Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name:
Izabella Bartalus
Stefana-Bianca Chelemen
Group:
30231
Email:
bartalusiza08@gmail.com
stefanac288@gmail.com

Teaching Assistant: Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Chapter 1

A1: Search

1.1 Introducere

Pac-man este un joc cu mai multe entitati: Pacman, fantome, food-dots si power-pellets. Jucatorul il controleaza pe Pacman cu scopul de a manca toate food-dots-urile. Pacman moare in cazul in care este mancat de o fantoma, dar daca Pacman mananca o power-pellet, acesta va avea o abilitate temporara de a manca fantome. Scopul nostru este de a construi agenti care sa il controleze pe Pacman si de a castiga. Actiunile posibile sunt Nord, Sud, Est, Vest, Stop, depinzand de prezenta peretilor. Cu fiecare pas, Pacman pierde 1 punct, pentru fiecare food-dot mancat primeste 10 puncte, iar pentru terminarea jocului primeste 500 puncte.

1.2 Definirea problemei

Pentru a dezvolta jocul Pac-man, ne-am propus sa il imbunatatim prin implementarea unor algoritmi multi-agent-search precum ReflexAgent si ExpectimaxAgent.

Prin termenul multi-agent ne referim la un mediu competitiv in care actiunile agentilor sunt in conflict. In jocul nostru, mediul multi-agent este definit prin faptul ca fantomele din joc definesc fiecare cate un agent care isi "face planuri" impotriva agentului principal: Pacman. Fiecare agent isi alege actiunea curenta bazata pe propria perceptie, computand miscarea optima pana cand unul castiga.

Un reflex agent ia toate actiunile legale posibile, calculeaza scorul starilor accesibile cu aceste actiuni si selecteaza starile care rezulta intr-o stare cu scor maxim. In cazul in care mai multe stari au scor maxim, se alege random una dintre ele. Agentul inca pierde de multe ori. Adversarul agentului functioneaza pe principiul privirii inainte tinand cont de miscarile oponentului.

Intr-un mediu multi-agent precum cel specificat mai sus, putem folosi algoritmul de cautare Minimax, algoritm de cautare limitata in adancime. Plecand de la pozitia curenta, generam multimea de pozitii succesoare posibile. Un agent este numit MAX, iar celalalt MIN. Actiunile agentului MAX se adauga primele, apoi pentru fiecare stare rezultata se adauga actiunea MIN-urilor si asa mai departe. In acest scop, structura de date folosita este arborele. Se aplica functia de evaluare si se alege cea mai buna stare. Valoarea minimax asigura strategia optima pentru MAX.

Algoritmul Expectimax este o variatie a algoritmului Minimax, in care nodurile maxime functioneaza exact in acelasi mod in care functioneaza pentru Minimax, aici cu ajutorul acestor noduri se calculeaza media valorilor tuturor copiilor(succesorilor) al unui nod ales la intamplare. Deci MAX alege nodul in care avem sansa de a avea probabilitatea obtinerii unui scor mult mai mare.

1.3 Implementare

PSEUDOCOD pentru algoritmul Expectimax:
def value(state):
if the state is a terminal state: return the state's utility
if the next agent is MAX: return max_value(state)
if the next agent is EXP: return exp_value(state)
def max-value(state):
initialize v = -inf
for each successor of state
v = max(v,value(successor))
return v
def exp-value(state):
initialize v=0
for each successor of state
p = probability(successor)
v += p*value(successor)
return v

1.4 Testare si rezultate obtinute

ReflexAgent

Dupa cum s-a mentionat mai sus (v. Definirea problemei), un reflex agent ia toate actiunile legale posibile, calculeaza scorul starilor accesibile cu aceste actiuni si selecteaza starile care rezulta intr-o stare cu scor maxim. Pentru a imbunatati calitatile acestui agent in asa fel incat sa selecteze o actiune mai buna, am inclus in valoarea returnata de catre fiecare stare si locatia mancarii si cea a fantomelor. Astfel, acum luam in considerare distanta celui mai apropiat aliment fata de pozitia pe care o are Pacman in starea actuala, cat si pozitia fantomelor fata de pozitia lui Pacman. Toate aceste modificari au fost facute in cadrul fuctiei evaluationFunction din clasa ReflexAgent.

Testarea pe testClassic:

python2 pacman.py -p ReflexAgent -l testClassic

Rezultatul obtinut:
Pacman emerges victorious! Score: 480
Average Score: 480.0
Scores: 480.0
Win Rate: 1/1 (1.00)
Record: Win

Cu o singura fantoma:

python2 pacman.py –frameTime 0 -p ReflexAgent -k 1 -l mediumClassic

Rezultatul obtinut:
Pacman emerges victorious! Score: 1066

Average Score: 1066.0
Scores: 1066.0
Win Rate: 1/1 (1.00)
Record: Win

Cu doua fantome:

python2 pacman.py –frameTime 0 -p ReflexAgent -k 2 -l mediumClassic

Rezultatul obtinut:
Pacman died! Score: 246
Average Score: 246.0
Scores: 246.0
Win Rate: 0/1 (0.00)
Record: Loss

Deoarece este o functie de evaluare medie, agentul va pierde in majoritatea cazurilor in care exista doua fantome.

ExpectimaxAgent

Algoritmul de cautare Expectimax este un algoritm utilizat pentru a maximiza utilitatea asteptata. Este o varianta a algoritmului Minimax. In timp ce Minimax presupune ca adversarul (minimizatorul) joacă in mod optim, Expectimax nu face aceasta presupunere. Acest lucru este util pentru modelarea mediului multi-agent in care agentii adversari nu sunt optimi sau actiunile lor se bazeaza pe intamplare.

Testare pe mediumClassic:

python2 pacman.py -p ExpectimaxAgent -l mediumClassic
Pacman emerges victorious! Score: 1062
Average Score: 1062.0
Scores: 1062.0
Win Rate: 1/1 (1.00)
Record: Win

Testarea implementarii la anumite adancimi si pe anumite layout-uri:

Adancime = 4 si layout = minimaxClassic:

python2 pacman.py –frameTime 0 -p ExpectimaxAgent -l minimaxClassic -a depth=4

Rezultatul obtinut:
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores: 516.0
Win Rate: 1/1 (1.00)
Record: Win

Adancime = 2 si layout = mediumClassic:

python2 pacman.py -p ExpectimaxAgent -l mediumClassic -a depth=2

Pacman died! Score: 292
Average Score: 292.0
Scores: 292.0
Win Rate: 0/1 (0.00)
Record: Loss

Adancime = 2 si layout = testClassic:

python2 pacman.py –frameTime 0 -p ExpectimaxAgent -l testClassic -a depth=2

Rezultatul obtinut:
Pacman emerges victorious! Score: 498
Average Score: 498.0
Scores: 498.0
Win Rate: 1/1 (1.00)
Record: Win

Adancime = 3 si layout = trappedClassic:

python2 pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3

Rezultatul obtinut:
Pacman emerges victorious! Score: 532
Average Score: 532.0
Scores: 532.0
Win Rate: 1/1 (1.00)
Record: Win

1.5 Concluzii

Pentru a face o comparatie intre algoritmii Minimax si Expectimax, o sa ii testam pe fiecare de cate 5 ori pe layout-ul minimaxClassic, utilizand urmatoarele adancimi: 2, 3, 4.

Adancimea	**Tip agent**	**Rata castig**	**Medie scor**
2	Minimax	2/5	-93
2	ExpectiMax	3/5	107
3	Minimax	2/5	-91
3	Expectimax	3/5	106
4	MinimaxAgent	4/5	324
4	Expectimax	5/5	514

O a doua comparatie pe care o sa o facem intre cei doi algoritmi va fi tot in functie de adancimi: 2, 3, 4 dar de data aceasta vom folosi layout-ul trappedClassic.

Adancimea	**Tip agent**	**Rata castig**	**Medie scor**
2	Minimax	3/5	118
2	ExpectiMax	3/5	118
3	Minimax	0/5	-501
3	Expectimax	4/5	325
4	MinimaxAgent	0/5	-501
4	Expectimax	3/5	118

Precum ne asteptam, agentul Expectimax a avut o rata de castig mai mare si totodata un scor mai bun in comparatie cu agentul Minimax.

La fel cum am precizat si in capitolele anterioare, algoritmul Expectimax este o variatie a alogoritmului Minimax, dar, in timp ce Minimax presupune ca adversarul joaca in mod optim, Expectimax nu pleaca de la aceasta presupunere. Acest lucru este folositor penntru jocul nostru deoarece fantomele sunt agenti neoptimali, actiunile lor fiind bazate pe sansa. Deci, spre deosebire de Minimax, Expectimax isi poate asuma un risc" si poate ajunge intr-o stare cu o utilitate mai mare, deoarece adversarii sunt aleatori (nu optimi). Cu toate acestea Expectimax nu este un algoritm optim. Poate avea ca rezultat pierderea jocului (ajungand intr-o stare cu utilitate mai mica).

Appendix A

Original code Search

```
class ReflexAgent(Agent):

    def getAction(self, gameState):

         # Collect legal moves and successor states
        legalMoves = gameState.getLegalActions()

        # Choose one of the best actions
        scores = [self.evaluationFunction(gameState, action) for action in legalMoves] #pt fiecare actiune legala calculeaza scorul
        bestScore = max(scores)
        bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore] #lista cu indicii care au scor maxim
        chosenIndex = random.choice(bestIndices) # Pick randomly among the best

        return legalMoves[chosenIndex]

    def evaluationFunction(self, currentGameState, action):

        # returneaza starea succesoare a jocului dupa ce un agent efectueaza o actiune
        successorGameState = currentGameState.generatePacmanSuccessor(action)
        # pozitia fantomelor in starea succesoare
        newGhostStates = successorGameState.getGhostStates()
        # pozitia Pacmanului in starea succesoare
        newPos = successorGameState.getPacmanPosition()
        # pozitiile mancarii in starea succesoare
        newFood = successorGameState.getFood()
        # lista cu nr care reprezinta starea "speriata" a fantomelor din starea succesoare
        newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

        finalValue = 0

        # Daca starea succesoare este o stare de Win => se retun un scor ft mare
        if successorGameState.isWin():
            return 999999

        # lista cu pozitia unde exista mancare in starea succesoare
        foodPos = newFood.asList()
        # lista distantelor dintre Pacman in starea succesoare si pana la
```

```
    fiecare punct de mancare disponibil
        distToFood = []
        # insumam starea fantomelor speriate din starea succesoare
        ghostScaredTimes = sum(newScaredTimes)

        # calculam distanta Manhattan a Pacmanului din starea succesoare
    catre  mancarea disponibila  si daca este diferita de 0 o adaugam in
    lista
        for fPos in foodPos:
            dist = manhattanDistance(fPos, newPos)
            # daca distanta e diferita de 0 o adaugam in lista distantelor
            if dist != 0:
                distToFood.append(dist)

        # daca lungimea listei este 0 atunci distanta minima calculata este
    si ea 0,daca nu calculam nminimul din lista pentru a stii pozitia celui
    mai aporpiat food-dot
        if len(distToFood) == 0:
            minDistToFood = 0
        else:
            minDistToFood = min(distToFood)

        # Distanta Manhattan catre fiecare fantoma din joc in starea curenta
        # pozitia fantomelor in starea curenta
        posToGhostCurrState = []
        for ghost in currentGameState.getGhostStates():
            posToGhostCurrState.append(ghost.getPosition())
        # distanta Manhattan de la pozitia Pacmanul din starea succesoare
    catre pozitia fantomelor din starea curenta
        distanceCurrPosToGhost = []
        for pos in posToGhostCurrState:
            distanceCurrPosToGhost.append(manhattanDistance(newPos, pos))

        # Distanta Manhattan catre fiecare fantoma din joc din starea
    succesoare
        # pozitia fantomelor in starea succesoare
        positionsOfGhosts = []
        for ghost in newGhostStates:
            positionsOfGhosts.append(ghost.getPosition())
        # distanta Manhattan de la pozitia Pacmanul din starea succesoare
    catre pozitia fantomelor din starea succesoare
        distanceToGhost = []
        for pos in positionsOfGhosts:
            distanceToGhost.append(manhattanDistance(newPos, pos))

        # pozitia curenta a fantomei
        currGhostDist = manhattanDistance(newPos, currentGameState.
    getGhostPosition(1))

        # Daca fantomele sunt speriate alege distanta mai mica pt ca e mai
    buna
        if ghostScaredTimes > 0:
            if min(distanceCurrPosToGhost) < min(distanceToGhost):
                finalValue = currGhostDist + successorGameState.getScore()
                finalValue += 100
            else:
                finalValue -= 5
        # Daca fantomele nu sunt speriate, distranta mai mare de fantome e
    recomandata
        else:
```

```
            if min(distanceCurrPosToGhost) < min(distanceToGhost):
                finalValue -= 5
            else:
                finalValue = currGhostDist + successorGameState.getScore()
                finalValue += 100

        # calculam cantitatea de mancare ramasa
        foodLeft = len(foodPos)

        # reducem scorul total cu distanta minima pe care am aflat-o pana la cea mai apropiata mancare
        finalValue -= minDistToFood

        # aici marim scorul daca pozitia actuala este fix pe pozitia mancarii
        if newPos in currentGameState.getCapsules():
            finalValue += 100

        # marim scorul daca ramane doar un singur punct in care gasim mancare
        if foodLeft == 1:
            finalValue += 1000

        # marim scorul daca sunt mai putine mancaruri disponibile in starea succesoare
        if foodLeft < len(currentGameState.getFood().asList()):
            finalValue += 100

        # decrementam scorul daca Pacman se opreste
        if action == Directions.STOP:
            # penalitate pt stop
            finalValue -= 10

        # returnam scorul final
        return finalValue

class ExpectimaxAgent(MultiAgentSearchAgent):
        def value(self, gameState, agentIndex, nodeDepth):
        #nodeDepth =cat de adanc e arborele
        #agentIndex = indexul agentului,Pacman=0,fantomele 1,2 ....

        #daca indexul agentului e mai mare decat numarul tuturor agentilor
        if agentIndex >= gameState.getNumAgents():
            #resetam indexul agentului
            agentIndex = 0
            #crestem adancimea arborelui
            nodeDepth = nodeDepth + 1

        #daca este un nod terminal returnam functia de utilitate
        if nodeDepth == self.depth:
            return self.evaluationFunction(gameState)

        #daca urmatorul agent este de tipul MAX: apelam max_value
        if agentIndex == self.index:
            return self.max_value(gameState, agentIndex, nodeDepth)
        #daca agentul este de tip MIN: apelam exp_value
        else:
            return self.exp_value(gameState, agentIndex, nodeDepth)
        return 'None'

```

```
143     def max_value(self, gameState, agentIndex, nodeDepth):
144         #daca Pacman a castigat/pierdut returnam functia de utilitate
145         if gameState.isWin() or gameState.isLose():
146             return self.evaluationFunction(gameState)
147         v = -999999 #incepem cu valoarea de la - infinit
148         for legalActions in gameState.getLegalActions(agentIndex):   #luam
    actiunile posibile ale agentului
149             successor = gameState.generateSuccessor(agentIndex, legalActions
    ) #luam starea succesoare
150             aux = self.value(successor, agentIndex + 1, nodeDepth)    #
    apelam functia value pt fiecare stare succesoare
151             if aux > v: #calculam maximul dintre valori
152                 v = aux
153                 actionValue = legalActions
154         if nodeDepth == 0:
155             return actionValue
156         else:
157             return v
158 
159     def exp_value(self, gameState, agentIndex, nodeDepth):
160         if gameState.isWin() or gameState.isLose(): #daca Pacman a castigat/
    pierdut returnam functia de utilitate
161             return self.evaluationFunction(gameState)
162         v = 0 #pornim cu valoare de la 0
163         probability = 1.0 / len(gameState.getLegalActions(agentIndex)) #
    calculam probabilitatea pentru fiecare ramura
164         for legalActions in gameState.getLegalActions(agentIndex):  #luam
    actiunile posibile ale agentului
165             successor = gameState.generateSuccessor(agentIndex, legalActions
    )  #luam starea succesoare
166             aux = self.value(successor, agentIndex + 1, nodeDepth)   #apelam
     functia value pt fiecare stare succesoare
167             v = v + (aux * probability)  #pentru a calcula valoarea se
    inmulteste probabilitatea cu valoarea fiecarei stari succesoare
168             actionValue = legalActions
169         return v
170 
171     def getAction(self, gameState):
172         return self.value(gameState, 0, 0)
```