**Комп’ютерний практикум №4**

**Методи пошуку в умовах протидії**

**ПІБ:** Шляхтун Денис Михайлович.

**Група:** ІП-14.

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

**Завдання:** 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:** 22.

**Завдання для варіанту:** MiniMax, вплив глибини пошуку.

**Середовище і задача:** Пакмен – це гра, де Пакмен рухається по лабіринту та намагається з’їсти якомога більше харчових гранул, уникаючи привидів. Якщо Пакмен з’їсть всю їжу в лабіринті, він виграє. Великі білі крапки називаються капсулами, які дають Пакмену можливість з’їсти привида за обмежений час.

Агенти – Пакмен і привиди. Пакмен помирає, якщо зіштовхується з привидом, а якщо було підібрано капсулу, то навпаки Пакмен має можливість поглинути привида при зіткненні. Пакмен і привиди можуть рухатися у чотири сторони крім випадків руху в стіни. Пакмен отримує винагороду за кожну харчову гранулу, втрачає нагороду при русі, поразка має велику від’ємну нагороду. Кінцевими станами гри є смерть Пакмена – поразка та закінчення харчових гранул на карті – перемога.

**Метод вирішення задачі:** MiniMax – метод прийняття рішень для ігор з нульовою сумою. Стратегія опонента невідома, тому приймається як найгірша для агента. Це найкраща стратегія проти оптимального опонента, забезпечує нижню межу корисності проти будь-якого супротивника, не оптимальна проти відомої стратегії супротивника.

Використання – у стані *s* викликаємо .

**Реалізація методу:**

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

legalMoves = gameState.getLegalPacmanActions()

scores = [self.miniMax(gameState.generatePacmanSuccessor(action), 0, 1) for action in legalMoves]

bestScore = max(scores)

bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]

chosenIndex = random.choice(bestIndices)

return legalMoves[chosenIndex]

def miniMax(self, gameState: GameState, curDepth, agentIndex: int):

if gameState.isLose() or gameState.isWin() or curDepth >= self.depth:

return scoreEvaluationFunction(gameState)

actions = gameState.getLegalActions(agentIndex)

successors = [gameState.generateSuccessor(agentIndex, action) for action in actions]

nextAgentIndex = (agentIndex+1)%gameState.getNumAgents()

if nextAgentIndex == 0:

curDepth+=1

result = [self.miniMax(successor, curDepth, nextAgentIndex) for successor in successors]

if agentIndex == 0:

return max(result)

else:

return min(result)

**Результати застосування розробленого методу:**

Відповідно до методичного матеріалу, було протестовано алгоритм для глибини від 1 до 4 на карті minimaxClassic і значення методу для початкового стану співпали із вказаними до методичного матеріалу. На знімку результат проходження 10 запусків гри для глибини 4 на карті minimaxClassic:

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Алгоритм показав свою працездатність, перемоги становлять більше 50%. Повна оцінка алгоритму наведена у розділі «Задача дослідження впливу параметра алгоритму».

**Оцінка результатів:** Загалом алгоритм призначений при невідомій стратегії руху привидів, тобто припускається, що вони завжди будуть рухатися найгірше для нас. Оскільки поведінка привидів випадкова, у них нема певної описаної чи відомої нам стратегії, то застосування методу можна назвати доцільним.

На більших картах Пакмен рухається майже на одному місці, якщо знаходиться достатньо далеко від харчових гранул, адже глибина алгоритму не дозволяє «побачити» їх, але одночасно його важко вбити привидами. Із простої модифікації – зміна функції оцінки станів, щоб вона враховувала не лише різницю поточної кількості поглинутих харчових гранул та кількості кроків по карті.

**Власна функція оцінки станів:** Основна ідея функції оцінки станів полягає у тому, що Пакмен при виконанні алгоритму майже не переміщається при віддаленості від харчових гранул. Додамо відстань від харчових гранул до Пакмена. Чим далі Пакмен від їжі, тим гірше, тому від загальної оцінки стану поля буде відніматися манхетенська відстань від їжі. Враховується середня відстань не до усіх харчових гранул, а до 5 частини найближчих, щоб не було ситуацій по типу Пакмен не може визначитися, до якої купи харчових гранул йти, і рухається на місці.

Також були спроби аналогічно вимірювати відстань до привидів, але це стимулювало до схожої поведінки Пакмена, як до початку модифікації функції, тому такі спроби було відкинуто.

Результати наведені у розділі «Задача дослідження впливу параметра алгоритму», загалом модифікація має не гірші результати кількості виграшів та кращі результати оцінки на кінці кожної гри, на карті mediumClassic перевага по оцінці становила 165% на глибині алгоритму 3.

**Реалізація власної функції оцінки станів:**

def getFoodPos(currentGameState: GameState):

food = currentGameState.getFood()

res = []

for i in range(food.width):

for j in range(food.height):

if food[i][j]:

res.append((i, j))

return res

def manhattanDistanceSum(startPos, arrPos):

distance = []

for pos in arrPos:

distance.append(manhattanDistance(startPos, pos))

distance.sort(reverse=False)

right\_edge = int(len(distance)/5)

return float(sum(distance[0:right\_edge]))/(right\_edge+1)

def betterEvaluationFunction(currentGameState: GameState) -> float:

pacmanPos = currentGameState.getPacmanPosition()

foodPos = getFoodPos(currentGameState)

return currentGameState.getScore() - manhattanDistanceSum(pacmanPos, foodPos)

**Задача дослідження впливу параметра алгоритму чи фактору середовища:** Проведемо серію експериментів для різної глибини та різних функцій оцінки станів. Приклад запуску тестування на карті smallClassic з глибиною 4 з модифікованою функцією оцінки станів та кількістю запусків гри 100:

python pacman.py -l smallClassic -p MinimaxAgent -a depth=4,evalFn=better -q -n 100

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Карта | Стандартна функція | | Модифікована функція | | Глибина |
| Виграші | Оцінка | Виграші | Оцінка |
| minimaxClassic | 69/100 | 201.24 | 76/100 | 272.29 | 1 |
| 50/100 | 9.18 | 72/100 | 233.06 | 2 |
| 45/100 | -41.57 | 65/100 | 160.55 | 3 |
| 63/100 | 142.55 | 74/100 | 253.43 | 4 |
| smallClassic | 48/100 | 426.88 | 66/100 | 578.29 | 1 |
| 45/100 | 407.86 | 78/100 | 781.47 | 2 |
| 67/100 | 700.09 | 80/100 | 760.72 | 3 |
| 59/100 | 672.25 | 81/100 | 898.7 | 4 |
| openClassic | 18/20 | 199.7 | 20/20 | 1012.8 | 1 |
| 19/20 | 610.35 | 19/20 | 1036.05 | 2 |
| 19/20 | 844.5 | 19/20 | 1106.3 | 3 |
| mediumClassic | 9/20 | 580.7 | 10/20 | 883.1 | 1 |
| 15/20 | 1102.0 | 19/20 | 1581.5 | 2 |
| 13/20 | 992.4 | 17/20 | 1640.75 | 3 |
| 10/20 | 1005.0 | 17/20 | 1636.5 | 4 |

Тестування проведено на різних картах з глибиною алгоритму від 1 до 4 крім карти OpenClassic, де виконання програми з глибиною 4 тривало невиправдано довго.

На карті minimaxClassic різниця доволі випадкова між різною глибиною, закономірність прослідкувати неможливо.

На карті smallClassic для стандартної функції оцінки є значне покращення між глибиною 2 і 3, а для модифікованої – між глибиною 1 та 2, подальше збільшення глибини майже не приносило переваги, а у випадку стандартної функції навіть зменшило кількість виграшів, що може бути у межах випадкової похибки.

На карті openClassic для стандартної функції оцінки станів видиме покращення оцінки від глибини 1 до 2 і не таке значне від 2 до 3, але кількість виграшів майже не відрізняється; для модифікованої функції глибина майже не впливала на результати алгоритму.

На карті mediumClassic для стандартної функції оцінки значне покращення від глибини 1 до 2 і далі при збільшенні глибини результат лише погіршувався, що може бути частково випадковою похибкою; для модифікованої функції також значне покращення від глибини 1 до 2 і далі різниця майже не відчутна по оцінці, але менша кількість перемог.

Загалом у середньому найкраще поліпшення відбувається при переході від глибини 1 до 2, тому рекомендованим значенням глибини можна вважати саме 2. Подальше непостійне покращення оцінки алгоритму при збільшенні глибини нівелюється часом виконання, який зростає експоненційно за степенем глибини, часова складність алгоритму O(b^d).