Motivation

Why Bother? Why Solving Games Automatically?

- Game AIs for computer games (modern ones or board game adaptations).
- Purely for the sake of knowledge!
  E.g., can you always (force a) win in “Connect 4” when you start?
Why Bother? Why Solving Games Automatically?

- Games for computer games (modern ones or board game adaptations).
- Purely for the sake of knowledge!
  E.g., can you always (force a) win in “Connect 4” when you start?
- Because many real-world problems can be regarded a game!
  The other player(s) in the game might be other agents or surroundings.
  - Economics! Cf. game theory (look up: Nash Equilibrium and Prisoner’s Dilemma)
  - Robotics or Multi-Agent-Planning (though this is often cooperative, whereas we take a look at antagonistic games)

What are Games? Which Kinds Exist?

A game consists of a set of one or more players, a set of moves for the players, and a specification of payoffs (outcomes) for each combination of strategies (also called policy).

What kinds of restrictions can games have?
- Perfect information vs. imperfect information
  - (One-player games vs.) Two-player games vs. multi-player games
- Zero-sum games vs. non-zero-sum games
- Games with chance (randomness) vs. games without chance

What’s a Strategy?

A strategy defines a complete plan of action for a given player.

Given enough processing time an optimal strategy can be found for games of perfect information by enumerating paths of a game tree. However, in practice this can only be done for small games.
What are we Looking For?

What are we looking for?
- Game AI (strategy) vs. game theoretic outcome!
- Just because we have an AI that beats all humans, it doesn’t mean the game is solved!

What’s the game theoretic outcome?
- The outcome of the game assuming all players play rational.
- Rationality = optimization of expected reward.
- Outcome is known? → The respective game is “solved”.

MiniMax: How to Solve Small Games?

Using search to solve a game:
- If the game tree is “sufficiently small” we can search in it to find and extract a strategy.
- But we still need to do that efficiently!

Consider two players, MAX and MIN. MAX tries to maximize his/her own score, and player MIN tries to minimize it.

We assume that the players are rational, taking turns, and that the game is zero-sum.

MiniMax: The MiniMax Algorithm

The MiniMax algorithm allows each player to compute their optimal move on a game tree of alternating MAX and MIN nodes.

The value of a node is the payoff for a game that is played optimally from that node until the end of the game.

```
max-value(s)
  if state s is a leaf then
    return payoff(s)
  v := −∞
  forall successor states s' of s do
    v := max {v, min-value(s')}
  return v

min-value(s)
  if state s is a leaf then
    return payoff(s)
  v := ∞
  forall successor states s' of s do
    v := min {v, max-value(s')}
  return v
```
MiniMax: Example: Tic Tac Toe

MAX player plays X, MIN plays O. Outcomes (black boxes) are from the perspective of the MAX player (i.e., 1 is a win, -1 a loss, 0 a draw).

MAX

\[
\begin{array}{c}
\text{MAX} \\
\begin{array}{c}
\text{MAX} \\
\text{MIN} \\
\text{MAX} \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
1 \\
-1 \\
0 \\
\end{array}
\]

MIN

MIN

\[
\begin{array}{c}
\text{MIN} \\
\begin{array}{c}
\text{MIN} \\
\text{MAX} \\
\text{MIN} \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
1 \\
-1 \\
0 \\
\end{array}
\]

MAX

\[
\begin{array}{c}
\text{MAX} \\
\begin{array}{c}
\text{MIN} \\
\text{MAX} \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
1 \\
-1 \\
0 \\
\end{array}
\]
MiniMax: Example: Tic Tac Toe

MAX player plays X, MIN plays O. Outcomes (black boxes) are from the perspective of the MAX player (i.e., 1 is a win, -1 a loss, 0 a draw).

MAX player plays X, MIN plays O. Outcomes (black boxes) are from the perspective of the MAX player (i.e., 1 is a win, -1 a loss, 0 a draw).
MiniMax: Example: Tic Tac Toe

MAX player plays X, MIN plays O. Outcomes (black boxes) are from the perspective of the MAX player (i.e., 1 is a win, -1 a loss, 0 a draw).
Motivation: Example: Tic Tac Toe

MAX player plays X, MIN plays O. Outcomes (black boxes) are from the perspective of the MAX player (i.e., 1 is a win, -1 a loss, 0 a draw).

MiniMax: Space and Time Complexities

What is the runtime of MiniMax?
- Time: All nodes have to be visited! How many are there?
  - Assume each game ends after \( d \) moves (tree depth).
  - Each player has at most \( b \) moves (branching factor)
  - \( \rightarrow \) Runtime is in \( O(b^d) \) (exponential!)

What is the space requirement of MiniMax?
- We perform a depth-first search!
- So only the longest path needs to be stored.
  - \( \rightarrow \) Space is in \( O(b \cdot d) \) (linear)

MiniMax: Example: Tic Tac Toe

\[
\begin{array}{cccc}
\text{MAX} & & \text{MIN} & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{array}
\]

\( \alpha / \beta \) Pruning: Can we do better?

- MiniMax suffers from the problem that the number of game states it has to examine is always exponential in the number of moves.
- \( \alpha / \beta \) pruning is a method for reducing the number of nodes that need to be evaluated by only considering nodes that may be reached in game play.
- Alpha-beta pruning places bounds on the values appearing anywhere along a path:
  - \( \alpha \) is the best (highest) value found so far for MAX
  - \( \beta \) is the best (lowest) value found so far for MIN
- \( \alpha \) and \( \beta \) propagate down the game tree.
- \( v \) propagates up the game tree.

\[
\begin{align*}
\text{max-value} & (s, \alpha, \beta) \\
\text{if} & \text{ state } s \text{ is a leaf then} \\
& \text{return} \ \text{payoff}(s) \\
& v := -\infty \\
& \text{forall successor states } s' \text{ of } s \text{ do} \\
& \quad v := \max\{v, \text{min-value}(s', \alpha, \beta)\} \\
& \quad \text{if } v \geq \beta \text{ then} \\
& \quad \quad \text{return } v \\
& \quad \quad \alpha := \max\{\alpha, v\} \\
& \text{return } v \\
\end{align*}
\]

\[
\begin{align*}
\text{min-value} & (s, \alpha, \beta) \\
\text{if} & \text{ state } s \text{ is a leaf then} \\
& \text{return} \ \text{payoff}(s) \\
& v := \infty \\
& \text{forall successor states } s' \text{ of } s \text{ do} \\
& \quad v := \min\{v, \text{max-value}(s', \alpha, \beta)\} \\
& \quad \text{if } v \leq \alpha \text{ then} \\
& \quad \quad \text{return } v \\
& \quad \quad \beta := \min\{\beta, v\} \\
& \text{return } v \\
\end{align*}
\]
**α/β Pruning: Idea Behind Pruning: When and Why?**

Min chooses the left move with $v = 5$ so there is no point investigating the branch below.

---

**α/β Pruning: Example: Tic Tac Toe**

Start with $\alpha = -1$ (rather than $-\infty$) and $\beta = 1$ (rather than $\infty$)

---

**α/β Pruning: Example: Tic Tac Toe**

Start with $\alpha = -1$ (rather than $-\infty$) and $\beta = 1$ (rather than $\infty$)
**α/β Pruning: Example: Tic Tac Toe**

Start with $\alpha = -1$ (rather than $-\infty$) and $\beta = 1$ (rather than $\infty$)

- **MAX**
  - $v = -1$
  - $\alpha = -1, \beta = 1$

- **MIN**
  - $v = ???$
  - $\alpha = ???, \beta = ???$

(4)

because $(v = -1) \leq (\alpha = -1)$

- **MAX**
  - $v = ???$
  - $\alpha = ???, \beta = ???$

(5)

- **MIN**
  - $v = -1$
  - $\alpha = -1, \beta = 1$

- **MAX**
  - $v = ???$
  - $\alpha = ???, \beta = ???$

(6)

- **MIN**
  - $v = ???$
  - $\alpha = ???, \beta = ???$

(7)

because $(v = 1) \geq (\beta = 1)
Motivation

Games?

Solving Small Games

Games with Chance

Solving Large Games

Game AI Success Story

α/β Pruning: Space and Time Complexities

What is the runtime (and space requirements) of α/β pruning?

- In the worst case: identical to MiniMax! If nothing can be pruned.
- On average: Complexities omitted. (Due to lack of time.)
- This can happen depending on the order in which edges are traversed/payoffs are discovered.
- In practice, it is very unlikely that no pruning occurs, so always choose α/β pruning over MiniMax!

---

Motivation

Games?

Solving Small Games

Games with Chance

Solving Large Games

Game AI Success Story

α/β Pruning: Assignment: AI for Cublino

- Perfect information, deterministic, zero-sum
- Branching factor? (Much higher than in, e.g., Connect 4)
- Game horizon? (I guess a bit higher than in Connect 4)

---

Motivation

Games?

Solving Small Games

Games with Chance

Solving Large Games

Game AI Success Story

How to Deal with Randomness?

- A random decision can be regarded as the move of yet another player!
- Certainly that’s not another MAX player! I.e., the “environment” (the random decision) will not always play in our favor!
- But what is it, then?
  - Another MIN player? (Too pessimistic...)
  - If we want to play rational, we maximize the expectation!
    \[
    \text{value}(s) = \sum_{\text{successor states } s'} P(s') \cdot \text{value}(s')
    \]
**Motivation**

Games?

Solving Small Games

Games with Chance

Solving Large Games

Game AI Success Story

---

**Illustration For a 2-Player Game With Throwing Two Dice, Counting Their Sum**

- **Max**
  - move 1 followed by dice throw
  - move n followed by dice throw

- **RAND**
  - $P(\text{sum} = 2) = \frac{1}{36}$
  - $P(\text{sum} = 7) = \frac{6}{36}$
  - $P(\text{sum} = 12) = \frac{1}{36}$

- **MIN**
  - move 1 followed by dice throw
  - move k followed by dice throw

- **RAND**
  - $P(\text{sum} = 2) = \frac{1}{36}$
  - $P(\text{sum} = 7) = \frac{6}{36}$
  - $P(\text{sum} = 12) = \frac{1}{36}$

- **Max**

---

**Solving Large Games**

---

**The “Size” of Games**

When is using MiniMax and $\alpha/\beta$ Pruning still feasible?

- Recall that the complexity of MiniMax (and $\alpha/\beta$) is exponential! I.e., in $O(b^d)$, with
  - $b$, the branching factor (available moves per state)
  - $d$, the depth (number of moves until game ends)
- For some games that is simply too large!
- So, let's take a look at some examples...

---

**The “Size” of Games: Tic Tac Toe**

Examples for (estimated) number of reachable (game) states:

(Source: [https://en.wikipedia.org/wiki/Game_complexity](https://en.wikipedia.org/wiki/Game_complexity))

- Rough maximum: $3^9 = 19,683$ (including invalid states)
- Actual maximum: 5,478
- Maximum after duplicating symmetries: 765
- There are still 26,830 possible games!
  (For those states with eliminated duplicates.)
  What's a “game”?
  A path in the MiniMax tree!
The "Size" of Games: Connect 4

Examples for (estimated) number of reachable (game) states:
(Source: https://en.wikipedia.org/wiki/Connect_Four)

- Rough maximum: $3^7 < 1.1 \cdot 10^{20}$ (including invalid states)
- Actual maximum: $4,531,985,219,092 \approx 4.5 \cdot 10^{12}$ (still including symmetries)
- First solved, independently, by James Dow Allen (October 1, 1988), and Victor Allis (October 16, 1988).
- Note that today it can also be solved using $\alpha/\beta$ pruning!

The "Size" of Games: Blokus

Examples for (estimated) number of reachable (game) states:
(Source: by Stephen Gould, previous year(s))

- approx. 58 moves, not all symmetries eliminated
- approx. 2·58 moves, symmetries as before
- approx. 116 moves, symmetries as before
- 21 pcs: 58 moves
- 58·116 = 6,728 moves
- 58·116·116 = 780,448 moves
- 58·116·116·58 = 45,265,984 \approx 4.5 \cdot 10^7 moves
- 20 pcs: ??? moves
The “Size” of Games: Chess

Examples for (estimated) number of reachable (game) states:
(Source: https://en.wikipedia.org/wiki/Shannon_number)
- Some maximum: $5 \cdot 10^{52}$
- Lower limit on game tree size: $10^{123}$
- More conservative estimate on lower limit of game tree size, eliminating obvious bad moves: $10^{40}$

How to deal with large games?

So, what to do for (too) large games?
- Don’t compute the entire game tree!
- Stop at certain nodes and estimate their payoff! But how?
  - hand-crafted heuristics

Title: Artificial Intelligence: A Modern Approach (3rd Ed.)
Authors: Stuart Russel and Peter Norvig
URL: https://aima.cs.berkeley.edu/

Estimate per piece:
- pawn: 1 pt
- knight/bishop: 3 pts
- rook: 5 pts
- queen: 9 pts

Estimate: Black: 7 pts versus White: 6 pts
→ Black leading! (Only very slightly.)

The “Size” of Games: Go

Examples for (estimated) number of reachable (game) states:
(Source: https://en.wikipedia.org/wiki/Shannon_number)
- Legal positions: $2.08168199382 \cdot 10^{170}$
- Lower limit on number of games: $10^{1048}$
- Upper limit on number of games: $10^{1071}$

How to deal with large games?

So, what to do for (too) large games?
- Don’t compute the entire game tree!
- Stop at certain nodes and estimate their payoff! But how?
  - hand-crafted heuristics
  - learned heuristics

Machine learning techniques are often used to find a good static evaluation function based on a linear combination of features:

$$\hat{v}(s) = w_1 f_1(s) + \cdots + w_n f_n(s)$$

Note the similarity to chess!
- $w_1 = 1$, $f_1(s) =$ number of pawns in $s$
- $w_2 = 3$, $f_2(s) =$ number of knights/bishops in $s$
- ...
How to deal with large games?

So, what to do for (too) large games?
- Don’t compute the entire game tree!
- Stop at certain nodes and estimate their payoff! But how?
  - hand-crafted heuristics
  - learned heuristics
  - simulate a game, use the outcome as estimate

Monte-Carlo Tree Search is a well-known algorithm exploiting this idea. It works in four phases:
- Selection (select a non-terminal leaf based on current strategy)
- Expansion (expand the selected node)
- Simulation (play a random game to the end)
- Backpropagation (use the outcome to update strategy)

Interested? See, e.g.,
https://www.youtube.com/watch?v=UXW2yZndl7U
(15:30, lecture by Dr. John Levine from Univ. of Strathclyde)

When to use heuristics?

- In standard MiniMax or alpha/beta pruning, we make a terminal test to obtain the payoff, or continue expanding. With heuristics, we instead make a cut-off test to check whether we should stop expansion and estimate the payoff of the current node.
- What about using a fixed depth as cut-off test? → Suffers from the horizon problem:

Mile Stones in AI Game Playing

1959 Arthur Samuel develops Checkers playing program
1997 IBM’s Deep Blue chess machine beats Garry Kasparov
2007 Checkers solved by University of Alberta
2011 IBM’s Watson wins Jeopardy! requiring natural language understanding
2015 Deep reinforcement learning algorithms learn to play Atari arcade games from scratch
2016 Google DeepMind’s AlphaGo beats Lee Sedol, Korea
2017 AlphaZero learns Go, Chess, and Shogi from scratch (and beats AlphaGo)
Sources / Copyright

Picture is public domain
https://commons.wikimedia.org/wiki/File:Puissance4_01.svg

Photo by A. Yobi Blumberg on Unsplash
https://unsplash.com/photos/22W19M-YsDE

Photo by Emile Perron on Unsplash
https://unsplash.com/photos/_jXn-gNzuGo

Photo by Michał Parzuchowski on Unsplash
https://unsplash.com/photos/oT-XbATcoTQ

Photo by Macau Photo Agency on Unsplash
https://unsplash.com/photos/as5EwdBWKqk

Unsplash pictures are free to use, see https://unsplash.com/license

Taken from https://www.brettspiele-report.de/cublino/
(We argue for fair use.)

Considered fair use for and by Wikipedia (https://en.wikipedia.org/wiki/Blokus). We also consider it fair dealing (Australian equivalent to US’s fair use) for illustrating the game in this lecture.

Picture is public domain

By Stuart Russel and Peter Norvig
from their book Artificial Intelligence: A Modern Approach (3rd Ed.)
Available freely for teaching on https://aima.cs.berkeley.edu/

Picture is public domain
https://nl.wikipedia.org/wiki/Checkers

Taken from the YouTube video https://www.youtube.com/watch?v=67w0QCMr9EY by channel wikipedia tts, licensed under CC BY (https://creativecommons.org/licenses/by/3.0/legalcode).

Considered fair use for and by Wikipedia (https://en.wikipedia.org/wiki/Space_Invaders). We also consider it fair dealing (Australian equivalent to US’s fair use) for illustrating the game in this lecture.