**Комп’ютерний практикум №4**

**Методи пошуку в умовах протидії**

**ПІБ: Мєшков Андрій Ігорович, Ткач Владислав Анатолійович**

**Група: ІП-15**

**Мета роботи:** ознайомитись з методами пошуку в умовах протидії та дослідити їх використання для інтелектуального агента в типовому ігровому середовищі.

***З*авдання:** обрати середовище, що моделює гру з нульовою сумою, та задачу, що містить декілька агентів, які протидіють один одному. В обраному середовищі вирішити поставлену задачу, реалізувавши один з методів пошуку в умовах протидії. Реалізувати власну функцію оцінки станів. Виконати дослідження впливу деякого фактору середовища.

**Номер варіанту: 23**

**Завдання для варіанту:** обрати середовище, що моделює гру з нульовою сумою, та задачу, що містить декілька агентів, які протидіють один одному. В обраному середовищі вирішити поставлену задачу, реалізувавши Альфа-бета відсікання. Реалізувати власну функцію оцінки станів. Виконати дослідження впливу Вплив глибини пошуку.

**Середовище і задача:**

Обране середовище для вирішення завдання - гра "Pacman" (імітаційна версія аркадної гри). У цій грі гравець керує Пакменом, який має збирати їжу в лабіринті, уникаючи зіткнень з привидами, що рухаються. Привиди намагаються зійти на Пакмена і втратити його. Гра має нульову суму, оскільки виграш одного гравця (наприклад, Пакмена) відповідає програш іншого (привидів) і навпаки.

Формалізація середовища:

* Агенти: Пакмен та привиди.
* Стани: конфігурація лабіринту та розташування Пакмена та привидів.
* Можливі дії агентів: для Пакмена - рух у чотирьох напрямках (вгору, вниз, вліво, вправо), для привидів - рух у напрямку Пакмена або рух у випадковому напрямку.
* Переходи: кожен агент робить крок у межах лабіринту, і гра переходить до наступного стану.
* Винагороди: Пакмен отримує винагороду за кожну зібрану одиницю їжі, а привиди отримують винагороду за з'їдання Пакмена.

**Метод вирішення задачі:** Метод альфа-бета відсікання є ефективним алгоритмом для пошуку оптимального рішення в деревоподібних структурах, таких як гри. Він призначений для випадків, коли можливі рішення представляються у вигляді дерева розгалужень (наприклад, граф пошуку), і ці рішення оцінюються за допомогою певної функції оцінки.

Основна ідея методу альфа-бета відсікання полягає в тому, щоб зменшити кількість вузлів, які потрібно оцінювати у дереві, шляхом відсікання піддерев, які точно не впливають на кінцевий результат. Це досягається завдяки двом параметрам, що використовуються для керування процесом обходу дерева:

1. Альфа (α) - найкращий (найбільший) варіант, який вже було знайдено для певного гравця або рівня глибини.

2. Бета (β) - найгірший (найменший) варіант, який дозволяє опоненту гравця або рівня глибини.

Процес відсікання відбувається наступним чином:

- Коли варіант перевищує або дорівнює значенню β, його можна відкинути, оскільки опонент не буде вибирати такий варіант.

- Коли варіант менше або дорівнює значенню α, він не буде вибрано гравцем, оскільки існує кращий варіант.

Цей метод дозволяє значно зменшити кількість варіантів, які потрібно розглядати, що робить його ефективним для розв'язання складних задач, таких як ігри з великою кількістю можливих ходів.

**Реалізація методу:**

class AlphaBetaAgent(MultiAgentSearchAgent):

def getAction(self, gameState: GameState) -> str:

def alphaBeta(state, depth, alpha, beta, agentIndex):

*if* state.isWin() or state.isLose() or depth == *self*.depth:

*return* *self*.evaluationFunction(state)

*if* agentIndex == 0:

*return* maxValue(state, depth, alpha, beta)

*else*:

*return* minValue(state, depth, alpha, beta, agentIndex)

def maxValue(state, depth, alpha, beta):

v = float('-inf')

actions = state.getLegalActions(0)

*if* not actions:

*return* *self*.evaluationFunction(state)

*for* action *in* actions:

successor = state.generateSuccessor(0, action)

v = max(v, alphaBeta(successor, depth, alpha, beta, 1))

*if* v >= beta:

*return* v

alpha = max(alpha, v)

*return* v

def minValue(state, depth, alpha, beta, agentIndex):

v = float('inf')

nextAgent = (agentIndex + 1) % state.getNumAgents()

nextDepth = depth + 1 *if* nextAgent == 0 *else* depth

actions = state.getLegalActions(agentIndex)

*if* not actions:

*return* *self*.evaluationFunction(state)

*for* action *in* actions:

successor = state.generateSuccessor(agentIndex, action)

v = min(v, alphaBeta(successor, nextDepth, alpha, beta, nextAgent))

*if* v <= alpha:

*return* v

beta = min(beta, v)

*return* v

bestAction = None

alpha = float('-inf')

beta = float('inf')

bestScore = float('-inf')

*for* action *in* gameState.getLegalActions(0):

successor = gameState.generateSuccessor(0, action)

score = alphaBeta(successor, 0, alpha, beta, 1)

*if* score > bestScore:

bestScore = score

bestAction = action

alpha = max(alpha, bestScore)

*return* bestAction

**Результати застосування розробленого методу:**

**Оцінка результатів:**

**Власна функція оцінки станів:**

**Реалізація власної функції оцінки станів:**

**Задача дослідження впливу параметра алгоритму чи фактору середовища:**