CMSC 373 Artificial Intelligence
Fall 2023
05-Game Playing

Deepak Kumar
Bryn Mawr College
2-Person Games

Checkers  Chess  Go  Konane  Tic Tac Toe

Perfect Information Game
Zero Sum Game
Samuel’s Checkers Program, 1950s


Writing Game Playing Programs

• The base algorithm for most 2-person, zero-sum, perfect information games is called the **Minimax Algorithm**.

• The algorithm explores possible moves the computer can play. And, in response examines possible moves the user can play. This is done for multiple levels. Information gathered from this exploration is then used to decide a move the computer should play.

• This strategy can be used for all 2-person, zero-sum, perfect information games. And, for imperfect information games as well.
Tic Tac Toe – Exploring Moves (depth = 2)

Minimax Search (MAX & MIN levels)
Minimax Search Algorithm

- A..U represent board positions
- A is the current board and it is the computer’s turn to move
- The computer has two possible moves, B & C
- Minimax answers the question: Which is a better move? B? C?
- If the computer makes move B, the user will have the moves D, E, F. Similarly for C the user will have G, H, I. etc.
- Minimax explores this as a search tree in a **depth-first** fashion.

Upon reaching a limit, the computer does an evaluation of the state of the board (see numbers in red).

A 4 in board state J represents the state of the board. Compared to -3 in board position M implies that J is a more desirable state than M.

- Each level is labelled as Max, or Min.
  - Max level means the maximum value will be selected (to favor the computer).
  - Min level means the minimum value will be selected as it is assumed that the user will always make the worst move possible for the computer.
- Through the search process, these values are **sifted up** the tree to enable the computer to decide about its move.

**Question:** Where do these values come from???
**Game Ply**

- Each pair of Max and Min levels is called a ply.
- Typically, one can specify how many plies to look ahead.
- Typically, more plies searched leads to better moves selected. The longer it takes.

**How Minimax Works**

The Minimax algorithm is a decision-making algorithm used in game theory and artificial intelligence to determine the best move for a player. It works by exploring the game tree, which represents all possible game states, and determining the best move based on a set of predefined rules. The algorithm uses a depth-first search and explores the tree to a predetermined depth, known as the ply of the game. At each node, the algorithm alternates between maximizing and minimizing values until it reaches the leaf nodes, which represent terminal states of the game. The backed-up value indicates the best move for the player at each node.
How Minimax Works

Where do those values come from???

- **Heuristics!!**
- Given a board state, $b$
  
  $e(b) = \text{an evaluation of state, } b \text{ to indicate its goodness for computer}$

  $e(b)$ is called the **static evaluation function**

  positive values represent a favorable state for the computer
  negative values represent a favorable state for the user

- Heuristics will vary from game to game. From programmer to programmer.
  It is more of an art.
  Quality of the heuristic function determines the quality of Minimax/game
A Heuristic for Tic Tac Toe

\[ e(b) = \text{[#of open lines for computer (x)]} - \text{[#open lines for user (o)]} \]

Which move should the computer make???
A Heuristic for Tic Tac Toe

\[ e(b) = \text{[#of open lines for computer (x)]} - \text{[#open lines for user (o)]} \]

Example Static Evaluation Functions

- **A simple heuristic for chess**
  
  \[
  \begin{align*}
  \text{pawn} &= 1 \\
  \text{knight} &= 3 \\
  \text{bishop} &= 3.5 \\
  \text{rook} &= 5 \\
  \text{queen} &= 9
  \end{align*}
  \]

  \[ e(b) = \text{[add up all the values of each piece for computer]} - \text{[add up all the values of each piece for user]} \]

- **Konane**

  \[ e(b) = \text{[#of moves available for computer]} - \text{[#of moves available for user]} \]

- **Go**

  Incorporates the difference between # of liberties, # of pieces, # of eyes
The Minimax Algorithm

**Function** `minimax(n)` **returns** `(pbv, move)`

- if `n` at depth bound
  - return `(e(n), move(n))`
- expand `n` to `n_1, n_2, ..., n_b` successors

  - if `n` is a MAX node:
    - `cbv = -\infty`, `bestMove = \emptyset`
    - for each `n_i in n_1, n_2, ..., n_b`
      - `bv, move = minimax(n_i)`
      - if `bv > cbv`
        - `cbv = bv`, `bestMove = move`
    - return `(cbv, bestMove)`

  - if `n` is a MIN node:
    - `cbv = \infty`, `bestMove = \emptyset`
    - for each `n_i in n_1, n_2, ..., n_b`
      - `bv, move = minimax(n_i)`
      - if `bv < cbv`
        - `cbv = bv`, `bestMove = move`
    - return `(cbv, bestMove)`

---

Which Move? What Value?

![Minimax Algorithm Diagram](image_url)
Which Move? What Value?

Complexity of Game Playing

- Minimax searches the entire tree up to level $d$
- With a branching factor, $b$ the complexity is $O(b^d)$

**Tic Tac Toe**
Average branching factor is 4, max depth is 9, i.e. $4^9 = 262,144$ states
The actual number is far less since many games end well before 9 moves.

**Konane**
Average branching factor is 10. A typical game lasts ~20 moves per player. Therefore, $10^{40}$ states!

**Chess**
Average branching factor is 31..35. A typical game lasts ~20 moves per player. Therefore, $31^{40}$ states!!

**Go**
Average branching factor is 250. A typical game lasts ~100 moves per player. Therefore, $200^{250}$ states!!
How to manage the Combinatorial Explosion?

- Only search to a limited ply (typically no more than 3-6)
  
  **Tic Tac Toe**  
  Average branching factor is 4. If limited ply is 3 (i.e. \(d=6\)), i.e. \(4^6 = 4096\) states
  
  **Konane**  
  Average branching factor is 10. If limited ply is 3 (i.e. \(d=6\)), \(10^6\) states
  
  **Chess**  
  Average branching factor is 31..35. If limited ply is 3 (i.e. \(d=6\)), \(31^6 = 887\) billion states. Still too large!
  
  **Go**  
  Fuhgeddaboudit!!!

---

Improving Minimax with \(\alpha-\beta\) Pruning

- Instead of searching the entire tree to level \(d\), we can reduce the number of states searched by **pruning the tree** as the search progresses.
Minimax with $\alpha$-$\beta$ Pruning

• Instead of searching the entire tree to level $d$, we can reduce the number of states searched by pruning the tree as the search progresses.

• Other improvements can also be made: move ordering is very common.

• The branching factor of the search can be effectively reduced to $\sqrt{b}$ allowing the search to go deeper in the same amount of time for $b$.

• In the end, all this and bigger faster computers have been very successful!
Minimax with $\alpha$-$\beta$ Pruning

Function \texttt{minimax-$\alpha$-$\beta$}(n, $\alpha$, $\beta$) returns $(pbv, move)$
- \textbf{if} $n$ at depth bound
  - \textbf{return} $(e(n), \text{move}(n))$
\textbf{expand} $n$ to $n_1, n_2, ..., n_b$ successors

\textbf{if} $n$ is a MAX node:
  \textit{bestMove} = $\emptyset$
  \textbf{for each} $n_i \in n_1, n_2, ..., n_b$
    - $bv, \text{move} = \text{minimax-$\alpha$-$\beta$}(n_i, \alpha, \beta)$
      \textbf{if} $bv > \alpha$
        - $\alpha = bv$, \textit{bestMove}=\text{move}
      \textbf{if} $\alpha \geq \beta$
        - \textbf{return} $(\beta, \textit{bestMove})$
    \textbf{return} $(\alpha, \textit{bestMove})$

\textbf{if} $n$ is a MIN node:
  \textit{bestMove} = $\emptyset$
  \textbf{for each} $n_i \in n_1, n_2, ..., n_b$
    - $bv, \text{move} = \text{minimax-$\alpha$-$\beta$}(n_i, \alpha, \beta)$
      \textbf{if} $bv < \beta$
        - $\beta = bv$, \textit{bestMove}=\text{move}
      \textbf{if} $\beta \leq \alpha$
        - \textbf{return} $(\alpha, \textit{bestMove})$
    \textbf{return} $(\beta, \textit{bestMove})$

---

Game Playing Successes

- **Checkers**
  Chinook (U. of Alberta)
  Beat the best player in the world in 1995 (though storied history!)
  In 2007, Chinook’s team declared that “Checkers is solved!”

- **Chess**
  In 1996, IBM’s Deep Blue beat Garry Kasparov in Philadelphia!

- **Go**
  We’ll see later in the course!
References
