Pacman-Search and MultiAgent Search-

Documentatie proiect

Facultatea de Automatica si Calculatoare

Coltea Alexandra Daria

FACULTATEA DE AUTOMATICA SI CALCULATOARE

UNIVERSITATEA TEHNICA DIN CLUJ-NAPOCA

Decembrie 2023

Chapter 1

Introducere

Proiectul rezolva jocul pacman in mod automat, prin utilizarea acentilor de cautare si a unor euristici potrivite. In Project 1 agentul Pacman gaseste căi prin lumea sa labirint, atât pentru a ajunge într-o anumită locație, cât și pentru a colecta mâncare în mod eficient. In Project 3 am implementat agenți pentru versiunea clasică a jocului Pacman, inclusiv fantome. Am implementat algoritmii minmax si alpha-beta prunning.

Chapter 2

Project1

Project 1 impelenteaza atat algoritmii bfs,dfs,A* si UCS, cat si euristici cum sunt cornerHeursitic si foodHeuristic.

2.1 DFS

Algoritmul DFS parcurge nodurile din graf incepand cu ”nodul” de stare curenta. Algortitmul are o stiva considerata frontiera nodurilor vizitate. Algoritmul pune in frontiera nodurile care nu au fost deja expandate si ia din frontiera noduri, atata vreme cat mai exista. La fiecare scoatere din frontiera a unui nod verificam daca acesta nu este Goal. Daca nodul este goal formam cu ajutorul legaturii parinte lista de actiuni de la radacina pana la nodul curent si o returnam. Daca nodul nu este goal , adaugam nodul in lista cu noduri deja expandate si continuam cautarea in graf cu succesorii nodului curent(ii adaugam in frontiera daca nu au fost deja expandati).

2.2 BFS

Similar algoritmului DFS, BFS parcurge nodurile din graf incepand cu ”nodul” de stare curenta. Diferenta majora intre cele doua este ca de aceasta data avem o coada considerata frontiera nodurilor vizitate. Algoritmul pune in frontiera nodurile care nu au fost deja expandate si ia din frontiera noduri(de aceasta data nodul cel mai vechi adaugat), atata vreme cat mai exista. La fiecare scoatere din frontiera a unui nod verificam daca acesta nu este Goal. Daca nodul este goal formam cu ajutorul legaturii

parinte lista de actiuni de la radacina pana la nodul curent si o returnam. Daca nodul nu este goal , continuam cautarea in graf cu succesorii nodului curent pe care ii adaugam in frontiera daca nu au fost deja expandati. Aici adaugam de asemenea starile care succed nodul curent in lista cu noduri expandate.

2.3 UCS

Algoritmul Uniform Cost Search foloseste pentru frontira sa o structura de tip Priority Queue. Aceasta ordoneaza nodurile introduse in frontira in functie de costul pe care il are actiunea de a ajunge la el. Astfel, in momentul in care scoatem noduri din frontira stim ca nodul scos va avea cel mai mic cost dintre cele disponibile. Verificam daca nodul curent este goal. Daca este formam calea de actiuni pana la el. Daca nodul curent nu este goal, adaugam nodul curent in lista de noduri expandate si adaugam in frontiera succesorii nodului curent care nu au fost deja vizitati. In momentul in care adaugam un nod nou in frontira, actualizam costul acestuia cu costul total pe care il avem pentru a ajunge de la starea initiala pana la el (costul nodului curent+costul nodului nou adaugat).

2.4 A*

Algoritmul de cautare A* se aseamana f bine cu algoritmul de cautare UCS, doar ca de aceasta data cand actualizam costul unui nod pe care il adaugam in frontira , luam in considerare si valoarea unei euristici care ne ajuta sa ajungem la rezultatul dorit mai rpd si precis.

2.5 Corners Problem

O stare in Corners Problem este reprezentata de o structura de date de tip tupla care contine o pozitie si o alta tupla de 4 valori boolene care sunt setate pe false daca in momentul in care sunt in pozitia curenta nu am vizitat un anumit corner, si pe true daca acel colt a fost vizitat : ex:(pozitieInitiala,(False,False,False,False)). Astfel, getStart state returneaza o astfel de tupla cu startingPosition() si o tupla de patru valori

False. Metoda isGoalState verifica daca toate valorile din subtupla corespunzatoate unei pozitii sunt setate pe True. Metoda getSuccesors preia starea curenta si pentru fieacre miscare valida, verifica daca pozitia in care rezulta pacman reprezinta un colt. Apoi adauga in lista de succesori pentru fiecare miscare valida o pereche pozitie si tupla(care contine atat colturile care au fost vizitate pana acolo, cat si colturile vizitate in pozitia actuala).

2.5.1 Corners Heuristic

In metoda cornersHeursitic calculez distanta manhattan maxima intre pacman si corners(cel mai indepartat nod) si o returnez. Este o euristica admisibila deoarece distanta returnata este cu siguranta mai mica decat distanta care trebuie parcursa si este non negativa deoarece nu returneaza niciodata o valoare negativa.

2.6 Food Problem

Metoda foodHeuristic calculeaza distanta reala prin labirint(mazeDistance) dintre pacman si cea mai apropiata mancare, apoi calculeaza distanta reala dintre cea mai apropiata mancare de pacman, si cea mai indepartata mancare de mancarea apropiata si returneaza o suma din cele doua. Este o euristaca buna deoarece distanata este mereu mai mica deact distanta reala care trebuie parcursa, dar totusi este o aproximare destul de apropiata a acesteia.

2.7 Find Path to Closest Dot

Metoda apeleaza algoritmul bfs pe problema data ca argument.

Chapter 3

Project3: MultiAgent

3.1 Reflex Agent

Metoda de evaluare a functiei din clasa Reflex Agent creeaza doua liste, una cu distantele manhattan dintre pacman si bucatile de mancare si cealalta cu distanta manhattan intre pacman si fantome. Ulterior calculam distanta cea mia mica pana la cea mai apropiata mancare , distanta cea mai mica pana la cea mai apropiata fantoma si calculam care este cea mai optima miscare pe are pacman o poate face(cu cel mai bun scor). Daca fanotma cea mai apropiata ese mai indepartata decat 5 o ”ignoram”. Pentru distanta dintre mancare si pacman folosim o pondere.

3.2 Minimax

Algoritmul minimax porneste de la starea initiala si alege random una dintre actiunile care ne duc la scor maxim. In functie de cat de mare este adancimea arborelui la care se doreste evaluarea , algoritmul actioneaza diferit. Deoarece algoritmul joaca perfect impotriva unui adversar perfect, dar in realitate adversarul nostru nu este perfect, exista situatii in care ne-ar avantaja o adancime mai mica . In implementarea alesa avem 3 metode, una de minimaxDecision care porneste de la pacman si apeleaza max value, a doua max value care se apeleaza doar atunci cand agentul este pacman si a treia min value care se apeleaza cand agentul curent(playerul) e fantoma. Max value la randul ei, ia actiunea optima si apeleaza min value pt urmatorul agent care e cu siguranta

fantoma. Min value verifica daca urmatorul player e o fantoma, moment in care apeleaza tot min value, sau daca urmatorul player e o pacman, moment in care apeleaza max value. Daca am ajuns cu min value la ultima fantoma , dar nu am atins adancimea dorita, indexul se reinitializeaza cu 0.

3.3 Alpha-Beta prunning

Algoritmul Alpha-Beta prunning este un minmax imbunatatit. Cu ajutorul atributelor Alpha si Beta evitam parcurgerea copiilor unor noduri. Alfa reprezinta cea mai mare valoare pe care am gasit-o pentru calea max, iar beta reprezinta cea mai mica valoare pentru care am gasit-o pana acum pe calea min . Astfel daca pentru max gasim o valoare mai mica decat alfa stim ca putem sa ignoram resul copiilor si sa nu mergem mai departe. La fel, daca pentru min gasim o valoare mai mare decat beta stim ca putem sa ignoram resul copiilor si sa nu mergem mai departe in procesare. Valorile alpha-beta se actualizeaza daca se gaseste o valoare mai buna pentru ele in arbore.