Documentatie PacMan

Poponet Tiberiu-Sergiu

Grupa 30231/2

În acest proiect, Pacman va căuta trasee prin labirint, atât pentru a ajunge la o locație specifică, ferind fantomele. cât și pentru a colecta mancare în mod eficient. Vom construi algoritmi de căutare generali și îi vom aplica în scenarii specifice Pacman.

Eu am implementat Q1, Q2, Q3, Q4 de la Project 1 si Q1 de la Project 3.

(Q1) Prima cerinta a fost implementarea DFS (Depth-First-Search), in fisierul search.py. Pentru reolvarea acestei cerinte, se inițializează o stivă (frontier), folosindu-ne de structurile definite de útil.py, pentru a urmări nodurile care trebuie explorate și o listă goală (expanded) pentru a ține evidența nodurilor deja explorate. Se adaugă nodul de start al problemei și o listă goală de acțiuni asociate la stivă. Începe o buclă care rulează până când găsește o soluție sau stiva devine goală. Se extrage un nod din vârful stivei și se verifică dacă este nodul scop. Dacă da, se întoarce lista de acțiuni asociată acelei soluții. Dacă nodul nu este scop, se adaugă la lista de noduri expandate și se explorează succesorii nodului curent. for new\_state, action, cost\_\_ in problem.getSuccessors(current):: Acesta este un ciclu care iterează prin toți succesori ai nodului curent. getSuccessors(current) returnează o listă de tupluri, fiecare tuplu reprezentând un succesori posibili al nodului curent. Functia getSuccessors returnează o listă de tuple, fiecare tupla reprezentând un succesor posibil al nodului curent. Fiecare tuplu este de forma (new\_state, action, cost), unde new\_state este starea succesorului, action este acțiunea de la nodul curent la acest succesor, iar cost este costul asociat cu acea acțiune. Pentru fiecare succesor, se verifică dacă nodul nu a fost deja explorat. Dacă succesorul nu a fost explorat, se adaugă la stivă împreună cu lista de acțiuni până la acel nod. Acest lucru asigură că DFS explorează cât mai adânc într-o direcție înainte de a reveni să exploreze alte ramuri ale arborelui de căutare. Algoritmul se repetă până când găsește o soluție sau până când toate nodurile posibile au fost explorate. Pentru aceasta cerinta, solution length este 130, iar nodes expanded este 146.

(Q2) A doua cerinta a fost implementarea BFS (Breadth-First-Search), in acelasi fisier search.py. . Pentru reolvarea acestei cerinte, se inițializează o coada (frontier), folosindu-ne de structurile definite de útil.py, pentru a urmări nodurile care trebuie explorate în ordinea în care au fost adăugate. Se adaugă nodul de start și o listă goală de acțiuni asociate nodului la coadă. Se inițializează o listă goală pentru a ține evidența nodurilor deja explorate. Se începe o buclă care rulează atâta timp cât coada nu este goală. În fiecare iterație, se explorează nodul de la începutul cozii. Se extrage nodul curent și lista de acțiuni asociată cu acest nod din începutul cozii. Se verifică dacă nodul curent este nodul țintă (nodul scop). Dacă este, se returnează lista de acțiuni care a condus la acest nod. Dacă nodul curent nu a fost încă explorat, se continuă procesul de expandare. Se adaugă nodul curent la lista de noduri explorate pentru a evita explorarea multiplă a aceluiași nod, apoi se iterează prin toți succesori ai nodului curent. Se construiește o nouă listă de acțiuni adăugând acțiunea corespunzătoare succesorului curent la lista de acțiuni de la nodul curent. Succesorul este adăugat la coadă împreună cu lista actualizată de acțiuni până la acel succesor. Acest proces continuă până când se găsește o soluție sau până când toate nodurile accesibile au fost explorate. . Pentru aceasta cerinta, solution length este 68, iar nodes expanded este 269.

(Q3) ) A treia cerinta a fost implementarea UCF (uniformCostSearch), in acelasi fisier search.py. . Pentru reolvarea acestei cerinte, se inițializează o coada de prioritati (frontier), folosindu-ne de structurile definite de útil.py, pentru a urmări nodurile. Nodurile cu costuri totale mai mici au o prioritate mai mare. Acest lucru asigură că nodurile cu costuri mai mici sunt explorate înaintea celor cu costuri mai mari. Se adaugă nodul de start, o listă goală de acțiuni asociată nodului și un cost total de 0 la coadă, cu o prioritate de 0. Se inițializează un set gol (expanded) pentru a ține evidența nodurilor deja explorate. Se începe o buclă care rulează atâta timp cât coada nu este goală. În fiecare iterație, se explorează nodul cu cel mai mic cost total. Se extrage nodul curent, lista de acțiuni asociată și costul total asociat cu acest nod din coada de priorități. Se verifică dacă nodul curent este nodul scop și se returnează lista de acțiuni care a condus la acest nod, altfel continuam cautarea. Se adaugă nodul curent la setul de noduri explorate pentru a evita explorarea multiplă a aceluiași nod. Se iterează prin toți succesori ai nodului curent. Se construiește o nouă listă de acțiuni adăugând acțiunea corespunzătoare succesorului curent la lista de acțiuni de la nodul curent. Se calculează noul cost total asociat cu succesorul curent. Succesorul este adăugat la coadă de priorități împreună cu lista actualizată de acțiuni și noul cost total. Acest lucru asigură că nodurile sunt explorate în ordinea costurilor totale crescătoare. Acest proces continuă până când se găsește o soluție (nodul scop) sau până când toate nodurile accesibile au fost explorate. Uniform Cost Search garantează găsirea celei mai scurte căi către soluție, indiferent de costurile asociate cu fiecare acțiune. Pentru aceasta cerinta, solution length ia valori ca: 68,74,152,7, iar nodes expanded este 269, 260,173,14.

(Q4) implementează algoritmul A\* (A-Star Search). Pentru reolvarea acestei cerinte, se inițializează o coada de prioritati (frontier), folosindu-ne de structurile definite de útil.py, pentru a urmări nodurile în ordinea costului total și a euristicelor. Nodurile cu costuri totale și euristici mai mici au o prioritate mai mare. Se obține starea de start a problemei. Se calculează valoarea euristicii pentru starea de start utilizând funcția euristică specificată sau o funcție implicită (în cazul în care heuristic este setată la nullHeuristic). Se adaugă nodul de start, o listă goală de acțiuni asociată nodului, costul total 0 și valoarea euristicii la coadă de priorități, cu o prioritate dată de suma costului total și euristicii. Se începe o buclă care rulează atâta timp cât coada nu este goală. În fiecare iterație, se explorează nodul cu cea mai mică valoare a costului total și euristicii. Se extrage nodul curent, lista de acțiuni asociată și costul total asociat cu acest nod din coada de priorități.Se verifică dacă nodul curent este nodul scop. Dacă este, se afișează un mesaj și se returnează lista de acțiuni care a condus la acest nod. Dacă nodul curent nu a fost încă explorat, se continuă procesul de expandare. Se adaugă nodul curent la setul de noduri explorate pentru a evita explorarea multiplă a aceluiași nod. Se verifică dacă succesorul nu a fost deja explorat. Se construiește o nouă listă de acțiuni adăugând acțiunea corespunzătoare succesorului curent la lista de acțiuni de la nodul curent. Se calculează noul cost total asociat cu succesorul curent. Se calculează estimarea euristică pentru succesorul curent. Se calculează suma dintre costul total și estimarea euristică pentru succesorul curent. Succesorul este adăugat la coadă de priorități împreună cu lista actualizată de acțiuni, noul cost total și suma costului total și a estimării euristice. Acest lucru asigură că nodurile sunt explorate în ordinea costurilor totale și a estimării euristice crescătoare. Pentru aceasta cerinta, solution length ia valoarea ca: 68, iar nodes expanded este 221.

Pentru P3, am implementat Q1. Acest proiect include fantome. Prima cerinta este optimizarea ReflexAgent din multiAgents.py, mai exact evaluationFunction. Scopul acestei funcții este să evalueze "calitatea" unei stări a jocului pentru a ghida deciziile agentului în alegerea următoarei acțiuni. Pasi:

Calcul Distante: foodDistances: Distanța până la cea mai apropiată bucată de mâncare. scaredGhostDistances: Distanța până la cea mai apropiată fantomă speriată. ghostDistances: Distanța până la cea mai apropiată fantomă nesperiată.

Scoruri și Penalizări: Adaugă la scor inversul distanței până la cea mai apropiată bucată de mâncare. Adaugă la scor inversul distanței până la cea mai apropiată fantomă speriată (dacă există). Scade din scor inversul distanței până la cea mai apropiată fantomă nesperiată (dacă există). Scade din scor numărul de bucăți de mâncare rămase (penalizare mai mare pentru mai multă mâncare rămasă).Adaugă la scor scorul global al stării succesor.

Returnare Scor: Returnează scorul evaluării. Scorul mediu este 1274.9.