CS 161 Fundamentals of Artificial Intelligence
Lecture 7

Game playing

Quanquan Gu

Department of Computer Science
UCLA

Feb 2, 2023

Outline

- Games
- Perfect play
  - minimax decisions
  - α–β pruning
- Resource limits and approximate evaluation
- Games of chance
- Games of imperfect information

Types of Games

	deterministic	chance
(full) perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information (partial	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

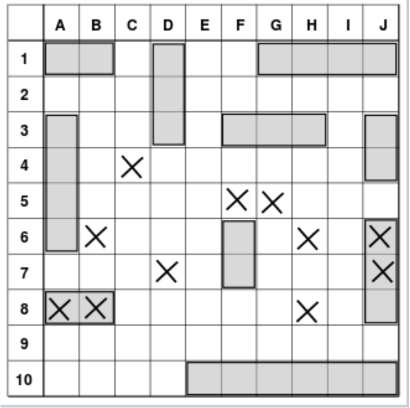
Games

Go: Perfect and Deterministic

Games

Monopoly: Perfect, Chance Introduced

Games



Battleship: Imperfect and Deterministic

Games

Bridge: Imperfect, Chance Introduced

Games vs. search problems

Can we use search strategies to win games?

• What would be the solution when applying search strategies to games?

• The solution will be a strategy that specifies a move for every possible opponent reply

Challenges

• Very very large search space

• Time limits

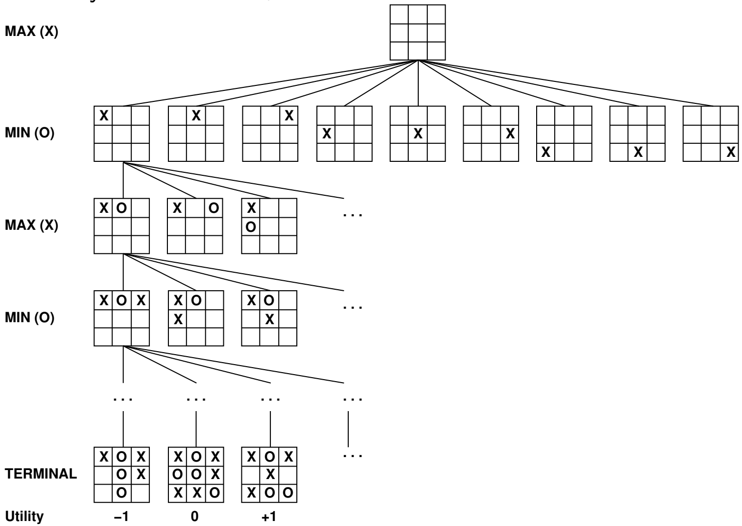
Game as a Search Problem

- **S0**: The initial state, which specifies how the game is set up at the start.
- **PLAYER(s)**: Defines which player has the move in a state.
- **ACTIONS(s)**: Returns the set of legal moves in a state.
- **RESULT(s, a)**: The transition model, which defines the result of a move.
- **TERMINAL-TEST(s)**: A terminal test, which is true when the game is over and false otherwise.
  - States where the game has ended are called terminal states.
- **UTILITY(s, p)**: A utility function that defines the final numeric value for a game that ends in terminal state s for a player p.
  - Also called an objective function or payoff function.
  - In chess, the outcome is a win, loss, or draw, with values $+1$, 0, or $1/2$.

Game tree (2-player, deterministic, turns)

Tic-tac-toe Game Tree

- Two Players: MAX: X; MIN O



Optimal Decisions in Games

How to find the optimal decisions in a deterministic, perfect-information game?

• **Idea:** choose the move with highest achievable payoff against the best play of the other player

• Important assumption: we assume that both players play optimally from beginning to the end of the game

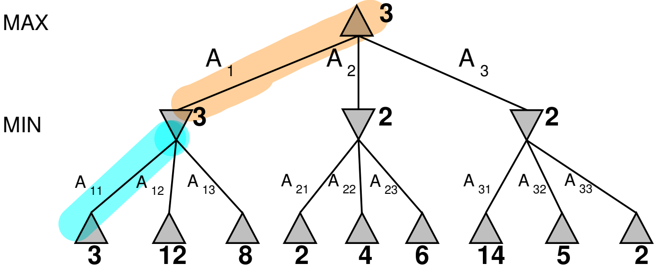
Minimax

- We refer to the two players as **MAX** and **MIN**.
    - Without loss of generality, we imagine that we are <u>MAX</u> playing against MIN

- We refer to the payoff as MINIMAX value
    - The player MAX will always choose the move with the <u>maximum</u> MINIMAX value
    - The player MIN will always choose the move with the <u>minimum</u> MINIMAX value

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Minimax algorithm



Minimax algorithm

max-value min-value

function MINIMAX-DECISION(*state*) **returns** *an action*
inputs: *state*, current state in game

return the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow -\infty$
for *a*, *s* in SUCCESSORS(*state*) **do** $v \leftarrow$ MAX(v, MIN-VALUE(s))
return *v*

---

function MIN-VALUE(*state*) **returns** *a utility value*
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow \infty$
for *a*, *s* in SUCCESSORS(*state*) **do** $v \leftarrow$ MIN(v, MAX-VALUE(s))
return *v*

$V = -\infty$

A_1	A_2	A_3
3	7	1
$V = \max(V, 3) = 3$	$V = \max(V, 7) = 7$	$V = \max(V, 1) = 7$

Properties of minimax

Complete??

Properties of minimax

Complete?? Only if tree is finite (chess has specific rules for this).
Note that a finite strategy can exist even in an infinite tree!
Optimal??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent.
Time complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent.
Time complexity?? $O(b^m)$
b is the branching factor, m is the maximum depth of the game tree
Space complexity??

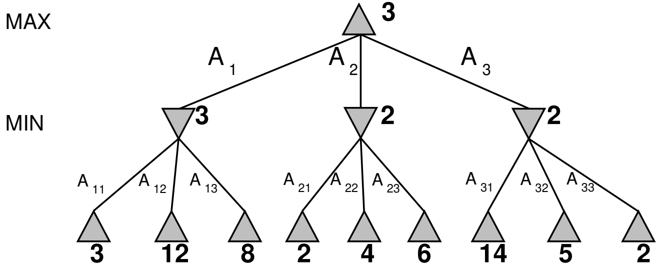
Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent.
Time complexity?? $O(b^m)$
Space complexity?? $O(bm)$ (depth-first exploration)

- Chess: $b \approx 35$, $m \approx 100$
  - Exact solutions could be completely infeasible for "reasonable" games

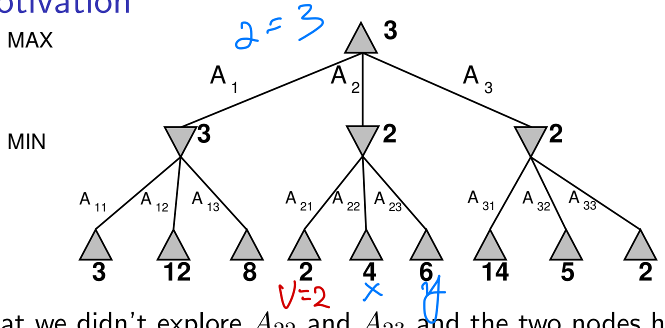
But do we need to explore every path and compute MINIMAX for every node?

α–β: Motivation



Do we really need to calculate the minimax value for every node?

α–β: Motivation

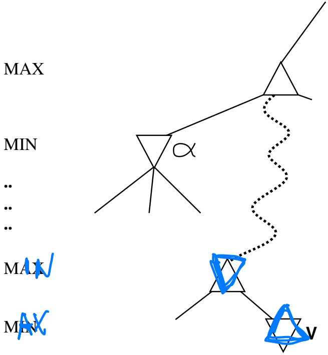


Say that we didn't explore A_{22} and A_{23} and the two nodes have unknown values x and y, then

$$
\begin{aligned}
\text{MINIMAX}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \qquad \text{where } z = \min(2,x,y) \leq 2 \\
&= 3.
\end{aligned}
$$

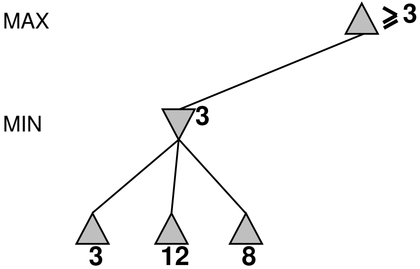
The value of the root is independent of x and y! No need to reveal them.

Why is it called α–β?

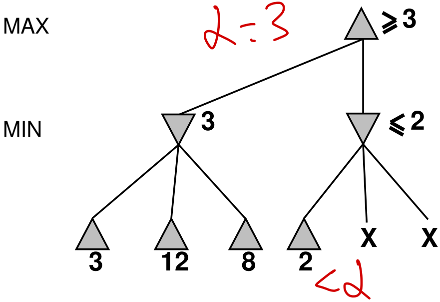


- α: the best value (to MAX) found so far off the current path. If v is worse than α, MAX will avoid it $\Rightarrow$ prune that branch
- β: similarly defined for MIN

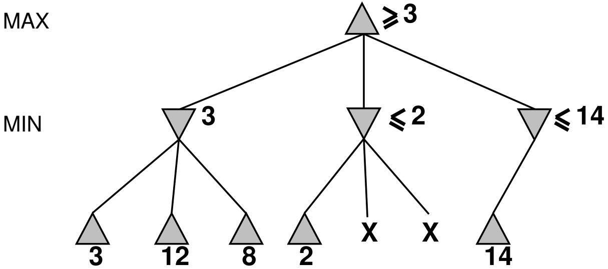
α–β pruning example



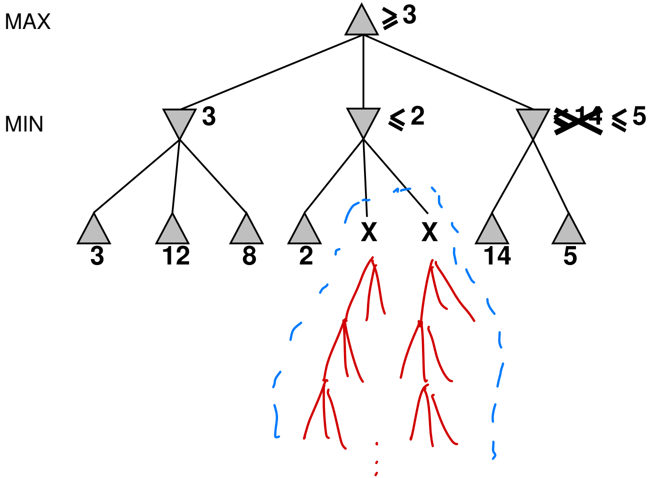
α–β pruning example



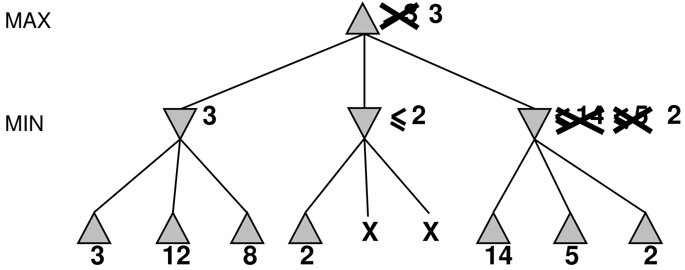
α–β pruning example



α–β pruning example



α–β pruning example



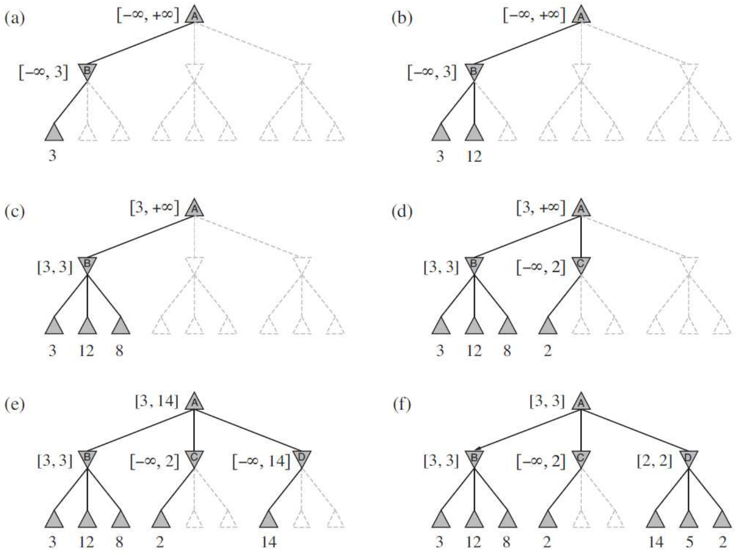
```
function ALPHA-BETA-DECISION(state) returns an action
return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

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

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

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

α–β pruning



Properties of α–β

- Pruning <u>does not</u> affect final result
- Good move ordering improves effectiveness of pruning
  - With "perfect ordering," time complexity $= O(b^{m/2})$

$(\sqrt{b})^m$

Chess: $b \approx 35$, $m \approx 100$
Unfortunately, 35^{50} is still infeasible!

prune $(b - \sqrt{b})$

Improvements under resource limits

Standard approach:

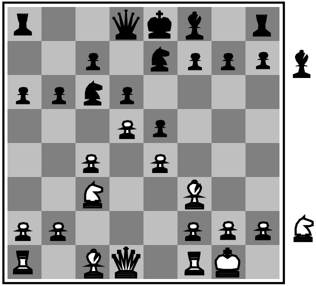
- Use CUTOFF-TEST instead of TERMINAL-TEST
  - e.g., depth limit
- Use EVAL instead of UTILITY
  - i.e., a <u>evaluation function</u> that estimates desirability of position

Suppose that we have 100 seconds and can explore 10^4 nodes/sec

- We can explore $10^6 \approx 35^{8/2}$ nodes per move
- α–β can reach depth 8
  - a pretty good chess program!

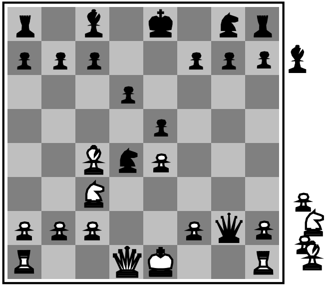
$35^{8/2}$

Evaluation functions



Black to move

White slightly better



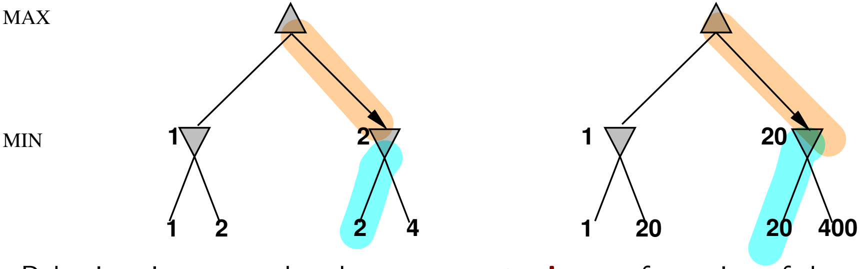
White to move

Black winning

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- Typically using linear weighted sum of features for chess
- A example of features:
  $f_1(s)$ = (number of white queens) – (number of black queens),

Digression: Exact values don't matter

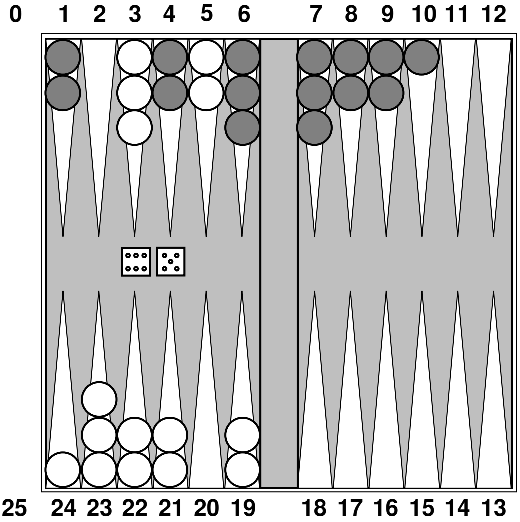


Behaviour is preserved under any **monotonic** transformation of the original EVAL

Deterministic games in practice

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.
- **Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- **Go:** Alpha Go, which is based on deep reinforcement learning, and Monte Carlo tree search

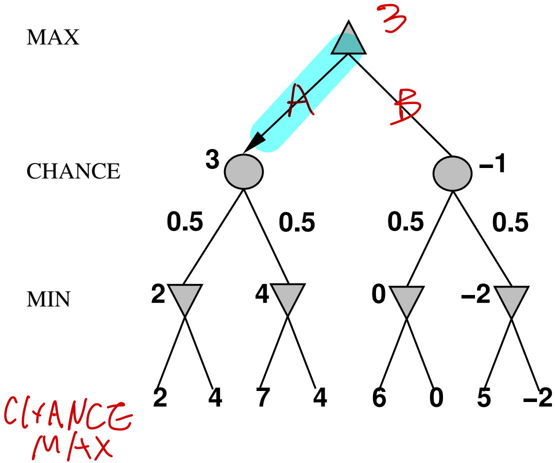
Nondeterministic games: backgammon



Nondeterministic (Stochastic) games in general

- In nondeterministic games, **chances** are introduced.
  - Example: dice roll, card-shuffling, coin-flipping, etc.

A simplified example with coin-flipping:



Algorithm for nondeterministic games

- EXPECTIMINIMAX gives perfect play
  - Just like MINIMAX, except we must also handle chance nodes

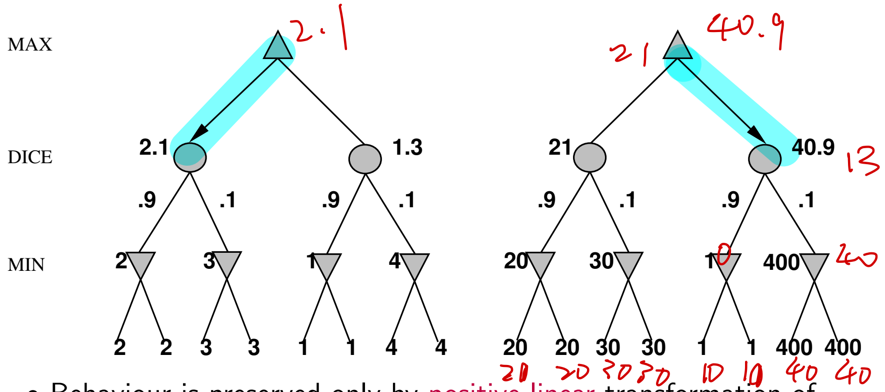
if *state* is a MAX node **then**
  return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

$P(s' = \text{Result}(s,a)) = \ldots$

if *state* is a MIN node **then**
  return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a chance node **then**
  return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

$$\sum_{s'} P(\text{Result}(s,a) = s') \cdot \text{MINIMAX-VALUE}(\text{Result}(s,a))$$

Digression: Exact values DO matter



- Behaviour is preserved only by positive linear transformation of EVAL
  - Hence EVAL should be proportional to the expected payoff

Games of imperfect information

Example

- In card games, opponent's initial cards are unknown

- Typically we can calculate a probability for each possible deal
  - Consider as a big dice roll at the beginning of the game
- Compute the minimax value of each action in each deal,
- Then choose the action with highest expected value over all deals

Special case: if an action is optimal for all deals, it's optimal.

The current best bridge program, approximates this idea by
1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

Games of imperfect information

Suppose that each deal s occurs with probability $P(s)$
The move we want is

$$\operatorname*{argmax}_a \sum_s P(s)\text{MINIMAX}(\text{RESULT}(s,a)) \tag{1}$$

In practice, the number of possible deals is rather large.
We resort to a Monte Carlo approximation:

- We take a random sample of N deals instead of adding up all the deals,
- The probability of deal s appearing in the sample is proportional to $P(s)$

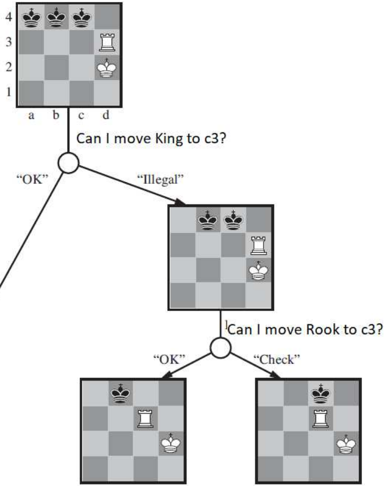
$\{s_1, s_2, s_3 \cdots s_N\}$

$$\operatorname*{argmax}_a \frac{1}{N} \sum_{i=1}^{N} \text{MINIMAX}(\text{RESULT}(s_i,a)) \tag{2}$$

Imperfect information example: Kriegspiel

- A partially observable variant of chess in which pieces can move but are completely invisible to the opponent.
- Rules:
  - White and Black each see a board containing only their own pieces.
  - A referee, who can see all the pieces
  - White proposes to the referee any move that would be legal if there were no black pieces. If the move is in fact not legal (because of the black pieces), the referee announces "illegal."
- Can white win the game?
- **Guaranteed checkmate**: for each possible percept sequence, leads to an actual checkmate for every possible board state, regardless of how the opponent moves

Example



Part of a guaranteed checkmate in the KRK endgame, shown on a reduced board. In the initial belief state, Black's king is in one of three possible locations.

Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you'll find a mound of jewels;
    take the right fork and you'll be run over by a bus.

Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
  take the left fork and you'll find a mound of jewels;
  take the right fork and you'll be run over by a bus.
Road A leads to a small heap of gold pieces
Road B leads to a fork:
  take the left fork and you'll be run over by a bus;
  take the right fork and you'll find a mound of jewels.

Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
  take the left fork and you'll find a mound of jewels;
  take the right fork and you'll be run over by a bus.
Road A leads to a small heap of gold pieces
Road B leads to a fork:
  take the left fork and you'll be run over by a bus;
  take the right fork and you'll find a mound of jewels.
Road A leads to a small heap of gold pieces
Road B leads to a fork:
  guess correctly and you'll find a mound of jewels;
  guess incorrectly and you'll be run over by a bus.

Proper analysis

* Intuition that the value of an action is the average of its values in all actual states is **WRONG**

With partial observability, value of an action depends on the **information state** or **belief state** the agent is in

Can generate and search a tree of information states

Leads to rational behaviors such as

- Acting to obtain information
- Signalling to one's partner
- Acting randomly to minimize information disclosure

Summary

Games are fun to work on! (and dangerous)
They illustrate several important points about AI
- perfection is unattainable $\Rightarrow$ must approximate
- good idea to think about what to think about
- uncertainty constrains the assignment of values to states
- optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design

Acknowledgment

The slides are adapted from Stuart Russell, Guy Van den Broeck et al.