Documentație - tema 1 IA

Proiect realizat de: Sîrghi Paula

Trif Sabrina-Ionela

Cuprins

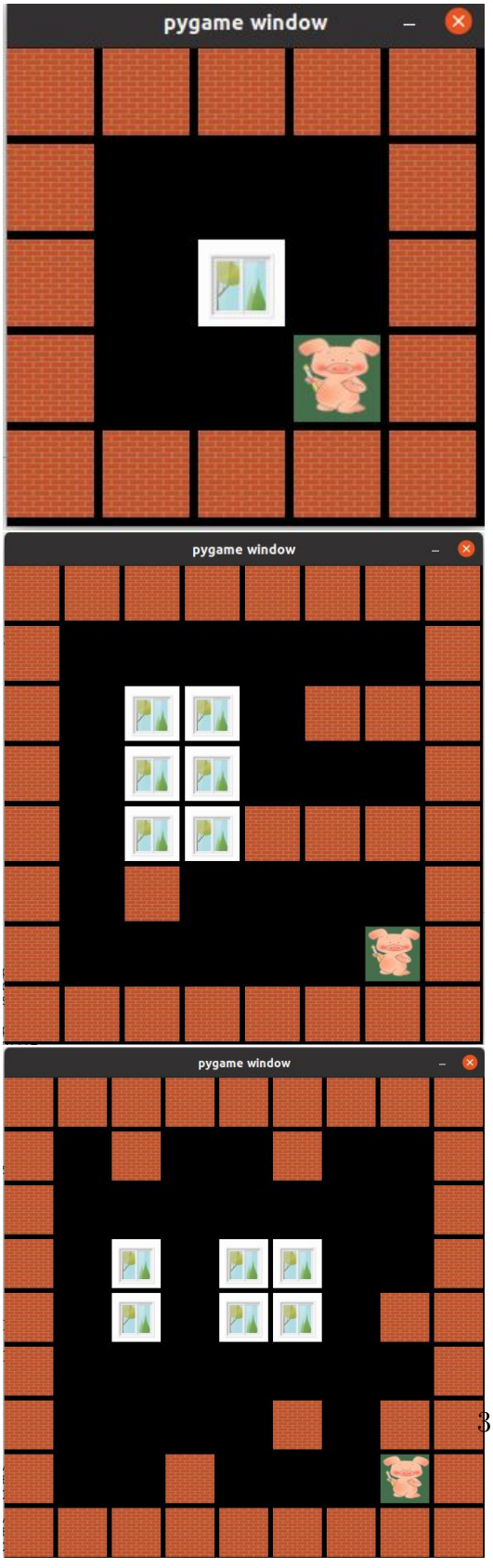
1. Enunțul problemei

House paint este un joc în care peretele unei case, care conține obstacole (gea, cărămizi) trebuie colorat. În momentul în care toate spațiile albe sunt colorate, atunci jocul este terminat. Pensula este așezată în colțul din dreapta-jos, iar utilizatorii pot face mișcari la dreapta, sânga, în sus și în jos pentru a deplasa pensula. Algoritmii de cautare aplicati vor gasi un drum de la pozitia de start pana la coltul din stanga sus al casei.

2. Modelarea problemei

Peretele casei este văzut ca o matrice, care poate conține:
0 pentru spațiile albe
1 pentru spațiile colorate
2 pentru caramizi
3 pentru geamuri
P pentru poziția curentă a pensulei

Jocul este terminat în momentul în care nu mai avem 0 în matrice. Operații posibile: N, S, E, V Starea inițială este tot timpul în partea din dreapta-jos, iar starea finală o regăsim în momentul în care exista doar 1 sau 2 în matrice în cazul navigării din butoane, iar în cazul folosirii algoritmilor de căutare starea finala va fi atinsă în momentul în care personajul va ajunge în poziția (1,1) din matrice. Observație! În starea finală, P poate fi găsit pe orice poziție marcată în starea inițială cu 0 sau P.


pygame window
pygame window
pygame window

Labirint1:
[2, 2, 2, 2, 2],
[2 ,0, 0, 0, 2],
[2, 0, 3, 0, 2],
[2, 0, 0, 'P', 2],
[2, 2, 2, 2, 2]

Labirint2:
[2,2,2,2,2,2,2,2],
[2,0,0,0,0,0,0,2],
[2,0,3,3,3,2,0,2],
[2,0,3,3,3,0,0,2],
[2,0,0,0,2,0,0,2],
[2,0,2,0,2,0,0,2],
[2,0,2,0,2,0,'P',2],
[2,2,2,2,2,2,2,2]

Labirint3:
[2,2,2,2,2,2,2,2,2],
[2,0,0,0,0,0,0,0,2],
[2,2,0,3,3,0,0,0,2],
[2,0,0,0,0,0,0,2,2],
[2,0,0,3,3,0,0,0,2],
[2,2,0,3,3,0,2,0,2],
[2,0,0,0,0,0,0,0,2],
[2,0,0,0,2,0,2,'P',2],
[2,2,2,2,2,2,2,2,2]

3. Metode de implementare

Aplicatia implementează patru algoritmi de căutare (BFS, DFS, Uniform Cost Search și AStar) și, totodată, oferă utilizatorului posibilitatea de a folosi tastele pentru a deplasa purcelușul în cele trei labirinturi.

În primul rând, clasa Directions conține toate cele 4 operații care se pot realiza in jocul ales de noi: Nord, Sud, Est, Vest. Funcția getSuccesors returneaza o lista de succesori (maxim 4) adică coordonatele locului in matrice și operația realizată pentru a ajunge din locul de unde am pornit pana la vecinul găsit.
BFS l-am implementat cu ajutorul unei cozi si o listă. Coada este frontiera, iar lista teritoriul, adica nodurile vizitate. Inițial am introdus in frontiera nodul de început si o lista goala, unde urmeaza sa fie adăugate acțiunile. Cât timp această coadă nu este goală extragem primul exement introdus și verificam dacă acesta nu este starea finală. Dacă incă nu am ajuns la sfârsit găsim succesorii nodului curent cu ajutorul funcției getSuccesors si îi verificăm pe toți și îi introducem în coadă alături de operațiie pe care le-a facut pentru a ajunge în aceea stare. La final returnăm lista cu operațiile realizate pentru a ajunge in starea finală.
Algoritmul DFS are ca scop găsirea unui drum de la colțul din stânga jos al labirintului la colțul din dreapta sus, evitând căsuțele cu obstacole. Pentru implementarea sa am folosit o coadă care va memora vecinii nodului în care se află personajul, respectiv frontiera domeniului și, de aceea, următoarea poziție a personajului va fi ultimul lement adăugat în coadă, expandând pe rând nodurile din aceasta atât timp cât mai are elemente. Totodată, pentru a reține teritoriul domeniului am folosit o listă.
Căutarea după cost are ca scop găsirea drumului de cost minim de la colțul din dreapta jos la colțul din stânga sus al labirintului. Pentru implementarea algoritmului am folosit o coadă de priorități care ordonează elementele sale astfel încât cele cu costul cel mai mic să fie ultimele din coadă, expandând pe rând nodurile din aceasta atât timp cât mai are elemente. Totodată, pentru a reține teritoriul domeniului am folosit o listă.
Avem doua funcții: manhattanDistance și euclidianDistance. Prima returnează suma a două module. Modulele scaderilor coordonatelor a două puncte. A doua returnează radical din suma scaderilor coordonatelor a două puncte ridicate la pătrat. A Star este implementat cu fiecare dintre aceste

funcții și cu ajutorul unei cozi cu priorități. Prima dată am introdus în coadă nodul de start alături de o listă goală și de f, o funcție compusă din cost și funcția euristică (manhattan sau euclidian). Apoi, cât timp coada nu este goala extragem câte un nod și verificăm dacă nu este cel în care dorim să ajungem. Dacă incă nu ne aflăm la final gasim succesorii nodului curent îi verificam pe toți, le calculam costul, euristica și apoi ii introducem în coadă. Returnăm lista cu operațiile necesare pentru a ajunge în punctul final.

4. Comparații între implementări

Algoritm	Complexitate	Structură De Date Folosită	Optimal	Descriere	Noduri expandate
BFS	O(V+E)	coada	da	Cele mai vechi noduri adăugate în stivă sunt expandate primele	labirint 1: 6, labirint 2: 14, labirint 3: 36
DFS	O(V)	stiva	nu	Cele mai noi noduri adăugate sunt luate expandate primele	labirint 1: 4 labirint 2: 14 labirint 3: 34
UCS	$O(b^{\frac{1+C}{e}})$	coadă de priorități	da	Selectează muchia în care poate ajunge, având costul cel mai mic până ajunge în punctul final	labirint 1: 4, labirint 2: 10, labirint 3: 12
AStar	$O(b^d)$	coadă de priorități	doar pentru varianta euristică	Căutare cu cost care estimează, pentru fiecare nod costul drumului care trece prin el cu ajutorul unei funcții	labirint 1: 4, labirint 2: 11, labirint 3: 14,

5. Rulare

Pentru a porni aplicația avem nevoie de pygame și de comanda: python3 game.py -l x, unde x poate fi orice număr între 1 și 3, fiecare reprezentând un labirint anume. Pentru a opri aplicația trebuie apăsată tasta ESC.
Pentru navigarea simplă, cea în care scopul este de a colora toate pătrățelele, folosim săgețile de pe tastatură. În acest caz, în momentul în care personajul colorează toate pătrățelele care nu sunt obstacole, aplicația se oprește.
Pentru folosirea algoritmilor, apăsăm tastele: b - pentru BFS, d - pentru DFS, c - pentru căutarea după cost uniform, a - pentru AStar cu distanța manhattan și e - pentru AStar cu distanța euclidiană. În acest caz, în momentul în care personajul ajunge în colțul din stânga sus, aplicația se oprește.