**UNIVERSITATEA DE STAT DIN MOLDOVA**

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**DEPARTAMENTUL DE INFORMATICĂ**

**MAMALIGA ARTUR**

**Мини-макс с альфа бета отсечением.**

**Информатика**

**Лабораторная №2.**

**CHIȘINĂU – 2025**

# **Задание**

Задание: реализовать оптимизированный алгоритм мини-макс с альфа-бета отсечением для глубоких игровых деревьев, а также проанализировать эффективность различных вариантов стратегии.

Требования:

Глубина дерева >= **5** уровней

Игроки – “**MAX**” и “**MIN**”

Листовые значения - **случайные**

Сравнение обычный минимакс против минимакс с альфа бета отсечением

Анализ эффективности – количество проверяемых узлов

### **Код:**

**Game\_tree**:

import numpy as np

np.random.seed(1)

# Node class for the game tree

class Node:

    def \_\_init\_\_(self, value=None, player=None, children=None):

        self.value = value

        self.player=player

        self.children = []

# Function to build a sample game tree

def build\_weighted\_tree(current\_depth,max\_depth,max\_children\_per\_node,values):

    if current\_depth > max\_depth:

        return None

    value = values.pop(0) if current\_depth == max\_depth else None

    player = "MAX" if current\_depth % 2 == 0 else "MIN"

    node = Node(value,player)

    for i in range(max\_children\_per\_node):

        child\_node = build\_weighted\_tree(current\_depth + 1, max\_depth, max\_children\_per\_node, values)

        if child\_node:

            node.children.append(child\_node)

    return node

def print\_tree(node, prefix="", is\_last=True):

    connector = "└── " if is\_last else "├── "

    value = node.value if node.value is not None else "None"

    print(prefix + connector + str(value) + " " + str(node.player))

    prefix += "    " if is\_last else "│   "

    child\_count = len(node.children)

    for i, child in enumerate(node.children):

        is\_last\_child = (i == (child\_count - 1))

        print\_tree(child, prefix, is\_last\_child)

def generate\_game\_tree(max\_depth,max\_children\_per\_node):

    shape = (1,pow(max\_children\_per\_node,max\_depth-1))

    values = np.random.randint(1,100, size=shape)[0].tolist()

    root = build\_weighted\_tree(1, max\_depth, max\_children\_per\_node,values)

    return root

Данный код занимается строительством бинарного древа с заданными шириной и глубиной, а также заполняет его псевдослучайными числами, используя сид 1.

**Minimax:**

def minimax(node, current\_depth, player\_name) -> float:

    if not hasattr(minimax, "counter"):

        minimax.counter = 0

    minimax.counter += 1

    if current\_depth == 1:

        return node.value

    if player\_name == "MAX":

        best\_value = float("-inf")

        for child in node.children:

            eval = minimax(child,current\_depth - 1, "MIN")

            best\_value = max(best\_value,eval)

        node.value = best\_value

        return best\_value

    else:

        best\_value = float("inf")

        for child in node.children:

            eval = minimax(child,current\_depth -1, "MAX")

            best\_value = min(best\_value,eval)

        node.value = best\_value

        return best\_value

**minimax\_ab\_pruning:**

def minimax\_ab\_pruning(node, current\_depth, player\_name, alpha=float("-inf"), beta=float("inf")) -> float:

    if not hasattr(minimax\_ab\_pruning, "counter"):

        minimax\_ab\_pruning.counter = 0

    minimax\_ab\_pruning.counter += 1

    if current\_depth == 1:

        return node.value

    if player\_name == "MAX":

        best\_value = float("-inf")

        for child in node.children:

            eval = minimax\_ab\_pruning(child,current\_depth - 1, "MIN", alpha, beta)

            best\_value = max(best\_value,eval)

            alpha = max(alpha,eval)

            if beta <= alpha:

                break

        node.value = best\_value

        return best\_value

    else:

        best\_value = float("inf")

        for child in node.children:

            eval = minimax\_ab\_pruning(child,current\_depth -1, "MAX", alpha, beta)

            best\_value = min(best\_value,eval)

            beta = min(beta,eval)

            if beta <= alpha:

                break

        node.value = best\_value

        return best\_value

**main:**

from game\_tree import generate\_game\_tree, print\_tree

from minimax import minimax

from minimax\_ab\_pruning import minimax\_ab\_pruning

import copy

import time

def main():

    tree\_depth = 10

    children\_per\_node = 3

    player = "MIN"

    root\_minimax = generate\_game\_tree(tree\_depth,children\_per\_node)

    root\_minimax\_ab\_pruning = copy.deepcopy(root\_minimax)

    # print("Initial tree:")

    # print\_tree(root\_minimax)

    # MiniMax

    start = time.perf\_counter()

    best\_value\_minimax = minimax(root\_minimax, tree\_depth, player)

    end = time.perf\_counter()

    t\_minimax = end - start

    print(f"MiniMax best value: {best\_value\_minimax}, time: {t\_minimax:.6f} sec")

    # MiniMax with alpha-beta pruning

    start = time.perf\_counter()

    best\_value\_ab = minimax\_ab\_pruning(root\_minimax\_ab\_pruning, tree\_depth, player)

    end = time.perf\_counter()

    t\_ab = end - start

    print(f"Alpha-Beta best value: {best\_value\_ab}, time: {t\_ab:.6f} sec")

    if t\_minimax > 0:

        speedup = abs((t\_minimax - t\_ab) / t\_minimax \* 100)

        print(f"Alpha-Beta is faster by {speedup:.2f}%")

    else:

        print("Timing error: minimax duration is 0")

    print("minimax times called")

    print(minimax.counter)

    print("minimax ab pruning times called")

    print(minimax\_ab\_pruning.counter)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# Вывод

**Output программы:**

* Пример с глубиной древа 5:

/lab02/src/main.py

MiniMax best value: 69, time: 0.000058 sec

Alpha-Beta best value: 69, time: 0.000048 sec

Alpha-Beta is faster by 17.27%

minimax times called

121

minimax ab pruning times called

69

* Пример с глубиной древа 10:

/lab02/src/main.py

MiniMax best value: 31, time: 0.009775 sec

Alpha-Beta best value: 31, time: 0.002166 sec

Alpha-Beta is faster by 77.84%

minimax times called

29524

minimax ab pruning times called

3837

* Пример с глубиной древа 12:

/lab02/src/main.py

MiniMax best value: 31, time: 0.091638 sec

Alpha-Beta best value: 31, time: 0.009147 sec

Alpha-Beta is faster by 90.02%

minimax times called

265720

minimax ab pruning times called

18722

Алгоритм оптимизации через альфа бета отсечение показывает свою эффективность на глубоких древах с увеличением числа нод. Общая сложность алгоритма minimax это O(b^d) ширина древа в степени глубины. Оптимизация дает в среднем O(b^(d/2)) общую сложность алгоритма. На деле же мы видим ускорение вплоть до 90%. И даже с небольшой глубиной древа в 5 единиц, алгоритм с оптимизацией вызывает рекурсивную функцию практически в 2 раза реже.