CSE 40171:
Artificial Intelligence

Adversarial Search: Games, Optimality, and Minimax

Homework #3 has been released
It is due at 11:59PM on 10/9

What is a game?

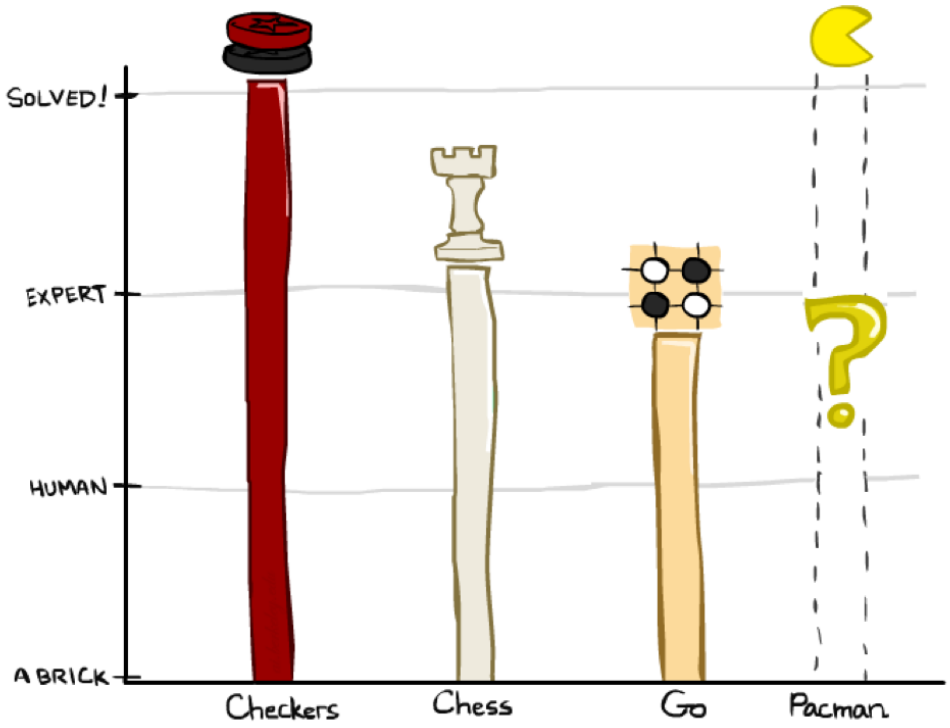
Game Playing State-of-the-Art

Checkers: 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

Chess: 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

Go: 2016: AlphaGo, a deep learning-based system, beat Lee Sedol, a 9-dan professional without handicaps, in a five game match. The win was a major milestone in data driven approaches to game playing.

Pacman



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Behavior from Computation

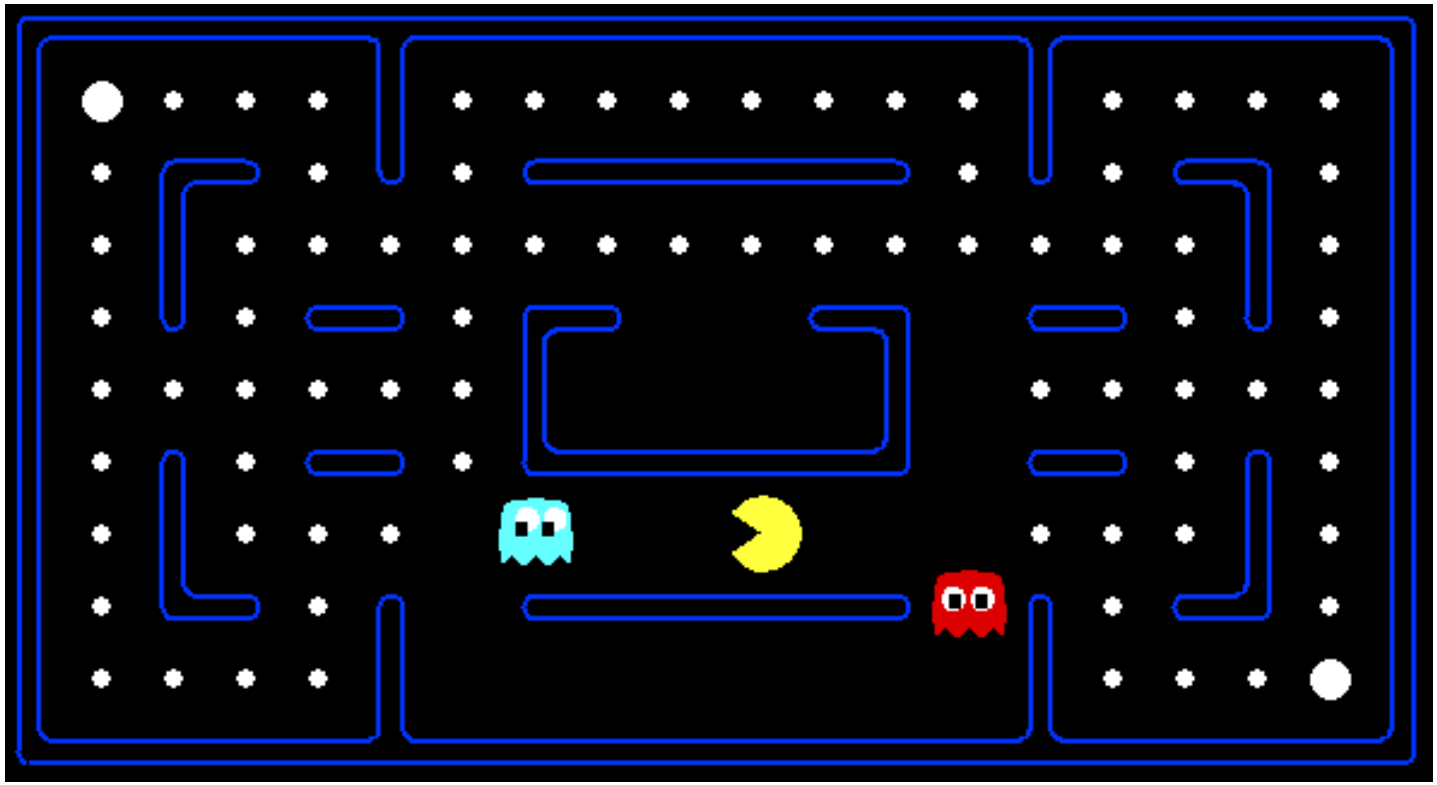
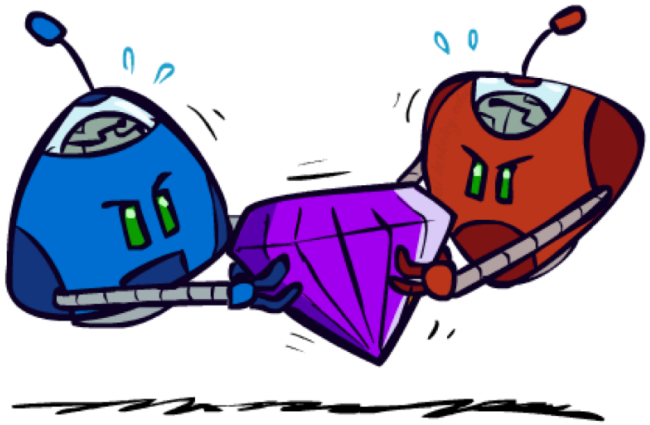


Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

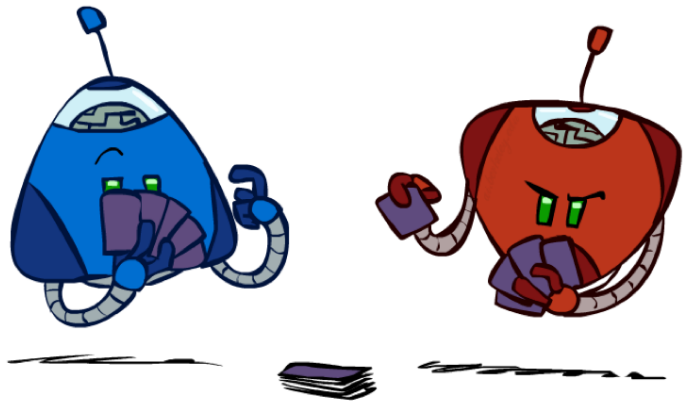
Adversarial Games



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Types of Games

Many different kinds of games!



Axes:

- Deterministic or stochastic?
- One, two, or more players?
- Zero sum?
- Perfect information (can you see the state)?

Want algorithms for calculating a **strategy (policy)** which recommends a move from each state

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

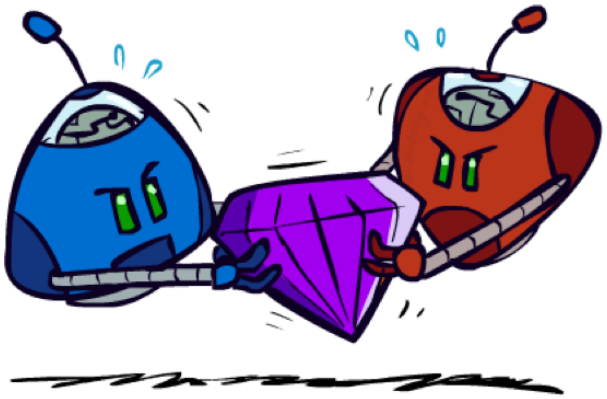
Formal Elements of a Game

- S_0: the **initial state**, which specifies how the game is set up at the start
- $\text{PLAYER}(s)$: Defines which player has the move in a state
- $\text{ACTIONS}(s)$: Returns the set of legal moves in a state
- $\text{RESULT}(s, a)$: the **transition model**, which defines the result of a move

Formal Elements of a Game

- 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** (a.k.a. objective or payoff function) defines the final numeric value for a game that ends in terminal state s for a player p.

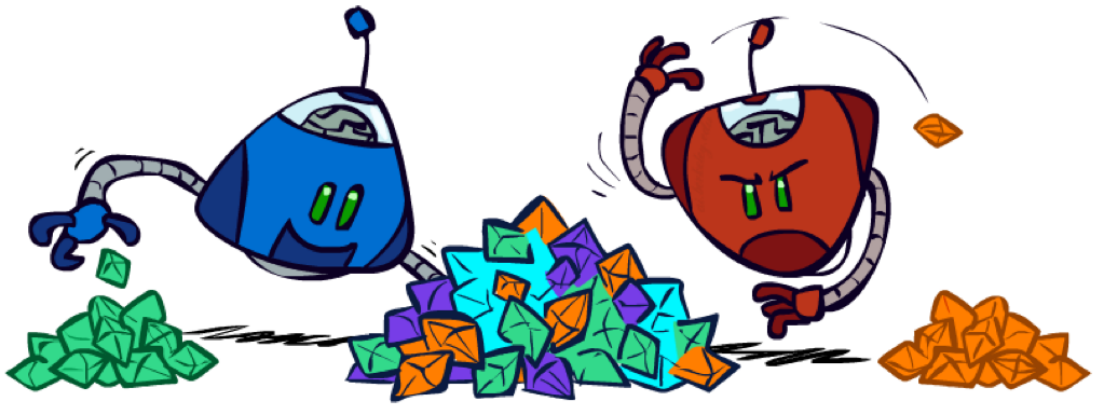
Zero-Sum Games



- Agents have opposite utilities (values on outcomes)
- Lets us think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

General Games



- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible
- More later on non-zero-sum games

Two Players

MAX

- Moves first
- High values are good for MAX

MIN

- Moves after MAX
- High values are bad for MIN

Game Trees

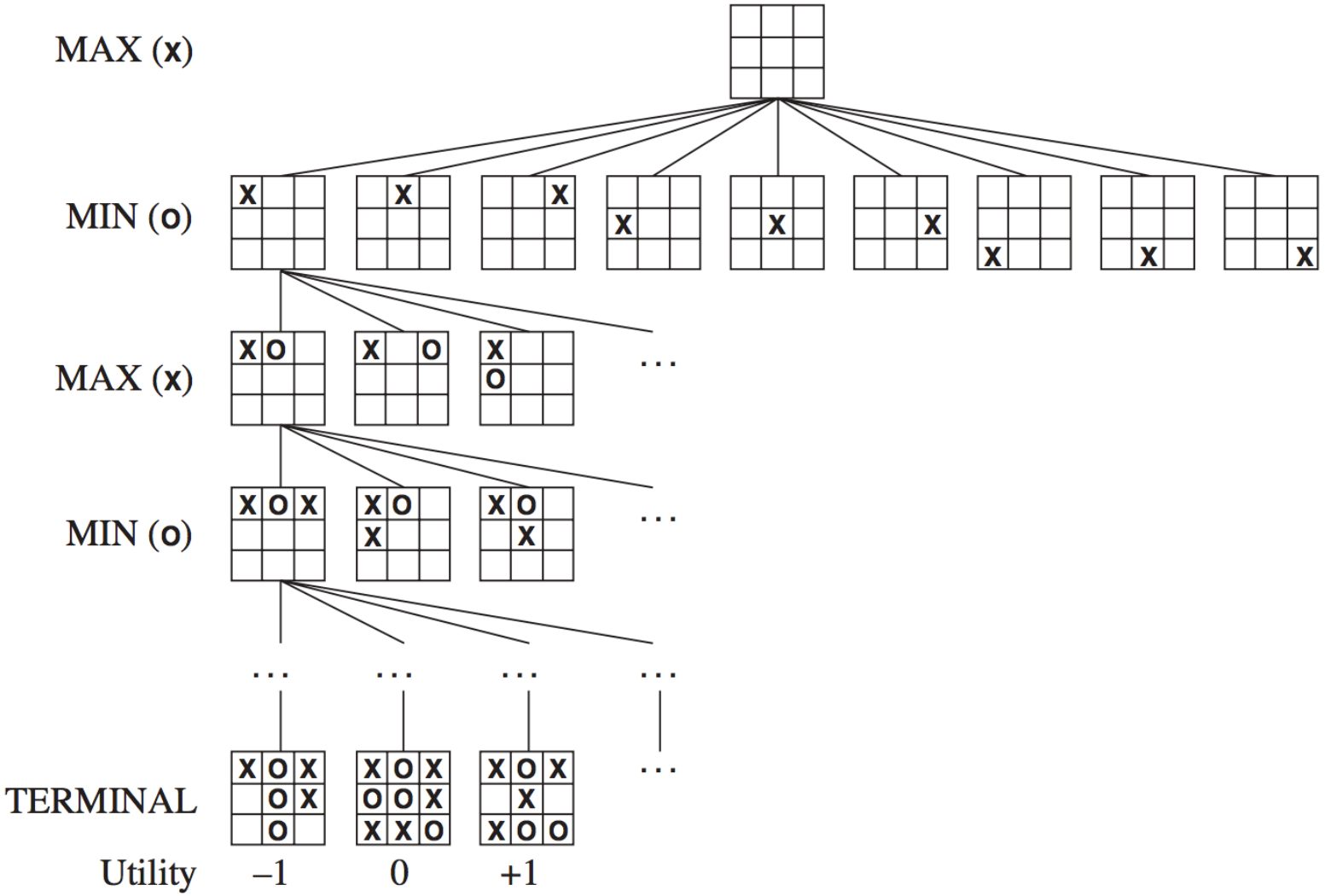
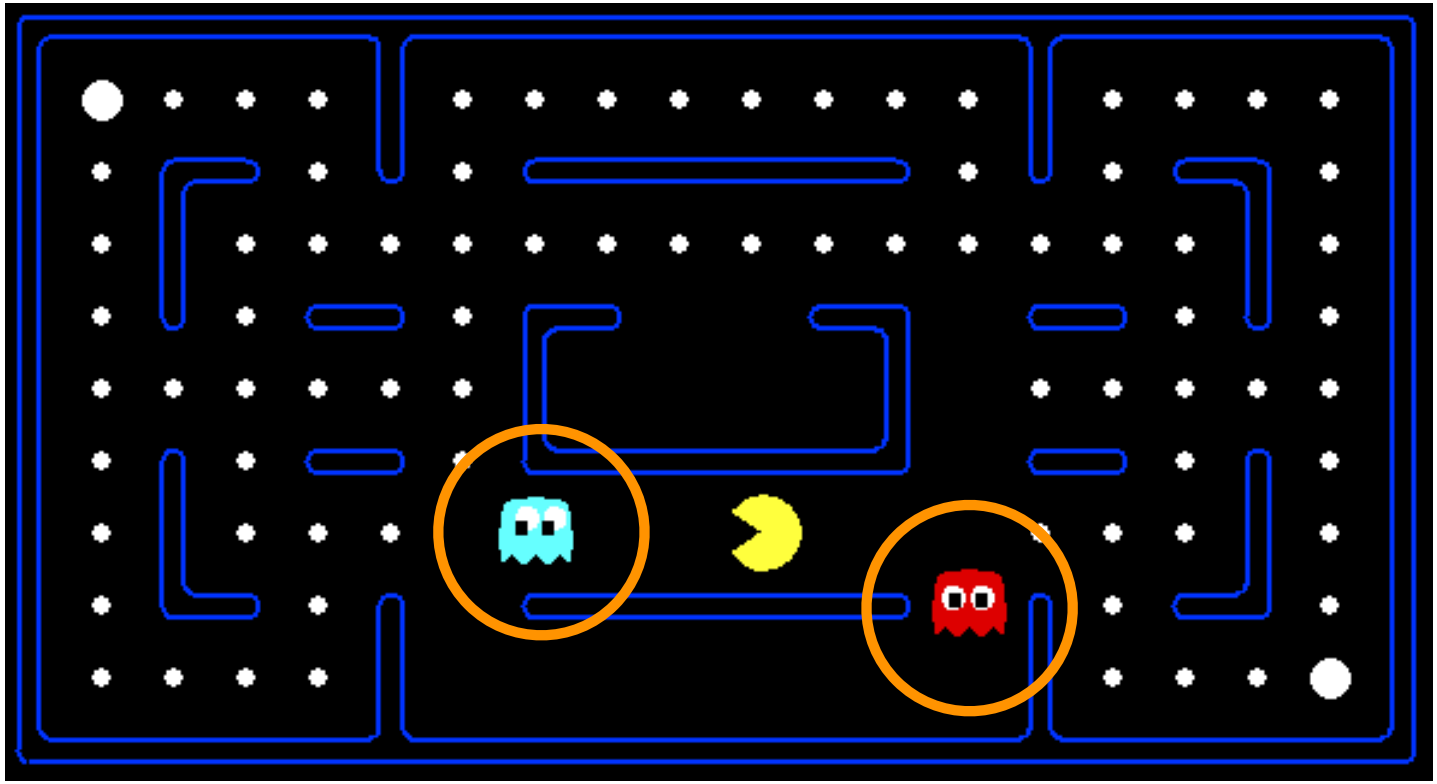


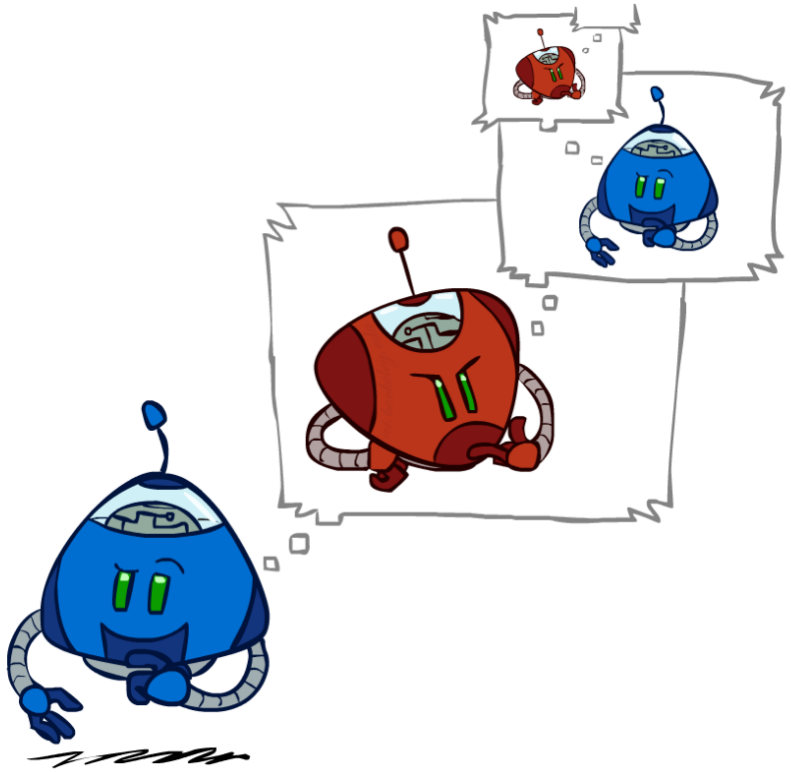
Image credit: Russell and Norvig

Optimal Decisions in Games

What is different about this compared to basic search?



Adversarial Search



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Single-Agent Trees

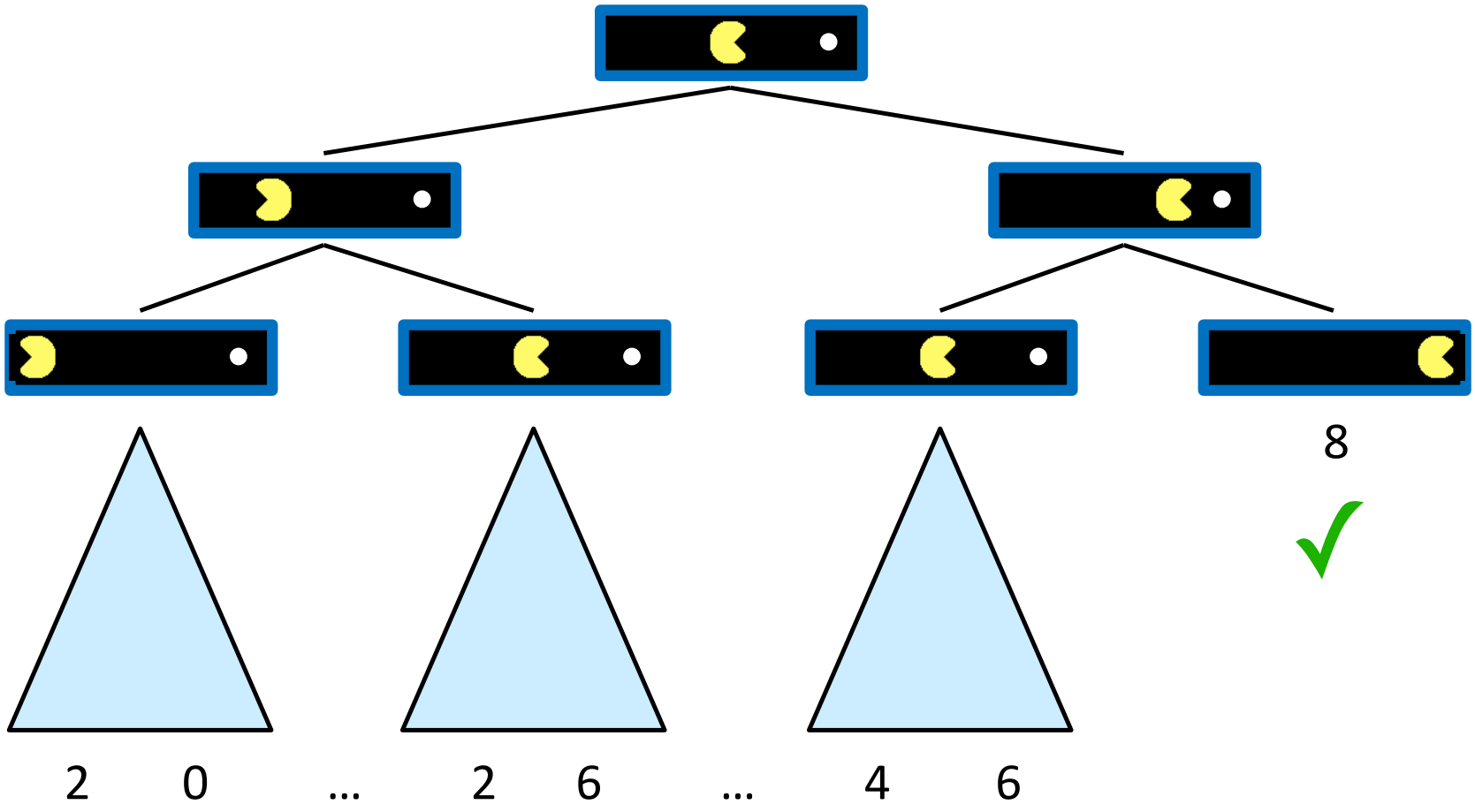


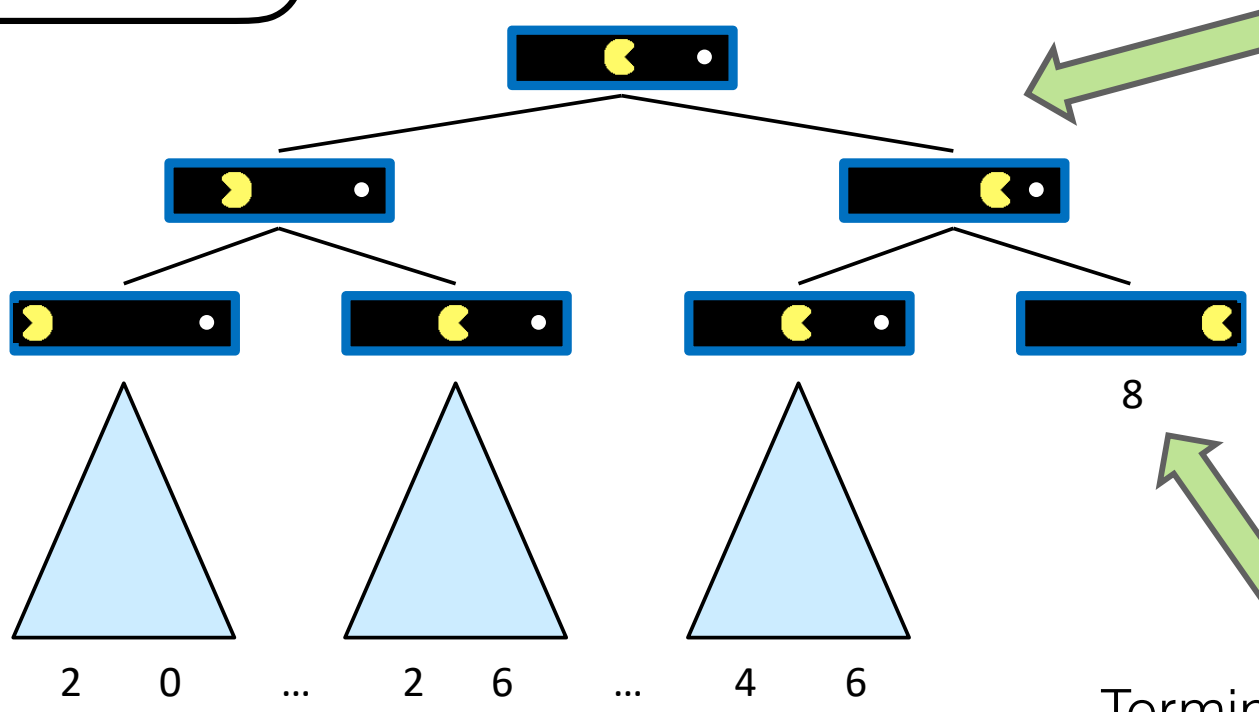
Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Value of a State

Value of a state: The best achievable outcome (utility) from that state

Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

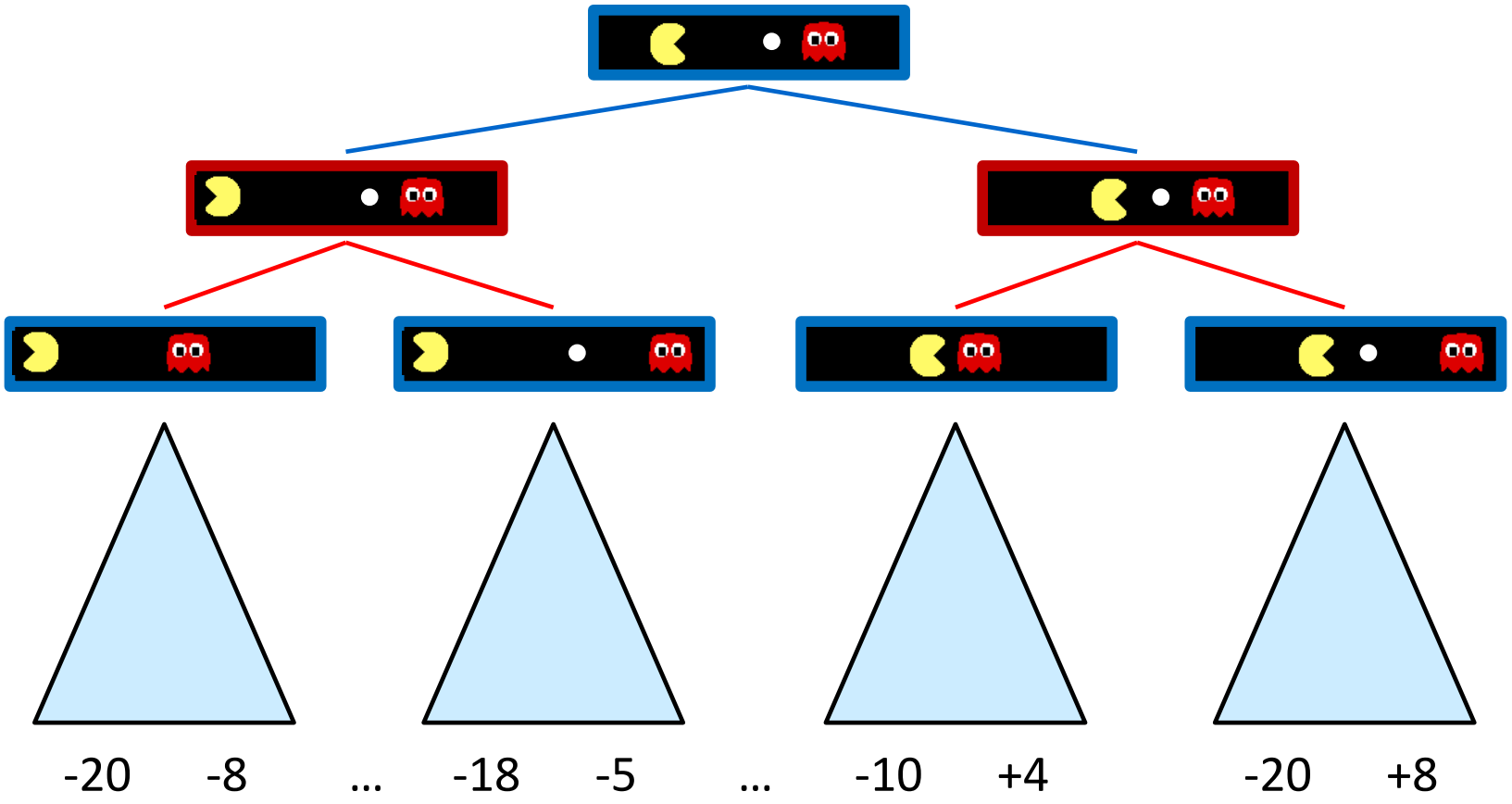


Terminal States:

$$V(s) = \text{known}$$

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Adversarial Game Trees



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

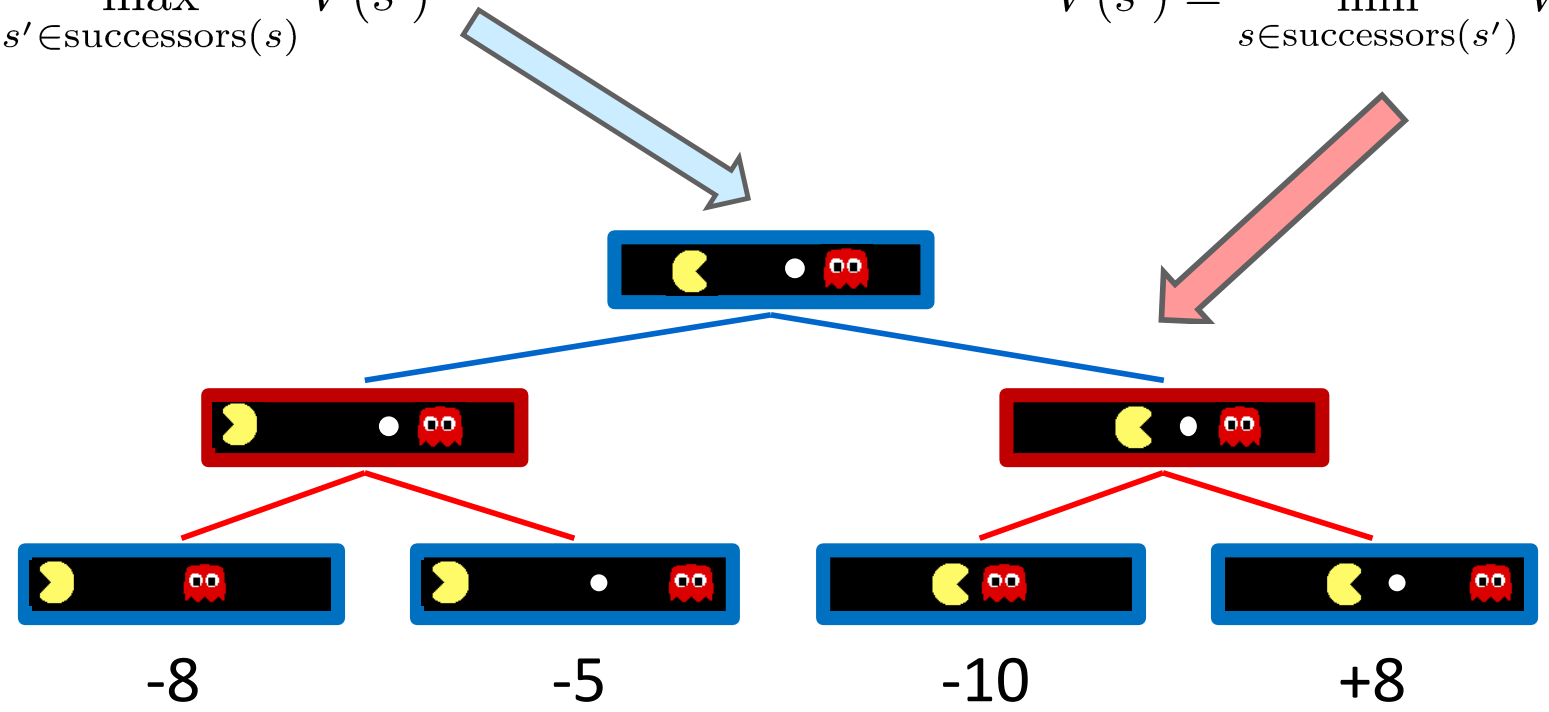
Minimax Values

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

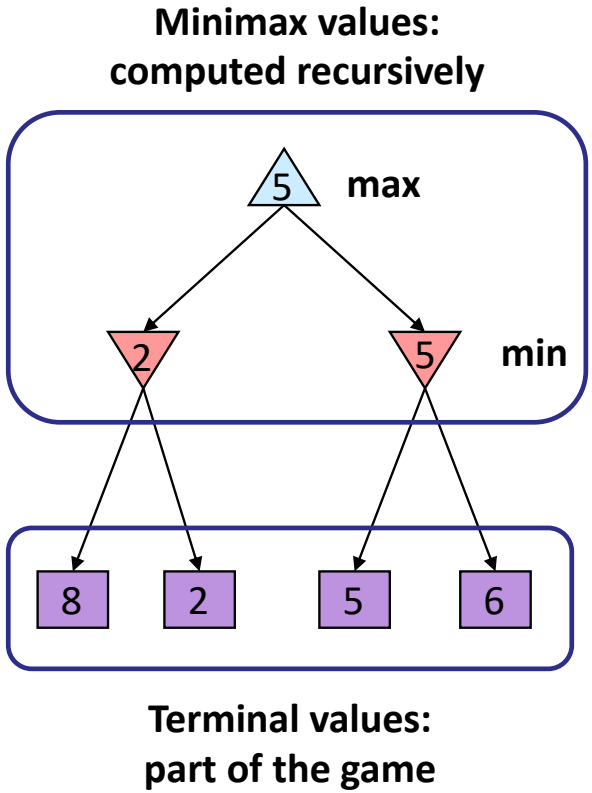
$$V(s) = \text{known}$$

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Adversarial Search (Minimax)

Minimax search:

- A state-space search tree
- Players alternate turns
- Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary



Minimax Implementation

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

Minimax Implementation (Dispatch)

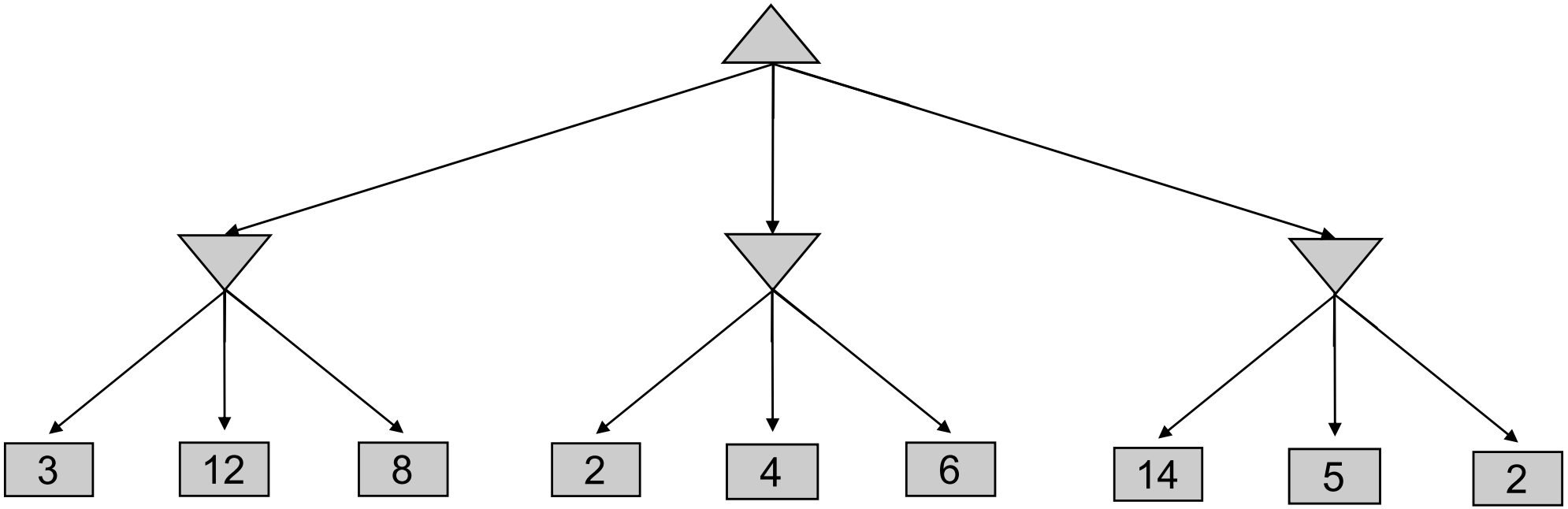
```
def value(state):
        if the state is a terminal state: return the state's utility
        if the next agent is MAX: return max-value(state)
        if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
        initialize v = -∞
        for each successor of state:
                v = max(v, min-
                    value(successor))
        return v
```

```
def min-value(state):
        initialize v = +∞
        for each successor of state:
                v = min(v, max-
                    value(successor))
        return v
```

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Minimax Example



Minimax Example

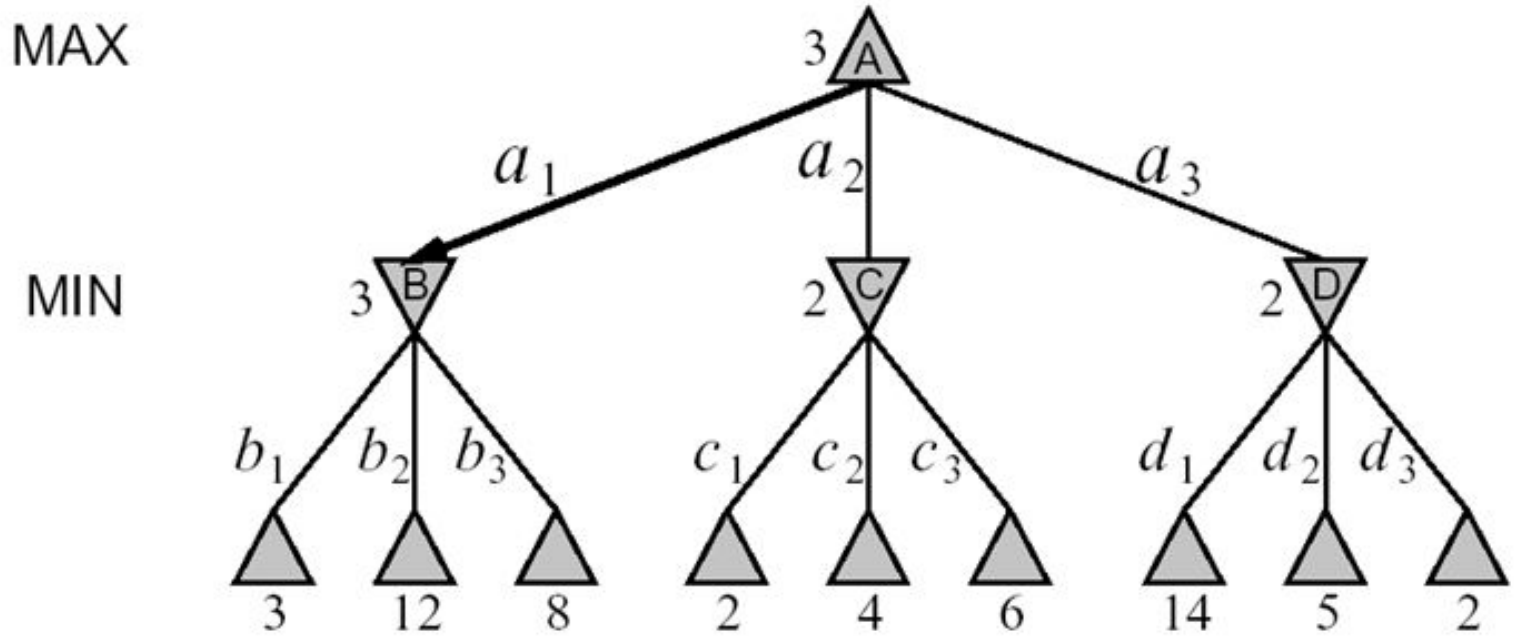
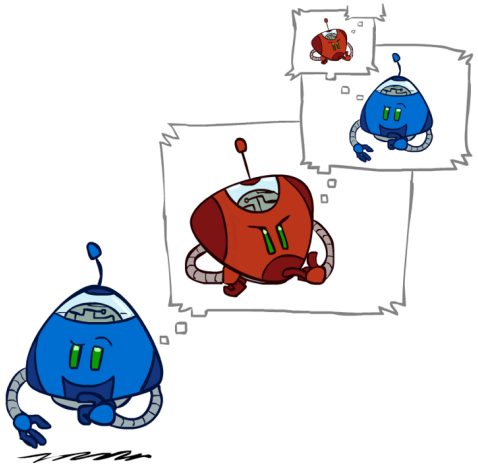


Image credit: Russell and Norvig

Minimax Efficiency

How efficient is minimax?

- Just like (exhaustive) DFS
- Time: $O(b^m)$
- Space: $O(bm)$

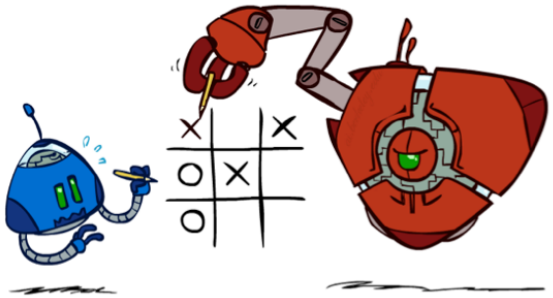


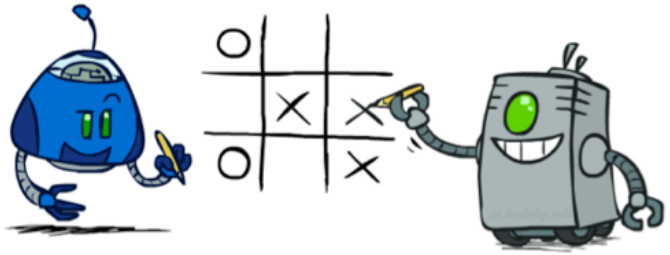
Example: For chess, $b \approx 35$, $m \approx 100$

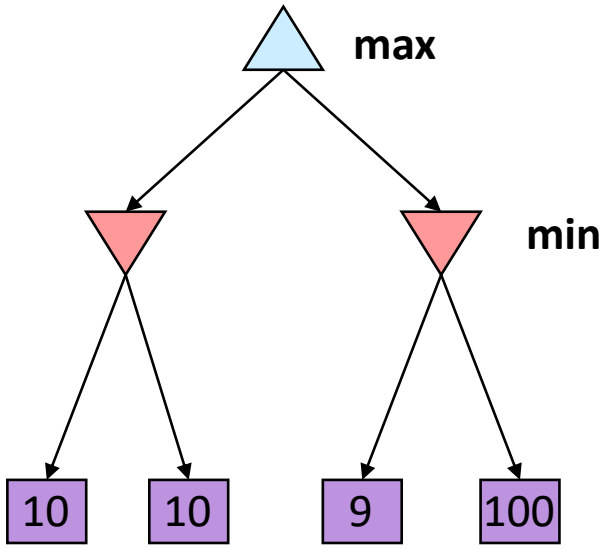
- Exact solution is completely infeasible
- But, do we need to explore the whole tree?

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Minimax Properties



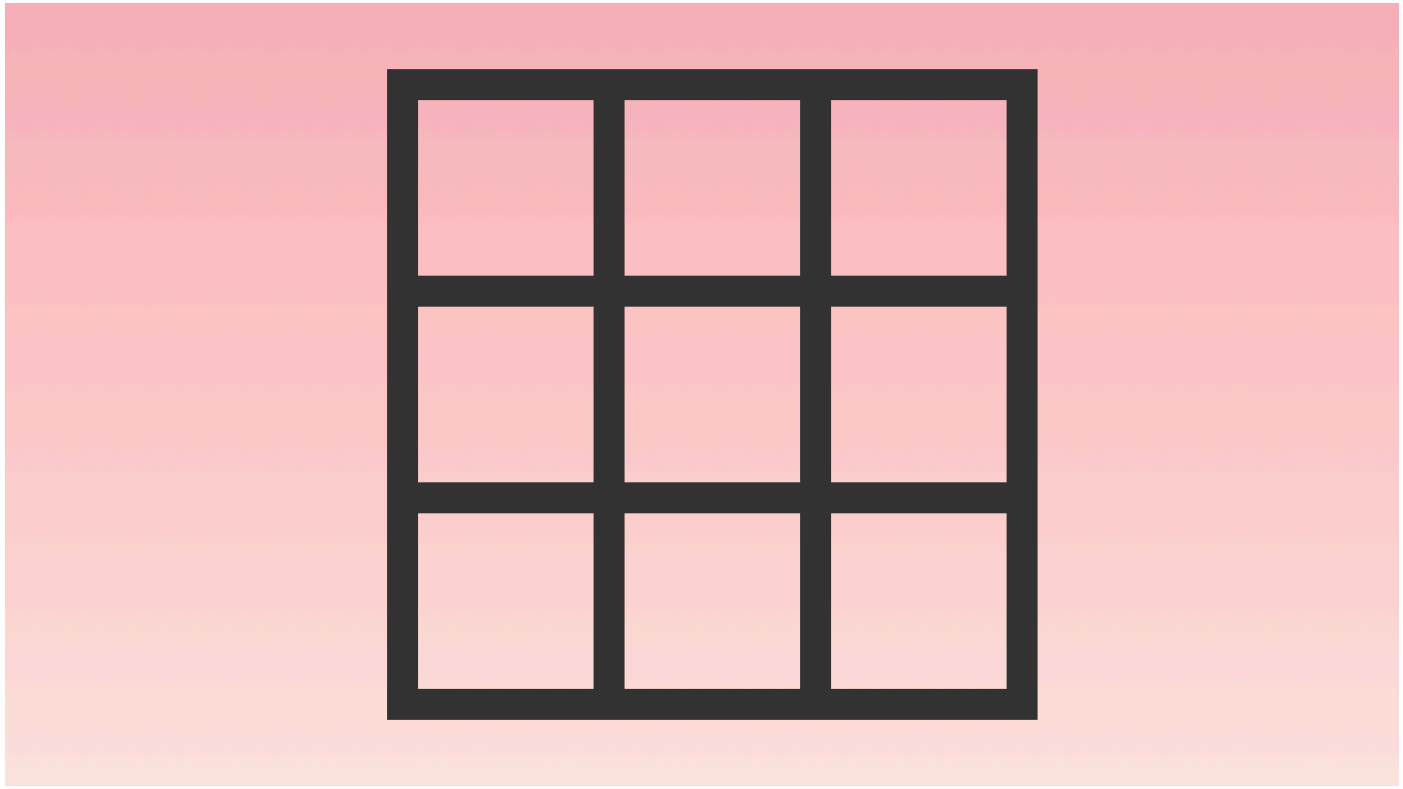




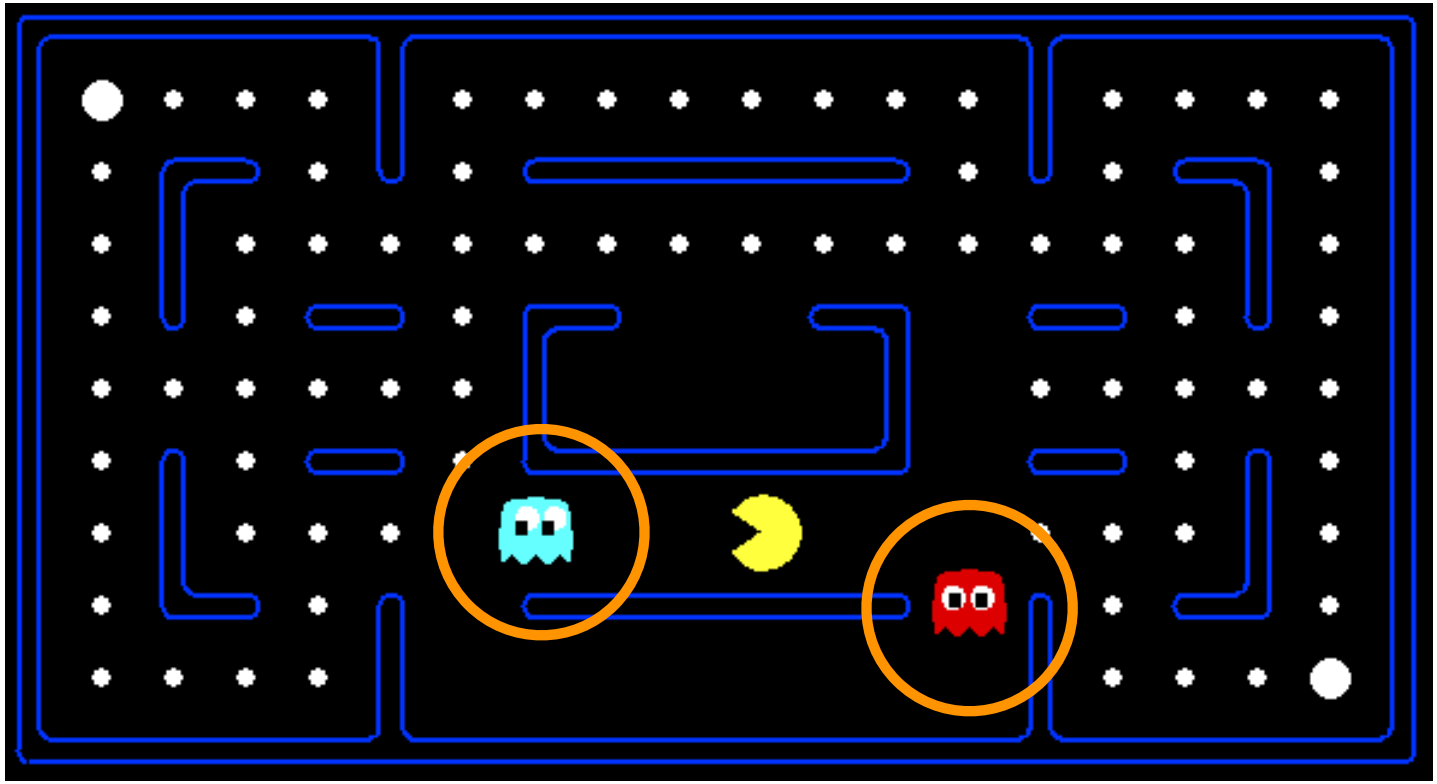
Optimal against a perfect player. Otherwise?

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Minimax Demo



But we have two of these guys — what do we do?



Multi-player Games

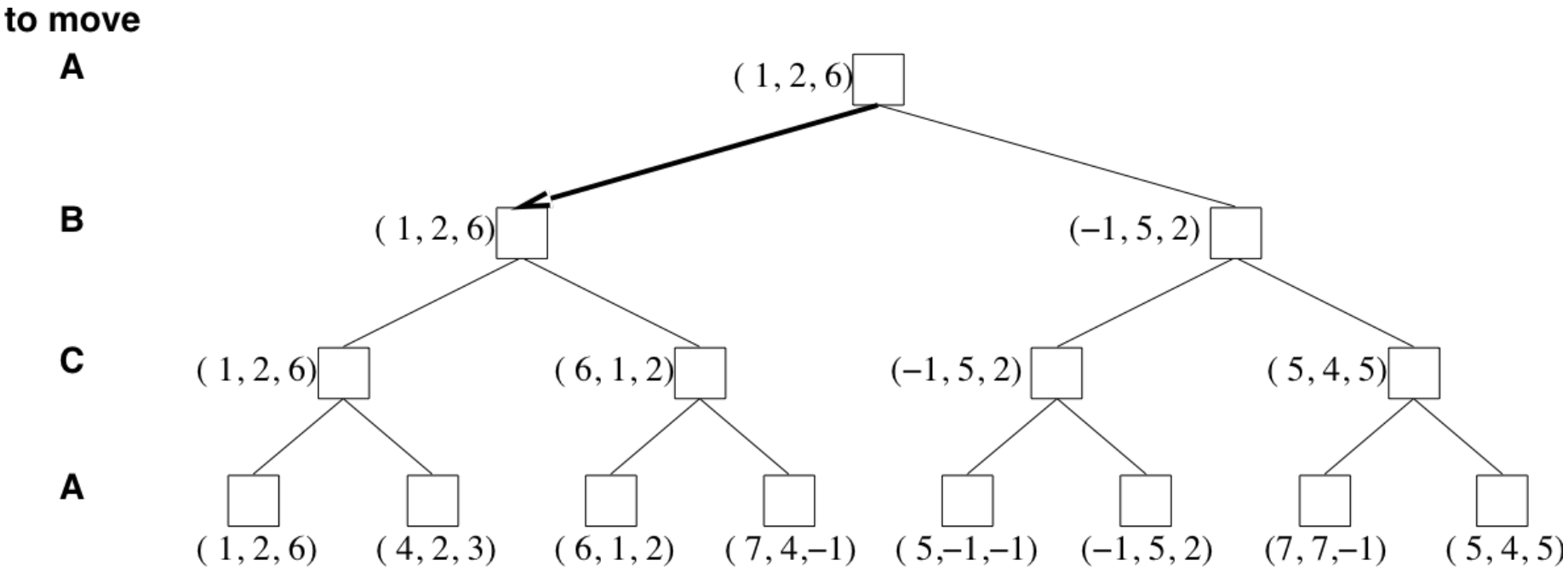
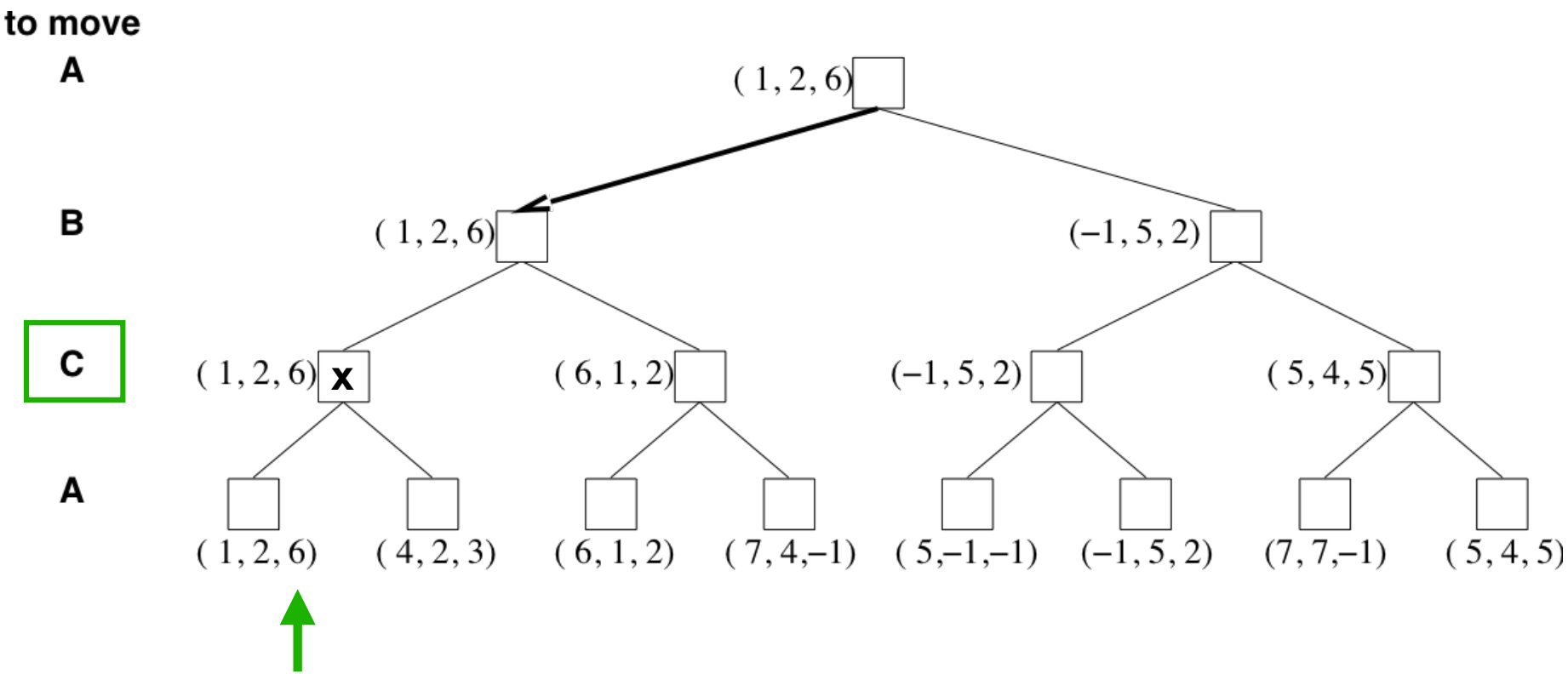


Image credit: Russell and Norvig

Multi-player Games



Now what if A and B begin to collaborate?

Image credit: Russell and Norvig

Multi-player Games

Diplomacy: Game 1 - Round 1

Minimax: GANs

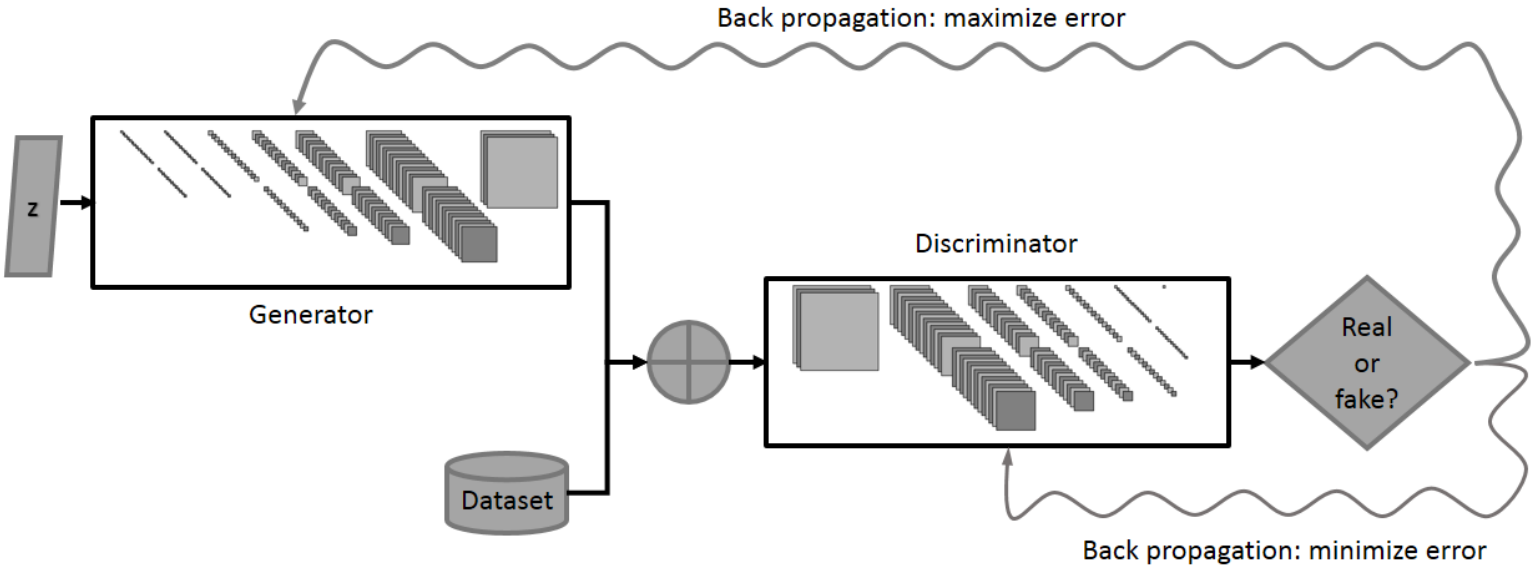


Image credits: https://devblogs.nvidia.com/photo-editing-generative-adversarial-networks-1/ &
https://research.nvidia.com/publication/2017-10_Progressive-Growing-of