# 

# **Отчет по**

# **Лабораторной работе №2.**

# **По предмету: Artificial Intelligence**

# 

# 

# 

# **Подготовил студент группы IA2303: Gutu Nicolae**

# **Проверяющий преподаватель: V. Trebiș**

# 

# 

# **Сhișinau, 2025**

**Задание:**

**Мини-макс с альфа-бета отсечением**

**Цель:**

Научиться реализовывать оптимизированный алгоритм мини-маĸс с альфа-бета отсечением для глубоĸих игровых деревьев, а таĸже анализировать эффеĸтивность различных вариантов стратегии.

**1)Мой вариант (№5)**

* **Глубина дерева:** 7 уровней
* **Ширина дерева (ветвление):** 2
* **Особенность:** *сделать часть листьев одинаковыми, чтобы проверять равенство путей*

**2) Краткий обзор**

## **Термины**

* **Игровое дерево** — ориентированное дерево возможных позиций; корень — текущая позиция, рёбра — ходы.
* **Уровни MAX/MIN** — на уровнях MAX ходит наш игрок (берём максимум), на MIN — соперник (берём минимум).
* **Лист** — позиция, где поиск останавливается: либо конец игры, либо достигнут **предел глубины**.
* **Оценочная функция** — число на листе: насколько хороша позиция для MAX (больше — лучше).
* **Ветвление (branching factor, b)** — среднее число ходов из позиции.
* **Глубина (depth, d)** — сколько «слоёв ходов» просматриваем.

**Классический minimax** рекурсивно сворачивает дерево:

* на уровнях MAX берёт max из значений потомков,
* на уровнях MIN — min.

## **Alpha–beta pruning (ускорение)**

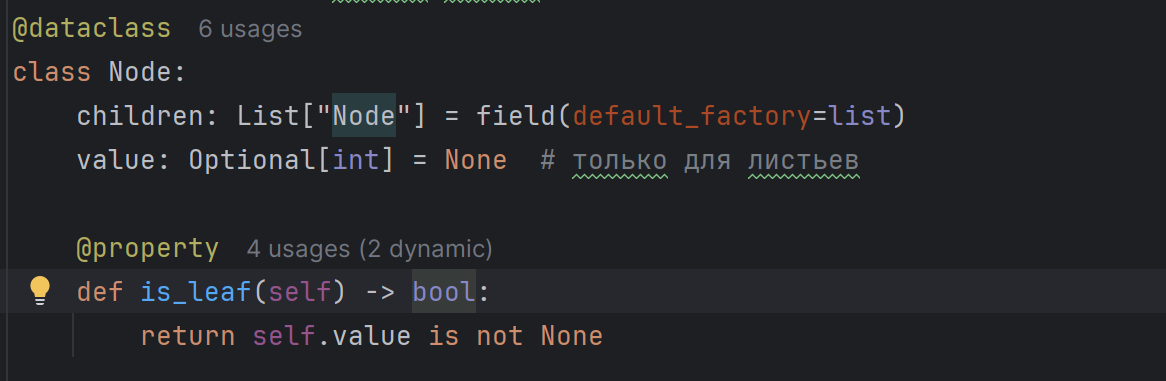
Поддерживаются границы:

* **α** — лучшая (нижняя) гарантия для MAX вдоль текущего пути.
* **β** — лучшая (верхняя) гарантия для MIN.

**Правило отсечения:** если в узле стало **α ≥ β**, оставшихся детей этого узла можно **не просматривать** — они не повлияют на итог.

**3) Разбор кода**

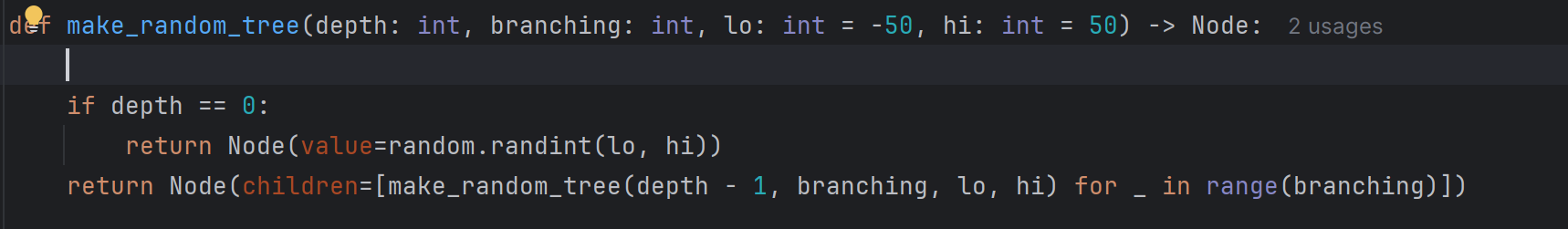
**3.1 Node Model**

*рис.1 Модель дерева*

Это модель узла дерева, в которой:

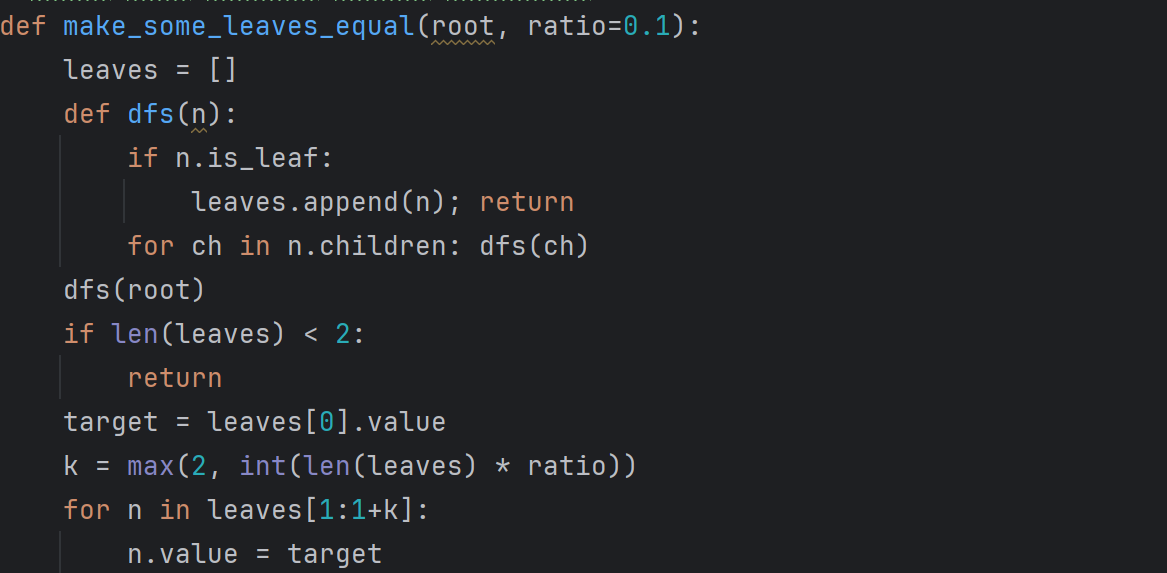
* children — варианты ходов;
* value хранится только у листьев;
* is\_leaf быстро отличает лист от узла

**3.2 Generating the tree**

*рис.2 Ровное дерево во случайными листьями*

Генератор «ровного» дерева. На глубине 0 создается лист со случайной оценкой;

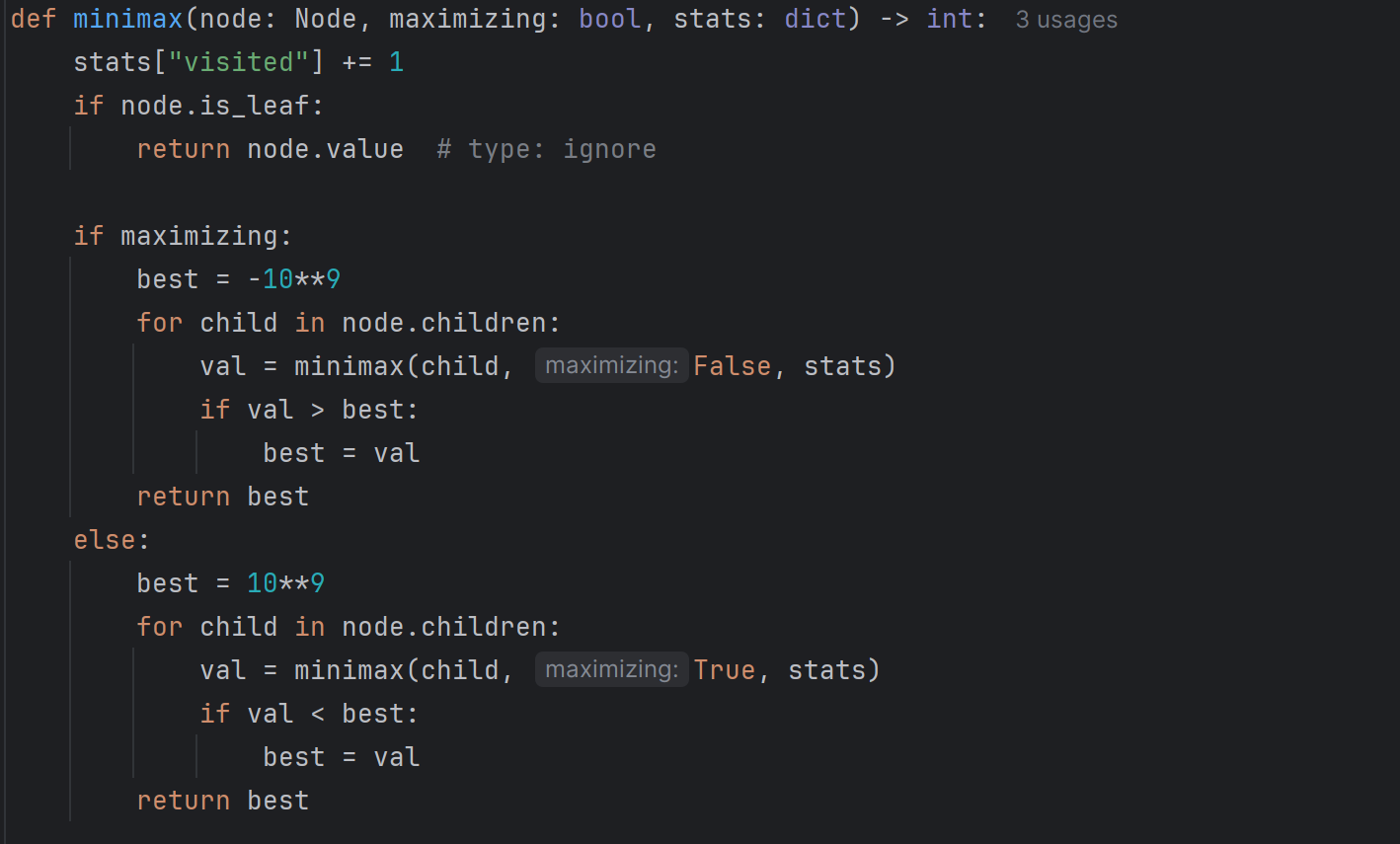
**3.3 Making leaves equal**



*\рис.3 Добавление одинаковых листьев*

Делаю часть листьев одинаковыми, сознательно создавая «ничьи» (равные оптимальные пути) для проверки корректности.

**3.4 Minimax calssical**

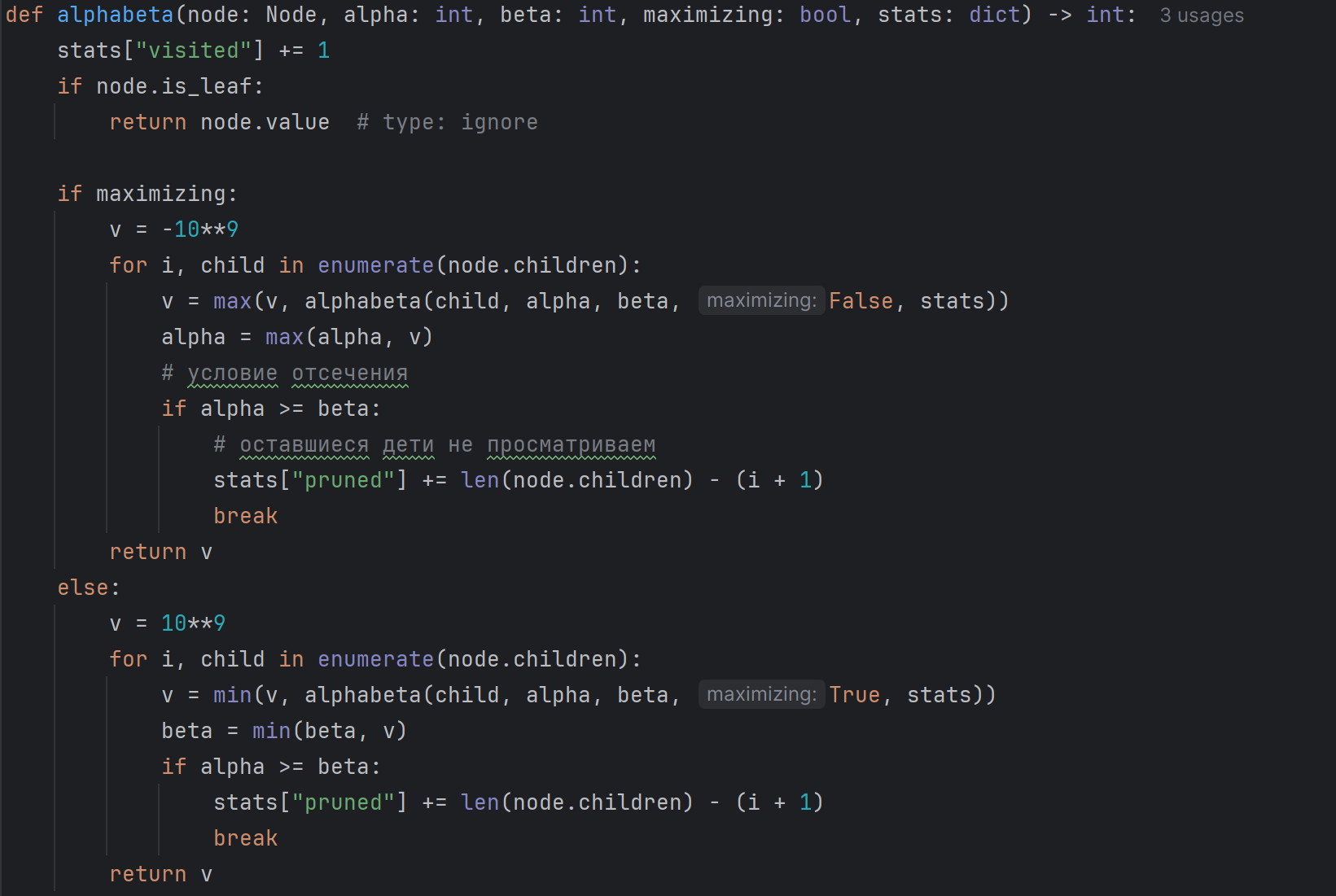


*рис.4 Mini-Max algorithm*

Классический minimax.

Сворачивает дерево снизу вверх: на MAX-уровнях берём максимум, на MIN — минимум; visited считает просмотренные узлы.

**3.5 AlhaBeta minimax**



*рис.5 Alpha-Beta algorithm with pruning*

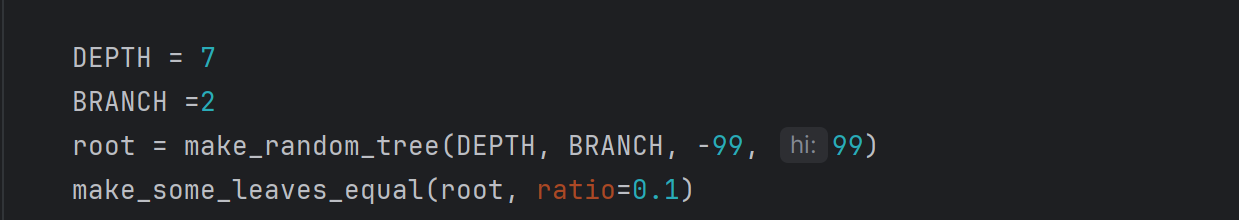
Minimax с α–β-отсечением:

* alpha — гарантия MAX (нижняя граница)
* beta — гарантия MIN (верхняя).
* При alpha >= beta оставшихся детей узла можно не смотреть;
* pruned — сколько таких детей пропущено.

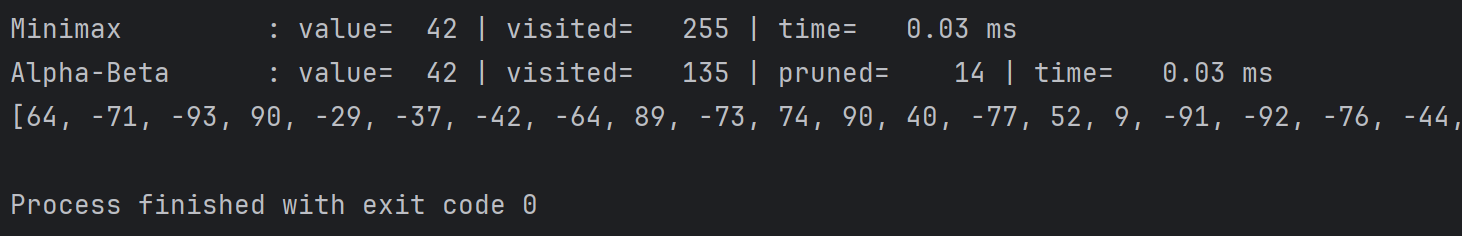
**4) Тестирование программы:**

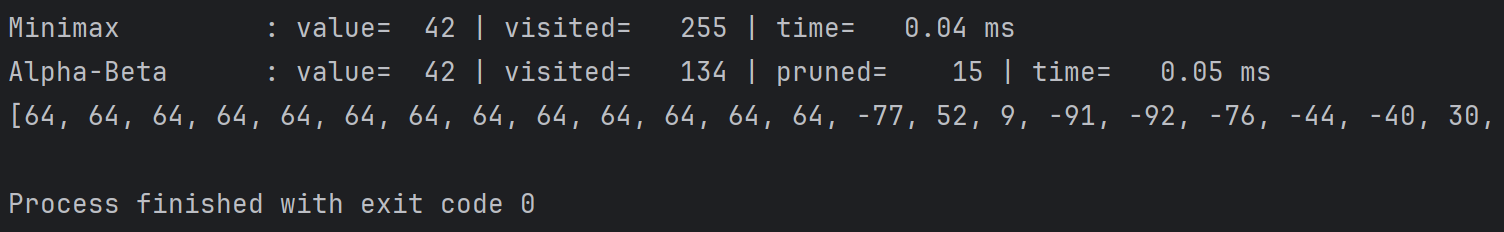
По условию моего варианта:

* Глубина: 7 единиц
* Количество ветвей: 2 единицы
* make\_some\_leaves\_equal: делает часть листьев одинаковыми, а именно 10% (ratio = 0.1)

*рис.6 Стартовые данные*

Результаты работы программы:

*рис. 7 Результаты со случайными листьями*



*рис. 8 Результаты c 10% одинаковых листьев*

Как видно по результатам работы: алгоритм отработал корректно в обоих случаях. При добавлении одинаковых листьев результат остался прежним, но в A|B-minimax было отрезано больше ветвей, что демонстрирует что при “ничейном результате” алгоритм сохраняет свою эффективность и не отрезает ветви, которые не могут дать лучший результат.