Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Програмування інтелектуальних інформаційних систем

**ЗВІТ**

До лабораторної роботи №3.

Тема: Рефлексія

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Виконав**  **студент** |  | ІТ-92, Бондаренко Дмитро Сергійович |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | ас. Баришич Л.М. |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2021

# **Завдання лабораторної роботи**

Темою цієї роботи є написання алгоритмів Minimax з alpha-beta pruning та Expectimax. Також ми напишемо другий тип ворогів, який шукає дорогу до гравця. Для алгоритмів буде стоврений файл, у який ми будемо записувати дані по грі, такі як тип алгоритму, чи гра переможна, час гри та кількість очок. Також ми напишемо функцію, яка буде оцінювати «силу» поточної позиції PacMan – чим більше число, тим краща позиція. Ця функція потрібна для того, щоб наші алгоритми працювали коректно видаючи при цьому значення, яке буде слугувати деякою «нагородою» за перехід у нове місце на ігровому полі.

# **Опис використаних технологій**

У цій роботі буде використана мова програмування Python, оскільки вона має зручну бібліотеку для відтворення графічного інтерфейсу, а й власне гри, tkinter.

# **Опис програмного коду**

Робота базується на грі, яку ми написали у минулих роботах. У цій лабораторній роботі ми додали один новий файл multiAgents.py, у якому ми реалізували всіх мультиагентів.

аа

У цьому файлі є представлена така іерархія коду:

* Метод scoreEvaluationFunction – оцінка "сили" поточної позиції. За увагу береться найближча відстань до їжі, найближча відстань до привидів та поточна кількість очок.
* Клас MultiAgentSearchAgent – базовий (абстрактний клас) для мультиагентів.
* Клас AlphaBetaAgent – агент minimax із alpha-beta pruning.
  + Метод getAction – повернення наступної дії, в залежності від наведеного алгоритму.
  + Метод term – перевірка того, чи є статус термінальним.
  + Метод min\_value – гравець мінімізатор (привиди).
  + Методmax\_value – гравець максимізатор (PacMan).
  + Метод alphabeta – вхідна точка в алгоритм. Тут же і відбувається Pruning.
* Клас ExpectimaxAgent – агент Expectimax.
  + Метод getAction – повернення наступної дії, в залежності від наведеного алгоритму.
  + Метод term – перевірка того, чи є статус термінальним.
  + Метод exp\_value – гравець мінімізатор (Наступний крок буде залежати від деякої вірогідності)
  + Метод max\_value – гравець максимізатор (PacMan).

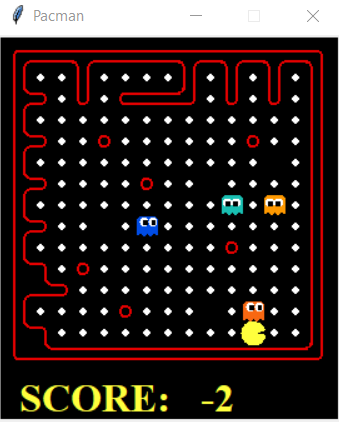
# Оцінка "сили" поточної позиції.  
# За увагу береться найближча відстань до їжі, найближча відстань до привидів та поточна кількість очок  
def scoreEvaluationFunction(currentGameState):  
 newPos = currentGameState.getPacmanPosition()  
 newFood = currentGameState.getFood()  
 newGhostStates = currentGameState.getGhostStates()  
 newFood = newFood.asList()  
 ghostPos = [(G.getPosition()[0], G.getPosition()[1]) for G in newGhostStates]  
  
 if currentGameState.isLose():  
 return -10000  
  
 if newPos in ghostPos:  
 return -10000  
  
 closestFoodDist = sorted(newFood, key=lambda fDist: util.manhattanDistance(fDist, newPos))  
 closestGhostDist = sorted(ghostPos, key=lambda gDist: util.manhattanDistance(gDist, newPos))  
  
 score = 0  
  
 fd = lambda fDis: util.manhattanDistance(fDis, newPos)  
 gd = lambda gDis: util.manhattanDistance(gDis, newPos)  
  
 if gd(closestGhostDist[0]) < 3:  
 score -= 300  
 if gd(closestGhostDist[0]) < 2:  
 score -= 1000  
 if gd(closestGhostDist[0]) < 1:  
 return -10000  
  
 if len(currentGameState.getCapsules()) < 2:  
 score += 100  
  
 if len(closestFoodDist) == 0 or len(closestGhostDist) == 0:  
 score += currentGameState.getScore() + 10  
 else:  
 score += (currentGameState.getScore() + 10 / fd(closestFoodDist[0]) + 1 / gd(closestGhostDist[0]) +  
 1 / gd(closestGhostDist[-1]))  
  
 return score  
  
  
# Базовий (абстрактний клас) для мультиагентів  
class MultiAgentSearchAgent(Agent):  
 def \_\_init\_\_(self, evalFn='scoreEvaluationFunction', depth='3'):  
 super().\_\_init\_\_()  
 self.index = 0 # Pacman is always agent index 0  
 self.evaluationFunction = util.lookup(evalFn, globals())  
 self.depth = int(depth)  
  
  
# Агент minimax із alpha-beta pruning  
class AlphaBetaAgent(MultiAgentSearchAgent):  
  
 # Повернення наступної дії, в залежності від наведеного алгоритму  
 def getAction(self, gameState):  
 GhostIndex = [i for i in range(1, gameState.getNumAgents())]  
  
 # Перевірка того, чи є статус термінальним  
 def term(state, d):  
 return state.isWin() or state.isLose() or d == self.depth  
  
 # Min гравець (Привиди)  
 def min\_value(state, d, ghost, A, B):  
  
 if term(state, d):  
 return self.evaluationFunction(state)  
  
 v = math.inf  
 for action in state.getLegalActions(ghost):  
 if ghost == GhostIndex[-1]:  
 v = min(v, max\_value(state.generateSuccessor(ghost, action), d + 1, A, B))  
 else:  
 v = min(v, min\_value(state.generateSuccessor(ghost, action), d, ghost + 1, A, B))  
  
 if v < A:  
 return v  
 B = min(B, v)  
  
 return v  
  
 # Max гравець (Pacman)  
 def max\_value(state, d, A, B):  
  
 if term(state, d):  
 return self.evaluationFunction(state)  
  
 v = -math.inf  
 for action in state.getLegalActions(0):  
 v = max(v, min\_value(state.generateSuccessor(0, action), d, 1, A, B))  
  
 if v > B:  
 return v  
 A = max(A, v)  
 print(self.evaluationFunction(state))  
 return v  
  
 # Вхідна точка в алгоритм. Тут же і відбувається Pruning  
 def alphabeta(state):  
  
 v = -math.inf  
 act = None  
 A = -math.inf  
 B = math.inf  
  
 for action in state.getLegalActions(0):  
 tmp = min\_value(gameState.generateSuccessor(0, action), 0, 1, A, B)  
  
 if v < tmp:  
 v = tmp  
 act = action  
  
 if v > B:  
 return v  
 A = max(A, tmp)  
  
 return act  
  
 return alphabeta(gameState)  
  
  
# Агент Expectimax  
class ExpectimaxAgent(MultiAgentSearchAgent):  
  
 # Повернення наступної дії, в залежності від наведеного алгоритму  
 def getAction(self, gameState):  
  
 GhostIndex = [i for i in range(1, gameState.getNumAgents())]  
  
 # Перевірка того, чи є статус термінальним  
 def term(state, d):  
 return state.isWin() or state.isLose() or d == self.depth  
  
 # Min гравець (Наступний крок буде залежати від деякої вірогідності)  
 def exp\_value(state, d, ghost):  
  
 if term(state, d):  
 return self.evaluationFunction(state)  
  
 v = 0  
 prob = 1 / len(state.getLegalActions(ghost))  
  
 for action in state.getLegalActions(ghost):  
 if ghost == GhostIndex[-1]:  
 v += prob \* max\_value(state.generateSuccessor(ghost, action), d + 1)  
 else:  
 v += prob \* exp\_value(state.generateSuccessor(ghost, action), d, ghost + 1)  
 return v  
  
 # Max гравець (Pacman)  
 def max\_value(state, d): # maximizer  
  
 if term(state, d):  
 return self.evaluationFunction(state)  
  
 v = -math.inf  
 for action in state.getLegalActions(0):  
 v = max(v, exp\_value(state.generateSuccessor(0, action), d, 1))  
 print(self.evaluationFunction(state))  
 return v  
  
 res = [(action, exp\_value(gameState.generateSuccessor(0, action), 0, 1)) for action in  
 gameState.getLegalActions(0)]  
 res.sort(key=lambda k: k[1])  
  
 return res[-1][0]

Логіка логування була додана до методу process із файлу pacman.py. Це метод перевіряє, чи пора закінчувати гру, якщо так, то записує інформацію про гру у файл та виводить відповідне повідомлення:

# Перевіряє, чи пора закінчувати гру, якщо так, то запишемо інформацію про гру у файл та  
# виведемо відповідне повідомлення.  
def process(self, state, game, timestart):  
 if (state.isWin() or state.isLose()) and (game.agents[0].\_\_class\_\_.\_\_name\_\_ == 'ExpectimaxAgent' or  
 game.agents[0].\_\_class\_\_.\_\_name\_\_ == 'AlphaBetaAgent'):  
 #header = ['AlgorithmAgent', 'IsWon', 'GameTime', 'Points']  
 data = [game.agents[0].\_\_class\_\_.\_\_name\_\_, state.isWin(), datetime.now() - timestart, state.data.score]  
  
 with open('stats.csv', 'a', encoding='UTF8') as f:  
 writer = csv.writer(f)  
  
 # write the header  
 #writer.writerow(header)  
  
 # write the data  
 writer.writerow(data)  
 if state.isWin():  
 self.win(state, game)  
 if state.isLose():  
 self.lose(state, game)

# **Скріншоти роботи програмного застосунку**

Під час роботи ми додали новий тип привидів, які шукають шлях до PacMan. Цей тип ворогів базується Евклідовій квадратичній відстані. Для того, щоб додати цей тип ворогів ми маємо вказати у параметрах його назву як параметр (–g DirectionalGhost) та кількість привидів (наприклад –k 2). Логіка така, що ми вказуємо кількість ворогів вказаного типу, а потім на поле (в залежності від вибраного поля, кількість ворогів може бути строго визначена) додається така ж кількість ворогів, які рухаються випадково. У нашому випадку параметри будуть наступні: -g DirectionalGhost -k 2 -l randomClassic, де –l randomClassic – випадкова мапа із грою класичного типу:



При цьому привиди синього та помаранчового кольору будуть постійно атакувати PacMan, а інші два будуть рухатися випадково.

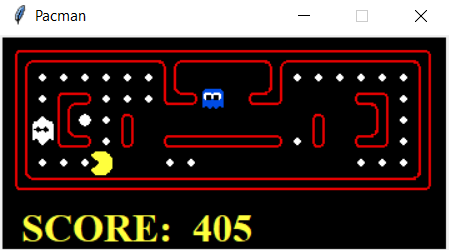
Тепер розберемо безпосередньо роботу самих алгоритмів.

**Minimax з alpha-beta pruning**

Для початку варто зрозуміти, що із себе представляє minimax. Minimax - це свого роду алгоритм зворотного відстеження, який використовується у прийнятті рішень та теорії ігор для пошуку оптимального ходу для гравця (максимізатора), припускаючи, що його опонент (мінімізатор) також грає оптимально. Алгоритм мінімакс може обрізати велике дерево пошуку, застосувавши налаштування, відоме як alpha-beta pruning, що може значно скоротити час обчислень. Ця зміна дозволяє значно швидше виконувати пошук, вирізаючи непотрібні гілки, які не потрібно шукати, оскільки вже існує кращий доступний хід. Назва пояснюється тим, що для мінімаксної функції потрібно передати 2 додаткові параметри, а саме-альфа та бета.

* Альфа – це найкраще значення, яке зараз максимізатор може гарантувати на цьому рівні або вище.
* Бета – це найкраще значення, яке на даний момент мінімізатор може гарантувати на цьому рівні або вище.

Два нові параметри використовуються для обрізки тих гілок, які не можуть вплинути на остаточне рішення. Щоразу, коли максимальний бал, у якому гравець-мінімізатор (бета-гравець) впевнений, стане меншим за мінімальний бал, у якому впевнений гравец-максимізатор (альфа), тобто, якщо бета<альфа, гравцю-максимізатору не потрібно розглядати подальших нащадків цього вузла, оскільки вони ніколи не будуть досягнуті у реальній грі. При цьому такий алгоритм буде швидше, ніж звичайний minimax оскільки у нас буде менше гілок для перевірки. В нашому випадку, при цьому, глибина дерева буде 3:



При цьому отримаємо наступний лог у .csv файл:

AlgorithmAgent,IsWon,GameTime,Points

AlphaBetaAgent,True,0:00:28.757097,1698

Expectimax

Алгоритм пошуку Expectimax - це алгоритм теорії ігор, який використовується для максимізації очікуваної корисності. Це різновид алгоритму Minimax. Хоча Minimax припускає, що противник (мінімізатор) грає оптимально, Expectimax - ні.

Цей алгоритм враховує також невизначеність навколишнього середовища. Вузли максимізаторів дерева працюють як у minimax, але вузли мінімізаторів замінюються випадковими вузлами, значення яких не є значенням мінімуму їхніх дочірніх елементів, але воно буде обчислюватися як очікування вартості їхніх дочірніх елементів.

Цей алгоритм, виходячи із того, що він залежить не від опонента а від деякого шансу, працює краще, за умови випадкового пересування ворогів. На відміну від minimax із alpha beta pruning ми не можемо обрізати шляхи до деяких вершин оскільки ми не маємо точних меж можливих значень. Для перевірки алгоритму ми знову оберемо глибину дерева 3:



При цьому отримаємо наступний лог у .csv файл:

AlgorithmAgent,IsWon,GameTime,Points

ExpectimaxAgent,True,0:00:23.531719,1538

**Висновок:** У цій роботі ми реалізували алгоритми minimax з alpha-beta pruning та expectimax. При цьому, ми зрозуміли, що expecitmax буде краще працювати, якщо твій опонент не грає оптимально (тобто випдково), але алгоритм при цьому буде працювати повільніше, тому, що ми не можемо проігнорувати прохід по деяким вершинам, як ми це можемо зробити у minimax з alpha-beta pruning. Також ми створили агента для привидів, який атакує Pacman, написали функцію оцінки «сили» на основі якої працюють написані у цій роботі алгоритми, а також написали логіку логування інформації про гру у файл під назвою stats.csv.