Learning Adversarial Search algorithms

The Art of Unlosable Tic-Tac-Toe

Eric Han

June 7, 2024

Introduction

Experience

My *Singaporean* educational journey to CS, R&D:

- [2008] **Secondary School** — Informal learning; scripting, games
  - *Interest in computing*: why & how computers work
- [2010] **Pioneer JC** — H2 Computing
  - *Interest in research*: A*STAR IHPC Quest 2009 (Bronze) - K-Means
- [2018] **B.Com. NUS** — Com. Sci w Honors
  - A*STAR Scholarship: Internships working on R&D projects
- [2024] **PhD. NUS** — AI/ML tackling scaling and robustness
  - First-author publications in AAAI, ICML.
  - *Interest in teaching*: Fulfilling

Working experiences:

- [2018-2024] **Teaching Assistant/Graduate Tutor, NUS** — Teach UG
- [B.Com.] **Research Intern, A*STAR IHPC** — ML Platform, Rec. Sys.

Expertise

Support, teach ($>$ 500 contact hours), grade, manage/mentor tutors for:

- AI/Machine Learning
  - **CS2109s** Introduction to AI and Machine Learning
  - **CS3243** Introduction to Artificial Intelligence
- Software Engineering
  - **CS3217** Software Engineering on Modern Application Platforms
  - **CS3203** Software Engineering Project
  - **CS2030/CS2030S** Programming Methodology II

Skilled with Linux (also administration), Windows, macOS:

- **Programming Languages** — Python, C++, Java, Mojo...
- **Databases** — Firebase, SQL...
- **Typesetting / Presentation Tools** — LaTex, Markdown (This slides!)...
- **Tools/Platforms** — Git, Mlflow, Plotly, Slurm, GCP...

Teaching Philosophy

Effective learning is driven by an *innate desire* to learn the subject rather than *need*:

1. **Creating a relaxed and safe environment** — Informal, casual, personal.
2. **Engaging students to facilitate learning in and after class** — Telegram, buddy
3. **Creating equal opportunities for all students to learn** — Reaching out

Teaching Excellence (Tutorials/Recitation)

	2109	**2109**	**3243**	**3243**	**3217**	**3203**	**3203**	**3203**	**3203**	**3203**
Score	4.8	4.6	4.8	4.5	3.8	4.6	4.4	4.8	4.1	3.3
Resp.	36	13	25	39	6	13	16	18	20	3
Nom.	47%	30%	32%	31%	0%	61%	31%	61%	10%	33%

Plans

For the teaching position, I am interested to

- Focus on improving teaching quality
- Contribute anywhere there are needs; Computing, Teaching & Research
- Curriculum development/improvement
- Casual research
  - Mentor for Undergraduate Research / FYP
- Involved in consultancy/policy

Mini-Lecture

Recap on environment properties

- **Fully / Partially Observable**: Can the agent see?
- **Single / Multi-Agent**: How many agents?
- **Deterministic / Stochastic**: Is there randomness in transition?
- **Episodic / Sequential**: Is there dependence on previous action?
- **Static / Dynamic**: Can the environment change while the agent is thinking?
- **Discrete / Continuous**: Discretized or varying continuously?

Recap on formulation

Un/Informed Search (Path): BFS, UCS, DFS, GBFS, A*

- State space
- Initial state
- Final state
- Action
- Transition

Local Search (Goal): Hill Climbing, Sim. Annealing, Beam, Genetic

- Initial state
- Transition
- Heuristic/Stopping criteria

Motivation: How can we win?

Ingredients needed to formulate a problem:

- **Initial state**: Starting configuration (representation)
- **Players**: Decision-makers within the game (2 players)
- **Actions**: Potential moves that the player can make
- **Transition**: Result of a move from a state
- **Terminal/Leaf test**: Checks if the game is over
- **Utility**: Reward for a terminal state and player

Tic-Tac-Toe

2P childhood game where (P_O, P_X) players take turns drawing their symbols on a 3x3 grid. The winner is the first player to get 3 of his/her symbol in a row, col. or diag.

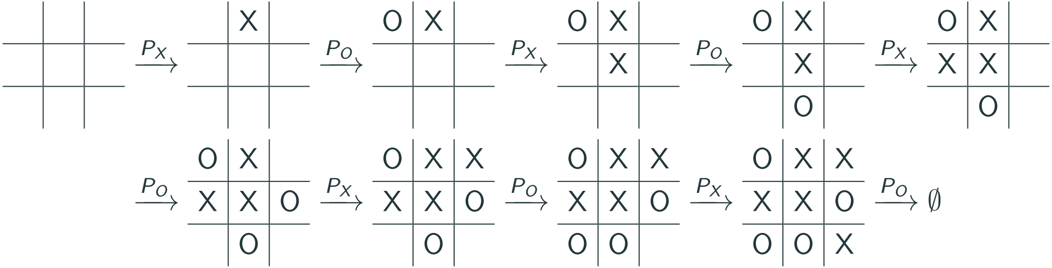
2P childhood game where (P_O, P_X) players take turns drawing their symbols on a 3x3 grid. The winner is the first player to get 3 of his/her symbol in a row, col. or diag.

$$\begin{array}{c|c|c} & & \\ \hline & & \\ \hline & & \end{array} \xrightarrow{P_X} \begin{array}{c|c|c} & X & \\ \hline & & \\ \hline & & \end{array} \xrightarrow{P_O} \begin{array}{c|c|c} O & X & \\ \hline & & \\ \hline & & \end{array} \xrightarrow{P_X} \begin{array}{c|c|c} O & X & \\ \hline & X & \\ \hline & & \end{array} \xrightarrow{P_O} \begin{array}{c|c|c} O & X & \\ \hline & X & \\ \hline & O & \end{array} \xrightarrow{P_X} \begin{array}{c|c|c} O & X & \\ \hline X & X & \\ \hline & O & \end{array}$$

$$\xrightarrow{P_O} \begin{array}{c|c|c} O & X & \\ \hline X & X & O \\ \hline & O & \end{array} \xrightarrow{P_X} \begin{array}{c|c|c} O & X & X \\ \hline X & X & O \\ \hline & O & \end{array} \xrightarrow{P_O} \begin{array}{c|c|c} O & X & X \\ \hline X & X & O \\ \hline O & O & \end{array} \xrightarrow{P_X} \begin{array}{c|c|c} O & X & X \\ \hline X & X & O \\ \hline O & O & X \end{array} \xrightarrow{P_O} \emptyset$$

Tic-Tac-Toe

2P childhood game where (P_O, P_X) players take turns drawing their symbols on a 3x3 grid. The winner is the first player to get 3 of his/her symbol in a row, col. or diag.



Recap — Environment Properties

Fully Observable, 2 Agent, Deterministic, Sequential, Static, Discrete

Modeling Tic-Tac-Toe [Discussion]

$$\begin{array}{c|c|c} & & \\ \hline & & \\ \hline & & \end{array} \xrightarrow{P_X} \begin{array}{c|c|c} & X & \\ \hline & & \\ \hline & & \end{array} \xrightarrow{P_O} \begin{array}{c|c|c} O & X & \\ \hline & & \\ \hline & & \end{array} \xrightarrow{P_X} \cdots \begin{array}{c|c|c} O & X & X \\ \hline X & X & O \\ \hline O & O & \end{array} \xrightarrow{P_X} \begin{array}{c|c|c} O & X & X \\ \hline X & X & O \\ \hline O & O & X \end{array}$$

2P childhood game where (P_O, P_X) players take turns drawing their symbols on a 3x3 grid. The winner is the first player to get 3 of his/her symbol in a row, col. or diag.

- **Initial state**:
- **Players**:
- **Actions**:
- **Transition**:
- **Terminal/Leaf test**:
- **Utility**:

Modeling Tic-Tac-Toe

$$S_0 = \begin{array}{c|c|c} 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \end{array} \xrightarrow{a_0=(1,X)} \begin{array}{c|c|c} & X & \\ \hline & & \\ \hline & & \end{array} \xrightarrow{\cdots} \cdots \begin{array}{c|c|c} O & X & X \\ \hline X & X & O \\ \hline O & O & \end{array} \xrightarrow{a_8=(8,X)} \begin{array}{c|c|c} O & X & X \\ \hline X & X & O \\ \hline O & O & X \end{array} = S_9$$

- **Initial state**: S_0, 1D array of $[\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset]$; possible elements: $O, X, \emptyset$
- **Players**: P_X-max, P_O-min
- ***i*-th Actions**: $a_i = (c_i, y) : c_i \in [0, 8]$ where $S_i[c_i] = \emptyset$, symbol $y \in X, O$
- **Transition**: $T(S_i, a_i) = S_{i+1}$, where $S_{i+1}[j] = \begin{cases} y & \text{if } j = c_i \\ S_i[j] & \text{otherwise} \end{cases}$
- **Terminal/Leaf test**: Row, col. or diag having same symbols or no moves
- **Utility**: $U(S_i, p)$ is 0 if draw, 1 if p wins, -1 if p loses

Modeling Tic-Tac-Toe

- **Initial state**: S_0, 1D array of $[\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset]$; possible elements: $O, X, \emptyset$
- **Players**: P_X-max, P_O-min
- i-**th Actions**: $a_i = (c_i, y) : c_i \in [0, 8]$ where $S_i[c_i] = \emptyset$, symbol $y \in X, O$
- **Transition**: $T(S_i, a_i) = S_{i+1}$, where $S_{i+1}[j] = \begin{cases} y & \text{if } j = c_i \\ S_i[j] & \text{otherwise} \end{cases}$
- **Terminal/Leaf test**: $\boxed{\text{Row, col. or diag having same symbols or no moves}}$
- **Utility**: $U(S_i, p)$ is 0 if draw, 1 if p wins, -1 if p loses

FAQ: Can I describe and not write math?

Yes, but it must be **clear**; ie. Able to translate into code without additional assumptions; you should (at min) describe how the state is represented.

Modeling Tic-Tac-Toe in Python

- **Initial state**: S_0, 1D array of $[\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset]$; possible elements: $O, X, \emptyset$
- ***i*-th Actions**: $a_i = (c_i, y) : c_i \in [0, 8]$ where $S_i[c_i] = \emptyset$, symbol $y \in X, O$

```
...
class TicTacToe(object):
  def __init__(self, Si=[ E ] * 9):
    self.Si = Si
  def actions(self):
    e_cis = [ ci for ci, Si_ci in enumerate(self.Si) if Si_ci == E ]
    return [ (ci, y) for y in SYMBOLS for ci in e_cis ]
```

Code will be made available: https://eric-han.com/teaching/demo/ttt.py

Zero-sum game

Zero-sum game is a game where one player gain is equals to another's loss, where the total utility of the game is the **same/constant** (ie. no improvement).

Tic-Tac-Toe is zero-sum

- If P_X wins P_O loses: $\sum U = 1 - 1 = 0$
- If P_O wins P_X loses: $\sum U = 1 - 1 = 0$
- If P_O, P_X draws: $\sum U = 0 + 0 = 0$

So, for Tic-Tac-Toe: $U(S_i, X) = -U(S_i, O)$

Intuition: If you played enough, you notice you keep getting draws.

Question

Can we come up with an algorithm to play Tic-Tac-Toe?

Tic-Tac-Toe game-tree

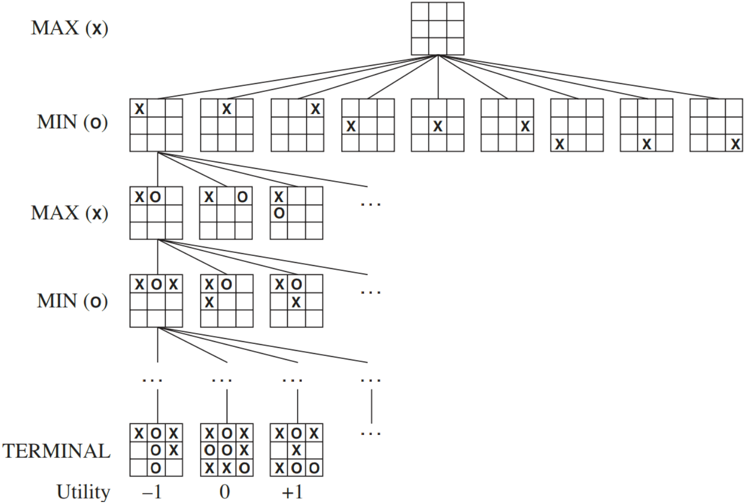


Figure 1: Game-tree (R&N 3rd Ed) — Initial, Players, Actions, Transition, Terminal, Utility

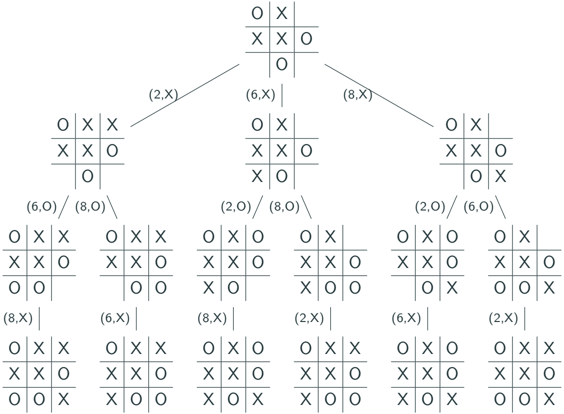
Intuition: Simulate the game until the end with an imaginary optimal opponent.

$$
\begin{array}{c|c|c} & & \\ \hline & & \\ \hline & & \end{array}
\xrightarrow{P_X}
\begin{array}{c|c|c} & \text{X} & \\ \hline & & \\ \hline & & \end{array}
\xrightarrow{P_O}
\begin{array}{c|c|c} \text{O} & \text{X} & \\ \hline & & \\ \hline & & \end{array}
\xrightarrow{P_X}
\begin{array}{c|c|c} \text{O} & \text{X} & \\ \hline & \text{X} & \\ \hline & & \end{array}
\xrightarrow{P_O}
\begin{array}{c|c|c} \text{O} & \text{X} & \\ \hline & \text{X} & \\ \hline & \text{O} & \end{array}
\xrightarrow{P_X}
\begin{array}{c|c|c} \text{O} & \text{X} & \\ \hline \text{X} & \text{X} & \\ \hline & \text{O} & \end{array}
$$

$$
\xrightarrow{P_O}
\boxed{
\begin{array}{c|c|c} \text{O} & \text{X} & \\ \hline \text{X} & \text{X} & \text{O} \\ \hline & \text{O} & \end{array}
\xrightarrow{P_X}
\begin{array}{c|c|c} \text{O} & \text{X} & \text{X} \\ \hline \text{X} & \text{X} & \text{O} \\ \hline & \text{O} & \end{array}
\xrightarrow{P_O}
\begin{array}{c|c|c} \text{O} & \text{X} & \text{X} \\ \hline \text{X} & \text{X} & \text{O} \\ \hline \text{O} & \text{O} & \end{array}
\xrightarrow{P_X}
\begin{array}{c|c|c} \text{O} & \text{X} & \text{X} \\ \hline \text{X} & \text{X} & \text{O} \\ \hline \text{O} & \text{O} & \text{X} \end{array}
\xrightarrow{P_O} \emptyset
}
$$

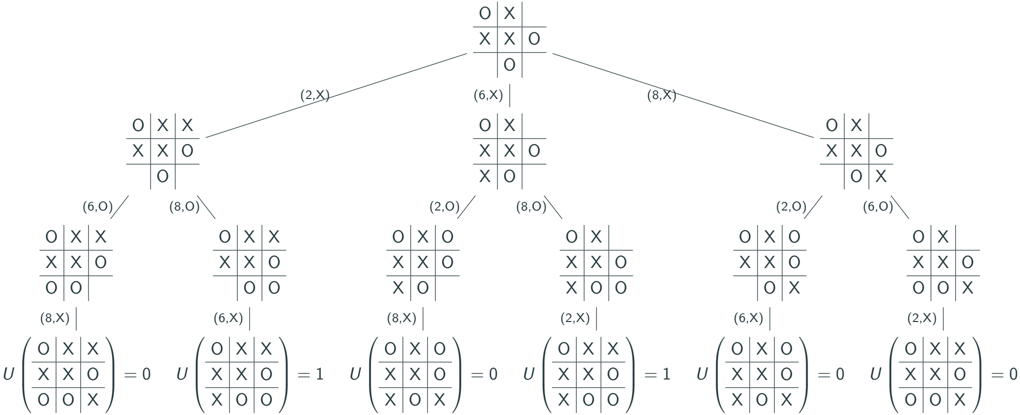
I am max player P_X, trying to find the best move at $\begin{array}{c|c|c} \text{O} & \text{X} & \\ \hline \text{X} & \text{X} & \text{O} \\ \hline & \text{O} & \end{array}$

Intuition: Simulate the game until the end with an imaginary opponent.



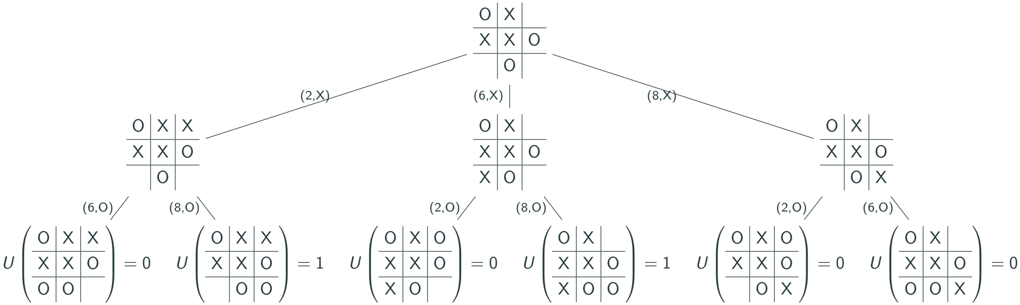
We can fill in the utility values for the leaf nodes!

Utility for X: $U(S_i, X)$ is 0 if draw, 1 if X wins, -1 if X loses



We know we want the best action later, so we choose the best action (max) there!

For the best action chosen, we inherit its corresponding value!



We don't know how the P_O will play this move, so

- Assume that P_O wants to win and plays optimally like me.
- We imagine that P_O chooses the best action (min) there!

Minimax algorithm

Intuition: Simulate the game until the end with an imaginary *optimal* opponent.

$$
\begin{array}{c|c|c}
\text{O} & \text{X} & \\ \hline
\text{X} & \text{X} & \text{O} \\ \hline
 & \text{O} &
\end{array}
$$

(2,X) (6,X) (8,X)

$$
U\left(\begin{array}{c|c|c}
\text{O} & \text{X} & \text{X} \\ \hline
\text{X} & \text{X} & \text{O} \\ \hline
 & \text{O} &
\end{array}\right) = 0 \quad
U\left(\begin{array}{c|c|c}
\text{O} & \text{X} & \\ \hline
\text{X} & \text{X} & \text{O} \\ \hline
\text{X} & \text{O} &
\end{array}\right) = 0 \quad
U\left(\begin{array}{c|c|c}
\text{O} & \text{X} & \\ \hline
\text{X} & \text{X} & \text{O} \\ \hline
 & \text{O} & \text{X}
\end{array}\right) = 0
$$

Now I can just pick the move that is the best (max value):

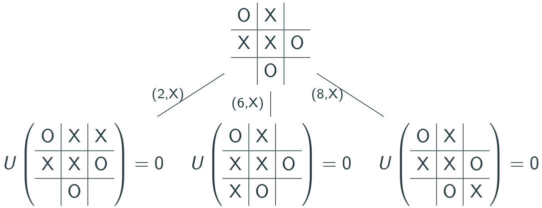
- All 3 moves would, at worse-case, end up in draws.

Minimax algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  return arg max $_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

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

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

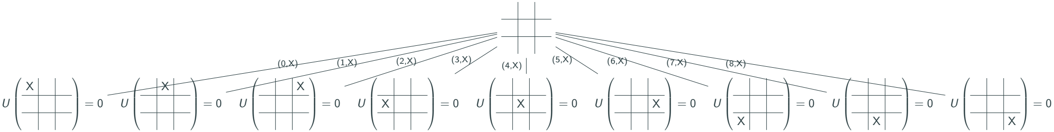


Conceptually, at every level (min or max),

- Evaluate all of the successor's values
- Pick the action with the best value

But we evaluate it in a DFS fashion.

Minimax Tic-Tac-Toe example



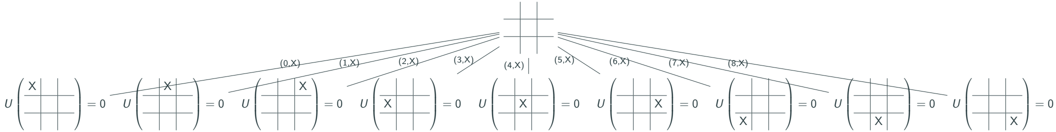
Computing it out for all possible actions for S_0:

- All successor states have values of $U(.) = 0$
- All actions would lead to draws
- No matter what you play, Tic-Tac-Toe is Unlosable

Intuition from Primary Sch: How can that be?

Center is better, corners are next best and then the rest.

Minimax Tic-Tac-Toe example



Computing it out for all possible actions for S_0:

- All successor states have values of $U(.) = 0$
- All actions would lead to draws
- No matter what you play, Tic-Tac-Toe is Unlosable
- Assuming *optimal* opponent

Minimax analysis

With b branching factor and m max depth,

- Time:
  - $O(b^m)$ (From DFS)
- Space:
  - $O(bm)$ (From DFS)
- Completeness:
  - Yes if finite (From DFS)
- Optimality:
  - Yes on $U(.)$, assuming *optimal* opponent

Minimax analysis

With b branching factor and m max depth,

- Time:
  - $O(b^m)$ (From DFS)
- Space:
  - $O(bm)$ (From DFS)
- Completeness:
  - Yes if finite (From DFS)
- Optimality:
  - Yes on $U(.)$, assuming *optimal* opponent

Can we do better for Tic-Tac-Toe?

We know that an action cannot be reused!

Minimax Tic-Tac-Toe analysis

With b branching factor and m max depth,

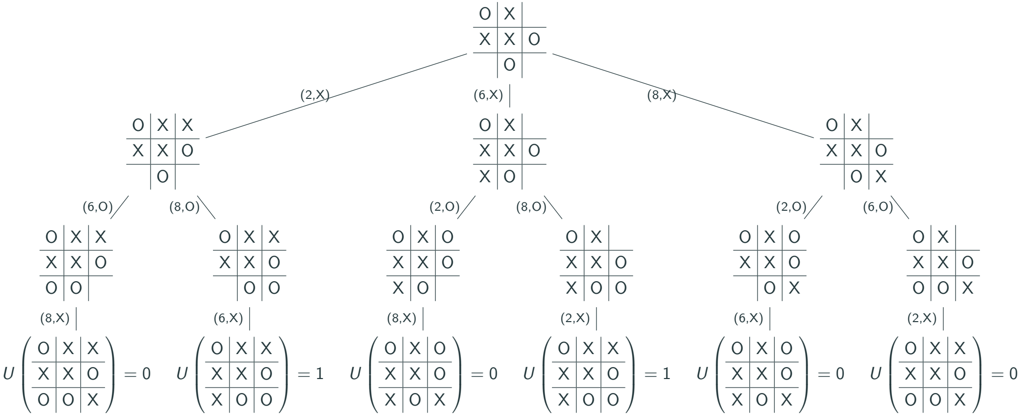
- States:
  - 1D array of size 9, with possible $O, X, \emptyset$ elements — $O(3^9)$
  - Can be reduced further by removing illegal states
- Time:
  - $O(b^m), m = 9$
  - 9! terminal nodes, so $\sum_i^9 i!$ nodes to explore
- Space:
  - $O(bm)$
- Completeness:
  - Yes if finite (From DFS)
- Optimality:
  - Yes on $U(.)$, assuming *optimal* opponent

Minimax limitations

What happens when:

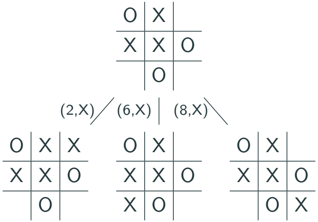
- Large game trees — Chess: $N = 10^{40}$, $b \sim 35$, Go: $N = 2.1 \times 10^{170}$, $b \sim 250$
  - Borrowing from IDS — Max depth/Cutoff
  - Is it necessary to evaluate everything? — Alpha-Beta Pruning
  - … (More during tutorials)
- Non-optimal agent or we have randomness — Games with Dice, 2048, etc…
  - Use statistics to capture randomness — Expectimax

Intuition: Stop at a time or depth limit.



Cutoff

Intuition: Stop at a time or depth limit.



Previously, we can *always* propagate the $U(.)$ values, but not now:

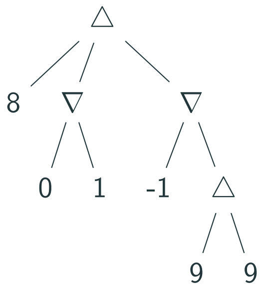
- Heuristic[1] value for non-terminal states: `UTILITY(state)`
- Add test for cutoff condition: `TERMINAL-TEST(state)`

[1] Must design carefully (More during tutorials)

Alpha-Beta Pruning algorithm

Intuition: Skip if there is *already* a better move found, track using bounds.

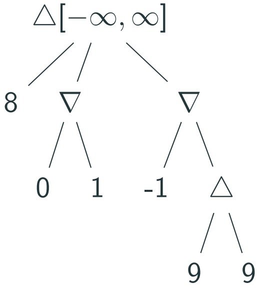
- Assign bounds to each of the nodes
- Starting with $[-\infty, \infty]$
- Go from left to right



Commonly used notation — $\triangle$: Max, ∇: Min

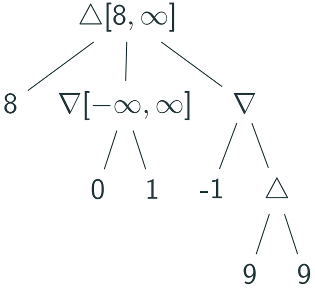
Alpha-Beta Pruning algorithm

We start by initializing $[\infty, \infty]$.



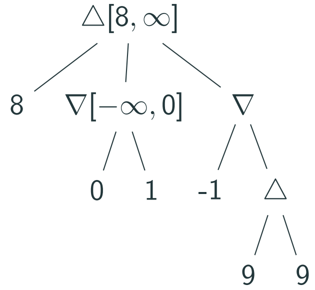
Alpha-Beta Pruning algorithm

Then we discover the leaf node 8, we update its parent max node to at least 8.



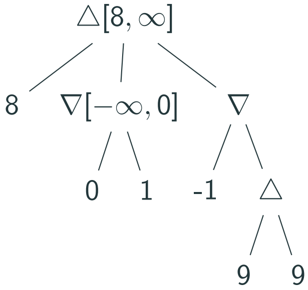
Alpha-Beta Pruning algorithm

Now we discover leaf node 0, we update its parent min node to at most 0.



Alpha-Beta Pruning algorithm

Now we discover leaf node 0, we update its parent min node to at most 0.

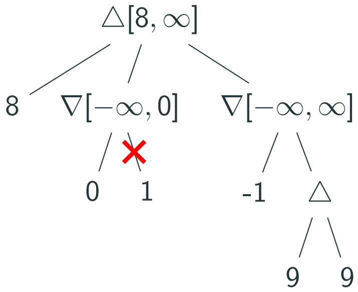


Notice that, from root:

- $\triangle[8, \infty]$ would always prefer the node $8 > [-\infty, 0]$:
- So, $\nabla[-\infty, 0]$ is never explored.
- We will not need to evaluate 1 at all $>$ Pune!

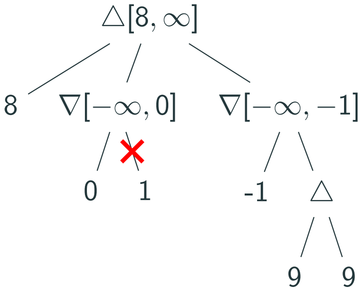
Alpha-Beta Pruning algorithm

$\triangle[8, \infty]$ would always prefer the node $8 > [-\infty, 0]$, so we prune leaf node 1.



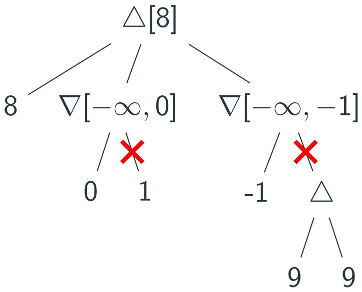
Alpha-Beta Pruning algorithm

Now we discover leaf node -1, we update its parent min node to at most -1.



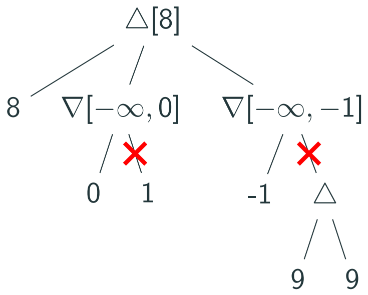
Alpha-Beta Pruning algorithm

$\triangle[8, \infty]$ would always prefer the node $8 > [-\infty, -1]$, so we prune the rest.



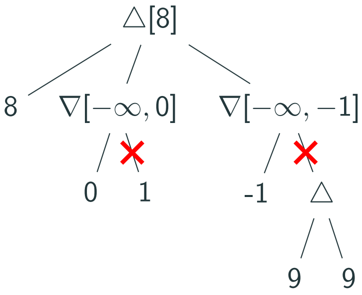
Alpha-Beta Pruning algorithm

- Bounds needs to be checked along the path
- Can we summarize all of the bounds searched so far?
  - α: Minimum score that the Max player knows it can guarantee
  - β: Maximum score that the Min player already knows it can guarantee



Alpha-Beta Pruning algorithm

- α: Minimum score that the Max player knows it can guarantee
- β: Maximum score that the Min player already knows it can guarantee



Instead of checking if root node would always prefer the node $8 > [-\infty, 0]$:

- $\alpha = 8, \beta = \infty$ is passed in from parent: allowing you to reason along the path.
- Nodes after 0, ie. node 1, is pruned because v less eq α, ie. $0 \leq 8$

Alpha-Beta Pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, -∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← -∞
  for each a in ACTIONS(state) do
      v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
  return v

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

Conceptually, we use

- α: Minimum score that the Max player knows it can guarantee
- β: Maximum score that the Min player already knows it can guarantee

to reason on the bounds on nodes to decide if we can prune.

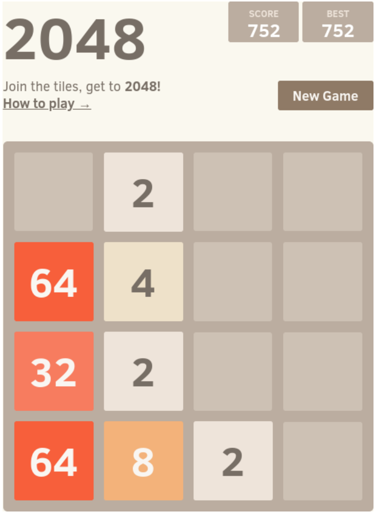
(Tracing during tutorials)

Alpha-Beta analysis

Pruning does not affect result:

- Time:
  - Worst Case: Same as Minimax
  - Best Case: $O(b^{\frac{m}{2}})$ for perfect ordering (More during tutorials)
    - Save on static evaluation (evaluating the values) and move generation (generating nodes).
    - Can explore twice as deep on the best case.
- Space: Same as Minimax
- Completeness: Same as Minimax
- Optimality: Same as Minimax

2048



Play 2048: https://play2048.co/

- Player plays Up, Down, Left or Right
- The game will randomly spawn either 2 or 4 in one of the empty cells
- If there is no empty cells, you lose.
- Tiles will fall in that direction
- Tiles with same value will be merged

Modeling 2048 [Discussion/Exercise]

- **Initial state**:
- **Players**:
- **Actions**:
- **Transition**:
- **Terminal/Leaf test**:
- **Utility**:

How to model the randomness?

[2] See R&N Section 5.5

Modeling 2048 [Discussion/Exercise]

- **Initial state**:
- **Players**:
- **Actions**:
- **Transition**:
- **Terminal/Leaf test**:
- **Utility**:

How to model the randomness?

- Model it adversarially as a min player using minimax
- Model it using expectation $\sum_r P(r) * U(r)$ — Expectimax[2]
  - Chance nodes, representing each possible outcome

[2] See R&N Section 5.5

2048: Minimax vs Expectimax

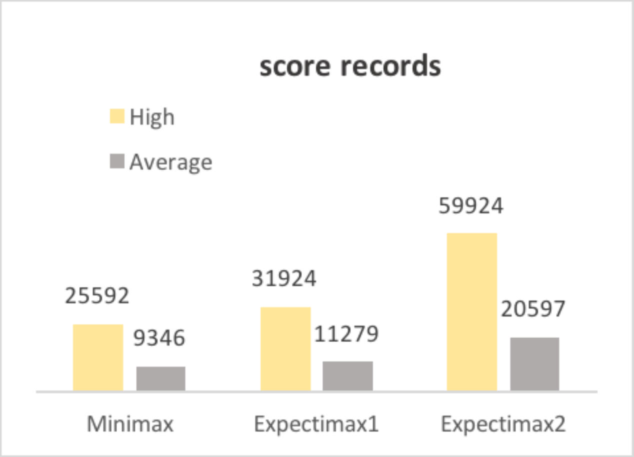


Figure 2: Expectimax1,2 are using different heuristics[3]

[3] https://cs229.stanford.edu/proj2016/report/NieHouAn-AIPlays2048-report.pdf

Tic-Tac-Toe: Minimax vs Expectimax

Max-Expectation-Max-Expectation...

Minimax will pick any action, but what if we model using a random agent?

$$U\left(\begin{array}{c|c|c} \text{X} & & \\ \hline & & \\ \hline & & \end{array}\right) = U\left(\begin{array}{c|c|c} & & \text{X} \\ \hline & & \\ \hline & & \end{array}\right) = U\left(\begin{array}{c|c|c} & & \\ \hline & & \\ \hline \text{X} & & \end{array}\right) = U\left(\begin{array}{c|c|c} & & \\ \hline & & \\ \hline & & \text{X} \end{array}\right) = 0.995$$

$$U\left(\begin{array}{c|c|c} & & \\ \hline & \text{X} & \\ \hline & & \end{array}\right) = 0.990$$

$$U\left(\begin{array}{c|c|c} & \text{X} & \\ \hline & & \\ \hline & & \end{array}\right) = U\left(\begin{array}{c|c|c} & & \\ \hline \text{X} & & \\ \hline & & \end{array}\right) = U\left(\begin{array}{c|c|c} & & \\ \hline & & \text{X} \\ \hline & & \end{array}\right) = U\left(\begin{array}{c|c|c} & & \\ \hline & & \\ \hline & \text{X} & \end{array}\right) = 0.987$$

Additional

Reading

1. R&N Chapter 5 — Adversarial Search
2. Alpha-Beta: IBM Deep Blue — https://www.sciencedirect.com/science/article/pii/S0004370201001291
3. What Game Theory Reveals About Life, The Universe, and Everything — https://youtu.be/mScpHTIi-kM?si=CLagrjz3WVi-EkXG
4. Expectimax for 2048, `16384: 94%` — https://github.com/nneonneo/2048-ai
5. National Museum of Mathematics Tic-Tac-Toe — https://momath.org/wp-content/uploads/2021/08/Alyssa-Choi-Tic-Tac-Toe.pdf

Experiment

1. Implement `minimax` in the https://eric-han.com/teaching/demo/ttt.py
2. Slides available https://eric-han.com/teaching/demo/ttt.pdf