Outline

• Overview
• Minimax search
• Adding alpha-beta cutoffs
• Additional refinements
• Iterative deepening
Overview

Old beliefs

Games provided a structured task in which it was very easy to measure success or failure.

Games did not obviously require large amounts of knowledge, thought to be solvable by straightforward search.
Overview

Chess

The average branching factor is around 35.

In an average game, each player might make 50 moves.

One would have to examine $35^{100}$ positions.
Overview

• Improve the *generate procedure* so that only good moves are generated.

```text
CuuDuongThanCong.com
```
Overview

• Improve the *generate procedure* so that only good moves are generated.

_plausible-moves vs. legal-moves_
Overview

• Improve the test procedure so that the best moves will be recognized and explored first.

CuuDuongThanCong.com

https://fb.com/tailieudientucntt
Overview

• Improve the test procedure so that the best moves will be recognized and explored first.

  less moves to be evaluated
Overview

• It is not usually possible to search until a goal state is found.

• It has to evaluate individual board positions by estimating how likely they are to lead to a win.

  Static evaluation function

• Credit assignment problem (Minsky, 1963).
Overview

• Good plausible-move generator.
• Good static evaluation function.
Minimax Search

- **Depth-first** and **depth-limited** search.
- At the player choice, **maximize** the static evaluation of the next position.
- At the opponent choice, **minimize** the static evaluation of the next position.
Minimax Search

Two-ply search
Minimax Search

**Player**(Position, Depth):

for each \( S \in \text{SUCCESSORS}(\text{Position}) \) do

\[
\text{RESULT} = \text{Opponent}(S, \text{Depth} + 1)
\]

\[
\text{NEW-VALUE} = \text{PLAYER-VALUE}(	ext{RESULT})
\]

if NEW-VALUE > MAX-SCORE, then

\[
\text{MAX-SCORE} = \text{NEW-VALUE}
\]

\[
\text{BEST-PATH} = \text{PATH}(	ext{RESULT}) + S
\]

return

\[
\text{VALUE} = \text{MAX-SCORE}
\]

\[
\text{PATH} = \text{BEST-PATH}
\]
Minimax Search

\textbf{Opponent}(Position, Depth):

for each \( S \in \text{SUCCESSORS}(\text{Position}) \) do

\begin{align*}
\text{RESULT} &= \text{Player}(S, \text{Depth} + 1) \\
\text{NEW-VALUE} &= \text{PLAYER-VALUE}(\text{RESULT}) \\
\text{if NEW-VALUE} &< \text{MIN-SCORE, then} \\
&\hspace{1cm} \text{MIN-SCORE} = \text{NEW-VALUE} \\
&\hspace{1cm} \text{BEST-PATH} = \text{PATH}(\text{RESULT}) + S
\end{align*}

\text{return}

\begin{align*}
\text{VALUE} &= \text{MIN-SCORE} \\
\text{PATH} &= \text{BEST-PATH}
\end{align*}
Minimax Search

Any-Player(Position, Depth):

for each \( S \in \text{SUCCESSORS}(\text{Position}) \) do

\[ \text{RESULT} = \text{Any-Player}(S, \text{Depth} + 1) \]

\[ \text{NEW-VALUE} = -\text{VALUE(RESULT)} \]

if \( \text{NEW-VALUE} > \text{BEST-SCORE} \), then

\[ \text{BEST-SCORE} = \text{NEW-VALUE} \]

\[ \text{BEST-PATH} = \text{PATH(RESULT)} + S \]

return

\[ \text{VALUE} = \text{BEST-SCORE} \]

\[ \text{PATH} = \text{BEST-PATH} \]
Minimax Search

\textbf{MINIMAX}(\text{Position, Depth, Player}): \\

- \textbf{MOVE-GEN}(\text{Position, Player}). \\
- \textbf{STATIC}(\text{Position, Player}). \\
- \textbf{DEEP-ENOUGH}(\text{Position, Depth})
Minimax Search

1. if DEEP-ENOUGH(Position, Depth), then return:
   \[
   \text{VALUE} = \text{STATIC(Position, Player)} \\
   \text{PATH} = \text{nil}
   \]

2. SUCCESSORS = MOVE-GEN(Position, Player)

3. if SUCCESSORS is empty, then do as in Step 1
Minimax Search

4. if SUCCESSORS is not empty:

   RESULT-SUCC = MINIMAX(SUCC, Depth+1, Opp(Player))

   NEW-VALUE = − VALUE(RESULT-SUCC)

   if NEW-VALUE > BEST-SCORE, then:

   BEST-SCORE = NEW-VALUE
   BEST-PATH = PATH(RESULT-SUCC) + SUCC

5. Return:

   VALUE = BEST-SCORE
   PATH = BEST-PATH
Adding Alpha-Beta Cutoffs

• At the player choice, **maximize** the static evaluation of the next position.
  > $\alpha$ threshold

• At the opponent choice, **minimize** the static evaluation of the next position.
  < $\beta$ threshold
Adding Alpha-Beta Cutoffs

Maximizing ply
Player

Minimizing ply
Opponent

Maximizing ply
Player

Minimizing ply
Opponent

α cutoff

β cutoff
Adding Alpha-Beta Cutoffs

Maximizing ply
Player

Minimizing ply
Opponent

Maximizing ply
Player

Minimizing ply
Opponent

α cutoff

β cutoff

Maximizing ply
Player

Minimizing ply
Opponent

α cutoff

β cutoff
**Player** (Position, Depth, α, β):

for each $S \in$ SUCCESSORS(Position) do

RESULT = Opponent($S$, Depth + 1, α, β)

NEW-VALUE = PLAYER-VALUE(RESULT)

if NEW-VALUE > α, then

$\alpha$ = NEW-VALUE

BEST-PATH = PATH(RESULT) + $S$

if $\alpha \geq \beta$ then return

VALUE = $\alpha$

PATH = BEST-PATH

return

VALUE = $\alpha$

PATH = BEST-PATH
Opponent(Position, Depth, $\alpha$, $\beta$):

for each $S \in$ SUCCESSORS(Position) do

RESULT = Player($S$, Depth + 1, $\alpha$, $\beta$)

NEW-VALUE = PLAYER-VALUE(RESULT)

if NEW-VALUE < $\beta$, then

$\beta = $ NEW-VALUE

BEST-PATH = PATH(RESULT) + $S$

if $\beta \leq \alpha$ then return

VALUE = $\beta$

PATH = BEST-PATH

return

VALUE = $\beta$

PATH = BEST-PATH
Any-Player(Position, Depth, α, β):

for each $S \in$ SUCCESSORS(Position) do

RESULT = Any-Player($S$, Depth + 1, $-\beta$, $-\alpha$)

NEW-VALUE = $-\text{VALUE}(RESULT)$

if NEW-VALUE $> \alpha$, then

$\alpha = \text{NEW-VALUE}$

BEST-PATH = PATH(RESULT) + $S$

if $\alpha \geq \beta$ then return

VALUE = $\alpha$

PATH = BEST-PATH

return

VALUE = $\alpha$

PATH = BEST-PATH
Additional Refinements

- Futility cutoffs
- Waiting for quiescence
- Secondary search
- Using book moves
- Not assuming opponent’s optimal move
Additional Refinements

- Futility cutoffs
Iterative Deepening

Iteration 1

Iteration 2

Iteration 3
Homework

Exercises 1-7, 9 (Chapter 12 – Al Rich & Knight)