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?

- 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?
- Because many real-world problems can be regarded a game!
  The other player(s) in the game might be other agents or surroundings.
  - Robotics or Multi-Agent-Planning (though this is often cooperative, whereas we take a look at antagonistic games)
  - Economics! Cf. game theory (look up: Nash Equilibrium and Prisoner's Dilemma)
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 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”.

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.
Solving Small Games

Games with Chance

Solving Large Games

Defeating Dragons with AI

Game AI Success Story

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.

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

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

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).
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).
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).
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).

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)
  - 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.
- Space is in $O(b \cdot d)$ (linear)
α/β 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.
- α/β 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.
- α/β pruning places bounds on the values appearing anywhere along a path:
  - α is the best (highest) value found so far for MAX
  - β is the best (lowest) value found so far for MIN
- α and β propagate down the game tree. v propagates up the game tree.

Motivation — Idea Behind Pruning: When and Why?

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

Keep in mind:
- α is the best value found so far for MAX, initialize with $-\infty$.
- β is the best value found so far for MIN, initialize with $\infty$.

α/β Pruning — Idea Behind Pruning: When and Why?

α/β Pruning — The MiniMax Algorithm Extended By α/β Pruning

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

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

Max

Pruning — Idea Behind Pruning: When and Why?

Pruning — Can we do better?

Pruning — Example: Tic Tac Toe

v = 5

α = $-\infty$

β = $\infty$

...
Game AI Success Story

Solving Large Games

Solving Small Games

Defeating Dragons with AI

Motivation

Pruning — Example: Tic Tac Toe

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

$\alpha = -1, \beta = -1$ (4)

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

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

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$

$\alpha = -1, \beta = -1$
α/β Pruning — Example: Tic Tac Toe

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

Max

Min

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

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

Pascal Bercher

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)

- 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: $3^{7^6} < 1.110^{20}$ (including invalid states)
- Actual maximum: 4,531,985,219,092 ≈ $4.5 \cdot 10^{12}$ (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))

- 20 pcs ≈ 58 moves
- 58 · 116 = 6,728 moves
- 58 · 116 · 116 = 780,448 moves
- 58 · 116 · 116 · 58 ≈ 45,265,984 ≈ 4.5 · 10^7 moves

Approximately 58 moves, not all symmetries eliminated

The "Size" of Games: Chess

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

- Some maximum: 5 · 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
  - 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=UXW2yZd17U (15:30, lecture by Dr. John Levine from Univ. of Strathclyde)
Motivation

Games?
Solving Small Games
Games with Chance
Solving Large Games
Defeating Dragons with AI
Game AI Success Story

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:

![Black to move](image)

White can promote a pawn into a queen on his next move! So the cut-off test should be negative in this state.

The Assignment: Tsuro of the Seas

Tsuro of the Seas: Ultra-short introduction

![Tsuro of the Seas](image)

Figure: YouTube video: https://www.youtube.com/watch?v=ziQS8rcT5EA (we just take a glance from 5:04 to 5:58) Code: z-i-Q-S-8-r-c-T-5-E-A

Regarding the game rules: Please stick to the ones officially provided by Steve Blackburn!

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)