Machine Intelligence

12. Multi-Agent Systems: Adversarial Search and Game Theory

Because we are not alone. . .

Álvaro Torralba

AALBORG UNIVERSITET

Fall 2022

Thanks to Thomas D. Nielsen and Jörg Hoffmann for slide sources

Agenda

From Single to Multi Agent Systems

So far ...

We have modeled an agent that decides/plans in a world with/without uncertainty.

New Dimension: Other Agents

Agent acts in an environment containing other agents. Other agents might have competing/conflicting objectives.

To solve Multi-Agent systems we need to reason about:

- Competing objectives of other agents
- What other agents will do to achieve their objectives
- Possibility to collaborate to achieve common objectives

Game theory: Study of mathematical models of strategic interactions among rational agents
$\rightarrow$Has its origin in **Economics** and goes much further than playing board games

Why AI Game Playing?

Many good reasons:

- Playing a game well clearly requires a form of "intelligence".
- Games capture a pure form of competition between opponents.
- Games are abstract and precisely defined, thus very easy to formalize.

→ Game playing is one of the oldest sub-areas of AI (ca. 1950).

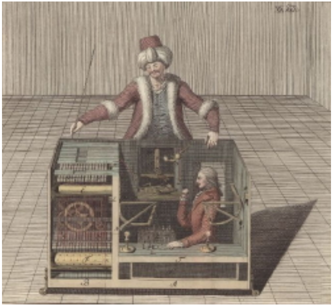
Why AI Game Playing?

Many good reasons:

- Playing a game well clearly requires a form of "intelligence".
- Games capture a pure form of competition between opponents.
- Games are abstract and precisely defined, thus very easy to formalize.

→ Game playing is one of the oldest sub-areas of AI (ca. 1950).

→ The dream of a machine that plays Chess is, indeed, *much* older than AI! (von Kempelen's "Schachtürke" (1769), Torres y Quevedo's "El Ajedrecista" (1912))



"Schachtürke" (1769)

"El Ajedrecista" (1912)

Our Agenda for This Chapter

- **Games:** What are we dealing with in this chapter?
  $\rightarrow$ **Basic definitions and game properties.**

Our Agenda for This Chapter

- **Games:** What are we dealing with in this chapter?

  $\rightarrow$ **Basic definitions and game properties.**

- **Solving Games with Minimax Search:** How to solve a game?

  $\rightarrow$ **Minimax is the canonical (and easiest to understand) algorithm for solving games, i.e., computing an optimal policy.**

Our Agenda for This Chapter

- **Games:** What are we dealing with in this chapter?

  → **Basic definitions and game properties.**

- **Solving Games with Minimax Search:** How to solve a game?

  → **Minimax is the canonical (and easiest to understand) algorithm for solving games, i.e., computing an optimal policy.**

- **Evaluation Functions:** How to evaluate a game position?

  → **Heuristic functions for games, and how to obtain them.**

Our Agenda for This Chapter

- **Games:** What are we dealing with in this chapter?

  → **Basic definitions and game properties.**

- **Solving Games with Minimax Search:** How to solve a game?

  → **Minimax is the canonical (and easiest to understand) algorithm for solving games, i.e., computing an optimal policy.**

- **Evaluation Functions:** How to evaluate a game position?

  → **Heuristic functions for games, and how to obtain them.**

- **Simultaneous and Non-Zero-Sum Games**: How to go beyond

  → **Some basic notions of Game Theory to understand how we deal with other complex games.**

Our Agenda for This Chapter

- **Games:** What are we dealing with in this chapter?

  → **Basic definitions and game properties.**

- **Solving Games with Minimax Search:** How to solve a game?

  → **Minimax is the canonical (and easiest to understand) algorithm for solving games, i.e., computing an optimal policy.**

- **Evaluation Functions:** How to evaluate a game position?

  → **Heuristic functions for games, and how to obtain them.**

- **Simultaneous and Non-Zero-Sum Games**: How to go beyond

  → **Some basic notions of Game Theory to understand how we deal with other complex games.**

Agenda

Definition of a Game

Definition (Game). *A* **game state space** *consists of:*

- *Set of states S divided in* **terminal states** S^T *and* **intermediate states** S^I
- *Set of players/agents P, each $p \in P$ with its own set of actions A^p and its* **utility function** u^p*.*
- *A utility function assigns a utility (score) to each terminal state*
- *On intermediate states S^I one or more players can play an action, resulting in a new state*
- *On terminal states $s \in S^T$, the game ends, and each player receives a score equal to the utility function of $u^p(s)$*

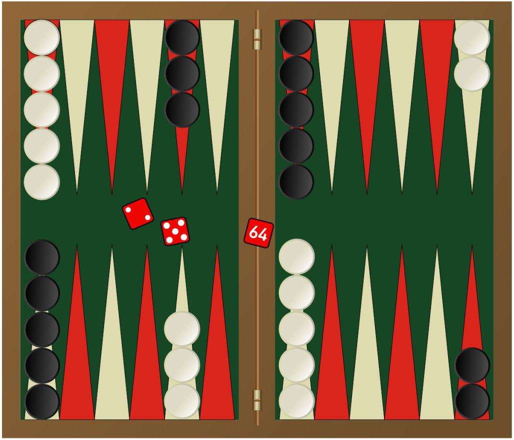
$\rightarrow$ Each agent will pick actions to **maximize their own utility**

This is a very general view of games, so we also consider some specific sub-classes that can be solved with adversarial search (see next slides)
At the end of the lecture, we will consider how to deal with more complex games.

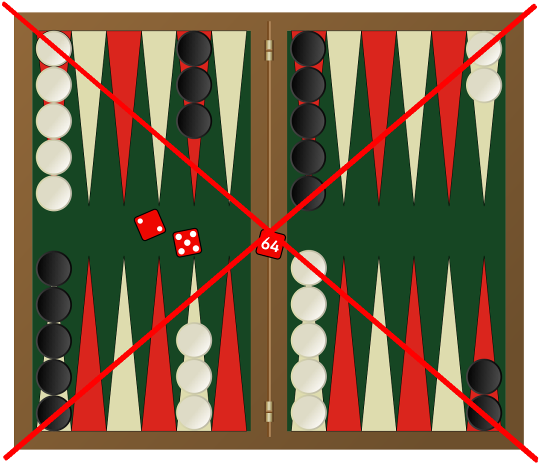
The Problem

→ "Adversarial search" = Game playing against an opponent.

Which Games?

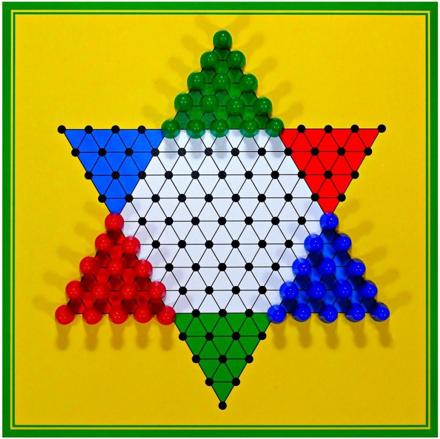


Which Games?

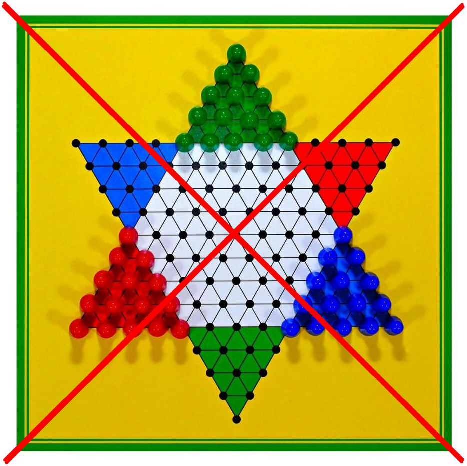


→ No chance element.

Which Games?



Which Games?



→ **Exactly two players.**

Which Games?

Which Games?

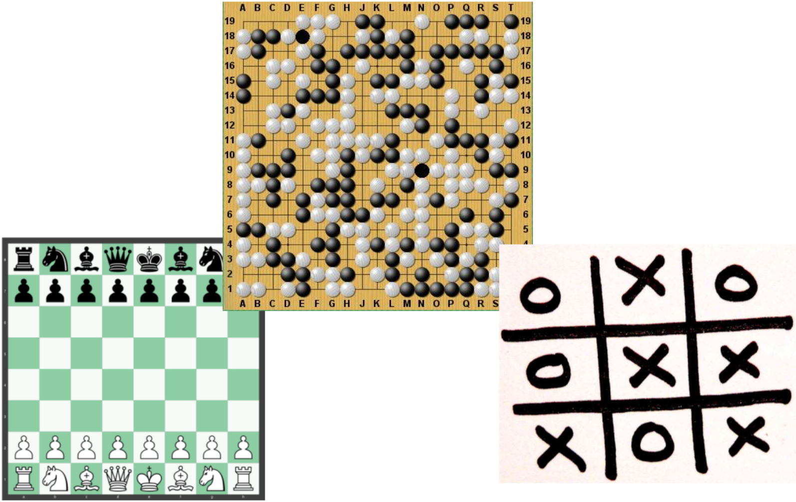
→ **Game state fully observable.**

Which Games?

Which Games?

→ Player utilities are diametrically opposed.
(Else: game theory, equilibria, auctions, . . .)

These Games!



These Games!

Restrictions:

- The game state is **fully observable**.
- The outcome of each move is **deterministic**.

These Games!

Restrictions:

- The game state is **fully observable**.
- The outcome of each move is **deterministic**.
- Game states **discrete, finite** number of possible moves and game states.
- There are **no infinite runs** of the game: a **terminal state** is always reached after a finite number of steps.

These Games!

Restrictions:

- The game state is **fully observable**.
- The outcome of each move is **deterministic**.
- Game states **discrete, finite** number of possible moves and game states.
- There are **no infinite runs** of the game: a **terminal state** is always reached after a finite number of steps.
- **Two-player zero-sum** game: two players, terminal states have utility with $\text{utility}(\text{player1}) = -\text{utility}(\text{player2})$.

These Games!

Restrictions:

- The game state is **fully observable**.
- The outcome of each move is **deterministic**.
- Game states **discrete, finite** number of possible moves and game states.
- There are **no infinite runs** of the game: a **terminal state** is always reached after a finite number of steps.
- **Two-player zero-sum** game: two players, terminal states have utility with utility(player1) $=$ $-$utility(player2).
- Our formulation (equivalent): single **utility function** u, players Max vs. Min trying to **maximize vs. minimize** u.

These Games!

Restrictions:

- The game state is **fully observable**.
- The outcome of each move is **deterministic**.
- Game states **discrete, finite** number of possible moves and game states.
- There are **no infinite runs** of the game: a **terminal state** is always reached after a finite number of steps.
- **Two-player zero-sum** game: two players, terminal states have utility with utility(player1) $=$ $-$utility(player2).
- Our formulation (equivalent): single **utility function** u, players Max vs. Min trying to **maximize vs. minimize** u.
- **Turn-taking**: Players move alternatingly. Max begins.

An Example Game

- Game states: Positions of figures.
- Moves: Given by rules.
- Players: White (Max), Black (Min).
- Terminal states: Checkmate.
- Utility function: $+100$ if Black is checkmated, 0 if stalemate, -100 if White is checkmated.

(A Brief Note On) Formalization

Definition (Game State Space). *A* **game state space** *is a 6-tuple* $\Theta = (S, A, T, I, S^T, u)$ *where:*

- S, A, T, I: *States, actions, deterministic transition relation, initial state. As in classical search problems, except:*
  - S *is the disjoint union of* S^{Max}, S^{Min}, **and** S^T.
  - A *is the disjoint union of* A^{Max} **and** A^{Min}.
  - *For* $a \in A^{Max}$, **if** $s \xrightarrow{a} s'$ **then** $s \in S^{Max}$ **and** $s' \in S^{Min} \cup S^T$.
  - *For* $a \in A^{Min}$, **if** $s \xrightarrow{a} s'$ **then** $s \in S^{Min}$ **and** $s' \in S^{Max} \cup S^T$.
- S^T *is the set of* **terminal states**.
- $u : S^T \mapsto \mathbb{R}$ *is the* **utility function**.

(A Brief Note On) Formalization

Definition (Game State Space). *A* **game state space** *is a 6-tuple* $\Theta = (S, A, T, I, S^T, u)$ *where:*

- S, A, T, I*: States, actions, deterministic transition relation, initial state. As in classical search problems, except:*
  - S *is the disjoint union of* S^{Max}, S^{Min}, **and** S^T.
  - A *is the disjoint union of* A^{Max} **and** A^{Min}.
  - *For* $a \in A^{Max}$, **if** $s \xrightarrow{a} s'$ **then** $s \in S^{Max}$ **and** $s' \in S^{Min} \cup S^T$.
  - *For* $a \in A^{Min}$, **if** $s \xrightarrow{a} s'$ **then** $s \in S^{Min}$ **and** $s' \in S^{Max} \cup S^T$.
- S^T *is the set of* **terminal states***.*
- $u : S^T \mapsto \mathbb{R}$ *is the* **utility function***.*

Commonly used terminology: state=**position**, terminal state=**end state**, action=**move**.

(A round of the game – one move Max, one move Min – is often referred to as a "move", and individual actions as "half-moves". We do *NOT* do that here.)

Why Games are Hard to Solve

Why Games are hard to solve, part 1: $\rightarrow$ **What is a "solution" here?**

Why Games are Hard to Solve

Why Games are hard to solve, part 1: $\rightarrow$ **What is a "solution" here?**

Definition (Strategy/Policy). *Let Θ be a game state space, and let $X \in \{Max, Min\}$. A* **policy** *for X is a function $p^X : S^X \mapsto A^X$ so that a is applicable to s whenever $p^X(s) = a$.*

- We don't know how the opponent will react, and need to **prepare for all possibilities**.
- A policy is **optimal** if it yields the best possible utility for X assuming perfect opponent play (not formalized here).

Why Games are Hard to Solve

Why Games are hard to solve, part 1: $\rightarrow$ **What is a "solution" here?**

Definition (Strategy/Policy). *Let Θ be a game state space, and let $X \in \{Max, Min\}$. A* **policy** *for X is a function $p^X : S^X \mapsto A^X$ so that a is applicable to s whenever $p^X(s) = a$.*

- We don't know how the opponent will react, and need to **prepare for all possibilities**.
- A policy is **optimal** if it yields the best possible utility for X assuming perfect opponent play (not formalized here).
- In (almost) all games, computing a policy is infeasible. Instead, compute the next move "on demand", given the current game state.

Why Games are Hard to Solve

Why Games are hard to solve, part 1: $\rightarrow$ **What is a "solution" here?**

Definition (Strategy/Policy). *Let Θ be a game state space, and let $X \in \{Max, Min\}$. A* **policy** *for X is a function $p^X : S^X \mapsto A^X$ so that a is applicable to s whenever $p^X(s) = a$.*

- We don't know how the opponent will react, and need to **prepare for all possibilities**.
- A policy is **optimal** if it yields the best possible utility for X assuming perfect opponent play (not formalized here).
- In (almost) all games, computing a policy is infeasible. Instead, compute the next move "on demand", given the current game state.

Why Games are hard to solve, part 2:

- **Number of reachable states:** in Chess 10^{40}; in Go 10^{100}.
- Chess: branching factor ca. 35, hence 1000 per move/counter-move lookahead. Go: 200, hence 40000.

Agenda

"Minimax"?

→ We want to compute an optimal move for player "Max". In other words: **"We are Max, and our opponent is Min."**

Remember:

- Max attempts to *maximize* the utility $u(s)$ of the terminal state that will be reached during play.
- Min attempts to *minimize* $u(s)$.

“Minimax”?

$\rightarrow$ We want to compute an optimal move for player “Max”. In other words: **“We are Max, and our opponent is Min.”**

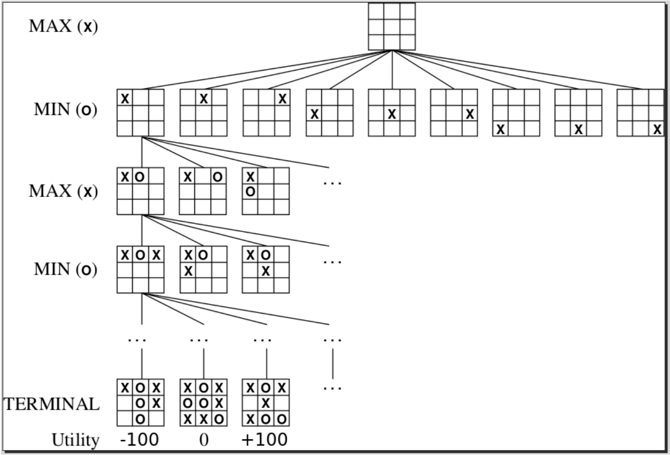
Remember:

- Max attempts to *maximize* the utility $u(s)$ of the terminal state that will be reached during play.
- Min attempts to *minimize* $u(s)$.

So what?

- The computation alternates between minimization and maximization $\Longrightarrow$ hence “Minimax”.

Example Tic-Tac-Toe



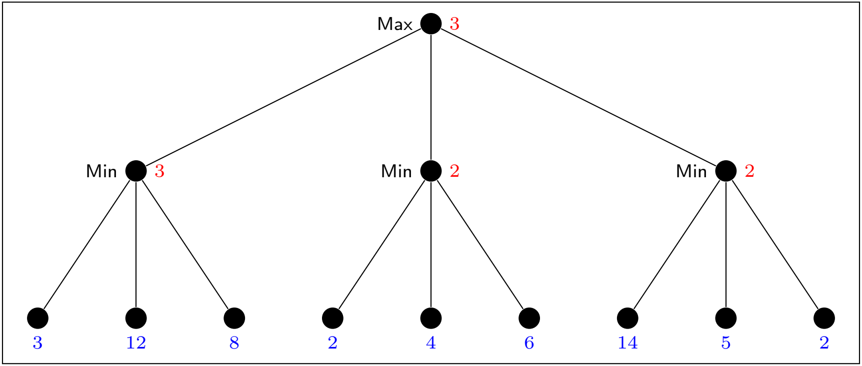
- Game tree, current player marked on the left.
- Last row: terminal positions with their utility.

Minimax: Outline

We max, we min, we max, we min . . .

1. Depth-first search in game tree, with Max in the root.
2. Apply utility function to terminal positions.
3. Bottom-up for each inner node n in the tree, compute the utility $u(n)$ of n as follows:
   - If it's Max's turn: Set $u(n)$ to the maximum of the utilities of n's successor nodes.
   - If it's Min's turn: Set $u(n)$ to the minimum of the utilities of n's successor nodes.
4. Selecting a move for Max at the root: Choose one move that leads to a successor node with maximal utility.

Minimax: Example



- **Blue numbers:** Utility function u applied to terminal positions.
- **Red numbers:** Utilities of inner nodes, as computed by Minimax.

Minimax: Pseudo-Code

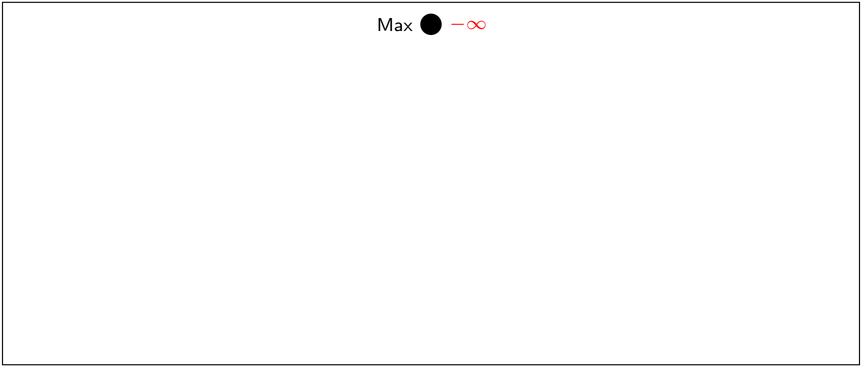
Input: State $s \in S^{Max}$, in which Max is to move.

function Minimax-Decision(s) **returns** an action
  $v \leftarrow$ *Max-Value*(s)
  return an *action* $a \in$ *Actions(s)* yielding value v

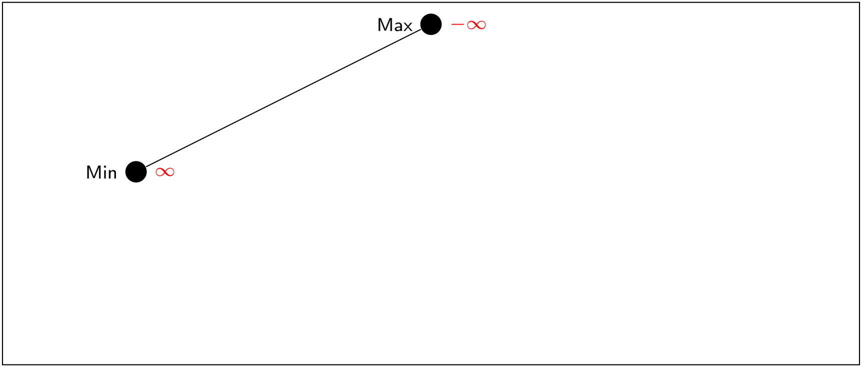
function Max-Value(s) **returns** a utility value
  if *Terminal-Test*(s) **then return** $u(s)$
  $v \leftarrow -\infty$
  for each $a \in$ *Actions(s)* **do**
    $v \leftarrow \max(v,$*Min-Value*(*ChildState*$(s, a)))$
  return v

function Min-Value(s) **returns** a utility value
  if *Terminal-Test*(s) **then return** $u(s)$
  $v \leftarrow +\infty$
  for each $a \in$ *Actions(s)* **do**
    $v \leftarrow \min(v,$*Max-Value*(*ChildState*$(s, a)))$
  return v

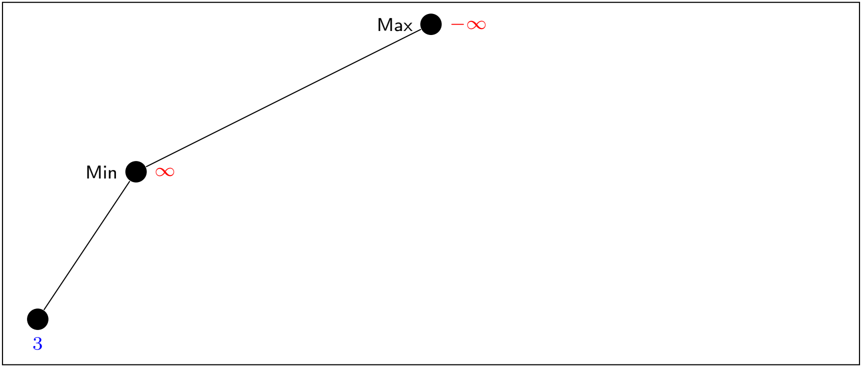
Minimax: Example, Now in Detail



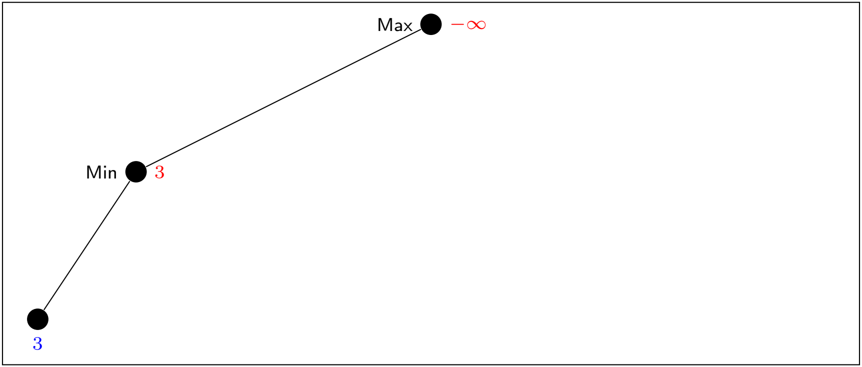
Minimax: Example, Now in Detail



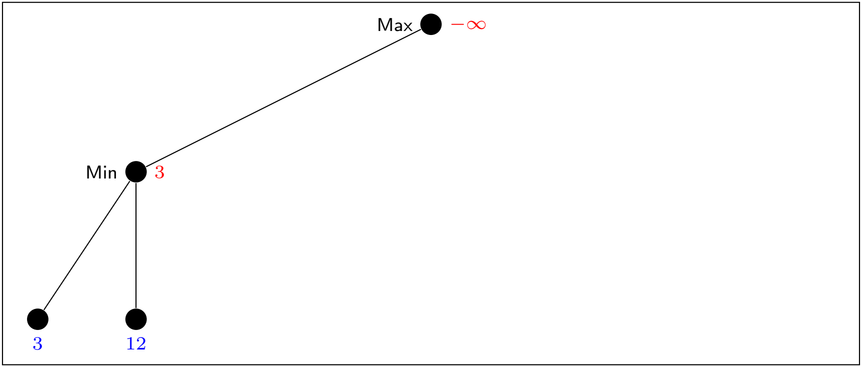
Minimax: Example, Now in Detail



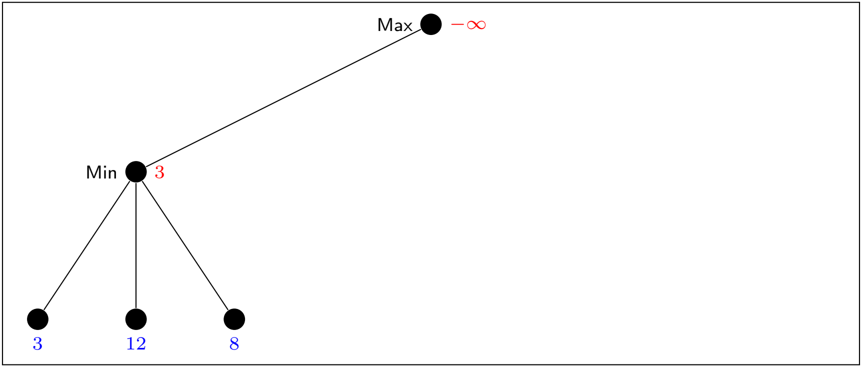
Minimax: Example, Now in Detail



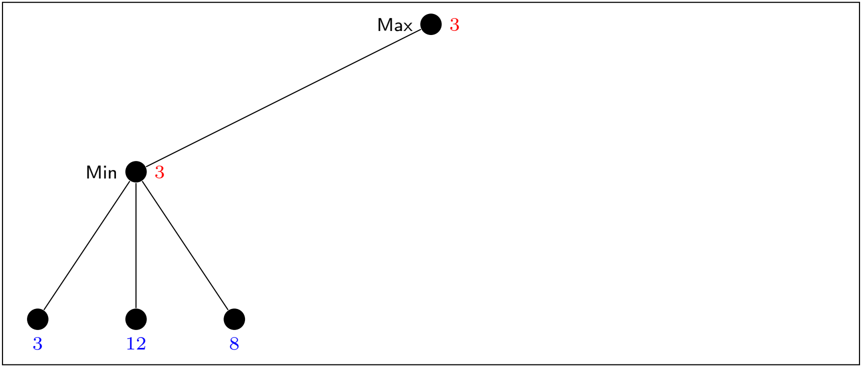
Minimax: Example, Now in Detail



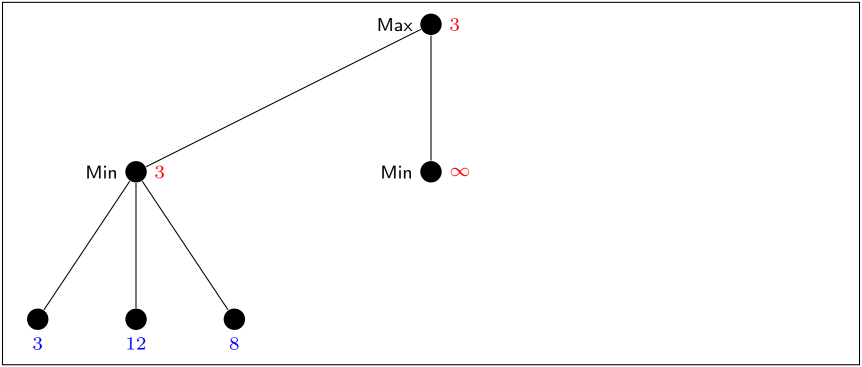
Minimax: Example, Now in Detail



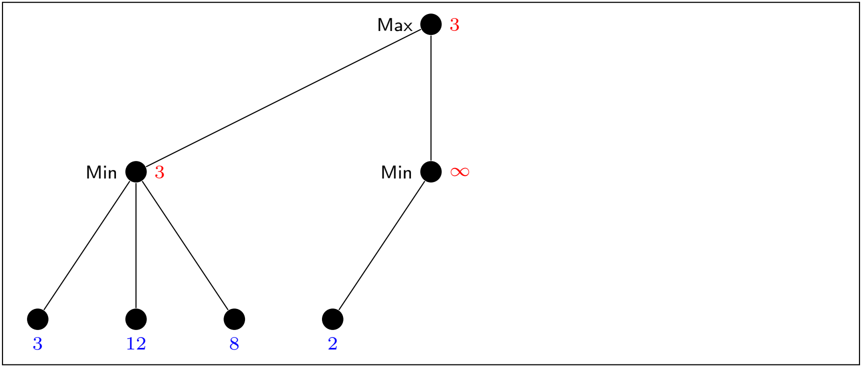
Minimax: Example, Now in Detail



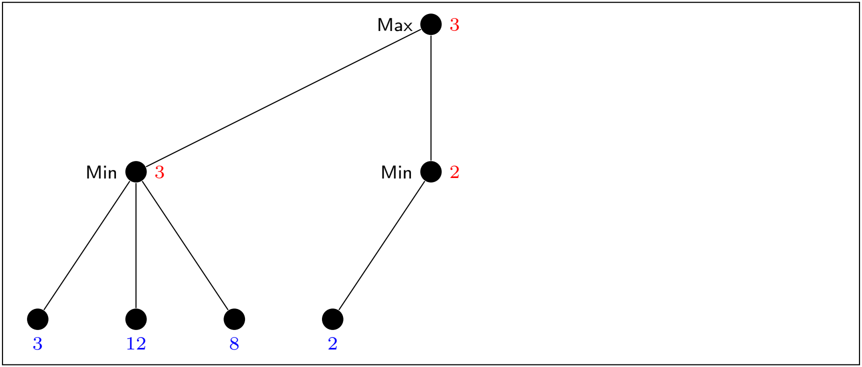
Minimax: Example, Now in Detail



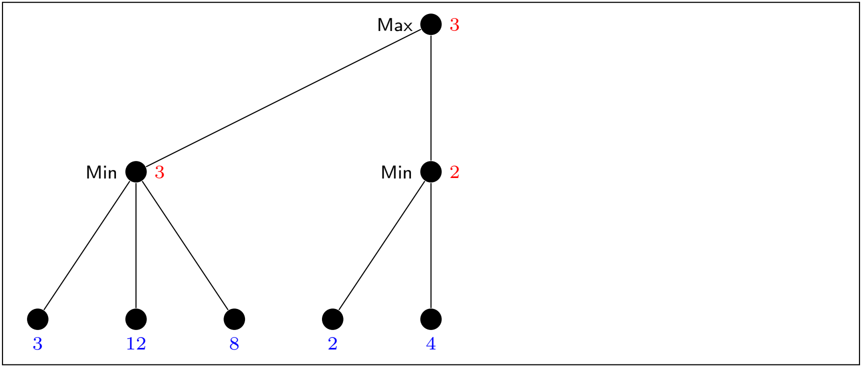
Minimax: Example, Now in Detail



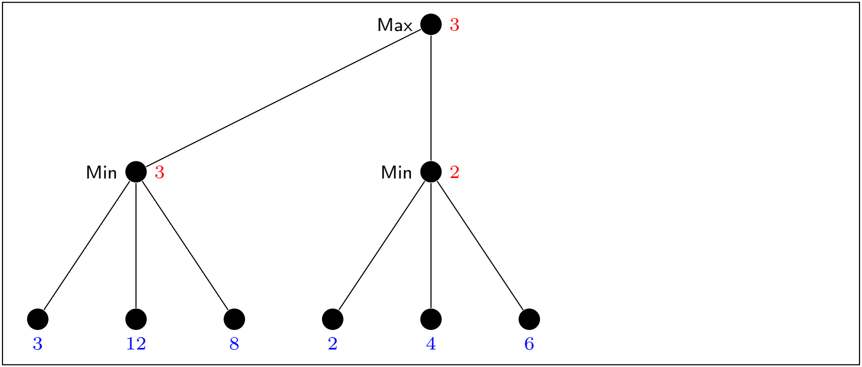
Minimax: Example, Now in Detail



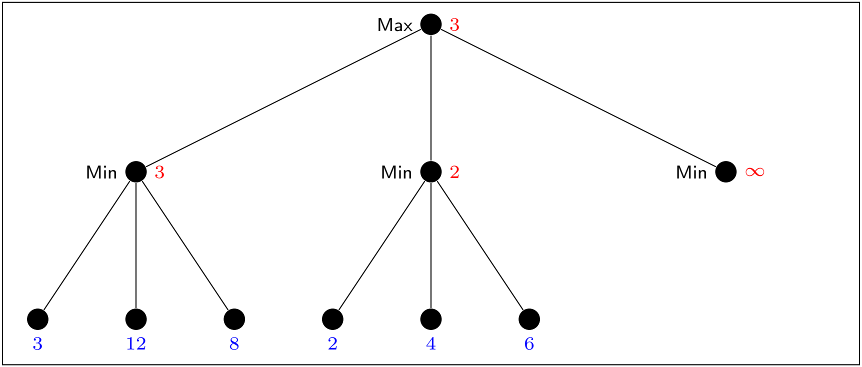
Minimax: Example, Now in Detail



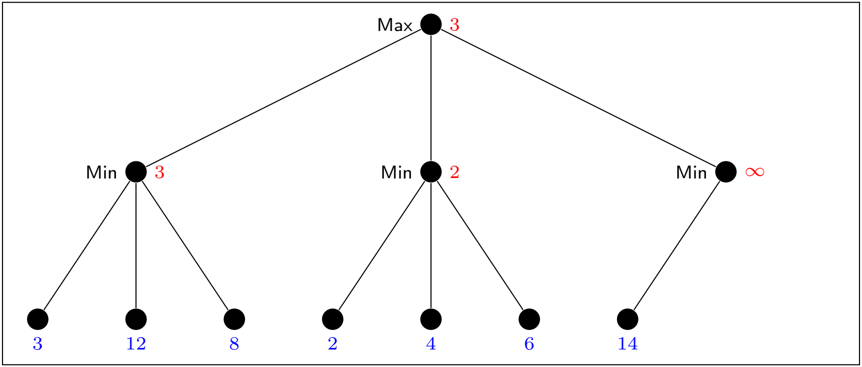
Minimax: Example, Now in Detail



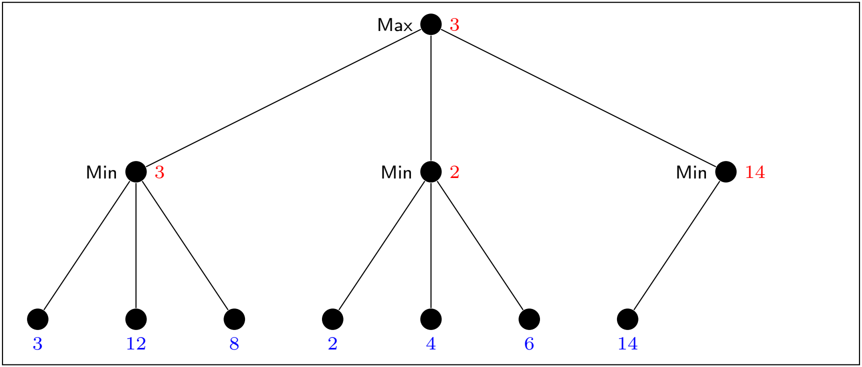
Minimax: Example, Now in Detail



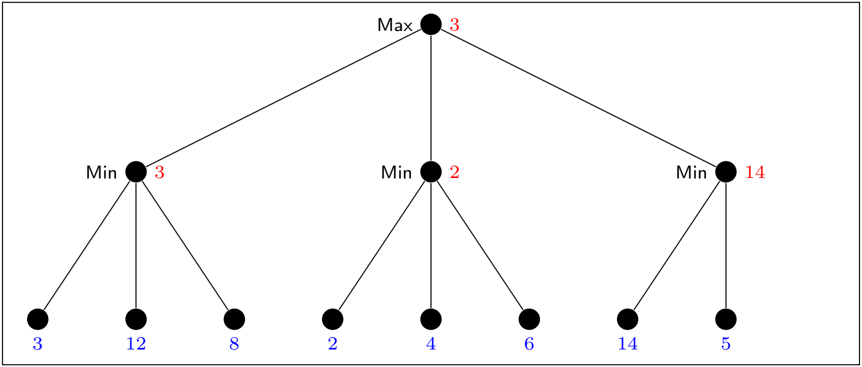
Minimax: Example, Now in Detail



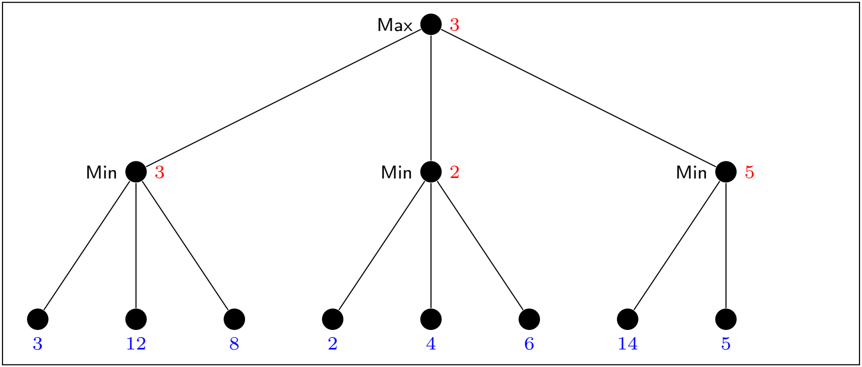
Minimax: Example, Now in Detail



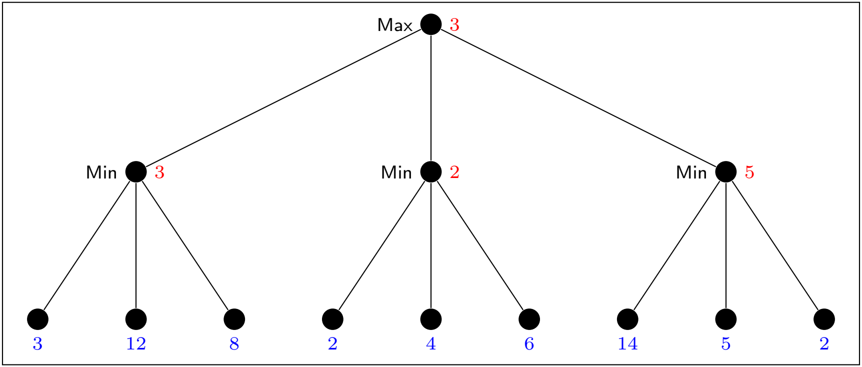
Minimax: Example, Now in Detail



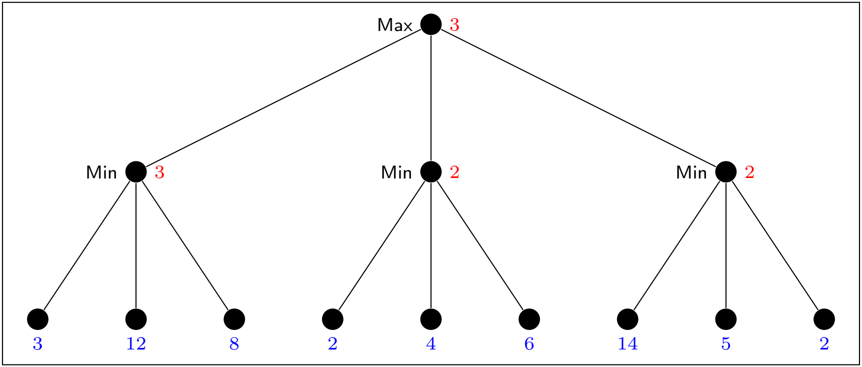
Minimax: Example, Now in Detail



Minimax: Example, Now in Detail

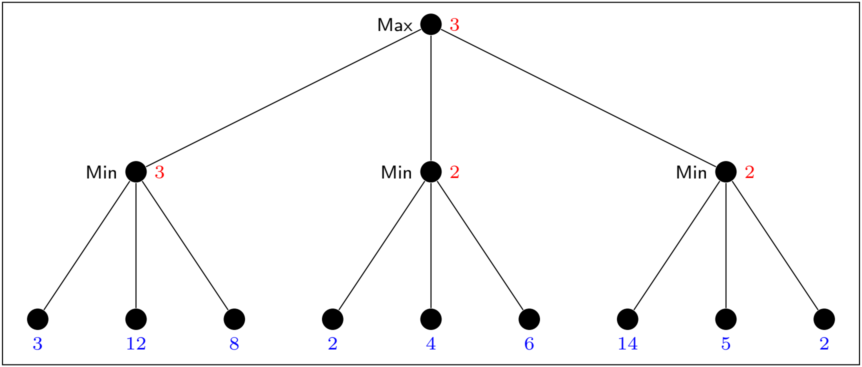


Minimax: Example, Now in Detail



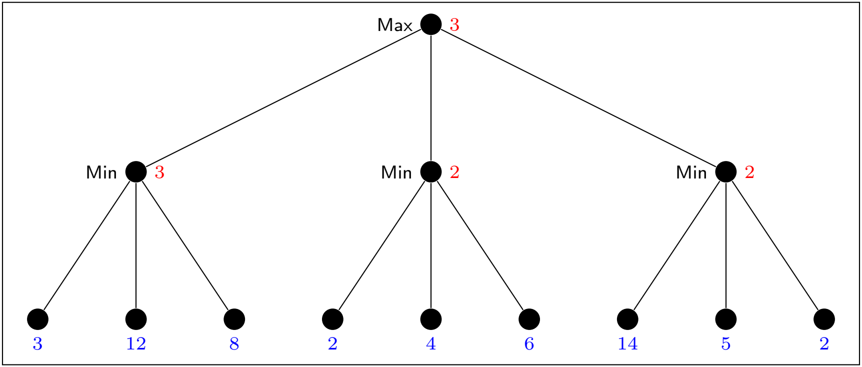
→ **So which action for Max is returned?**

Minimax: Example, Now in Detail



→ **So which action for Max is returned?** Leftmost branch.

Minimax: Example, Now in Detail



→ **So which action for Max is returned?** Leftmost branch. Note: The maximal possible pay-off is higher for the rightmost branch, but assuming perfect play of Min, it's better to go left. (Going right would be "relying on your opponent to do something stupid".)

Minimax, Pro and Contra

Pro:

- Returns an optimal action, assuming perfect opponent play.
- Extremely simple.

Minimax, Pro and Contra

Pro:

- Returns an optimal action, assuming perfect opponent play.
- Extremely simple.

Contra:

Minimax, Pro and Contra

Pro:

- Returns an optimal action, assuming perfect opponent play.
- Extremely simple.

Contra:

- **Completely infeasible (search tree way too large).**

Minimax, Pro and Contra

Pro:

- Returns an optimal action, assuming perfect opponent play.
- Extremely simple.

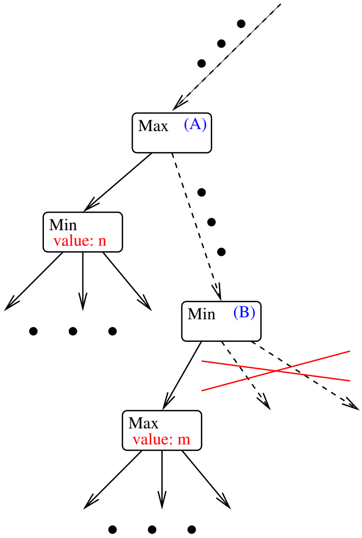
Contra:

- **Completely infeasible (search tree way too large).**

Remedies:

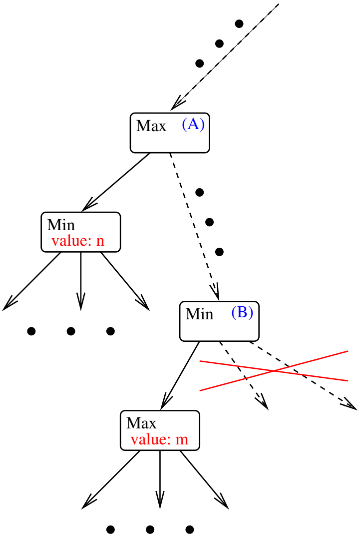
- Limit search depth, apply **evaluation function** at cut-off states.
- Sparse search (MCTS) instead of exhaustive search.
- **Alpha-beta** pruning reduces search yet preserves optimality.

Alpha-Beta Pruning: Idea



Say $n > m$**.**

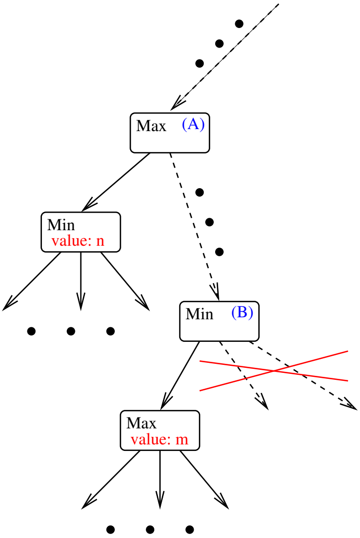
Alpha-Beta Pruning: Idea



Say $n > m$.

→ By choosing to move left in Max node (A), Max already can get utility at least n in this part of the game.

Alpha-Beta Pruning: Idea

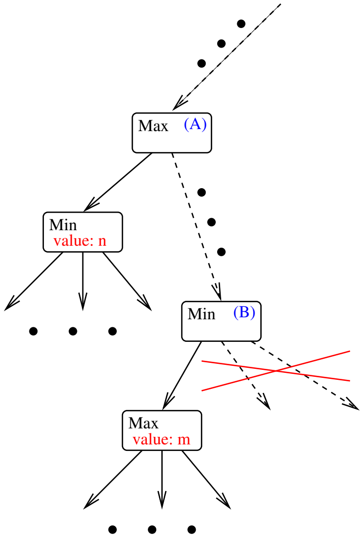


Say $n > m$.

$\rightarrow$ By choosing to move left in Max node (A), Max already can get utility at least n in this part of the game.

Say that below a different move at (A), in Min node (B) Min can force Max to get value $m < n$.

Alpha-Beta Pruning: Idea



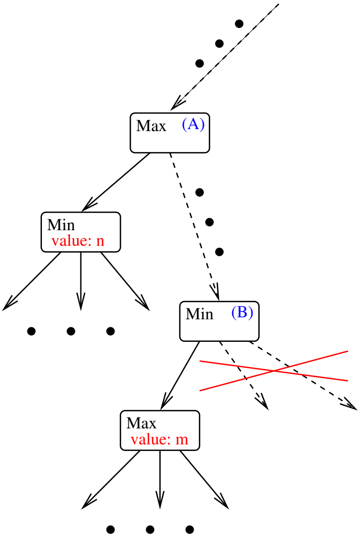
Say $n > m$.

$\rightarrow$ By choosing to move left in Max node (A), Max already can get utility at least n in this part of the game.

Say that below a different move at (A), in Min node (B) Min can force Max to get value $m < n$.

Then we already know that (B) will not be reached during the game, given the policy we currently compute for Max (Max can prevent the game from reaching (B), e.g. by moving left at (A)).

Alpha-Beta Pruning: Idea



Say $n > m$.

$\rightarrow$ By choosing to move left in Max node (A), Max already can get utility at least n in this part of the game.

Say that below a different move at (A), in Min node (B) Min can force Max to get value $m < n$.

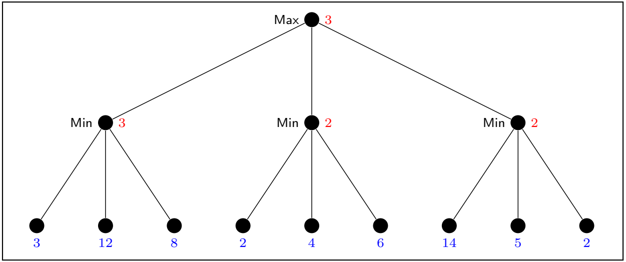
Then we already know that (B) will not be reached during the game, given the policy we currently compute for Max (Max can prevent the game from reaching (B), e.g. by moving left at (A)).

Hence we can spare ourselves the effort of searching the other children of (B).

Alpha Pruning: The Idea in Our Example

Question:

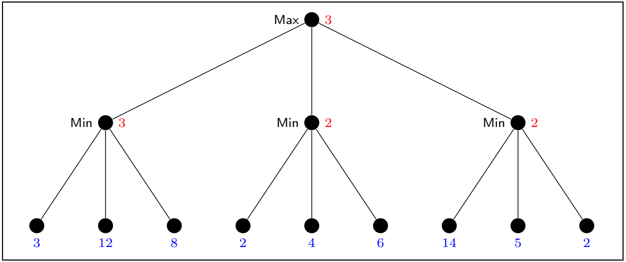
Can we save some work here?

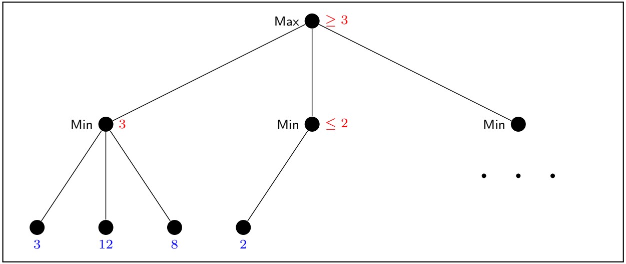


Alpha Pruning: The Idea in Our Example

Question:

Can we save some work here?





f **Answer**: Yes!

→ We already know at this point that the middle action won't be taken by Max.

Solving Checkers

J.Schaeffer et al.: *Checkers Is Solved*. Science, July 2007

- Schaeffer et al. proved: there is no winning strategy for either player: perfect play by both players will always result in a draw
- checkers has approximately $5 \cdot 10^{20}$ different positions
- in the proof only about 10^{14} positions were explored
- reduction by several techniques, including α-β-pruning.

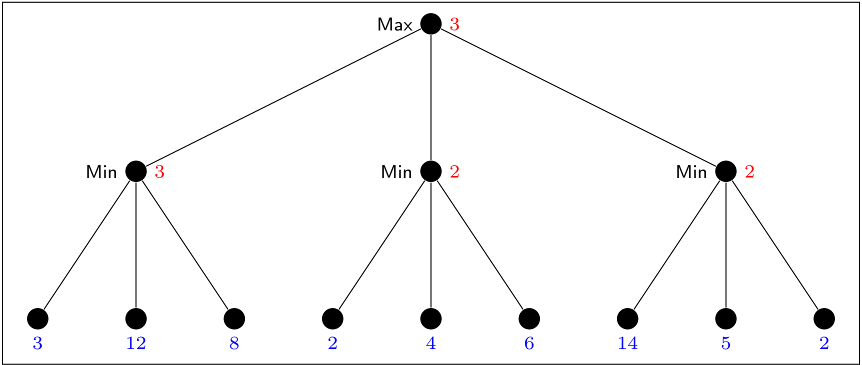
Agenda

1. Introduction
2. Games
3. Solving Games with Minimax Search
4. Evaluation Functions
5. Simultaneous and Non-Zero Sum Games
6. Conclusion

Minimax With Depth Limit

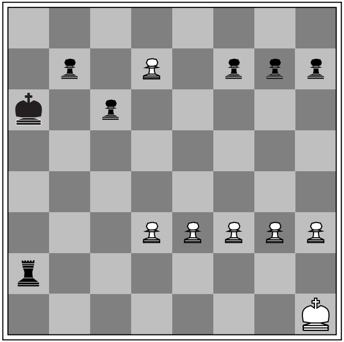
What if we cannot search until the end of the game? Search until a limited depth and use an evaluation function!

Notation: blue: evaluation function value on cut-off states; red: non-cut-off state value as computed by Minimax with depth limit 2.



→ Search pretends that states at depth limit d (number of actions i.e. half-moves) are terminal; requires evaluation function to estimate their values (see later).

Questionnaire



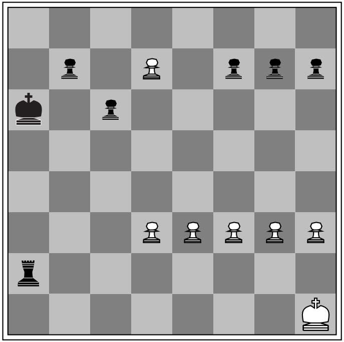
Black to move

Question!

Who's gonna win here?

(A): White **(B):** Black

Questionnaire



Black to move

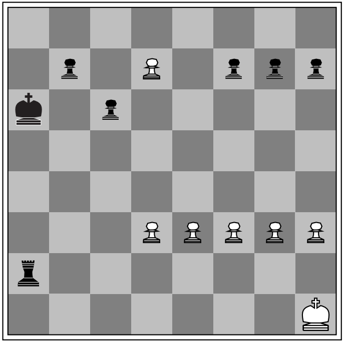
Question!

Who's gonna win here?

(A): White **(B):** Black

- White wins (Pawn cannot be prevented from becoming a queen.)

Questionnaire



Black to move

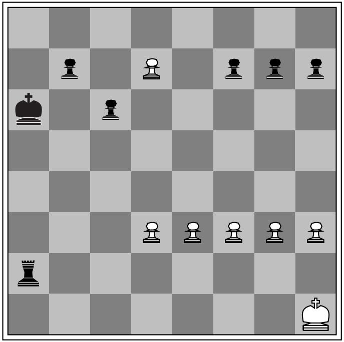
Question!

Who's gonna win here?

(A): White **(B):** Black

- White wins (Pawn cannot be prevented from becoming a queen.)
- Black has a large advantage in material. If cut-off is here, then the evaluation function will say "-100, black wins".

Questionnaire



Black to move

Question!

Who's gonna win here?

(A): White **(B):** Black

- White wins (Pawn cannot be prevented from becoming a queen.)
- Black has a large advantage in material. If cut-off is here, then the evaluation function will say "-100, black wins".
- The loss for black is **beyond our horizon** unless we search extremely deeply: Black can hold off the end by repeatedly giving check to White's king.

→ In other words: Minimax is not robust to inaccurate cut-off evaluations.

Evaluation Functions

Definition: *Given a game with states* S*, a* **(heuristic) evaluation function** *is a function* $h : S \mapsto \mathbb{R}$*.*

- h **estimates the expected utility of** s. (In particular, we can use $h := u$ on terminal states)
- In Minimax: Impose depth limit, use h at (non-terminal) cut-off states.

Evaluation Functions

Definition: *Given a game with states* S, *a* **(heuristic) evaluation function** *is a function* $h : S \mapsto \mathbb{R}$.

- h **estimates the expected utility of** s. (In particular, we can use $h := u$ on terminal states)
- In Minimax: Impose depth limit, use h at (non-terminal) cut-off states.

How to obtain h**?**

Evaluation Functions

Definition: *Given a game with states S, a* **(heuristic) evaluation function** *is a function $h : S \mapsto \mathbb{R}$.*

- **h estimates the expected utility of s**. (In particular, we can use $h := u$ on terminal states)
- In Minimax: Impose depth limit, use h at (non-terminal) cut-off states.

How to obtain h?

- Relaxed game: Possible in principle, too costly in practice.

Evaluation Functions

Definition: *Given a game with states S, a* **(heuristic) evaluation function** *is a function $h : S \mapsto \mathbb{R}$.*

- **h estimates the expected utility of s.** (In particular, we can use $h := u$ on terminal states)
- In Minimax: Impose depth limit, use h at (non-terminal) cut-off states.

How to obtain h?

- Relaxed game: Possible in principle, too costly in practice.
- **Encode human expert knowledge.**
- **Learn from data.**

Position Evaluation in Chess

Position Evaluation in Chess

- **Material**: Pawn 1, Knight 3, Bishop 3, Rook 5, Queen 9.
  $\rightarrow$ Rule of thumb:
  3 points advantage $\implies$ safe win.

Position Evaluation in Chess

- **Material**: Pawn 1, Knight 3, Bishop 3, Rook 5, Queen 9.
  $\rightarrow$ Rule of thumb:
  3 points advantage $\implies$ safe win.
- **Mobility**: How many fields do you control?
- **King safety**, **Pawn structure**, ...

Linear Feature-Based Evaluation Functions

Functions taking the form:

$$\mathbf{h(s) := w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)}$$

f_i are **features**, w_i are **weights**.

Linear Feature-Based Evaluation Functions

Functions taking the form:

$$\mathbf{h(s) := w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)}$$

f_i are **features**, w_i are **weights**.

How to obtain such functions?

- Features f_i designed by human experts.
- Weights w_i set by experts, or learned automatically (see later).

Linear Feature-Based Evaluation Functions

Functions taking the form:

$$\mathbf{h(s) := w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)}$$

f_i are **features**, w_i are **weights**.

How to obtain such functions?

- Features f_i designed by human experts.
- Weights w_i set by experts, or learned automatically (see later).

Discussion: Pro/Con

- **Very fast.** (Unless there are many features or computing their value is very expensive; incremental computation helps)

Linear Feature-Based Evaluation Functions

Functions taking the form:

$$\mathbf{h(s) := w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)}$$

f_i are **features**, w_i are **weights**.

How to obtain such functions?

- Features f_i designed by human experts.
- Weights w_i set by experts, or learned automatically (see later).

Discussion: Pro/Con

- **Very fast.** (Unless there are many features or computing their value is very expensive; incremental computation helps)
- **Very simplistic.** For example, assumes that features are independent. (But, e.g., value of Rook depends on Pawn structure)
- **Human knowledge crucial in design of features.**

Feature-Based Evaluation in Chess

- **Material**: Pawn (Bauer) 1, Knight (Springer) 3, Bishop (Läufer) 3, Rook (Turm) 5, Queen (Dame) 9.
  $\rightarrow$ Rule of thumb:
  3 points advantage $\implies$ safe win.
- **Mobility**: How many fields do you control?
- **King safety**, **Pawn structure**, ...

$$\rightarrow h(s) = \Delta pawn(s) + 3 * \Delta knight(s) + 3 * \Delta bishop(s) + 5 * \Delta rook(s) + 9 * \Delta queen(s) \quad (\Delta\text{: \#White}-\text{\#Black})$$

Feature-Based Evaluation in Chess

- **Material**: Pawn (Bauer) 1, Knight (Springer) 3, Bishop (Läufer) 3, Rook (Turm) 5, Queen (Dame) 9.
  $\rightarrow$ Rule of thumb:
  3 points advantage $\implies$ safe win.
- **Mobility**: How many fields do you control?
- **King safety**, **Pawn structure**, ...

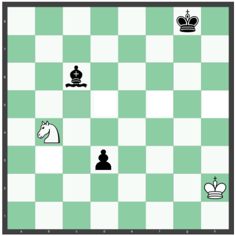
$\rightarrow h(s) = \Delta pawn(s) + 3 * \Delta knight(s) + 3 * \Delta bishop(s) +$
$5 * \Delta rook(s) + 9 * \Delta queen(s)$ (Δ: #White−#Black)

$+w_k kingsafety(s) + w_p pawnstructure(s)$**?**

Feature-Based Evaluation in Chess???

- **Material**: Pawn (Bauer) 1, Knight (Springer) 3, Bishop (Läufer) 3, Rook (Turm) 5, Queen (Dame) 9.
  $\rightarrow$ Rule of thumb:
  3 points advantage $\Longrightarrow$ safe win.
- **Mobility**: How many fields do you control?
- **King safety**, **Pawn structure**, . . .

Questionnaire

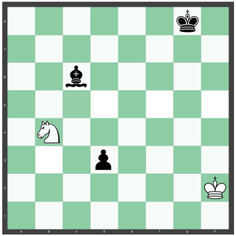


- White to move.
- $h(s) = \Delta pawn(s) + 3 * \Delta knight(s) + 3 * \Delta bishop(s) + 5 * \Delta rook(s) + 9 * \Delta queen(s)$.
  (Δ: #White−#Black)

Question!

Say Minimax with depth limit d uses h at cut-off states. Which move does it choose in this state with a depth limit of $d = 1$ half-moves (i.e. considering only the first action) For which values of d does it choose the best action?

Questionnaire



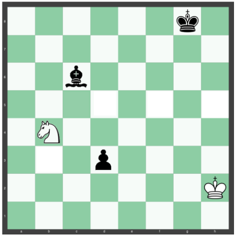
- White to move.
- $h(s) = \Delta pawn(s) + 3 * \Delta knight(s) + 3 * \Delta bishop(s) + 5 * \Delta rook(s) + 9 * \Delta queen(s)$.
  (Δ: #White−#Black)

Question!

Say Minimax with depth limit d uses h at cut-off states. Which move does it choose in this state with a depth limit of $d = 1$ half-moves (i.e. considering only the first action) For which values of d does it choose the best action?

$\rightarrow$ With $d = 1$, Minimax chooses to capture the black bishop due to the superior material advantage.

Questionnaire



- White to move.
- $h(s) = \Delta pawn(s) + 3 * \Delta knight(s) + 3 * \Delta bishop(s) + 5 * \Delta rook(s) + 9 * \Delta queen(s)$.
  (Δ: #White−#Black)

Question!

Say Minimax with depth limit d uses h at cut-off states. Which move does it choose in this state with a depth limit of $d = 1$ half-moves (i.e. considering only the first action) For which values of d does it choose the best action?

→ With $d = 1$, Minimax chooses to capture the black bishop due to the superior material advantage.

→ The best action is to capture the black pawn, as this is the only way to prevent it from turning into a queen. To see this, we need $d \geq 4$.

Supervised Learning of Evaluation Functions (for Reference)

Human expert annotates states with evaluation-function value
$\Rightarrow$ standard supervised learning problem.

- Set of annotated states, i.e., state/value pairs (s, v).
- Learn ML model that predicts output v from input s.

Supervised Learning of Evaluation Functions

(for Reference)

Human expert annotates states with evaluation-function value $\Rightarrow$ standard supervised learning problem.

- Set of annotated states, i.e., state/value pairs (s, v).
- Learn ML model that predicts output v from input s.

Possible ML methods: arbitrary

- Classic approach: learn weights in feature-based evaluation function.
- Recent breakthrough successes: **neural networks**!

Policies & Supervised Learning (for Reference)

Definition: *By p, we denote a (combined)* **policy** *for both players. A* **probabilistic policy** *returns a probability distribution over actions instead.*

- An optimal policy captures perfect play from both players.
- (Probabilistic) policies can be used as **search guidance in MCTS: action selection in tree, action selection in rollouts**.

Policies & Supervised Learning (for Reference)

Definition: *By p, we denote a (combined)* **policy** *for both players. A* **probabilistic policy** *returns a probability distribution over actions instead.*

- An optimal policy captures perfect play from both players.
- (Probabilistic) policies can be used as **search guidance in MCTS: action selection in tree, action selection in rollouts**.

Supervised learning of policies:

Human expert annotates states with preferred moves
$\Rightarrow$ standard supervised classification problem.

- Way more natural for humans; side effect of expert game play.
- e.g. KGS Go Server: 30 million positions with expert moves, used for training in AlphaGo.

Learning from Self-Play (for Reference)

Self-play for reinforcement learning:

Repeat: play a game using the current h and/or p; at each step (s_t, a_t) along the game trace, reinforce the game outcome in h and/or p.

- Evaluation function learning: update weights in h to reduce the error $h(s_t) - \mathit{game\text{-}outcome\text{-}value}$.
- Probabilistic policy learning: update weights in p to increase (game won)/decrease (game lost) the likelihood of choosing a_t in s_t.

Learning from Self-Play (for Reference)

Self-play for reinforcement learning:

Repeat: play a game using the current h and/or p; at each step (s_t, a_t) along the game trace, reinforce the game outcome in h and/or p.

- Evaluation function learning: update weights in h to reduce the error $h(s_t) - \mathit{game\text{-}outcome\text{-}value}$.
- Probabilistic policy learning: update weights in p to increase (game won)/decrease (game lost) the likelihood of choosing a_t in s_t.

Self-play to generate data for supervised learning:

Fix policy p. Repeat: play game using p; annotate the states in each game trace with the game outcome value.

- Use this data for supervised learning of evaluation function.

Learning from Self-Play (for Reference)

Self-play for reinforcement learning:

Repeat: play a game using the current h and/or p; at each step (s_t, a_t) along the game trace, reinforce the game outcome in h and/or p.

- Evaluation function learning: update weights in h to reduce the error $h(s_t) - \mathit{game\text{-}outcome\text{-}value}$.
- Probabilistic policy learning: update weights in p to increase (game won)/decrease (game lost) the likelihood of choosing a_t in s_t.

Self-play to generate data for supervised learning:

Fix policy p. Repeat: play game using p; annotate the states in each game trace with the game outcome value.

- Use this data for supervised learning of evaluation function.
- Might sound strange, but actually successful: used in AlphaGo.

Agenda

Beyond Turn-based Two-Player Zero-sum Games

In this section we consider how to deal with games that:

- Are not turn-based

  → **players play simultaneously**

- Are not zero-sum

  → **players have their own objectives, sometimes cooperate, sometimes compete**

Beyond Turn-based Two-Player Zero-sum Games

In this section we consider how to deal with games that:

- Are not turn-based

  $\rightarrow$ **players play simultaneously**
- Are not zero-sum

  $\rightarrow$ **players have their own objectives, sometimes cooperate, sometimes compete**

Similar techniques could be applied for:

- Are not fully observable $\rightarrow$ **e.g. the player does not know the cards that other players**
- Are not deterministic $\rightarrow$ **e.g. we can roll a die**

Beyond Turn-based Two-Player Zero-sum Games

In this section we consider how to deal with games that:

- Are not turn-based

  → **players play simultaneously**

- Are not zero-sum

  → **players have their own objectives, sometimes cooperate, sometimes compete**

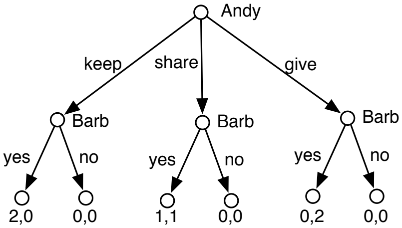
Similar techniques could be applied for:

- Are not fully observable → **e.g. the player does not know the cards that other players**
- Are not deterministic → **e.g. we can roll a die**

→ Side note: In exchange, we are going to simplify things by considering games where the state space is very small (consisting on 1 state :))

Non Zero-Sum Games

The sharing "game": Andy and Barb share two pieces of pie:



- Utilities in the leaves do not always sum to the same number!
- It's not about who wins, each player tries to maximize their utility but sometimes players can co-operate

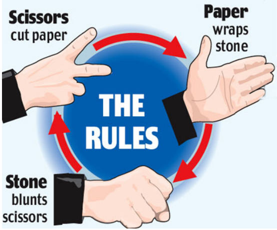
Extensive Form Representation

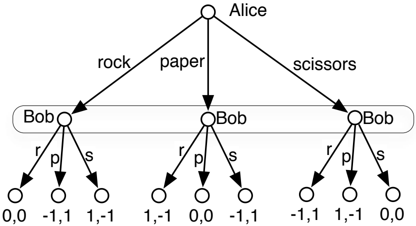
Representation by **game tree**:

- tree whose nodes are labeled with agents
- outgoing arcs labeled by actions of agent
- leaves labeled with one utility value for each agent

Games with Simultaneous Moves: Rock Paper Scissors

Representation of game with simultaneous moves:





Collect in an **information set** the nodes that the agent (Bob) can not distinguish (at all nodes in an information set the same actions must be possible).

The notion of information set is also useful on games with imperfect information:

- Unobserved, random moves by nature (dealing of cards).
- Hidden moves by other agent

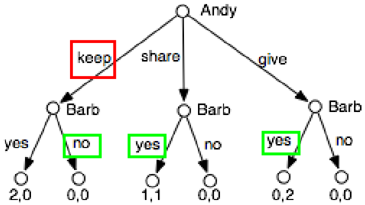
Strategies for Incomplete Information Games

A **(pure) strategy/policy** for one agent is a mapping from information sets to (possible) actions.

→ A strategy tells us which action the player should select under any circumstance within the game

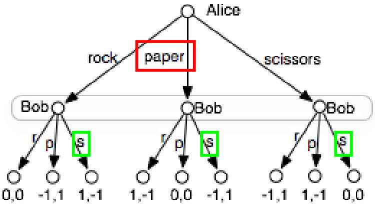
Example strategies for A and strategies for B:

Turn-based Game



Barb can select a different action depending on Andy's choice

Simultaneous-move Game

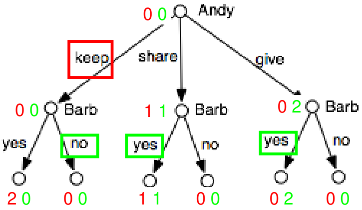


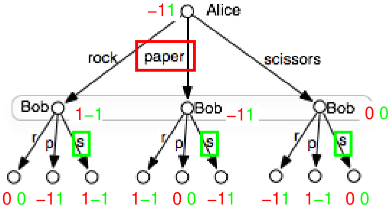
Bob does not know Alice's choice. The three states are an information set and Bob needs to choose the same action for all of them

A **strategy profile** consists of a strategy for each agent.

Normal Form

For each strategy profile, utilities of game are determined:





Share game:

- Strategy Andy: *keep*
- Strategy Barb: *no if keep*, *yes if share*, *yes if give*
- Utilities: 0 for Andy, 0 for Barb

Rock Paper Scissors:

- Strategy Alice: *paper*
- Strategy Bob: *scissors*
- Utilities: -1 for Alice, 1 for Bob

Normal form of a game: Can view game as a big n-dimensional matrix (n is the number of players) consisting of the utility for each strategy profile. For two players:

- Columns: Choice of action by A (possibly: action=strategy)
- Rows: Choice of action by B (possibly: action=strategy)
- Each cell: Utilities determined by these choices

Normal form representations

Share game

Barb	Andy: keep	share	give
$k \rightarrow y, s \rightarrow y, g \rightarrow y$	2 0	1 1	0 2
$k \rightarrow y, s \rightarrow y, g \rightarrow n$	2 0	1 1	0 0
$k \rightarrow y, s \rightarrow n, g \rightarrow y$	2 0	0 0	0 2
$k \rightarrow y, s \rightarrow n, g \rightarrow n$	2 0	0 0	0 0
$k \rightarrow n, s \rightarrow y, g \rightarrow y$	0 0	1 1	0 2
$k \rightarrow n, s \rightarrow y, g \rightarrow n$	0 0	1 1	0 0
$k \rightarrow n, s \rightarrow n, g \rightarrow y$	0 0	0 0	0 2
$k \rightarrow n, s \rightarrow n, g \rightarrow n$	0 0	0 0	0 0

Rock Paper Scissors

Bob	Alice: rock	paper	scissors
rock	0 0	1 -1	-1 1
paper	-1 1	0 0	1 -1
scissors	1 -1	-1 1	0 0

Difference between perfect and imperfect information not directly visible in normal form representation!

Question!

How many non-terminal states does the share game have?

(A): 4 **(B):** 8 **(C):** 24 **(D):** 2^{24}

Normal form representations

Share game

Barb	Andy: keep	share	give
$k \to y, s \to y, g \to y$	2 0	1 1	0 2
$k \to y, s \to y, g \to n$	2 0	1 1	0 0
$k \to y, s \to n, g \to y$	2 0	0 0	0 2
$k \to y, s \to n, g \to n$	2 0	0 0	0 0
$k \to n, s \to y, g \to y$	0 0	1 1	0 2
$k \to n, s \to y, g \to n$	0 0	1 1	0 0
$k \to n, s \to n, g \to y$	0 0	0 0	0 2
$k \to n, s \to n, g \to n$	0 0	0 0	0 0

Rock Paper Scissors

Bob	Alice: rock	paper	scissors
rock	0 0	1 -1	-1 1
paper	-1 1	0 0	1 -1
scissors	1 -1	-1 1	0 0

Difference between perfect and imperfect information not directly visible in normal form representation!

Question!

How many non-terminal states does the share game have?

(A): 4 **(B):** 8 **(C):** 24 **(D):** 2^{24}

$\to$ 4: three where Barb moves and one where Andy moves

And Rock-paper-scissors?

Normal form representations

Share game

Barb \ Andy	keep	share	give
$k \rightarrow y, s \rightarrow y, g \rightarrow y$	2 0	1 1	0 2
$k \rightarrow y, s \rightarrow y, g \rightarrow n$	2 0	1 1	0 0
$k \rightarrow y, s \rightarrow n, g \rightarrow y$	2 0	0 0	0 2
$k \rightarrow y, s \rightarrow n, g \rightarrow n$	2 0	0 0	0 0
$k \rightarrow n, s \rightarrow y, g \rightarrow y$	0 0	1 1	0 2
$k \rightarrow n, s \rightarrow y, g \rightarrow n$	0 0	1 1	0 0
$k \rightarrow n, s \rightarrow n, g \rightarrow y$	0 0	0 0	0 2
$k \rightarrow n, s \rightarrow n, g \rightarrow n$	0 0	0 0	0 0

Rock Paper Scissors

Bob \ Alice	rock	paper	scissors
rock	0 0	1 -1	-1 1
paper	-1 1	0 0	1 -1
scissors	1 -1	-1 1	0 0

Difference between perfect and imperfect information not directly visible in normal form representation!

Question!

How many non-terminal states does the share game have?

(A): 4 **(B):** 8 **(C):** 24 **(D):** 2^{24}

$\rightarrow$ 4: three where Barb moves and one where Andy moves

And Rock-paper-scissors? $\rightarrow$1

Note: the amount of strategies is exponential on the size of the state space (so double exponential in games with exponentially many states) so this representation can only be done in practice for this tiny games. Still, it is a good concept to understand the notions of more complex games, such as poker for example.

Nash Equilibrium

Consider optimal strategy profile for share game:

	Andy		
Barb	keep	share	give
$k \to y, s \to y, g \to y$	2 0	1 1	0 2
$k \to y, s \to y, g \to n$	2 0	1 1	0 0
$k \to y, s \to n, g \to y$	2 0	0 0	0 2
$k \to y, s \to n, g \to n$	2 0	0 0	0 0
$k \to n, s \to y, g \to y$	0 0	1 1	0 2
$k \to n, s \to y, g \to n$	0 0	1 1	0 0
$k \to n, s \to n, g \to y$	0 0	0 0	0 2
$k \to n, s \to n, g \to n$	0 0	0 0	0 0

Nash Equilibrium

Consider optimal strategy profile for share game:

Barb	Andy: keep	Andy: share	Andy: give
$k \to y, s \to y, g \to y$	2 0	1 1	0 2
$k \to y, s \to y, g \to n$	2 0	1 1	0 0
$k \to y, s \to n, g \to y$	2 0	0 0	0 2
$k \to y, s \to n, g \to n$	2 0	0 0	0 0
$k \to n, s \to y, g \to y$	0 0	1 1	0 2
$k \to n, s \to y, g \to n$	0 0	1 1	0 0
$k \to n, s \to n, g \to y$	0 0	0 0	0 2
$k \to n, s \to n, g \to n$	0 0	0 0	0 0

The two strategies are in **Nash Equilibrium**:

- no agent can improve utility by switching strategy while other agent keeps its strategy
- this also means: agent will stick to strategy when it knows the strategy of the other player

If every player maximizes their own utility, they will tend to a Nash Equilibrium
Are there other Nash Equilibriums?

Nash Equilibrium

Consider optimal strategy profile for share game:

Barb	Andy: keep	Andy: share	Andy: give
$k \to y, s \to y, g \to y$	2 0	1 1	0 2
$k \to y, s \to y, g \to n$	2 0	1 1	0 0
$k \to y, s \to n, g \to y$	2 0	0 0	0 2
$k \to y, s \to n, g \to n$	2 0	0 0	0 0
$k \to n, s \to y, g \to y$	0 0	1 1	0 2
$k \to n, s \to y, g \to n$	0 0	1 1	0 0
$k \to n, s \to n, g \to y$	0 0	0 0	0 2
$k \to n, s \to n, g \to n$	0 0	0 0	0 0

The two strategies are in **Nash Equilibrium**:

- no agent can improve utility by switching strategy while other agent keeps its strategy
- this also means: agent will stick to strategy when it knows the strategy of the other player

If every player maximizes their own utility, they will tend to a Nash Equilibrium
Are there other Nash Equilibriums? Yes! The Nash Equilibrium is not unique

Prisoner's Dilemma

Alice and Bob are arrested for burglary. They are separately questioned by police. Alice and Bob are both given the offer to *testify*, in which case

- they will receive a sentence of 5 years each if both testify
- if only one testifies, that person will receive 1 year, and the other 10 years
- if neither testifies, both will get 2 years

	Alice	
Bob	*testify*	*not testify*
testify	-5 -5	-10 -1
not testify	-1 -10	-2 -2

Question!

Which of the following are Nash Equilibria?

(A): Both testify
(B): Only Alice testifies
(C): Only Bob testifies
(D): No one testifies

→ The only Nash equilibrium is Alice:*testify*, Bob:*testify*
Even if by not testifying they could get a better pay-off
Nash equilibria do not represent cooperative behavior!

Nash Equilibrium in Rock-Paper-Scissors

	Alice		
Bob	rock	paper	scissors
rock	0 0	1 -1	-1 1
paper	-1 1	0 0	1 -1
scissors	1 -1	-1 1	0 0

Question!

What is the Nash Equilibrium in Rock Paper Scissors:

Nash Equilibrium in Rock-Paper-Scissors

	Alice		
Bob	rock	paper	scissors
rock	0 0	1 -1	-1 1
paper	-1 1	0 0	1 -1
scissors	1 -1	-1 1	0 0

Question!

What is the Nash Equilibrium in Rock Paper Scissors:

There is no Nash equilibrium if we only consider pure strategies!

Mixed Strategies

Bob	Alice rock	Alice paper	Alice scissors
rock	0 0	1 -1	-1 1
paper	-1 1	0 0	1 -1
scissors	1 -1	-1 1	0 0

A **mixed strategy** is a probability distribution over actions.

Mixed Strategy for Alice: $r : 1/3$ $p : 1/3$ $s : 1/3$
Mixed Strategy for Bob: $r : 1/3$ $p : 1/3$ $s : 1/3$

Expected utility for Alice = expected utility for Bob =

$$1/9(0 + 1 - 1 - 1 + 0 + 1 + 1 - 1 + 0) = 0$$

Mixed Strategies

Bob	Alice rock	Alice paper	Alice scissors
rock	0 0	1 -1	-1 1
paper	-1 1	0 0	1 -1
scissors	1 -1	-1 1	0 0

A **mixed strategy** is a probability distribution over actions.

Mixed Strategy for Alice: $r : 1/3\ p : 1/3\ s : 1/3$
Mixed Strategy for Bob: $r : 1/3\ p : 1/3\ s : 1/3$

Expected utility for Alice = expected utility for Bob =

$$1/9(0 + 1 - 1 - 1 + 0 + 1 + 1 - 1 + 0) = 0$$

Suppose Alice plays some other strategy: $r : p_r\ p : p_p\ s : p_s$. Expected utility for Alice then:

$$1/3(p_r \cdot 0 + p_p \cdot 1 - p_s \cdot 1 - p_r \cdot 1 + p_p \cdot 0 + p_s \cdot 1p_r \cdot +1 - p_p \cdot 1 + p_s \cdot 0) =$$
$$1/3(p_p + p_r + p_s - p_p - p_r - p_s) = 0$$

- If Bob plays $r : 1/3\ p : 1/3\ s : 1/3$, Alice can not do better than playing $r : 1/3\ p : 1/3\ s : 1/3$ also.
- Same for Bob
- Both playing $r : 1/3\ p : 1/3\ s : 1/3$ is a (the only) Nash equilibrium

Key Results Regarding Nash Equilibria

- Nash Equilibria represent rational play under the assumption that all players maximize their utility
  "if we assume that players are rational, know the full structure of the game, the game is played just once, and there is just one Nash equilibrium, then players will play according to that equilibrium."
- Every (finite) game has a Nash equilibrium (using mixed strategies)
- There can be multiple Nash equilibria
- Playing a Nash equilibrium strategy profile does not necessarily lead to optimal utilities for the agents (prisoner's dilemma)

Agenda

Summary

- Games that are 2-player turn-taking zero-sum discrete and finite can be understood as a simple extension of classical search problems.
- Each player tries to reach a **terminal state** with the best possible **utility** (maximal vs. minimal).
- **Minimax** searches the game depth-first, max'ing and min'ing at the respective turns of each player. It yields perfect play, but takes time $O(b^d)$ where b is the branching factor and d the search depth.
- Except in trivial games (Tic-Tac-Toe), Minimax needs a **depth limit** and apply an **evaluation function** to estimate the value of the cut-off states.
- **Nash Equilibria** characterize the strategies of rational players that maximize their own utility. When considering mixed strategies, every game has a Nash Equilibria.