Meeting 6: Alpha-Beta Pruning, Real-Time Decisions (and Chance)
Reading: AIAMA 5.3-5.4 (some of 5.5 and 5.6)

Today’s Schedule:
- Review minimax search
- Pruning using $\alpha - \beta$
- Real-Time Decisions
- Stochastic and Imperfect Information Games
**Minimax Search**

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

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    \( v \leftarrow -\infty \)
    for each \( a \) in \( \text{ACTIONS}(state) \) do
        \( v \leftarrow \max(v, \text{MIN-VALUE}(\text{RESULT}(s, a))) \)
    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 \( \text{ACTIONS}(state) \) do
        \( v \leftarrow \min(v, \text{MAX-VALUE}(\text{RESULT}(s, a))) \)
    return \( v \)
```
def minimax_decision(state, game):
    player = game.to_move(state)

    def max_value(state):
        ...

    def min_value(state):
        ...

    # Body of minimax_decision:
    return argmax(game.actions(state),
                  lambda a: min_value(game.result(state, a)))
The argmax function

def argmin(seq, fn):
    best = seq[0]; best_score = fn(best)
    for x in seq:
        x_score = fn(x)
        if x_score < best_score:
            best, best_score = x, x_score
    return best

def argmax(seq, fn):
    return argmin(seq, lambda x: -fn(x))
def max_value(state):
    if game.terminal_test(state):
        return game.utility(state, player)
    v = -infinity
    for a in game.actions(state):
        v = max(v, min_value(game.result(state, a)))
    return v

def min_value(state):
    if game.terminal_test(state):
        return game.utility(state, player)
    v = infinity
    for a in game.actions(state):
        v = min(v, max_value(game.result(state, a)))
    return v
Recall this example from last time

Assume Max goes first in the following game tree. What move should be made?
Pruning using $\alpha - \beta$

```plaintext
function ALPHA-BETA-SEARCH(state) returns an action
    $v \leftarrow$ MAX-VALUE(state, $-\infty$, $+\infty$)
    return the action in ACTIONS(state) with value $v$

function MAX-VALUE(state, $\alpha$, $\beta$) 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-VALUERESULT(s,a), $\alpha$, $\beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ MAX($\alpha$, $v$)
    return $v$

function MIN-VALUE(state, $\alpha$, $\beta$) 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-VALUERESULT(s,a), $\alpha$, $\beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ MIN($\beta$, $v$)
    return $v$
```

Figure 5.7 The alpha-beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure ??, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain $\alpha$ and $\beta$ (and the bookkeeping to pass these parameters along).
def alphabeta_full_search(state, game):
    player = game.to_move(state)

    def max_value(state, alpha, beta):
        ...

    def min_value(state, alpha, beta):
        ...

    return argmax(game.actions(state),
                  lambda a: min_value(game.result(state, a),
                                      -infinity, infinity))
def max_value(state, alpha, beta):
    if game.terminal_test(state):
        return game.utility(state, player)
    v = -infinity
    for a in game.actions(state):
        v = max(v, min_value(game.result(state, a), alpha, beta))
        if v >= beta:
            return v
        alpha = max(alpha, v)
    return v

def min_value(state, alpha, beta):
    if game.terminal_test(state):
        return game.utility(state, player)
    v = infinity
    for a in game.actions(state):
        v = min(v, max_value(game.result(state, a), alpha, beta))
        if v <= alpha:
            return v
        beta = min(beta, v)
Warmup #1

Consider the following game tree with heuristic values at a ply depth of 3 indicated. Perform alpha-beta search, describing where pruning occurs on the graph (if any). Indicate the best move for max to make. Assume max goes first.
Function Call Trace for the above example

Left branch

<table>
<thead>
<tr>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 -inf inf</td>
<td>s2 -inf inf</td>
</tr>
<tr>
<td>s4 -inf inf</td>
<td>s8 -inf inf</td>
</tr>
</tbody>
</table>

Min returning: 8

<table>
<thead>
<tr>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>s9 8 inf</td>
<td>s10 -inf 8</td>
</tr>
</tbody>
</table>

Min returning: 7

Max returning: 8

<table>
<thead>
<tr>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>s5 -inf 8</td>
<td>s11 5 8</td>
</tr>
</tbody>
</table>

Min returning: 5

Max returning: 5

Min returning: 5

Right Branch

<table>
<thead>
<tr>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3 5 inf</td>
<td>s6 5 inf</td>
</tr>
<tr>
<td>s12 5 inf</td>
<td>s13 5 inf</td>
</tr>
</tbody>
</table>

Min returning: 2

Max returning: 3

Min returning: 3

Pruning at s3 3 <= 5

Max returning: 5

Max returning: 5

Min returning: 5
Remarks on $\alpha - \beta$ pruning

- It reached the same conclusion as minimax, using the same ply-depth cutoff, but does so faster
- Because it is faster, given a fixed time-limit, it can search to a larger ply-depth
- How effective $\alpha - \beta$ is depends on the order moves are considered

```
for each a in ACTIONS(state) do
    The best moves are "killer" moves
```

- This typically translates into being able to search twice as deep
Warmup # 2

How would you handle a game tree in high-speed checkers, where the time limit for a move is under 5 seconds?
Time limits and real-time decisions

- Most games have a time limit and a state space that prohibits reaching terminal states.
- So, we cut-off the search at a specific depth and use a heuristic, called the evaluation function, to estimate what would be the backed-up value.

Some questions arise:

- How do we determine the depth?
- Does the branching factor depend on the depth?
- What do we do when we run out of time?, guess?
Should we just use a fixed depth cutoff?

- Not all game states are equal, some are more “exciting” than others
- In some states the moves generate drastically different evaluation values
- States that are stable relative to adjacent states are *quiescent*.

So, we can decide to search more deeply from nodes that are **not** quiescent.
So what was Deep Blue?

A layering of ideas

- minimax +
- $\alpha - \beta$ +
- iterative deepening +
- quiescence search +
- opening move ordering via a database (book) +
- pre-evaluated end games (bidirectional search) +
- parallel evaluation

Doing so it reached ply depths of 14-16 levels!
Consider a simple Stochastic Game: 6 sided die (1-6), each player takes turns tossing, adding up the number each time. The first to exceed 3 wins.

The players begin by putting $1 into a pot. Each turn the player can double the pot or keep it the same before the toss.

▶ sketch the game tree for a few ply depths
▶ how should we compute the best decision?
In a well shuffled deck of standard 52 cards, what is the chance of getting the initial pair Ace, Spades and 9, Clubs in a two player game of blackjack?
Game Tree for Blackjack

Sketch the game tree for black jack. Some questions to consider.

- What would be a good evaluation function?
- Why do real BJ tables use more than 1 deck?
- You may have heard of card counters, what do they do?
Next Actions

- Reading on Constraint Satisfaction, AIAMA 6.1-6.2
- Take warmup before noon on Tuesday 2/10.

Reminder! PS1 is due 2/16. You should be able to do Exercises 1-4 and EDP 1 and 2 now. Don’t procrastinate!