PACMAN - MultiAgent

Salajan Madalina, Negrut Ciprian

December 4, 2022

Abstract

In acest proiect, Pacman trebuie sa gaseasca drumul optim prin labirint pentru a ajunge la o anumita locatie si pentru a colecta mancarea in cat mai putini pasi. Pe parcursul acestui proiect ne-am imbunatatit gandirea prin implementarea unor noi algoritmi de decizie care il ajuta pe Pacman sa obina un scor cat mai bun cand are de a face cu fantomele.

1 Introducere

Pacman este unul dintre cele mai populare jocuri din lume. Goal-ul acestui joc este de a avea un scor cat mai mare mancand toate punctele din labirint.

Am implementat algoritmi de cautare neinformati care l-au ajutat pe Pacman sa isi indeplineasca visul. Insa acesti algoritmi nu l-au ajutat pe Pacman sa obtina un scor prea mare din cauza ca nu sunt prea eficienti. Prin urmre, am fost nevoiti sa implementam algoritmi de cautare informati care sa maximizeze scorul lui Pacman si sa il ajute sa faca fata fantomelor.

2 Cerinte

2.1 Cerinta 1: Reflex Agent

In REFLEX AGENT, prin metoda evaluationFunction se genereaza un scor pentru urmatoarea mutare a fiecarui agent. In calcularea scorului, am tinut cont: distanta minima pana la mancare, distanta maxima pana la mancare, distanta minima pana la fantoma si daca pacman-ul a mancat un dot.

Am rulat urmatoarele comenzi pentru a testa implementarea:

- python pacman.py -p ReflexAgent -l testClassic
- python pacman.py –frameTime 0 -p ReflexAgent -k 1
- python pacman.py –frameTime 0 -p ReflexAgent -k 2
- python autograder.py -q q1 –no-graphics

Putem observa rezultatele in Table 1.

Maze form	Depth	Score
Test Classic	-	1115
With two ghost	-	-248

Table 1: Reflex Agent results.

```
def evaluationFunction(self, currentGameState: GameState, action):
    successorGameState = currentGameState.generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood()
    newGhostStates = successorGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

    maxDistanceToFoods = 0
    minDistanceToFoods = 0
```

```
distanceToGhosts = [manhattanDistance(newPos, gp) for gp in successorGameState.getGhostPos
distanceToFoods = [manhattanDistance(newPos, gp) for gp in newFood.asList()]
if len(distanceToFoods) != 0:
    maxDistanceToFoods = max(distanceToFoods)
    minDistanceToFoods = min(distanceToFoods)

minDistanceToGhosts = min(distanceToGhosts)

for action in currentGameState.getLegalActions(0):
    if currentGameState.generateSuccessor(0, action).getPacmanPosition() in newFood.asList
        return successorGameState.getScore() - (maxDistanceToFoods + minDistanceToFoods +
               minDistanceToGhosts
return successorGameState.getScore() - (maxDistanceToFoods + minDistanceToFoods) + max(dis
```

2.2 Cerinta 2: Minimax

Algortimul MINIMAX se imparte in doua parti. MAX vrea sa gaseasca o solutie pentru a obine o victorie, dar MIN are rolul de a-l incurca pe MAX. In algoritmul nostru MAX este PACMAN, iar MIN este/sunt GHOST-urile. La fiecare miscare de a lui MIN, MAX trebuie sa aleaga valoarea cea mai mica, iar MAX, la fiecare miscare a lui MIN, trebuie sa aleaga valoarea cea mai mare, adica cu cel mai bun scor pentru PACMAN. In cazul in care avem mai mult de o fantoma [GHOST], trebuie sa lasam si restul fantomelor sa ia o decizie la urmatoarea miscare, iar ulterior pacmanul.

Am rulat urmatoarele comenzi pentru a testa implementarea:

- python autograder.py -q q2
- python autograder.py -q q2 –no-graphics
- python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
- python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3

Putem observa rezultatele in Table 2. Codul este prezentat in Figura **??**.

Maze form	Depth	Score
Test Classic	4	84
Minimax Clasic	4	516
Trapped Classic	3	-501

Table 2: Minimax results.

```
def max_value(gameState, pacman, depth, move):
    if gameState.isLose() or gameState.isWin() or depth == self.depth:
        return self.evaluationFunction(gameState), None
    v = float("-inf")

    for action in gameState.getLegalActions(pacman):
        v2, a2 = min_value(gameState.generateSuccessor(pacman, action), 1, depth, move)
        if v2 > v:
            v, move = v2, action
    return v, move

def min_value(gameState, indexAgent, depth, move):
    if gameState.isLose() or gameState.isWin() or depth == self.depth:
        return self.evaluationFunction(gameState), None
    v = float("+inf")
    if indexAgent + 1 == gameState.getNumAgents():
        for action in gameState.getLegalActions(indexAgent):
            v2, a2 = max_value(gameState.generateSuccessor(indexAgent, action),
                               0, depth + 1, move)
            if v2 < v:
                v, move = v2, action
```

```
        else:
            for action in gameState.getLegalActions(indexAgent):
                v2, a2 = min_value(gameState.generateSuccessor(indexAgent, action),
                            indexAgent + 1, depth, move)
                if v2 < v:
                    v, move = v2, action
        return v, move

    pacman = 0
    value, move = max_value(gameState, pacman, 0, Directions.STOP)
    return move
```

2.3 Cerinta 3: Alpha-Beta Pruning

Algortimul ALPHA-BETA PRUNING se bazeaza in mare parte pe algoritmul MINIMAX, dar se bazeaza pe reducerea expandarilor nodurilor urmatoare. Avem doi parametrii noi fata de MINIMAX, alfa si beta. Alfa jucand un rol de minim, iar beta un rol de maxim. In cazul in care pe nodul maxim deja avem un alfa [o valoare minima], iar cand expandam urmatorul nod, pe minim si gasim o valoare mai mica decat alfa, nu mai are rost sa expandam copii acelui nod, pentru ca nu o sa gasim o valoare mai mare decat alfa curent. Prin aceasta logica rezulta reducere numarului de noduri expandate.

Am rulat urmatoarele comenzi pentru a testa implementarea:

- python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
- python autograder.py -q q3
- python autograder.py -q q3 –no-graphics

Putem observa rezultatele in Table 3. Codul este prezentat in Figura **??**.

Maze form	Depth	Score
Small Classic	3	795
Test Clasic	-	84

Table 3: Alpha-Beta Pruning results.

```
    def max_value(gameState, pacman, depth, move, alfa, beta):
        if gameState.isLose() or gameState.isWin() or depth == self.depth:
            return self.evaluationFunction(gameState), None
        v = float("-inf")

        for action in gameState.getLegalActions(pacman):
            v2, a2 = min_value(gameState.generateSuccessor(pacman, action),
                        1, depth, move, alfa, beta)
            if v2 > v:
                v, move = v2, action
                alfa = max(alfa, v)
            if v > beta:
                return v, move
        return v, move

    def min_value(gameState, indexAgent, depth, move, alfa, beta):
        if gameState.isLose() or gameState.isWin() or depth == self.depth:
            return self.evaluationFunction(gameState), None
        v = float("+inf")

        if indexAgent + 1 == gameState.getNumAgents():
            for action in gameState.getLegalActions(indexAgent):
                v2, a2 = max_value(gameState.generateSuccessor(indexAgent, action),
                        0, depth + 1, move, alfa, beta)
                if v2 < v:
```

```
                v, move = v2, action
                beta = min(beta, v)
            if v < alfa:
                return v, move
        else:
            for action in gameState.getLegalActions(indexAgent):
                v2, a2 = min_value(gameState.generateSuccessor(indexAgent, action),
                        indexAgent + 1, depth, move,
                                alfa, beta)
                if v2 < v:
                    v, move = v2, action
                    beta = min(beta, v)
                if v < alfa:
                    return v, move
        return v, move

    pacman = 0
    alfa = float("-inf")
    beta = float("+inf")
    value, move = max_value(gameState, pacman, 0, Directions.STOP, alfa, beta)
    return move
```

References

In realizarea implementarii acestor algoritmi, descrisi mai sus, ne-au fost de mare ajutor:

- Cartea: "Artificial Intelligence, A Modern Approach - Fourth Edition" de Stuart Russell si Peter Norving (AIMA)
- Indrumatorul de laborator: "Intoduction to Artificial Intelligence" de Anca Marginean, Adrian Groza si Radu Razvan Slavescu