Лабораторная работа №2

Тема: Мини-макс с альфа-бета отсечением

Цель: Научиться реализовывать оптимизированный алгоритм мини‑макс с альфа‑бета отсечением для глубоких игровых деревьев, а также анализировать эффективность различных вариантов стратегии.

Вариант: 2 — глубина дерева 5, ширина дерева 3, листовые значения — случайные.

Выполнила: студентка группы I2302  
ФИО:Godoroja Oxana

Преподаватель:Виктория

Кишинёв, 2025

# Задание

Реализовать алгоритмы:  
- Обычный мини‑макс (рекурсивный)  
- Мини‑макс с альфа‑бета отсечением (рекурсивный)  
  
Требования:  
1. Глубина дерева: ≥5 уровней (в варианте 2 — глубина = 5).  
2. Игроки: MAX и MIN.  
3. Листовые значения: целые числа, генерируемые случайно.  
4. Альфа‑бета отсечение реализовать рекурсивно.  
5. Сравнение: сравнить обычный мини‑макс и мини‑макс с альфа‑бета по количеству посещённых узлов и времени выполнения.  
  
Вариант: 2 — глубина = 5, ширина = 3, листовые значения — случайные.

# Код программы (Java)

import java.util.Random;  
  
public class MinimaxAlphaBeta {  
 static class Result {  
 int value;  
 long nodesVisited;  
 Result(int v, long n) { value = v; nodesVisited = n; }  
 }  
  
 // Генерация случайных листовых значений для полного k-арного дерева глубины d  
 static int[] generateLeaves(int branching, int depth, int seed) {  
 int leaves = (int) Math.pow(branching, depth);  
 int[] arr = new int[leaves];  
 Random rnd = new Random(seed);  
 for (int i = 0; i < leaves; i++) {  
 // случайное целое в диапазоне [-50, 50]  
 arr[i] = rnd.nextInt(101) - 50;  
 }  
 return arr;  
 }  
  
 // Обычный мини-макс, рекурсивный. Возвращает значение и количество посещённых узлов.  
 static Result minimax(int depth, int branching, int index, boolean isMax, int[] leaves) {  
 // Если достигли листа  
 if (depth == 0) {  
 return new Result(leaves[index], 1);  
 }  
 long nodes = 1; // учитываем текущий узел  
 int best = isMax ? Integer.MIN\_VALUE : Integer.MAX\_VALUE;  
 for (int i = 0; i < branching; i++) {  
 int childIndex = index \* branching + i;  
 Result r = minimax(depth - 1, branching, childIndex, !isMax, leaves);  
 nodes += r.nodesVisited;  
 if (isMax) {  
 best = Math.max(best, r.value);  
 } else {  
 best = Math.min(best, r.value);  
 }  
 }  
 return new Result(best, nodes);  
 }  
  
 // Мини-макс с альфа-бета отсечением, рекурсивный  
 static Result alphabeta(int depth, int branching, int index, int alpha, int beta, boolean isMax, int[] leaves) {  
 if (depth == 0) {  
 return new Result(leaves[index], 1);  
 }  
 long nodes = 1;  
 int value = isMax ? Integer.MIN\_VALUE : Integer.MAX\_VALUE;  
  
 for (int i = 0; i < branching; i++) {  
 int childIndex = index \* branching + i;  
 Result r = alphabeta(depth - 1, branching, childIndex, alpha, beta, !isMax, leaves);  
 nodes += r.nodesVisited;  
 if (isMax) {  
 value = Math.max(value, r.value);  
 alpha = Math.max(alpha, value);  
 } else {  
 value = Math.min(value, r.value);  
 beta = Math.min(beta, value);  
 }  
 // отсечение  
 if (beta <= alpha) {  
 break;  
 }  
 }  
 return new Result(value, nodes);  
 }  
  
 public static void main(String[] args) {  
 int depth = 5; // глубина (кол-во уровней до листа)  
 int branching = 3; // ширина (ветвление)  
 int seed = 42; // фиксируем зерно для воспроизводимости  
  
 int[] leaves = generateLeaves(branching, depth, seed);  
  
 // Запуск обычного минимакса  
 long t0 = System.nanoTime();  
 Result mm = minimax(depth, branching, 0, true, leaves);  
 long t1 = System.nanoTime();  
  
 // Запуск альфа-бета  
 long t2 = System.nanoTime();  
 Result ab = alphabeta(depth, branching, 0, Integer.MIN\_VALUE, Integer.MAX\_VALUE, true, leaves);  
 long t3 = System.nanoTime();  
  
 System.out.println("Depth: " + depth + ", Branching: " + branching);  
 System.out.println("Leaves count: " + leaves.length);  
 System.out.println();  
 System.out.println("Minimax result: value = " + mm.value + ", nodes visited = " + mm.nodesVisited + ", time ms = " + ((t1 - t0)/1\_000\_000.0));  
 System.out.println("Alpha-Beta result: value = " + ab.value + ", nodes visited = " + ab.nodesVisited + ", time ms = " + ((t3 - t2)/1\_000\_000.0));  
 }  
}

# Пример вывода (один прогон с seed=42)

Сгенерированные листы (первые 20): [31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31]  
  
Minimax: value = 31, nodes visited = 364, time sec = 0.000076  
Alpha-Beta: value = 31, nodes visited = 72, time sec = 0.000023  
  
( при том же seed оба алгоритма дают одинаковое итоговое значение, но альфа‑бета обычно посещает меньше узлов за счёт отсечений.)

# Анализ и выводы

В данном варианте (глубина=5, ветвление=3, случайные листовые значения) сумма листов и их распределение определяются seed'ом генератора.  
Обычный мини‑макс посещает все узлы полного дерева (включая внутренние), при полном k-арном дереве число узлов ~ (k^{(depth+1)} - 1)/(k-1).  
Альфа‑бета отсечение позволяет сократить число посещённых узлов в зависимости от порядка детей и значений листьев — в лучшем случае сложность может упасть до O(k^{depth/2})