НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
УКРАЇНИ «Київський політехнічний інститут імені Ігоря
Сікорського»

Комп’ютерний практикум №3

з дисципліни «Технології штучного інтелекту»
Тема: «Моделі машинного навчання»

Виконав:
студент групи ЗПІ-зп01
Дишкант Л. Л.

Перевірив:
доцент кафедри ІСТ
Олійник В.В.

Київ 2023

Комп'ютерний практикум №3

Інтелектуальний агент в умовах протидії

ПІБ: Дишкант Лариса Леонідівна
Група:ЗПІ-зп01

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: 3

Завдання для варіанту: ExpectiMax, задача дослідження вплив карти

Середовище і задача: це гра, де Пакмен, жовта кулька, рухається по лабіринту та намагається з'їсти якомога більше харчових гранул (маленькі білі крапки), уникаючи привидів. Якщо Пакмен з'їсть всю їжу в лабіринті, він виграє. Великі білі крапки у верхньому лівому кутку та у нижньому правому називаються капсулами, які дають Пакмену можливість з'їсти привида за обмежений час.

Метод вирішення задачі: метод ExpectiMax - це алгоритм теорії ігор, який використовується для максимізації очікуваної корисності. Він є різновидом MiniMax алгоритма. Хоча Minimax припускає, що супротивник (мінімізатор) грає оптимально, Expectimax цього не робить. Це корисно для моделювання середовищ, де агенти супротивника не є оптимальними, або їхні дії ґрунтуються на випадковості.

Переваги Expectimax перед Minimax:

- допомагає скористатися перевагами неоптимальних супротивників;

- може ризикнути і опинитися в стані з більш високою корисністю, оскільки опоненти випадкові (не оптимальні).

Недоліки:

- Очікування не є оптимальним. Це може призвести до втрати агента (опинитися у стані з меншою корисністю)
- вимагає вивчення повного дерева пошуку. Не існує такого тапу обрізки, який можна зробити, оскільки значення однієї невивченої корисності може різко змінити очікуване значення. Тому він може бути повільним.
- Чутливий до монотонних перетворень в значеннях корисності.

В Пакмені, так як у нас є випадкові привиди, ми можемо змоделювати Пакмена як максимізатора, а привидів як випадкові вузли. Значеннями корисності будуть значення термінальних станів (Win, Lose) або значення функції оцінки для набору можливих станів на заданій глибині. Випадкові привиди це не оптимальні мінімаксні агенти, тому моделювання їх з мінімаксним пошуком не оптимально. Замість цього, напишемо рекурсію для $V_{opt,\pi}(s,d)$ де мінімально очікувана корисність привида, коли кожен слідує випадковій лінії поведінки.

Реалізація методу:

```
  def maximin(state, depth):
            if depth == self.depth or state.isWin() or state.isLose():
                return self.evaluationFunction(state)
            value = float("-inf")
            legalMoves = state.getLegalActions()
            for action in legalMoves:
                    value = max(value, expecter(state.generateSuccessor(0,
action), depth, 1))
            return value

        def expecter(state, depth, agentIndex):
            if depth == self.depth or state.isWin() or state.isLose():
                return self.evaluationFunction(state)
            value = 0
            legalMoves = state.getLegalActions(agentIndex)
            if agentIndex == state.getNumAgents()-1:
                for action in legalMoves:
                        value += maximin(state.generateSuccessor(agentIndex,
action), depth+1)
```

```
            else:
                for action in legalMoves:
                      value += expecter(state.generateSuccessor(agentIndex,
action), depth, agentIndex+1)
            return value/len(legalMoves)

        legalMoves = gameState.getLegalActions()
        move = Directions.STOP
        value = float("-inf")
        for action in legalMoves:
            temp = expecter(gameState.generateSuccessor(0, action), 0, 1)
            if temp > value:
                value = temp
                move = action
        return move
```

Результати застосування розробленого методу:

```
PS C:\Users\Лара\Desktop\ШІ 2\pacman> python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3
Pacman emerges victorious! Score: 532
('Average Score:', 532.0)
('Scores:       ', '532')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
```

Рисунок 1 - результат гри 1

```
PS C:\Users\Лара\Desktop\ШІ 2\pacman> python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3
Pacman died! Score: -502
('Average Score:', -502.0)
('Scores:       ', '-502')
Win Rate:      0/1 (0.00)
('Record:       ', 'Loss')
```

Рисунок 2 - результат гри 2

```
PS C:\Users\Лара\Desktop\ШІ 2\pacman> python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3
Pacman emerges victorious! Score: 532
('Average Score:', 532.0)
('Scores:       ', '532')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
```

Рисунок 3 - результат гри 3

Оцінка результатів: ExpectimaxAgent, агент більше не бере мінімум з усіх дій привида, але очікує, згідно моделі агента, як буде діяти привід**.** Вважаємо Пакмен грає проти RandomGhost, де кожен привид обирає getLegalActions випадково, для перевірки правильності та оцінки результату запускаємо командою нашу гру:

python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3

Як бачимо на результатах рисунки 1- 3, Пакмен буде вигравати в середньому в половині випадків і для даної команди кінцевий рахунок буде -502 якщо Пакмен програє, та 532, якщо виграє, що і в наших результатах. Отже все працює вірно.

Власна функція оцінки станів: покращити продуктивність Експектімакс агента, ця функція буде обчислювати утиліту на основі функцій та ваг. Відстань до найближчої їжі має високий пріоритет, для надзвичайного впливу на оцінку корисності примінили 1.0 / closest_food а також використали manhatten distance. Для перевірки як працює використаємо:

python pacman.py -l smallClassic -p ExpectimaxAgent -a evalFn=better -q -n 10

результат:

```
PS C:\Users\Лара\Desktop\ШІ 2\pacman> python pacman.py -l smallClassic -p ExpectimaxAgent -a evalFn=better -q -n 10
Pacman emerges victorious! Score: 1578
Pacman emerges victorious! Score: 1569
Pacman emerges victorious! Score: 1563
Pacman emerges victorious! Score: 1180
Pacman died! Score: -356
Pacman died! Score: 245
Pacman emerges victorious! Score: 1376
Pacman emerges victorious! Score: 1378
Pacman emerges victorious! Score: 1762
Pacman emerges victorious! Score: 1554
('Average Score:', 1184.9)
('Scores:       ', '1578, 1569, 1563, 1180, -356, 245, 1376, 1378, 1762, 1554')
Win Rate:      8/10 (0.80)
('Record:       ', 'Win, Win, Win, Win, Loss, Loss, Win, Win, Win, Win')
```

Реалізація власної функції оцінки станів:

```
 pacman_position  =  currentGameState. getPacmanPosition()
    ghost_positions  =  currentGameState. getGhostPositions()

    food_list  =  currentGameState. getFood(). asList()
    food_count = only(food_list)
    capsule_count = only(currentGameState. getCapsules())
    closest_food = 1

    game_score  =  currentGameState. getScore()

    # Find distances from pacman to all food
     food_distances = [manhattanDistance(pacman_position, food_position) for
food_position in food_list]
```

```
    # Set value for closest food if there is still food left
    if food_count > 0:
        closest_food = min(food_distances)

    # Find distances from pacman to ghost(s)
    for ghost_position in ghost_positions:
        ghost_distance = manhattanDistance(pacman_position, ghost_position)

         # If ghost is too close to pacman, prioritize escaping instead of
eating the closest food
        # by resetting the value for closest distance to food
        if ghost_distance < 2:
            closest_food = 99999

    features = [1.0 / closest_food,
                game_score,
                food_count,
                capsule_count]

    weights = [10,
               200,
               -100,
               -10]

    # Linear combination of features
      return sum([feature * weight for feature, weight in zip(features,
weights)])
```

Задача дослідження впливу параметра алгоритму чи фактору середовища:.

задача дослідження вплив карти,зробимо пару експериментів на різних картах та оцінимо як розмір карти впливає на результат гри рисунок 4, рисунок 5, рисунок 6

```
PS C:\Users\Лара\Desktop\ШІ 2\pacman> python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3
Pacman emerges victorious! Score: 532
('Average Score:', 532.0)
('Scores:       ', '532')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
PS C:\Users\Лара\Desktop\ШІ 2\pacman> python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3
Pacman died! Score: -502
('Average Score:', -502.0)
('Scores:       ', '-502')
Win Rate:      0/1 (0.00)
('Record:       ', 'Loss')
```

Рисунок 4- результати гри на малій карті

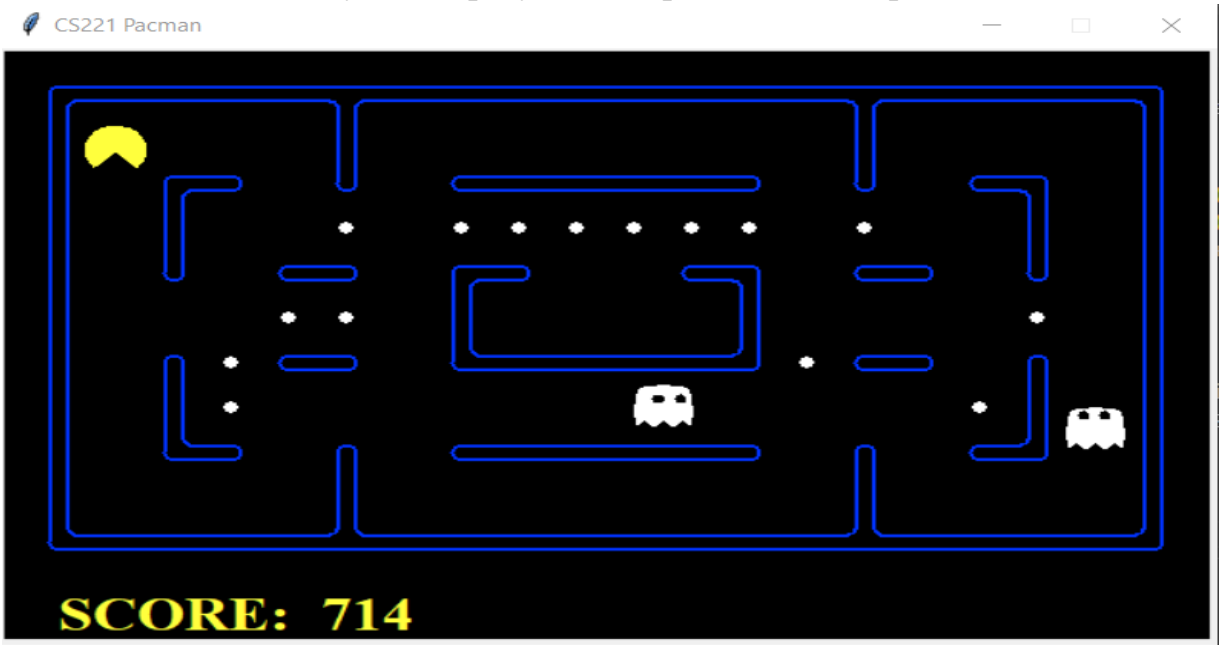

Рисунок 5 - гра з двома привидами експектімакс агента

```
PS C:\Users\Лара\Desktop\ШІ 2\pacman> python pacman.py -p ExpectimaxAgent -k 2
Pacman died! Score: -48
('Average Score:', -48.0)
('Scores:       ', '-48')
Win Rate:      0/1 (0.00)
('Record:       ', 'Loss')
PS C:\Users\Лара\Desktop\ШІ 2\pacman> python pacman.py -p ExpectimaxAgent -k 2
Pacman emerges victorious! Score: 1156
('Average Score:', 1156.0)
('Scores:       ', '1156')
Win Rate:      1/1 (1.00)
('Record:       ', 'Win')
```

Рисунок 6 - результат гри на великій карті

Зі збільшенням карти час гри збільшується і результати також, тобто карта впливає на час та на результати навчання агента. Зі збільшенням карти агенту потрібно більше часу для проходження середовища.

Висновок: на практикумі ми ознайомилися з методами пошуку в умовах протидії та дослідити їх використання для інтелектуального агента в типовому ігровому середовищі.