Introduction to Artificial Intelligence

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

So far, we discussed behavior of a single agent.

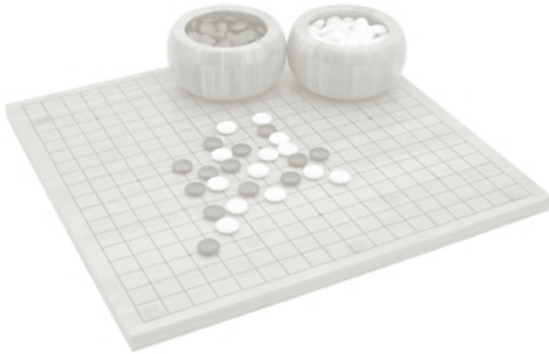
What if there are other agents in the environment that influence agent's welfare?

Game theory, a branch of economics, views any multi agent environment as a game (impact of each agent on the others is "significant").

The most common games in AI – **deterministic, turn taking, two-player, zero-sum games of perfect information**

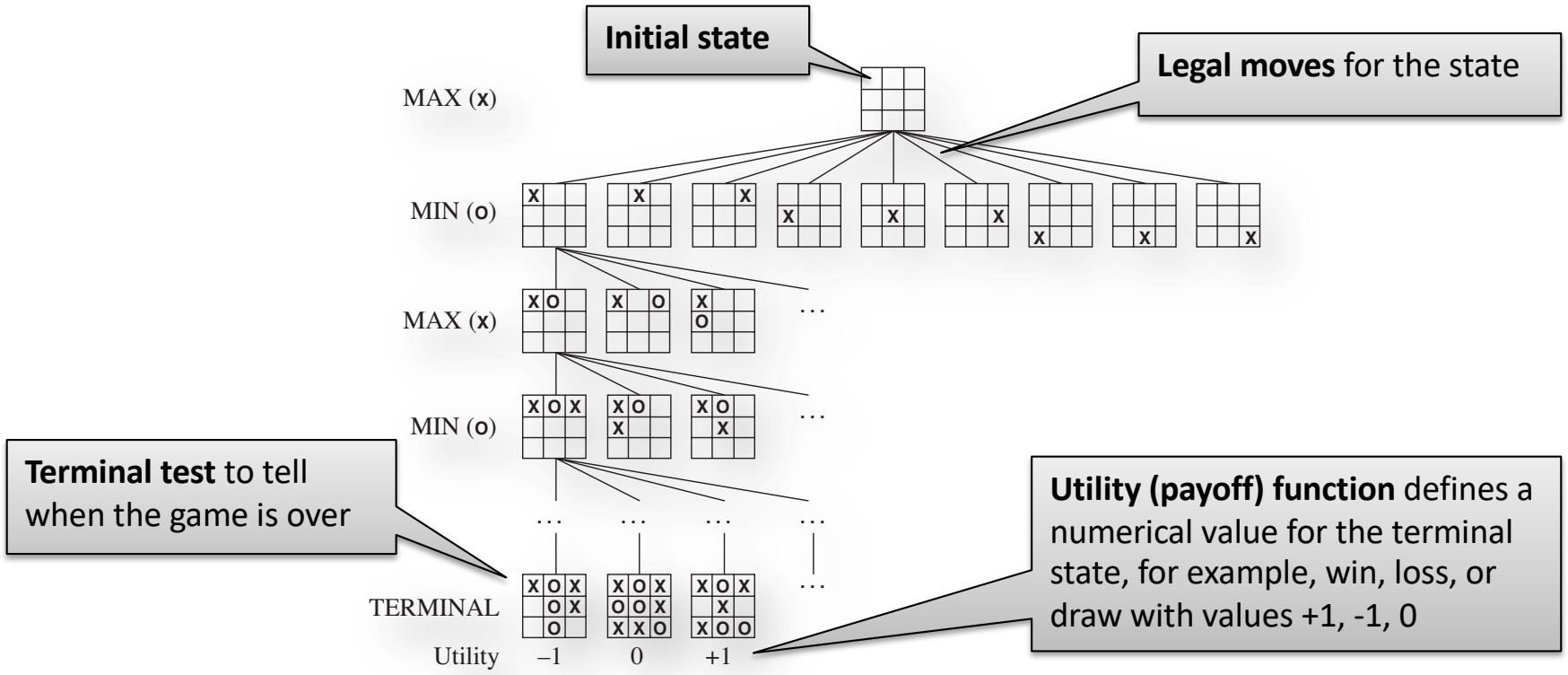
Examples: Chess, Go, …

easy to represent, a small number of actions, precise rules, but still hard to solve



We consider turn taking games with two players, MIN and MAX.

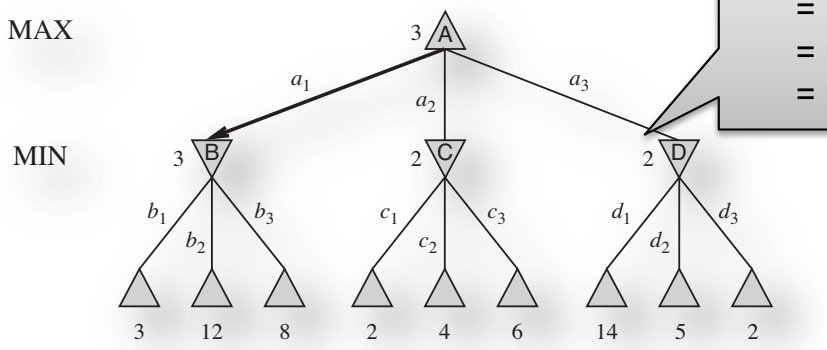
Game can be formally defined as a kind of search (called **adversarial search**) in a **game tree** representing all possible moves.



In adversarial search, the solution consists of a **strategy**: the move in the initial state and then moves for every possible response by the opponent.

Optimal strategy leads to outcomes at least as good as any other strategy when playing with infallible opponent.

Optimal strategy can be determined from the **minimax value** – utility of being in the corresponding state, assuming that both players play optimally.



MINIMAX-VALUE(**n**)

= UTILITY(**n**) if **n** is a terminal state

= $\max_{\mathbf{s}\in succ(\mathbf{n})}$ MINIMAX-VALUE(**s**) if MAX plays in **n**

= $\min_{\mathbf{s}\in succ(\mathbf{n})}$ MINIMAX-VALUE(**s**) if MIN plays in **n**

```
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
```

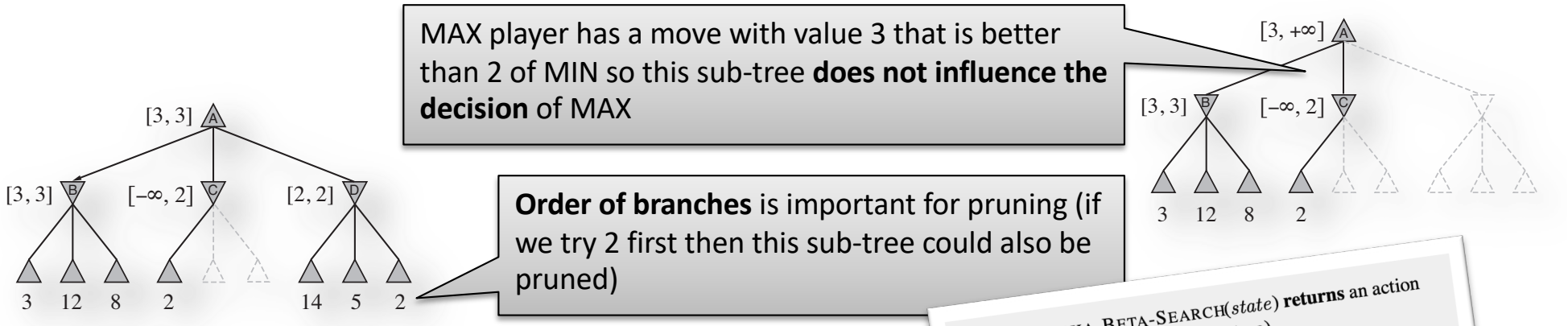
The **minimax decision** – action in the root

– calculated using the **minimax algorithm**

Not practical for real games but serves as basis for more practical algorithms.

It is not necessary to look at every node of the game tree to calculate the correct minimax decision.

Some **sub-trees can be pruned** as they will never be reached in actual play – a better move is available that will not lead to the sub-tree



Alfa-beta pruning uses bounds for best values for MIN and MAX long the path:

- α is the value of best (i.e. the highest-value) choice we have found so far at any choice point along the path for MAX
- β is the value of best (i.e. the lowest-value) choice we have found so far at any choice point along the path for MIN

Alpha-beta can solve a tree roughly twice as deep as minimax in the same amount of time.

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

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

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

In real games, we can not generate the entire game tree even if using alpha-beta pruning.

For example, the search tree for chess has about 35^{100} nodes (10^{154}), branching factor of about 35 and about 50 moves by each player

We can **cut off search** earlier and use evaluation function instead of utility function for (non-terminal) leaf nodes.

Evaluation function returns an estimate of the expected utility of the game from a given position (just as the heuristic function).

Most evaluation functions work by calculating various **features of the state**.

For example, in chess, the number of white and black pawns, bishops etc.

Numerical contributions from each feature are **combined**.

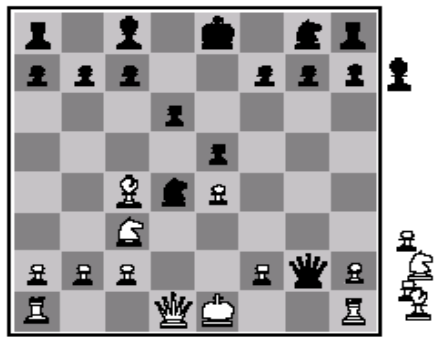
For example, chess uses **material value**: each pawn is worth 1, a knight or bishop is worth 3, a rook 5, and the queen 9

Evaluation function is then a weighted linear function:

$$\mathbf{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

This uses strong assumption that the contribution of each feature is independent of the values of the other features.

→ if not, use non-linear combinations

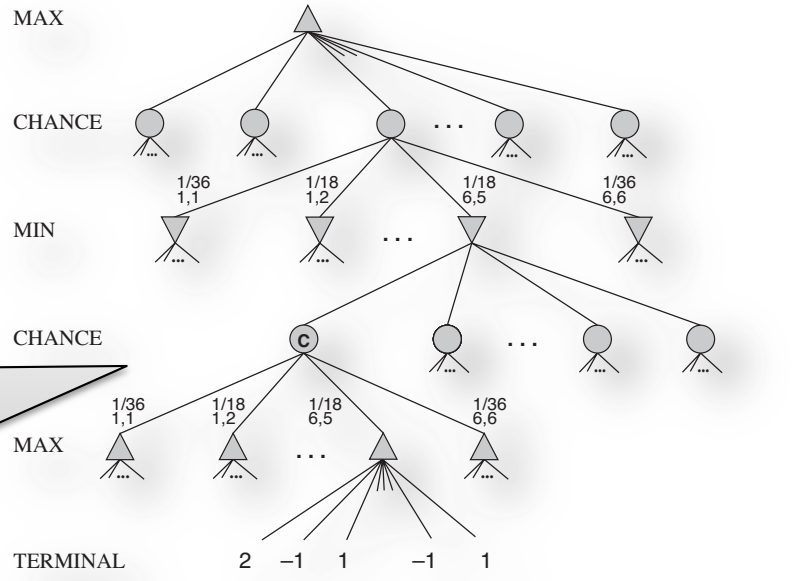


In real-life, **unpredictable external events** can put us into unforeseen situations.

This can be modeled in games by including a **random element**, such as the throwing of dice.

Stochastic games

We add **chance nodes** to a game tree and calculate **expected minimax value**

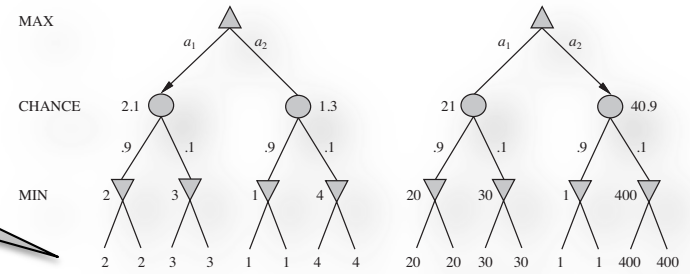


EXPECTMINIMAX-VALUE(**n**)

= UTILITY(**n**)	if **n** is a terminal state
= $\max_{\mathbf{s}\in succ(\mathbf{n})}$ EXPECTMINIMAX-VALUE(**s**)	if MAX plays in **n**
= $\min_{\mathbf{s}\in succ(\mathbf{n})}$ EXPECTMINIMAX-VALUE(**s**)	if MIN plays in **n**
= $\sum_{\mathbf{s}\in succ(\mathbf{n})}$ P(s) . EXPECTMINIMAX(**s**)	if **n** is a chance node

Evaluation function for stochastic games must be designed more carefully as absolute values are important, not only the ordering

ordering of leaf values is identical, but different best move is selected



Let us consider a restricted set of games, where all players take action simultaneously and the result of the game is based on this set of actions – **single-move games**.

- **players**, agents who will be making decisions
- **actions**, that the players can choose
- a **payoff function** that gives the utility to each player for each combination of actions by all the players

Example: two-finger Morra

Players O(dd) and E(ven) show one or two fingers and the utility is given by the payoff function.

	O: one	O: two
E: one	E=+2, O=-2	E=-3, O=+3
E: two	E=-3, O=+3	E=+4, O=-4

Players apply a **strategy** (policy) to select the action:

- **pure strategy** is a deterministic policy (single action)
- **mixed strategy** is randomized policy that selects actions according to a probability distribution; [7/12:one; 5/12:two]

Solution to a game is assignment of rational strategy to each player.

- Two alleged burglars, Alice and Bob, are caught red-handed near the scene of burglary and are interrogated separately.
- A prosecutor offers each a deal: if you testify against your partner as the leader of a burglary ring, you will go free while your partner will serve 10 years in prison.

	Alice: testify	**Alice: refuse**
Bob: testify	A=-5, B=-5	A=-10, B=0
Bob: refuse	A=0, B=-10	A=-1, B=-1

- However, if you both testify against each other, you will both get 5 years.
- If you both refuse to testify, you will serve only 1 year each for lesser charge of possessing stole property.

The rational decision is to testify, which is a **dominant pure strategy:**

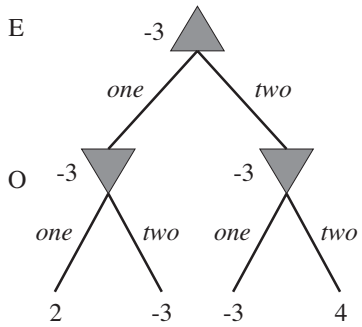
  – a strategy s for player p **dominates** strategy s' if the outcome for s is better for p than the outcome for s', for every choice of strategies by the other player(s)

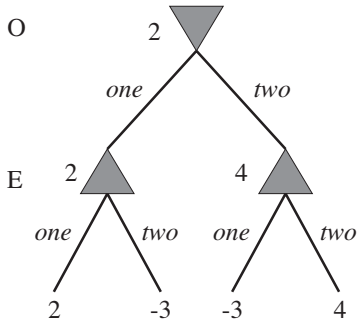
Nash equilibrium - no player benefits by switching strategy, given that every other player sticks with the same strategy

Outcome is **Pareto dominated** by another outcome if all players would prefer the other outcome. Outcome is **Pareto optimal**, if there is no other outcome that all players would prefer.

Prisoner's dilemma is due to having a dominant strategy equilibrium (testify, testify) that is Pareto dominated by outcome (refuse, refuse).

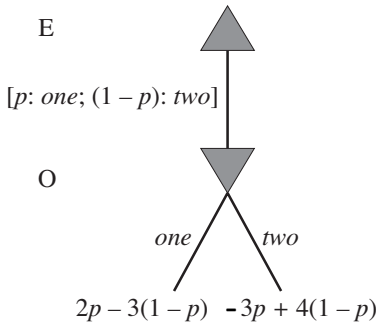
Two finger Morra game has no pure strategy.
How to find optimal mixed strategy for two-player, zero-sum games?

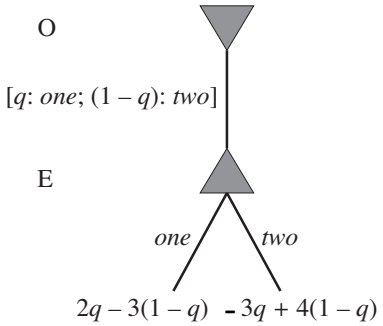


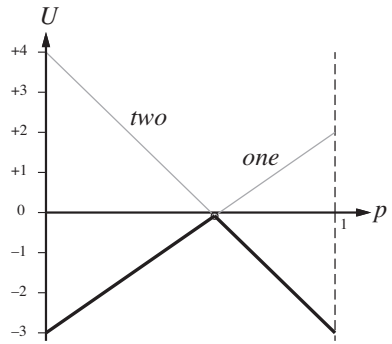


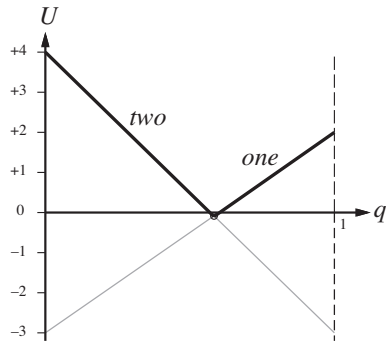
Assume that we play the game as a **turn-taking game** (the second player is favored as the first action is known).

- By trying both orders of players, we can calculate lower and upper bounds for utilities of both players (using minimax algorithm).
- Lower bound for the true utility of E is -3 (left) and upper bound is 2 (right).









Now, we apply the same approach to **mixed-strategies**.

- Once the first player has revealed his strategy, the second player might as well choose a pure strategy.
- What is the value of p to get the best utility for E (left)?
  - p=7/12 and the payoff -1/12
- What is the value of q to get the best utility for O (right)?
  - q=7/12 and the payoff -1/12

Optimal strategy for both players is

[7/12:one; 5/12:two]

Two-finger Morra game favors the player O.

→ Maximin technique

Repeated game is the simplest kind of a multiple-move game:

- players face the same choice repeatedly, but each time with knowledge of the history all players' previous choices
- payoffs are additive over time

Strategies for the repeated version of the prisoner's dilemma**:**

- the same players play 100 rounds
  - rational strategy is still to *testify* (the last game is not the repeated game etc.)
    - earning a total jail sentence of 500 years each
- 99% chance that the players meet again
  - the expected number of rounds is still 100, but neither player knows for sure which round will be the last
  - **perpetual punishment** strategy: each player *refuses* unless the other player has ever played *testify*
    - the expected future payoff is -100 ($\sum_{t=0}^{\infty} 0.99^t * (-1)$) if both players adopted this strategy
  - **tit-for-tat** strategy: starting with *refuse* and the echoing the other player's previous move on all subsequent moves
    - highly robust and effective against a wide variety of strategies

Agent design

- **game theory** can analyze the agent's decisions and compute the expected utility for each decision (under the assumption that other agents are acting optimally according to game theory)

Mechanism design

- **inverse game theory** make it possible to define the rules of the environment so that the collective good of all agents is maximized (when each agent adopts the game-theoretic solution that maximizes its own utility)
- Useful for solving complex problems in a distributed fashion.

Auction is a mechanism for selling some goods to members of a pool of bidders.

Bidder i may have a **private value** v_i for the item (for example, a collector may value some item differently than a regular person)

or the item has a **common value** (but different bidders may have different estimates of the item's true value).

At some time, each bidder gets a chance to make a **bid** b_i.

The highest bid, b_{max}, wins the item, but the price paid need not be b_{max} – this is determined by the **mechanism design**.

How to determine a **good mechanism**?

- maximize expected revenue for the seller
- maximize global utility - the winner of the auction is the agent who values the item the most (auction is **efficient**)

It is desirable that the bidders have a **dominant strategy** (works against all other strategies).

agent just bids without wasting time
contemplating other agents' possible strategies

ascending-bid auction (English auction)

- start with minimum (reserve) bid b_{min}
- if some bidder is willing to pay that amount, then ask for $b_{min} + d$
- continue until nobody is willing to bid anymore, then the last bidder wins the item, paying the price he bid

It has a **simple dominant strategy** (bid if price is not greater than v_i).

But it can **discourage competition**, if there is a known strong (rich) bidder.

It has also **high communication cost** (bidders need to meet or have high-speed secure connection to an auctioneer).

descending-bid auction (Dutch auction)

- start with high price and decrease it until somebody accepts the price

It is **fast**, hence used to sell fruits, flowers etc.

sealed-bid auction

- each bidder makes a single bid and communicates it to the auctioneer
- the highest bid wins and pays the price of the bid

No simple dominant strategy (if you believe that maximum of all the other bids is b_0 then you should bid $b_0 + \varepsilon$, if that is less then v_i).

sealed-bid second-price auction (Vickrey auction)

- each bidder makes a single bid and communicates it to the auctioneer
- the highest bid wins and pays the price of the second highest bid

The **dominant strategy** is now simply to bid v_i.

Consider another type of game, in which countries set their policy for controlling air pollution.

Each country has a choice

- they can **reduce pollution** at a cost of **-10** points for implementing the necessary changes
- or they can **continue to pollute**, which gives them a net utility of **-5** (in added health costs, etc.) and also contributes **-1** points to every other country (because the air is shared across countries)

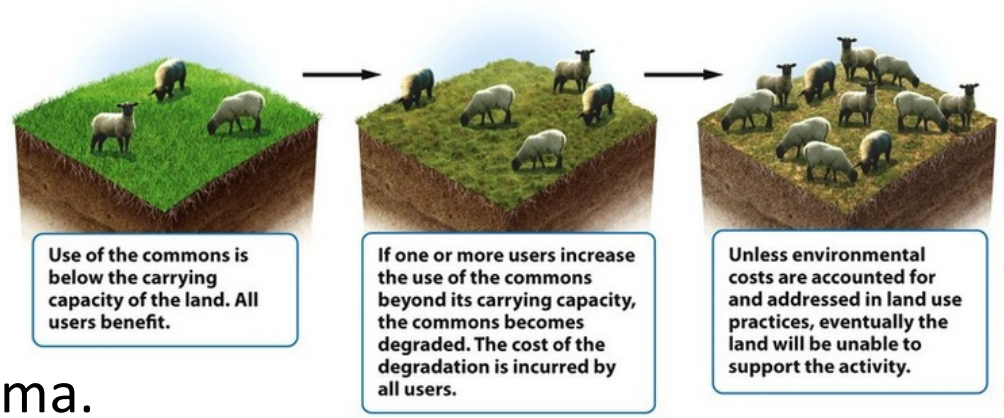
What is the strategy of each country?

- Clearly, the dominant strategy for each country is “continue to pollute”.

Tragedy of commons:

if nobody has to pay for using a common resource, then it tends to be exploited in a way that leads to a lower total utility for all agents.

It is similar to the prisoner’s dilemma.



A standard approach for dealing with the tragedy of commons is to change the mechanism to one that charges each agent for using the commons (a carbon tax).

We need to ensure that all **externalities** – effects on global utility that are not recognized in the individual agents' transactions – are made explicit.

Consider the problem of allocating some common goods, but there are more bidders than goods.

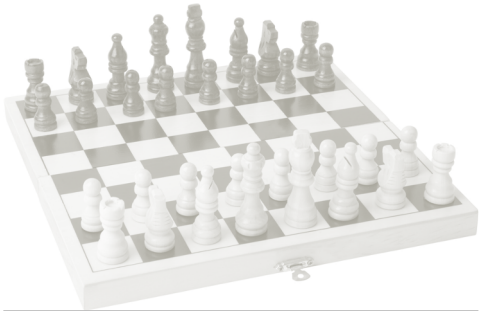
Vickrey-Clarks-Groves mechanism

1. the center asks **each agent to report its value** for receiving an item – b_i
2. the **center allocates the goods** to a subset A of the bidders. Let $b_i(A) = b_i$, if $i \in A$, otherwise 0. The center chooses A to maximize total reported utility $B = \Sigma_i b_i(A)$
3. each **agent pays a tax** equal to $W_{-i} - B_{-i}$, where
   $B_{-i} = \Sigma_{j \neq i} b_j(A)$
   W_{-i} = total global utility if i were not in the game
   each winner would pay a tax equal to the highest reported value among the losers (losers pay nothing)

The **dominant strategy** is to bid v_i.

Chees

- 1997 **Deep Blue** won over Kasparov 3.5 – 2.5
- 2006 „regular“ PC (DEEP FRITZ) beats Kramnik 4 – 2



Checkers

- 1994 **Chinook** became the official world champion
- 29. 4. 2007 solved – optimal policy leads to draw

Go

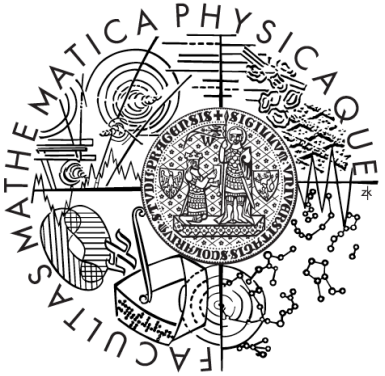
- branching factor 250 makes it challenging
- **AlphaGo** won over human champions (Lee Sedol, 2016), **AlphaGo Zero** won over AlphaGo (2017)
- using Monte Carlo methods for search and deep learning for action selection)

Poker

- **Deep Stack** and **Libratus** won over best humans (2017)

Soccer

- **RoboCup**: „By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, complying with the official rule of the FIFA, against the winner of the most recent World Cup.“



Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz