Artificial Intelligence **Spring 2005**
Professor Greenwald **Lecture #5**

Adversarial Search[1]

This lecture is concerned with search spaces that model parlor games such as go, chess, checkers, othello, and backgammon. At an abstract level, we are concerned with two-player, zero-sum games of perfect information in which the players' moves are sequential. Adversarial search algorithms are designed to return optimal paths, or winning strategies, through game trees, assuming the players are *adversaries*—rational and self-interested: i.e., they play to win.

1 Definition and Example

A *game tree* is a 7-tuple $\Gamma = \langle P, X, S, T, \delta, l, v \rangle$, where

- P is a set of n players
- X is a finite set of states
- $S \subseteq X$ is a nonempty set of *start* states
- $T \subseteq X$ is a nonempty set of *terminal* states
- $\delta : X \to 2^X$ is a state transition function
  $\delta(x)$ is the set of successor states of x
- $l : X \to P$ labels state x with the player who moves at x
- $v : T \to [-1, 1]^n$ maps terminal states into real-valued vectors
  $v_i(x) \in [-1, 1]$ is the payoff to player i at state x

Zero-sum games require that $\sum_{i=1}^{n} v_i(x) = 0$, for all $x \in T$. In two-player, zero-sum games the two players are viewed as adversaries; one player is the maximizer (MAX); the other is the minimizer (MIN). By definition, $v_{\text{MAX}}(x) = -v_{\text{MIN}}(x)$. Thus, it suffices to use one value to represent the value of a state: if $v(x) > 0$ ($v(x) < 0$), then MAX (MIN) is the winner; if $v(x) = 0$, then the game is a draw.

As an example, consider the zero-sum game of perfect information, m, p-NIM. Initially, there are p piles of m matches. To move, a player removes any number of matches from exactly one pile. The losing (or winning) player is s/he who removes the final match. The game tree representation of 2,2-NIM (for two players) is depicted in Figure 1, where it is modeled as follows:

[1]Copyright© Amy Greenwald, 2001–05

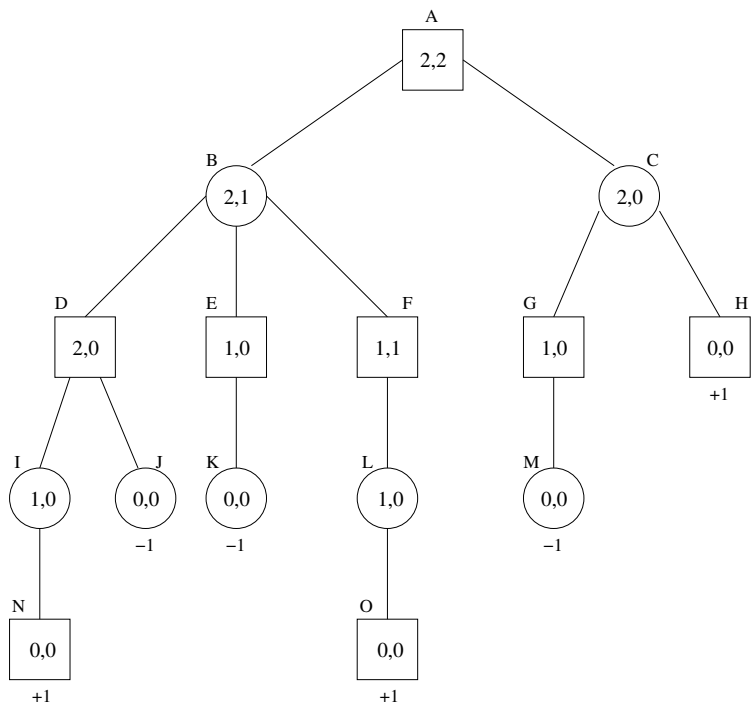

Figure 1: Game tree for 2,2-Nim. MAX nodes are squares; MIN nodes are circles. The minimax value of this game is -1: there is a winning strategy for MIN.

- $P = \{\textsc{Max}, \textsc{Min}\}$
- $X = Y \times P$: e.g., $((1,1), \textsc{Max}) \in X$
  $Y = \{(2,2), (2,1), (2,0), (1,1), (1,0), (0,0)\}$
  canonical representation where $a \geq b$ for all pairs (a, b)
- $S = \{((2,2), \textsc{Max}), ((2,2), \textsc{Min})\}$
- $T = \{((0,0), \textsc{Max}), ((0,0), \textsc{Min})\}$
- $\delta((2,2), \textsc{Max}) = \{((2,1), \textsc{Min}), ((2,0), \textsc{Min})\}$,
  $\delta((2,1), \textsc{Min}) \ = \{((1,1), \textsc{Max}), ((1,0), \textsc{Max}), ((2,0), \textsc{Max})\}, \ldots$
- $l((2,2), \textsc{Max}) = \textsc{Max}$,
  $l((2,1), \textsc{Min}) \ = \textsc{Min}$,
  $l((2,0), \textsc{Min}) \ = \textsc{Min}, \ldots$
- $v((0,0), \textsc{Max}) = (+1, -1)$,
  $v((0,0), \textsc{Min}) \ = (-1, +1)$

2 Minimax Search

The *minimax* value of a game tree is the value of the root node x, whenever MIN moves first, computed as the minimum value of x's successors, which are in turn computed as the maximum value of x's successors' successors, and so on.[2] If the minimax value of a game tree is $+1$, then there exists a winning strategy for MAX; if the minimax value is -1, then there exists a winning strategy for MIN; if the value of the root node is 0, then neither player has a winning strategy. The definition of the minimax value of a game tree suggests a breadth-first-search style computation. In practice, the minimax algorithm traverses nodes in depth-first-search (DFS) order to ensure that space is managed efficiently.

MINIMAX-DFS(Γ, x)	
Inputs	game tree Γ, root node x
Output	minimax value of game tree
Initialize	$O = \{x\}$, $v(n) = \perp$ for $n \notin T$

`while` (O is not empty) `do`

1. choose *first* node $n \in O$
2. if $n \in T$
   (a) if $l(n) = \text{MIN}$, let $v(n) = v_{\text{MIN}}(n)$
   (b) if $l(n) = \text{MAX}$, let $v(n) = v_{\text{MAX}}(n)$
3. if $v(n) \neq \perp$
   (a) if $n \neq x$
      i. let $m = \delta^{-1}(n)$
      ii. BACKUP1(Γ, n, m)
   (b) *delete* n from O
4. else if $v(n) = \perp$
   (a) if $l(n) = \text{MIN}$, $v(n) = +\infty$
      if $l(n) = \text{MAX}$, $v(n) = -\infty$
   (b) prepend $\delta(n)$ to *front* of O

`return` $v(x)$

Table 1: MiniMax Depth-First Search.

[2] The definition of the *maximin* value of a game is analogous to that of minimax, whenever the root node is labeled MAX.

BackUp1(Γ, n, m)	
Inputs	game tree Γ, node n, parent m
Output	updated game tree values

1. if $l(m) = \text{Min}$, $v(m) = \min\{v(m), v(n)\}$
   if $l(m) = \text{Max}$, $v(m) = \max\{v(m), v(n)\}$

Table 2: BackUp1 Subroutine: "backs up" values 1-level—from child to parent.

MiniMax-DFS maintains the following stack during its execution on the 2,2-NIM game tree in Figure 1. Nodes are subscripted with their values.

A
B C $A_{-\infty}$
D E F $B_{+\infty}$ C $A_{-\infty}$
I J $D_{-\infty}$ E F $B_{+\infty}$ C $A_{-\infty}$
N $I_{+\infty}$ J $D_{-\infty}$ E F $B_{+\infty}$ C $A_{-\infty}$
N_{+1} $I_{+\infty}$ J $D_{-\infty}$ E F $B_{+\infty}$ C $A_{-\infty}$
I_{+1} J $D_{-\infty}$ E F $B_{+\infty}$ C $A_{-\infty}$
J D_{+1} E F $B_{+\infty}$ C $A_{-\infty}$
J_{-1} D_{+1} E F $B_{+\infty}$ C $A_{-\infty}$
D_{+1} E F $B_{+\infty}$ C $A_{-\infty}$
E F B_{+1} C $A_{-\infty}$
K $E_{-\infty}$ F B_{+1} C $A_{-\infty}$
K_{-1} $E_{-\infty}$ F B_{+1} C $A_{-\infty}$
E_{-1} F B_{+1} C $A_{-\infty}$
F B_{-1} C $A_{-\infty}$
L $F_{-\infty}$ B_{-1} C $A_{-\infty}$
O $L_{+\infty}$ $F_{-\infty}$ B_{-1} C $A_{-\infty}$
O_{+1} $L_{+\infty}$ $F_{-\infty}$ B_{-1} C $A_{-\infty}$
L_{+1} $F_{-\infty}$ B_{-1} C $A_{-\infty}$
F_{+1} B_{-1} C $A_{-\infty}$
B_{-1} C $A_{-\infty}$
C A_{-1}
G H $C_{+\infty}$ A_{-1}
M $G_{-\infty}$ H $C_{+\infty}$ A_{-1}
M_{-1} $G_{-\infty}$ H $C_{+\infty}$ A_{-1}
G_{-1} H $C_{+\infty}$ A_{-1}
H C_{-1} A_{-1}
H_{+1} C_{-1} A_{-1}
C_{-1} A_{-1}
A_{-1}

Table 3: MiniMax-DFS on 2,2-NIM.

MiniMax-DFS (see Table 1) traverses the game tree in depth-first fashion, visiting all nonterminal nodes *twice*. The first time it visits a nonterminal node, its value is unknown; to evaluate the node, its children are pushed onto the stack. The second time it visits a nonterminal node, its value is known; thus, it is popped off the stack, and its value is backed up to its parent (See Table 2) (unless it is the root node). When MiniMax-DFS visits the root node for the second time, it terminates. At this point, the value of the root node is the minimax value of the game, since all of its children's values have been backed up, all of its children's children's values have also been backed up, and so on. Otherwise, all of its children would not have been popped off the stack, and it could not have been visited for the second time; similarly, all of its children's children would not have been popped off the stack, and its children could not have been visited for the second time, and so on.

3 $\alpha\beta$Pruning

Although the minimax solution always exists (Zermelo, 1912), it is intractable to compute the minimax solution in most interesting game trees. (On the contrary, in game trees like tic-tac-toe for which computing the minimax solution is tractable could be said to be uninteresting for precisely that reason!)

Sophisticated game-playing programs prune the search space. The $\alpha\beta$Pruning algorithm improves upon MiniMax: it is a technique that provably computes the MiniMax value of the game, but tends to visit only a fraction of the nodes of the MiniMax procedure because it prunes redundant subtrees.

A *variation* is a complete path, from root node to leaf node, through the game tree. The *principal variation* is the path through the game tree along which the minimax value is discovered. In particular, the value of the leaf node on the principal variation is the minimax value of the game.

At most, adversarial search algorithms need only consider those nodes that are candidates for the principal variation. The key idea underlying $\alpha\beta$Pruning is the following: *any subgame tree whose root node is provably* not *on the principal variation can be pruned.*

Example 1 Consider the chess game tree depicted in Figure 2. Once node B is determined to lead to checkmate for Max, there is no need to evaluate any of node A's other children, since node B—a win for Max—is at least as preferable to Max as node C. Therefore, node C can be pruned.

> Technically, $v(\mathrm{A}) = \max\{v(\mathrm{B}), v(\mathrm{C})\} \geq v(\mathrm{B}) = +1$ and $v(\mathrm{A}) \leq +1$; it follows that $v(\mathrm{A}) = +1$, regardless of the value of C.

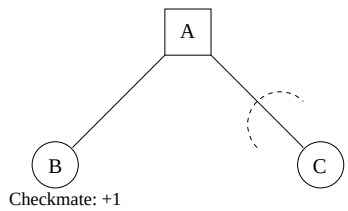

Figure 2: An example of $\alpha\beta$Pruning. Node C is not on the principal variation, since B is an optimal alternative for Max, so it can be pruned.

This example captures the following intuition: a node's value (e.g., C) *cannot* impact the minimax value of its parent (e.g., A) if its parent's minimax value is known. The subtrees of a node whose value is already known can be pruned, since these subtrees are not on the principal variation.

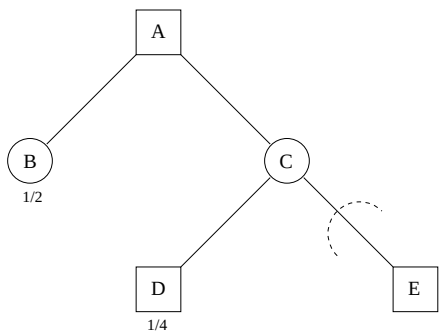

Figure 3: Another example of $\alpha\beta$Pruning. Node C is not on the principal variation, since B is a preferable alternative for Max, so node E can be pruned.

Example 2 Consider the game tree depicted in Figure 3. Once node D is determined to yield value 1/4, there is no further need to evaluate any of node C's children, since node B, with value 1/2, is at least as preferable to Max as node C. Therefore, node E can be pruned.

> Technically, since $v(\mathrm{C}) = \min\{v(\mathrm{D}), v(\mathrm{E})\} \leq v(\mathrm{D})$ and $v(\mathrm{D}) \leq v(\mathrm{B})$, it follows that $v(\mathrm{C}) \leq v(\mathrm{B})$. Therefore, $v(\mathrm{A}) = \max\{v(\mathrm{B}), v(\mathrm{C})\} = v(\mathrm{B})$, regardless of the value of E.

This example captures the following intuition: a node (e.g., C) is guaranteed *not* to be on the principal variation if any of the node's ancestors (e.g., A) presents a better alternative (e.g., B) for either player. The subtrees of such nodes can be pruned, since these subtrees are not on the principal variation.

3.1 $\alpha\beta$Pruning: High-Level

$\alpha\beta$Pruning associates an α-value and a β-value with every game. An α-value is a lower bound on the minimax value of the game: Max can guarantee that he wins at least this value. A β-value is an upper bound on the minimax value of the game: Min can guarantee that he loses no more than this value. At Max nodes, α-values are update; at Min nodes, β-values are updated. It is an invariant of $\alpha\beta$Pruning that $\alpha < \beta$. If after updating at a Max node x, α strictly exceeds β, then all subtrees rooted at x can be pruned, since there exists a preferable alternative for Min elsewhere in the game. If after updating at a Min node y, β falls strictly below α, then the subtree rooted at y can be pruned, since there exists a preferable alternative for Max elsewhere in the game. If α equals β at some node z, then the minimax value of the subtree rooted at z is determined, and all subtrees rooted at the children of z can be pruned.

3.2 $\alpha\beta$Pruning-DFS

Pseudocode that implements $\alpha\beta$Pruning via depth-first search is shown in Table 4. The depth-first search algorithm maintains both α- and β-values at all nodes. When a node is popped off the stack, a pruning test is immediately performed. (This test is described in the following paragraph.) If the node *cannot* be pruned, then the algorithm proceeds as in MiniMax-DFS. If it is a terminal node, it is evaluated, and its value is stored at the corresponding α- and β-values. Otherwise, a test is performed to determine whether the node's α- and β-values are initialized. If so, this visit is the second; all of its children's values have been backed up, so its value is now backed up, and it is deleted from the stack. If the node is not yet initialized, its values are initialized to those of its parents, and its children are prepended to the stack.

The pruning test is performed at all nodes other than the root node. The test simply asks whether a node's α-value is greater than or equal to its β-value, in which case the node can be pruned. As in the pseudocode, let n denote the node in question and m its parent node. Before any of m's children are evaluated, $\alpha(m) < \beta(m)$; otherwise the subtree rooted at m would have been pruned already. If m is labeled Min, then as m's children are evaluated, the value of β potentially decreases. If ever $\beta(m) < \alpha(m)$, then there exists an alternative for max at least as profitable; or, if $\beta(m) = \alpha(m)$, then the minimax value of m is known. If m is labeled Max, then as m's children are evaluated, the value of α potentially increases. If ever $\alpha(m) > \beta(m)$, then there exists an alternative for Min at least as desirable; or, if $\alpha(m) = \beta(m)$, then the minimax value of m is known. In all of these cases, the subtree rooted at m can be pruned.

Exercise Which subtrees are pruned by $\alpha\beta$Pruning in the game of 2,2-NIM?

$\alpha\beta$PRUNING-DFS(Γ, x)

Inputs	game tree Γ, root node x
Output	minimax value of game tree
Initialize	$O = \{x\}$, $\alpha(n) = \beta(n) = \perp$ for $n \notin T$

`while` (O is not empty) `do`

1. choose *first* node $n \in O$
2. if $n \neq x$
   (a) let $m = \delta^{-1}(n)$
   (b) if $\alpha(m) \geq \beta(m)$, *delete* n from O, `continue`
3. if $n \in T$
   (a) if $l(n) = \text{MIN}$, let $\beta(n) = \alpha(n) = v_{\text{MIN}}(n)$
   (b) if $l(n) = \text{MAX}$, let $\alpha(n) = \beta(n) = v_{\text{MAX}}(n)$
4. if $\alpha(n) \neq \perp$ and $\beta(n) \neq \perp$
   (a) if $n \neq x$
      i. let $m = \delta^{-1}(n)$
      ii. $\alpha\beta$BACKUP1(Γ, n, m)
   (b) *delete* n from O
5. else if $\alpha(n) = \beta(n) = \perp$
   (a) $\alpha(n) = -1$ and $\beta(n) = +1$
   (b) if $n \neq x$
      i. let $m = \delta^{-1}(n)$
      ii. $\alpha(n) = \max\{\alpha(n), \alpha(m)\}$
         $\beta(n) = \min\{\beta(n), \beta(m)\}$
   (c) prepend $\delta(n)$ to *front* of O

if $l(x) = \text{MIN}$, `return` $\beta(x)$
if $l(x) = \text{MAX}$, `return` $\alpha(x)$

$\alpha\beta$BACKUP1(Γ, n, m)

Inputs	game tree Γ, node n, parent m
Output	updated α and β values

1. if $l(m) = \text{MIN}$, $\beta(m) = \min\{\beta(m), \alpha(n)\}$
   if $l(m) = \text{MAX}$, $\alpha(m) = \max\{\alpha(m), \beta(n)\}$

Table 4: $\alpha\beta$Pruning Depth-First Search.

3.3 Complexity

In the best case, $\alpha\beta$pruning discovers the principal variation on its first traversal of the tree, and its effective branching factor[3] is $\sqrt{b}$. In this case, its complexity is only $O(b^{d/2})$—substantial savings over the minimax algorithm, which visits $O(b^d)$ nodes. In the worst-case, however, $\alpha\beta$pruning prunes no nodes at all.

The order in which nodes are explored greatly impacts the savings that can be achieved via $\alpha\beta$pruning. One reasonable ordering heuristic is to first perform iterative deepening search, and then to order successor nodes according to the backed-up values returned by the search at the previous depth.

4 Evaluation Functions

In spite of its effectiveness, $\alpha\beta$pruning does not provide sufficient pruning to compute the minimax solution in chess, for example. In general, in large game trees, search proceeds to some limited depth or for some limited time, at which point expansion of the search tree is truncated, and heuristic estimates of the minimax value are computed via an *evaluation function* and backed up the tree.

Most evaluation functions for the game of chess take into account material values: $e(\text{pawn}) = 1$, $e(\text{bishop}) = e(\text{knight}) = 3$, $e(\text{rook}) = 5$, and $e(\text{queen}) = 9$. Let W be the set of white's pieces; let B be the set of black's pieces. Now $w(n) = \sum_{p \in W} e(p)$ and $b(n) = \sum_{p \in B} e(p)$. One simple heuristic is to evaluate nodes as follows, assuming Max is white:

$$e(n) = \frac{w(n) - b(n)}{w(n) + b(n)}$$

The value $e(n) = 1$ predicts a sure win for white; the value $e(n) = -1$ predicts a sure win for black; the value $e(n) = 0$ predicts a draw.

But evaluation functions return heuristic estimates, which are not perfect. Thus, determining the depth or time at which to truncate search in game trees is a delicate matter. If $e(n)$ is changing rapidly, then single moves dramatically affect the (apparent) value of n. One popular heuristic is to allow search to proceed until *quiescence*. Another difficulty is that search algorithms cannot recognize drastic changes in the values of nodes if delay tactics push drastic moves beyond the horizon; this is called the *horizon effect*. One proposed solution to this problem is to engage in a secondary search beyond the seemingly best node (this technique is called *singular extension*.) If it is determined that this path degrades, then secondary search is performed on the second-best node; but it is impractical to conduct secondary search on all nodes.

[3] An *effective* branching factor b^* is that of a uniform tree of depth d if it were to contain N nodes. In other words, given that an algorithm expands N nodes, assuming depth d, b^* is the solution to the following equation: $N = 1 + b^* + \ldots + (b^*)^d$.

A Recursive Adversarial Search

In this appendix, we present recursive versions of the MiniMax and $\alpha\beta$-Pruning adversarial search algorithms. These algorithms search smaller and smaller game trees. Hence, we introduce the following notation: given (game) tree Γ, we let Γ_z denote the sub(game) tree of Γ rooted at node z.

A.1 Recursive MiniMax

This recursive minimax algorithm traverses nodes in depth-first-search order, by virtue of the fact that the algorithm is recursive. The initial call initializes $\alpha = -1$ (lower bound) and $\beta = +1$ (upper bound). The algorithm updates α values at Max nodes and β values at Min nodes. In the base case, the terminal nodes' values are returned. In the inductive step, two cases arise. If the root node x is a Min (Max) node, then as its successor's are evaluated, their values are "backed up"—"minned" ("maxed") with the value of x—until x's value is computed, at which point this updated value, β (α), is returned.

MiniMax(Γ_x, α, β)	
Inputs	game tree Γ_x rooted at x
	lower bound α
	upper bound β
Output	minimax value

1. if $x \in T$
   (a) if $l(x) =$ Min, `return` $v_{\text{MIN}}(x)$
   (b) if $l(x) =$ Max, `return` $v_{\text{MAX}}(x)$
2. if $l(x) =$ Min
   (a) for all $y \in \delta(x)$
      i. $\beta = \min\{\beta, \text{MiniMax}(\Gamma_y, \alpha, \beta)\}$
   (b) `return` β
3. if $l(x) =$ Max
   (a) for all $y \in \delta(x)$
      i. $\alpha = \max\{\alpha, \text{MiniMax}(\Gamma_y, \alpha, \beta)\}$
   (b) `return` α

Table 5: Recursive MiniMax.

MiniMax updates α- and β-values as follows during execution on 2,2-NIM.

1. $A : \alpha = \max\{-1, \textsc{MiniMax}(2,2\text{-Nim}_B, -1, +1)\}$
   (a) $B : \beta = \min\{+1, \textsc{MiniMax}(2,2\text{-Nim}_D, -1, +1)\}$
      i. $D : \alpha = \max\{-1, \textsc{MiniMax}(2,2\text{-Nim}_I, -1, +1)\}$
         A. $I : \beta = \min\{+1, \textsc{MiniMax}(2,2\text{-Nim}_N, -1, +1)\}$
            I. $N :$ `return` $v = +1$
         B. $I : \beta = \min\{+1, +1\} = +1$
         C. $I :$ `return` $\beta = +1$
      ii. $D : \alpha = \max\{-1, +1\} = +1$
      iii. $D : \alpha = \max\{+1, \textsc{MiniMax}(2,2\text{-Nim}_J, +1, +1)\}$
         A. $J :$ `return` $v = -1$
      iv. $D : \alpha = \max\{+1, -1\} = +1$
      v. $D :$ `return` $\alpha = +1$
   (b) $B : \beta = \min\{+1, +1\} = +1$
   (c) $B : \beta = \min\{+1, \textsc{MiniMax}(2,2\text{-Nim}_E, -1, +1)\}$
      i. $E : \alpha = \max\{-1, \textsc{MiniMax}(2,2\text{-Nim}_K, -1, +1)\}$
         A. $K :$ `return` $v = -1$
      ii. $E : \alpha = \max\{-1, -1\} = -1$
      iii. $E :$ `return` $\alpha = -1$
   (d) $B : \beta = \min\{+1, -1\} = -1$
   (e) $B : \beta = \min\{-1, \textsc{MiniMax}(2,2\text{-Nim}_F, -1, -1)\}$
      i. $F : \alpha = \max\{-1, \textsc{MiniMax}(2,2\text{-Nim}_L, -1, -1)\}$
         A. $L : \beta = \min\{+1, \textsc{MiniMax}(2,2\text{-Nim}_O, -1, -1)\}$
            I. $O :$ `return` $v = +1$
         B. $L : \beta = \min\{+1, +1\} = +1$
         C. $L :$ `return` $\beta = +1$
      ii. $F : \alpha = \max\{-1, +1\} = +1$
      iii. $F :$ `return` $\alpha = +1$
   (f) $B : \beta = \min\{-1, +1\} = -1$
   (g) $B :$ `return` $\beta = -1$
2. $A : \alpha = \max\{-1, -1\} = -1$
3. $A : \alpha = \max\{-1, \textsc{MiniMax}(2,2\text{-Nim}_C, -1, +1)\}$
   (a) $C : \beta = \min\{+1, \textsc{MiniMax}(2,2\text{-Nim}_G, -1, +1)\}$
      i. $G : \alpha = \max\{-1, \textsc{MiniMax}(2,2\text{-Nim}_M, -1, +1)\}$
         A. $M :$ `return` $v = -1$
      ii. $G : \alpha = \max\{-1, -1\} = -1$
      iii. $G :$ `return` $\alpha = -1$
   (b) $C : \beta = \min\{+1, -1\} = -1$
   (c) $C : \beta = \min\{-1, \textsc{MiniMax}(2,2\text{-Nim}_H, -1, -1)\}$
      i. $H :$ `return` $v = +1$
   (d) $C : \beta = \min\{-1, +1\} = -1$
   (e) $C :$ `return` $\beta = -1$
4. $A : \alpha = \max\{-1, -1\} = -1$
5. $A :$ `return` $\alpha = -1$

Table 6: MiniMax on 2,2-NIM.

B Recursive $\alpha\beta$Pruning

The recursive version of $\alpha\beta$PRUNING appears in Table 7. The initial call to this algorithm initializes $\alpha = -1$ (lower bound) and $\beta = +1$ (upper bound). Let x denote the root node of the game tree. If x is a terminal node, the value of x is returned. Otherwise, if x is nonterminal, then $\alpha\beta$MINIMAX proceeds to recursively call itself on each of x's children in turn, updating the value of α or β depending on whether x is labeled MAX or MIN. With each recursive call, α is initialized as the maximum value that can be achieved at any of MAX's choice points along the path to x, and β is initialized as the minimum value that can be achieved at any of MIN's choice points along the path to x. These values are updated as variations rooted at x are completed. If after updating at a MAX node, the value of α exceeds the value of β, then MIN can achieve β rather than α along an alternative path. No further variations in this subtree need be completed, so the algorithm breaks out of the loop and returns β. (Ordinarily, an updated value of α is returned at MAX nodes.) The algorithm operates symmetrically on MIN nodes.

$\alpha\beta$PRUNING$(\Gamma_x, \alpha, \beta)$

Inputs	game tree Γ_x rooted at x
	lower bound α
	upper bound β
Output	value in $[\alpha, \beta]$

1. if $x \in T$
   (a) if $l(x) =$ MIN, `return` $v_{\text{MIN}}(x)$
   (b) if $l(x) =$ MAX, `return` $v_{\text{MAX}}(x)$
2. else if $l(x) =$ MIN
   (a) for all $y \in \delta(x)$
      i. let $\beta = \min\{\beta, \alpha\beta\text{PRUNING}(\Gamma_y, \alpha, \beta)\}$
      ii. if $\beta \leq \alpha$, `return` α
   (b) `return` β
3. else if $l(x) =$ MAX
   (a) for all $y \in \delta(x)$
      i. let $\alpha = \max\{\alpha, \alpha\beta\text{PRUNING}(\Gamma_y, \alpha, \beta)\}$
      ii. if $\alpha \geq \beta$, `return` β
   (b) `return` α

Table 7: Recursive $\alpha\beta$Pruning.

$\alpha\beta$PRUNING updates α- and β-values as follows during its execution on the 2,2-NIM game tree in Figure 1. This trace differs from the MINIMAX trace in Table 6 in that five nodes are pruned. Pruning is evidenced by a MAX (MIN) node returning a β (α) value rather than an α (β) value.

1. $A : \alpha = \max\{-1, \alpha\beta\text{PRUNING}(\text{2,2-Nim}_B, -1, +1)\}$
   (a) $B : \beta = \min\{+1, \alpha\beta\text{PRUNING}(\text{2,2-Nim}_D, -1, +1)\}$
      i. $D : \alpha = \max\{-1, \alpha\beta\text{PRUNING}(\text{2,2-Nim}_I, -1, +1)\}$
         A. $I : \beta = \min\{+1, \alpha\beta\text{PRUNING}(\text{2,2-Nim}_N, -1, +1)\}$
            I. $N :$ `return` $v = +1$
         B. $I : \beta = \min\{+1, +1\} = +1$
         C. $I :$ `return` $\beta = +1$
      ii. $D : \alpha = \max\{-1, +1\} = +1$
      iii. $D :$ `return` $\beta = +1$
   (b) $B : \beta = \min\{+1, +1\} = +1$
   (c) $B : \beta = \min\{+1, \alpha\beta\text{PRUNING}(\text{2,2-Nim}_E, -1, +1)\}$
      i. $E : \alpha = \max\{-1, \alpha\beta\text{PRUNING}(\text{2,2-Nim}_K, -1, +1)\}$
         A. $K :$ `return` $v = -1$
      ii. $E : \alpha = \max\{-1, -1\} = -1$
      iii. $E :$ `return` $\alpha = -1$
   (d) $B : \beta = \min\{+1, -1\} = -1$
   (e) $B :$ `return` $\alpha = -1$
2. $A : \alpha = \max\{-1, -1\} = -1$
3. $A : \alpha = \max\{-1, \alpha\beta\text{PRUNING}(\text{2,2-Nim}_C, -1, +1)\}$
   (a) $C : \beta = \min\{+1, \alpha\beta\text{PRUNING}(\text{2,2-Nim}_G, -1, +1)\}$
      i. $G : \alpha = \max\{-1, \alpha\beta\text{PRUNING}(\text{2,2-Nim}_M, -1, +1)\}$
         A. $M :$ `return` $v = -1$
      ii. $G : \alpha = \max\{-1, -1\} = -1$
      iii. $G :$ `return` $\alpha = -1$
   (b) $C : \beta = \min\{+1, -1\} = -1$
   (c) $C :$ `return` $\alpha = -1$
4. $A : \alpha = \max\{-1, -1\} = -1$
5. $A :$ `return` $\alpha = -1$

Table 8: $\alpha\beta$PRUNING on 2,2-NIM. In step 1(a)iii, node J is pruned; in step 1e, nodes F, L, and O are pruned; and in Step 3c, node H is pruned.

Problems

#1 Consider the following two-player, alternating move, zero-sum game. The game begins with one pile of seven coins. At each player's turn, s/he picks one pile of coins and divides it into two piles such that the number of coins in the first pile is a multiple of the number of coins in the second. For example, a pile of four coins may be divided into piles of one and three coins, or two piles of two coins. The game is over when the largest pile contains only two coins. The losing player is s/he that makes the last move.

Draw the game tree for this game. Compute the minimax value of the game. Indicate which nodes in your game tree (if any) can be pruned via $\alpha\beta$pruning.

#2 Consider the labeling of 2 player game trees not with a single value, but with a pair of values: e.g., $(x, -x)$. The first (second) value denotes the *payoffs* to player 1 (2). Extend this labeling and the MiniMax algorithm to handle N player games.

#3 Trace $\alpha\beta$Pruning on the abbreviated game depicted in Figure 3.

#4 Checkers is a 2 player board game where the objective is to capture all of one's opponents pieces. There are two types of pieces, checkers, which can move only in the forward direction, and kings, which can move both forwards and backwards. Give a (sensible) heuristic evaluation function e for the game of checkers in terms of the number of checkers c and the number of kings k on the board for each of red and black.