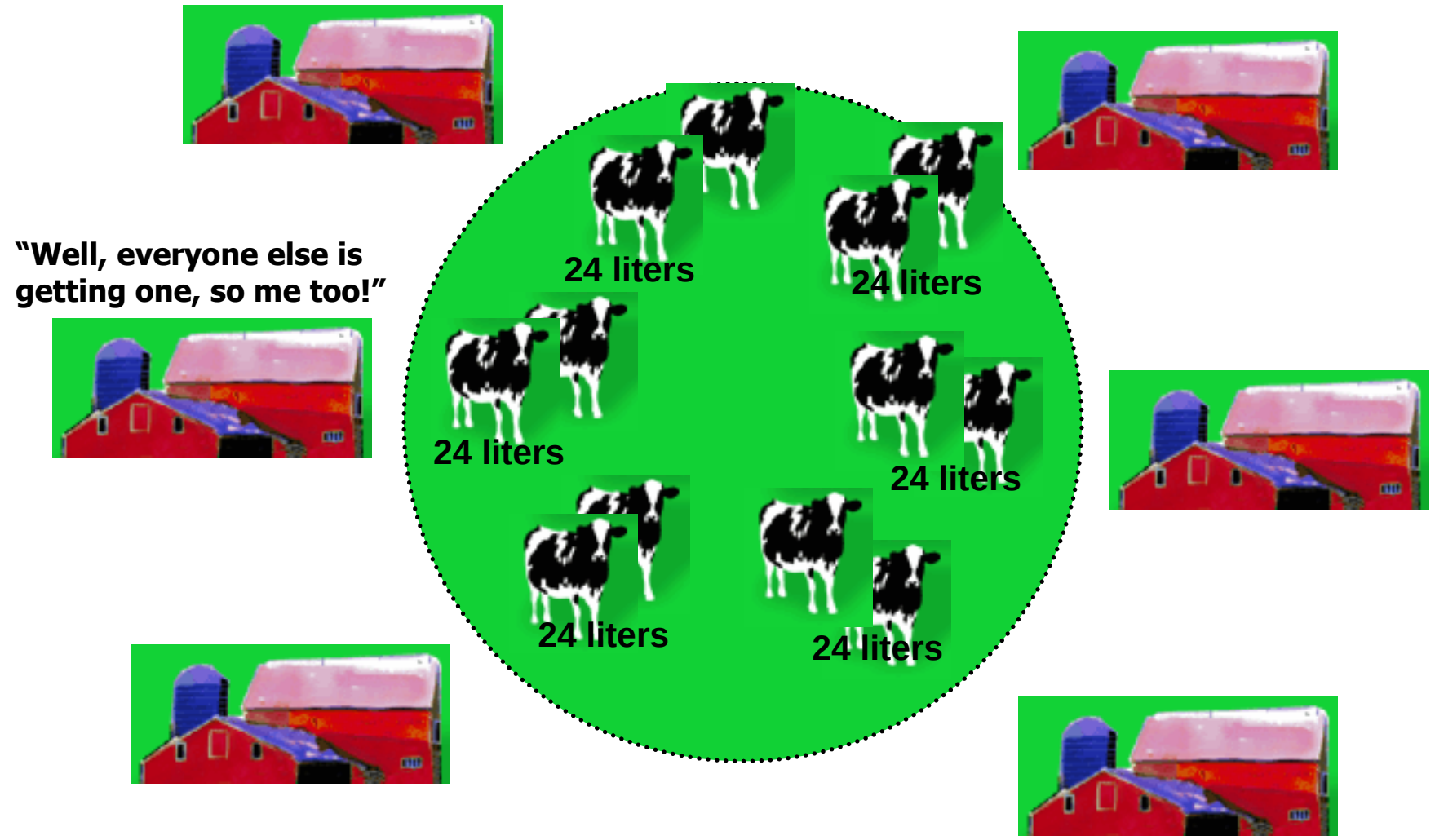
Total daily milk production for the commons: 162 liters (9 cows)



Total daily milk production for the commons: 160 liters (10 cows)

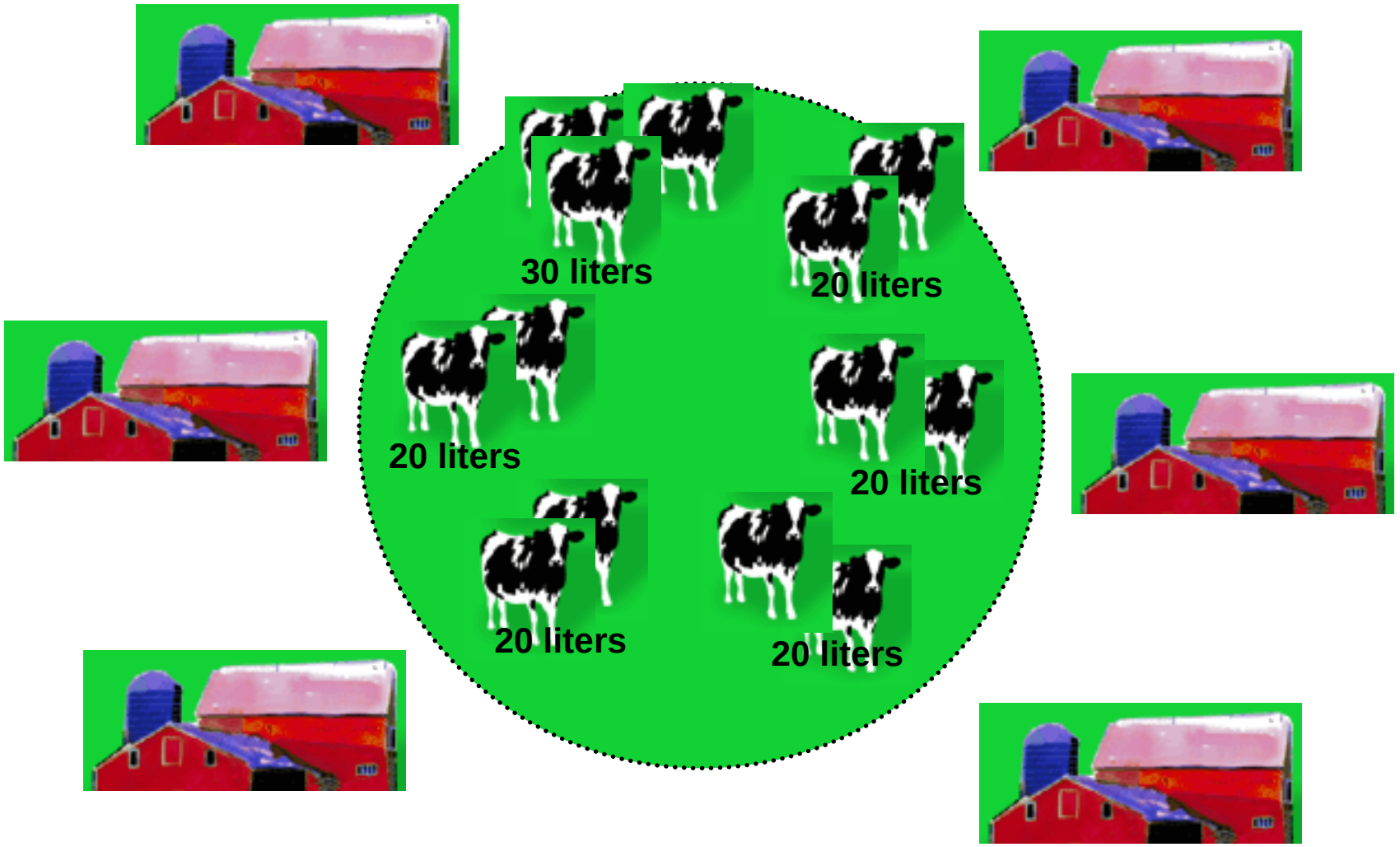


Total daily milk production for the commons: 154 liters (11 cows)

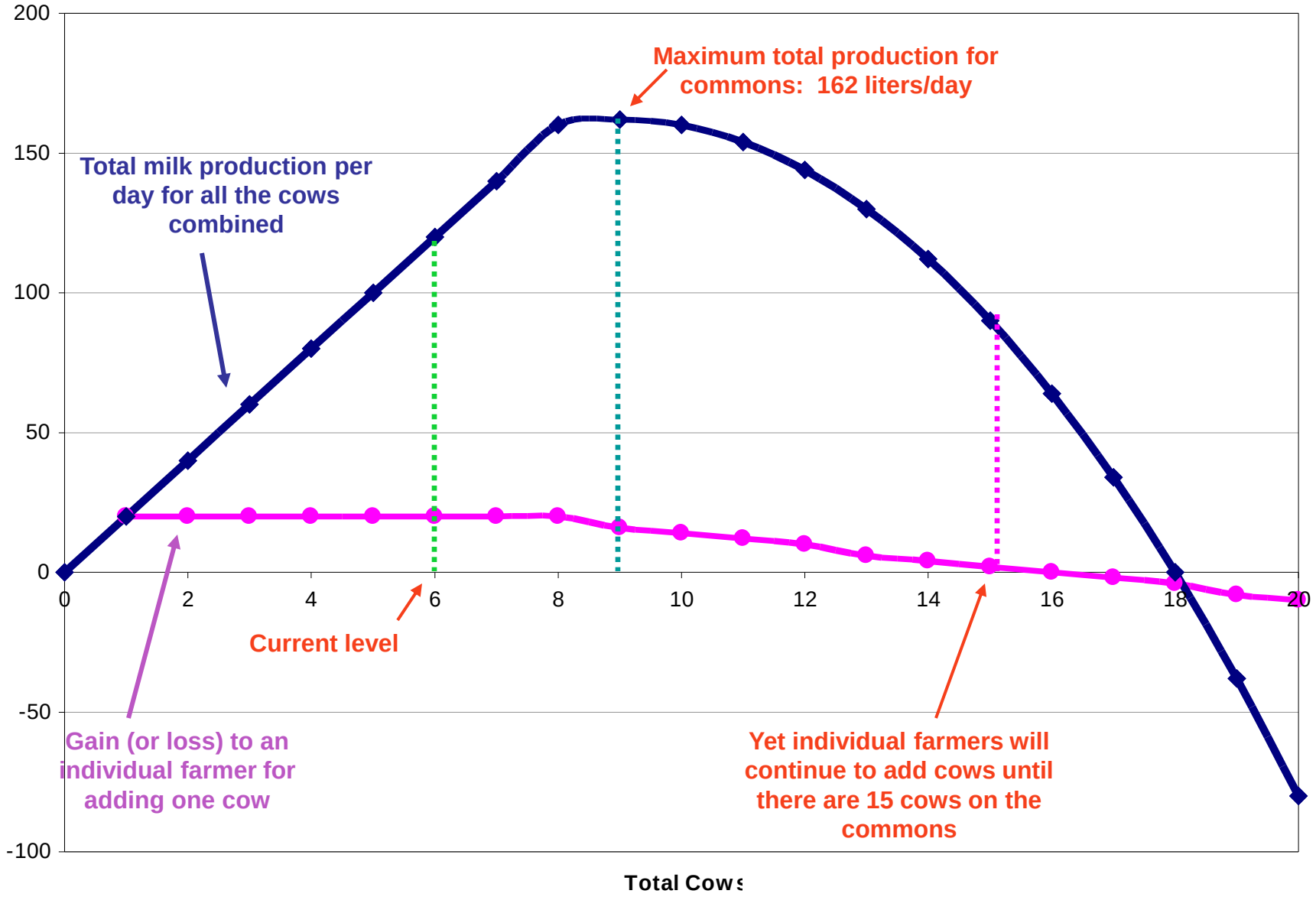


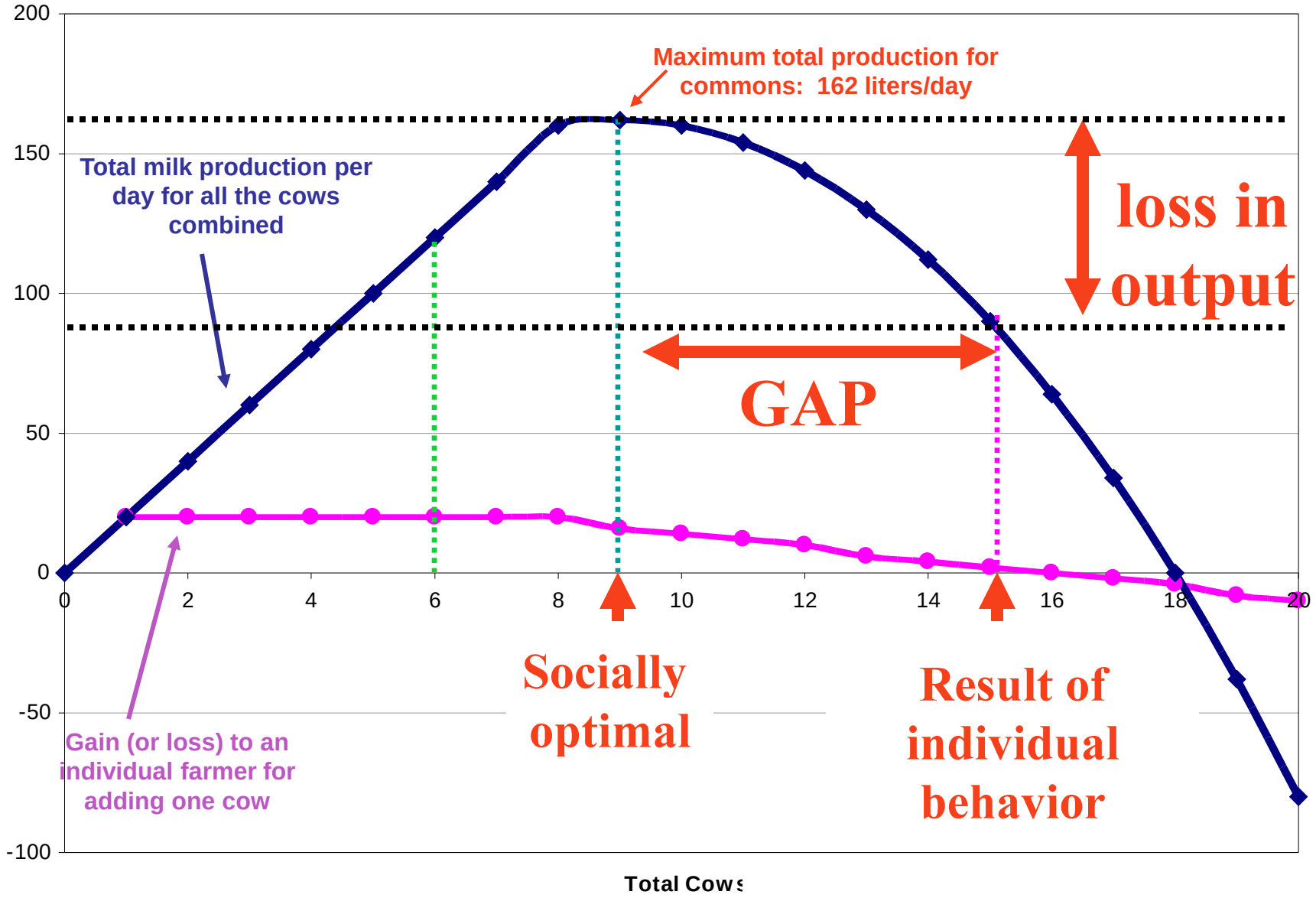
Total daily milk production for the commons: 144 liters (12 cows)

"Well, I can still increase milk production if I get a third cow"



Total daily milk production for the commons: 130 liters (13 cows)


Maximum total production for commons: 162 liters/day
Total milk production per day for all the cows combined
Current level
Gain (or loss) to an individual farmer for adding one cow
Yet individual farmers will continue to add cows until there are 15 cows on the commons
200
150
100
50
0
-50
-100
0
2
4
6
8
10
12
14
16
18
20
Total Cows

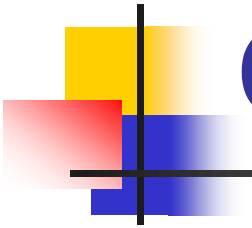

200
150
100
50
0
-50
-100
0
2
4
6
8
10
12
14
16
18
20
Maximum total production for commons: 162 liters/day
Total milk production per day for all the cows combined
loss in output
GAP
Socially optimal
Result of individual behavior
Gain (or loss) to an individual farmer for adding one cow
Total Cows

Solutions?

- Cooperative agreement between farmers
  - Profit sharing for the additional 3 cows
- Consolidation
  - One firm runs the entire set of farms and thus becomes a single profit center
- "State" regulations
  - Set an upper limit on the number of cows
  - Force redistribution
- Mechanism design
  - Incentives and penalties for individual players so that they are inclined to reach the social optimum

Social vs. individual benefit

- Sharing in P2P networks
- Pollution
- ... in general, the management of common property resources



Conclusions

- Game theory analyzes abstract models of multiagent interactions. This field deals with many types of "games" and decision situations
- In this lecture, we only referred to sequential and strategic games
- Minimax is a recursive algorithm that finds the optimal move of a player, given a certain search depth in the game tree
- Alpha-beta pruning is an often used optimization for minimax
- A strategy profile (i.e. a combination of strategies) is a Nash equilibrium if every player maximizes his/her payoff, given the strategies used by the other players. The strategies in a Nash equilibrium are stable