Raport końcowy projektu P1
Przeszukiwanie przestrzeni stanów

Metody sztucznej inteligencji (2021Z)

Autor: Maciej Krosman
Nr albumu: 257341

A. Algorytm przeszukiwania poinformowanego A*

Problemem projektu P1 jest zaplanowanie drogi dla robota mobilnego z punktu startowego do punktu końcowego w przestrzeni reprezentowanej przez tablicę komórek N x N zawierającej przeszkody algorytmem przeszukiwania poinformowanego A*. Punkty start i koniec znajdują się na krawędziach tablicy komórek, gdyż będzie to lepiej reprezentować rzeczywistość, w której robot ma za zadanie opuścić teren przeszkód w określonym punkcie.

Reprezentacja problemu:

stany: tablica komórek N x N, część komórek jest przeszkodami
akcje: przemieszczenie robota do jednej z 8 sąsiednich komórek
koszt akcji: d1 dla pustej komórki, d2 dla komórki z przeszkodą takie że d2 > d1

Sformułowanie problemu:

stan początkowy: pozycja startowa na krawędzi tablicy komórek
cel: pozycja końcowa na krawędzi tablicy komórek
rozwiązanie: lista kolejnych odwiedzanych komórek
koszt rozwiązania: suma kosztów akcji

Algorytm przeszukiwania poinformowanego A* – algorytm heurystyczny znajdowania najkrótszej ścieżki w grafie ważonym z dowolnego wierzchołka do wierzchołka spełniającego określony warunek (test celu). Algorytm jest zupełny i optymalny, co oznacza, że znajduje ścieżkę, jeśli tylko taka istnieje, i przy tym jest to ścieżka najkrótsza.

Algorytm ten opiera się na wybieraniu punktów ścieżki w oparciu o wartość funkcji oceny f(n)
f(n) = g(n) + h(n)
g(n) - koszt ruchu z punktu początkowego do aktualnej pozycji
h(n) - szacunkowy koszt przejścia z aktualnej pozycji do celu (funkcja heurystyczna)
oraz wykorzystanie listy otwartej (open), listy zamkniętej (closed) oraz relacji węzeł - rodzic

Pseudokod algorytmu przy wygenerowanej już tablicy komórek oraz pozycji startowej i końcowej:

```
AStar (tablica komórek, start, cel):
list open, closed
start.g = 0
start.h = 0
start.f = 0                          #wyliczenie wartości funkcji f, g, h dla węzła
If ( start == cel ) return start     #osiągnięto cel, więc zwracany jest węzeł start z którego można odczytać koszt i wypisać trasę
dodaj ( start, open )                #dodanie węzła start do listy open

while ( !pusta(open) )
    m = min_f ( open )               #przypisanie węzła o najmniejszej wartości f
    if ( m == cel ) return m                 #osiągnięto cel, więc zwracany jest węzeł m z którego można odczytać koszt i wypisać trasę
    usuń ( m, open )
    dodaj ( m, closed )              #przeniesienie węzła z listy open do closed

    foreach ( n w następniki(m) )
        kosztN = m.g + koszt_akcji(m, n)     #wyliczenie kosztu dotarcia do węzła n przez m

        if ( !(n w open) && !(n w closed) )
            n.g = kosztN
            n.h = funkcja_heurystyczna( n, cel )
            n.f = n.g + n.h          #wyliczenie wartości funkcji f, g, h dla węzła
            n.rodzic = m             #przypisanie węzłowi rodzica
            dodaj ( n, open )        #węzeł  nie należy do list więc jest dodawany do listy open
        else
            if ( n w open && kosztN < n.g )
                usuń ( n, open )
            if ( n w closed && kosztN < n.g )
                usuń ( n, closed )   #trasa przez m do n jest lepsza dlatego gorsza trasa do n jest usuwana z list
            n.g = kosztN
            n.h = funkcja_heurystyczna( n, cel )
            n.f = n.g + n.h          #wyliczenie wartości funkcji f, g, h dla węzła
            n.rodzic = m             #przypisanie węzłowi rodzica
            dodaj ( n, open )        #węzeł jest dodawany do listy open

return porażka
```

B. Koncepcja rozwiązania zadanego problemu

Założenia projektowe

Program ma porównać dwa rozwiązania problemu planowania ścieżki robota mobilnego dla zadanej tablicy komórek N x N w zależności od dwóch heurystyk kosztów resztkowych. Program zostanie wykonany w oparciu o język C++.

Program ma składać się z 3 modułów:

Metoda: Moduł odpowiedzialny za realizację przeszukiwania przestrzeni stanów problemu – funkcja implementująca algorytm przeszukiwania poinformowanego A*, dwie funkcje obliczające wartość heurystyki dla danego węzła oraz funkcje pomocnicze określające osiągnięcie celu czy też poprawność współrzędnych węzła.

Problem: Moduł odpowiedzialny za losowy generator problemu w postaci tablicy komórek o rozmiarze zależnym od N wraz z kosztami akcji oraz komórek start i cel wylosowanych na krawędziach tablicy komórek.

Sterowanie: Moduł odpowiedzialny za sprawdzenie wykonania programu. Wynik w postać wygenerowanej tablicy komórek, sekwencje wizytowanych komórek, kosztu wyniku dla obu heurystyk oraz czasu jaki potrzebowały rozwiązania.

Tablica komórek zostanie zrealizowana przez tablicę znaków których wartości będą odpowiadały komórce pustej, komórce przeszkodzie, komórce startowej oraz komórce celu. Węzeł będzie reprezentowany strukturą zawierającą współrzędne rodzica, wartości g, h oraz f. Zbiory open i closed będą realizowane poprzez 3 struktury danych: lista wartości f i współrzędnych, tablica „kwadratowa” węzłów oraz tablica „kwadratowa” bool-i. Pierwsza struktura reprezentująca zbiór open będzie na początku każdej iteracji sortowana malejąco. Druga struktura reprezentuje aktualnie najlepsze wartości ścieżek do węzła(będzie nazwana tablicą pomocniczą). Posiadanie wartości innych niż zainicjalizowane wskazuje na przynależność węzła do zbioru closed lub open, co pozwoli zmniejszyć ilość przeszukiwań listy open, gdyż będzie ona przeszukiwana jedynie gdy trzeba będzie usunąć z niej element. Trzecia struktura będzie tablicą closed gdzie wartość true będzie wskazywać na przynależność węzła o takich samych współrzędnych do zbioru closed.

Parametry które będą się zmieniały między testami będą wczytywane z pliku konfiguracyjnego tekstowego. Wyniki programu będą zapisywane w pliku wynikowym tekstowym zawierającym dokładną datę i godzinę wykonania w nazwie.

Dane przekazywane przez plik konfiguracyjny:

Rozmiar przestrzeni tablicy komórek
Koszt akcji na pustą komórkę
Koszt akcji na przeszkodę
Procent komórek które mają zostać wygenerowane jako przeszkody reprezentowane jako wartość z przedziału [0,100]
Liczba tablic komórek do wygenerowania i rozwiązania

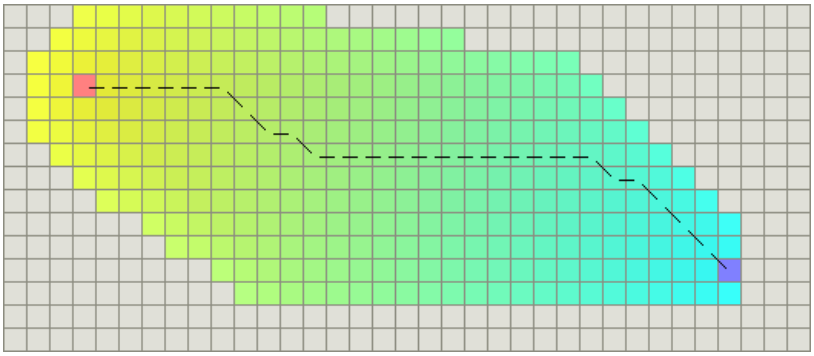
Dane wynikowe:

Dla każdego testu:
Wygenerowana tablica komórek
Dla każdej z obu funkcji heurystycznych:
Wygenerowana tablica komórek zawierająca trasę
Wynik sumy kosztów akcji trasy
Sekwencja komórek trasy od startu do celu
Czas pracy algorytmu z daną funkcją heurystyczną
Statystyka zbiorcza testów

W pliku wynikowym(monitorującym) pojawią się również wpisy z kroków działania algorytmu.

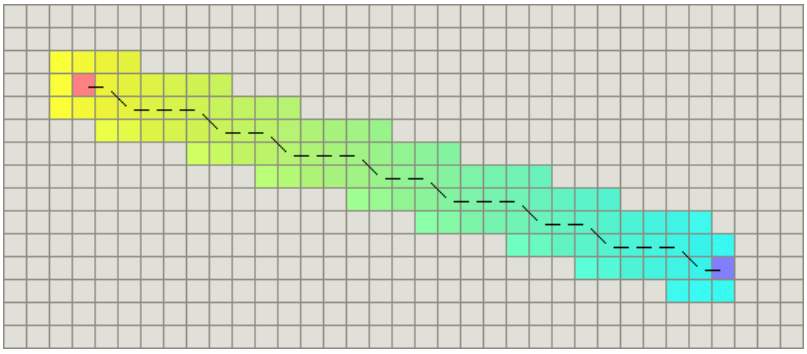
Generowanie tablicy komórek:
Program dla każdej komórki losuje wartość z przedziału <0,100> i porównuje ją z procentem komórek przeszkód zadanym w pliku konfiguracyjnym. Jeśli wylosowana wartość nie jest większa to komórka ta będzie miała nadany symbol przeszkody, a w reszcie przypadków symbol pustej komórki. Następnie wylosowuje komórki startu i celu.

Funkcje heurystyczne:
1.



```
funkcja heurystyczna(węzeł, cel)
   dx = abs(węzeł.x - cel.x)
   dy = abs(węzeł.y - cel.y)
   return D1 * sqrt(dx * dx + dy * dy)          #D1 jest kosztem akcji na puste pole
```

2.



```
funkcja heurystyczna(węzeł, cel)
   dx = abs(węzeł.x - cel.x)
   dy = abs(węzeł.y - cel.y)
      return D1 * max(dx, dy)                    #D1 jest kosztem akcji na puste pole
```

Źródło: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
Funkcja 2 będzie lepiej odzwierciedlać problem projektu gdyż uwzględnia w sobie format przestrzeni.

Statystyka i porównanie wyników przeprowadzona jest w trakcie pracy programu i podsumowana pod koniec jego działania.

Pseudokod programu uwzględniający najważniejsze funkcje

1. Pobranie zmiennych z pliku konfiguracyjnego
    a. Sprawdzenie poprawności parametrów
    b. W razie błędów przypisywane są wartości domyślne
2. Stworzenie pliku monitorująco-wynikowego z datą i godziną w nazwie
    a. Rozpoczęcie wpisu istotnych kroków programu do pliku wynikowego
3. Następujące funkcje są wykonywane wielokrotnie w zależności od ilości zadanych testów:
    a. Losowe wygenerowanie tablicy komórek, węzłów start oraz cel.
        i. Wypisanie ich w pliku wynikowym
    b. Zapisanie czasu startu
    c. Rozwiązanie problemu dla funkcji heurystycznej 1
        i. Wyliczenie czasu wykonania rozwiązania
        ii. Weryfikacja wyników
        iii. W przypadku poprawnych wyników wypisanie ich do pliku wynikowego:
            1. Tablica komórek z rozwiązaniem
            2. Koszt ścieżki
            3. Wyliczenie czasu wykonania
        iv. W przypadku błędnych rozwiązań wypisanie błędu
    d. Powtórzyć kroki b. i c. dla funkcji heurystycznej 2
4. Wyliczenie średnich wartości wynikowych dla wszystkich rozwiązań w zależności od funkcji heurystycznej.

C. Opis struktury i funkcji zrealizowanego programu

Kod programu jest zawarty w 3 plikach Sterowanie.cpp(moduł sterowanie), Problem.h(moduł problem) oraz Metoda.h(moduł metoda).
Plik wykonawczy nazywa się MSI-P1.exe.

Struktury i funkcje w module Problem:

```
struct rt {                          #struktura danych wyjściowych generowania losowego terenu
   vector<vector<char>> terrain;     #tablica symboli komórek terenu dla robota
   Pair start;                       #Pair == pair<int,int>, współrzędne węzłów start i dest(cel)
   Pair dest;
};

void outputTerrain(const vector<vector<char>>& terrain, ofstream& output) #wypisanie tablicy komórek do pliku i konsoli tekstowej

void outputTerrainWithPath(const vector<vector<char>>& terrain, vector<Pair> path, ofstream& output)   #naniesienie ścieżki na tablice komórek terenu i wypisanie jej do pliku i konsoli tekstowej

rt random_terrain(const int& N, const int& P, ofstream& oLog)       #funkcja wylosowuje tablice symboli NxN oraz współrzędne start i dest(cel) na krawędziach tablicy symboli
```

Moduł spełnia swoje założenia projektowe.

Struktury i funkcje w module Metoda:

```
struct cell {                        #struktura węzła
      double g, h, f;
      Pair parent;                   #Pair == pair<int,int>, współrzędne rodzica
      cell() :                       #konstruktor z wartościami domyślnymi, wskazującymi na to że rozważany
            parent(-1, -1),          węzeł nie należy ani do zbioru open ani closed
            f(-1),
            g(-1),
            h(-1)
      {}
};

struct aStarValues {                 #struktura danych wyjściowych funkcji implementującej algorytm A*
      vector<Pair> Path;             #tablica współrzędnych węzłów w ścieżce
      double f;                      #koszt ścieżki
      int openListSize;              #ilość węzłów w zbiorze open
      int closedListSize;            #ilość węzłów w zbiorze closed
};

bool isValid(const int& N, const Pair& current)   #funkcja sprawdza czy współrzędne należą do tablicy komórek
```

bool isDest(const Pair& current, const Pair& dest) #funkcja sprawdza czy sprawdzany węzeł jest celem

double calcH1(const Pair& current, const Pair& dest, map<char, double> actionCost) #zwraca wartość funkcji heurystycznej 1 dla zadanego węzła

double calcH2(const Pair& current, const Pair& dest, map<char, double> actionCost) #zwraca wartość funkcji heurystycznej 2 dla zadanego węzła

vector<Pair> makePath(const Pair& dest, const vector<vector<cell>>& cells) #zwraca tablicę współrzędnych węzłów w ścieżce na podstawie podanego węzła i tablicy pomocniczej

int countClosed(const vector<vector<bool>>& closed) #zwraca ilość węzłów w zbiorze closed

void outputPath(ofstream& output, vector<Pair> path) #wypisuje do pliku współrzędne węzłów ścieżki

aStarValues aStar(const vector<vector<char>>& terrain, const Pair& start, const Pair& dest, map<char, double> actionCost, const int& heuristic, ofstream& oLog) #dla zadanych tablicy znaków, współrzędnych start, cel, kosztów akcji oraz funkcji heurystycznej wykonuje algorytm A*

Moduł spełnia swoje założenia projektowe oraz dodatkowo niektóre funkcje wyliczają dodatkowe dane wynikowe takie jak liczba węzłów w zbiorach open i closed.

Struktury i funkcje w module Sterowanie:

Wczytywanie, tworzenie i zapisywanie do plików jest zrealizowane przy pomocy biblioteki <fstream> oraz klas ifstream oraz ofstream.

Wypisywanie do konsoli tekstowej jest wykonane przy pomocy biblioteki <iostream>.

Wyliczanie daty i czasu jest wykonane przy pomocy biblioteki <time.h> oraz poprzez funkcje:
time(), localtime_s(wylicza date i czas regionalny), asctime_s(przekształca date w obiekt string),
czas wykonania algorytmu jest obliczony za pomocą:

```
timer = clock();
//funkcja właściwa//
timer = clock() - timer;
float temp = (float)timer / CLOCKS_PER_SEC * 1000   #przeliczenie czasu na milisekundy
```

Moduł spełnia swoje założenia projektowe, sprawdza wykonanie funkcji, wczytuje i zapisuje w odpowiednich plikach parametry i rozwiązania.

W terminalu tekstowym są wypisywane jedynie rozwiązania(tablica komórek z ścieżką, współrzędne węzłów ścieżki, koszt ścieżki i inne dane do statystyki porównawczej) oraz statystykę porównawczą.

W pliku monitorująco-wynikowym wpisane są te same dane co w terminalu tekstowym oraz kroki wykonania programu.

Sposób uruchomienia programu

Program należy uruchamiać używając pliku MSI-P1.exe. W tym samym folderze co plik wykonawczy powinien znajdować się również plik konfiguracyjny config.txt, ponieważ w innym przypadku do wykonania programu zostaną użyte parametry domyślne. W razie gdyby plik monitorująco-wynikowy nie powstawał, problemem może być brak uprawnień dla MSI-P1.exe do tworzenia plików.

Podsumowanie wykonanych testów programu

Parametry wejściowe:
(rozmiar tablicy komórek do wylosowania,
prawdopodobieństwo wylosowania komórki przeszkody,
koszt akcji na wolną komórkę,
koszt akcji na przeszkodę,
ilość testów do przeprowadzenia)

Dla (100, 50, 1.0, 2.0, 100):
Średni dystans start->cel w heurystyce 1: 79.7988
Średni dystans start->cel w heurystyce 2: 71.19
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 1: 78.75
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 2: 77.74
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 1: 74.53
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 2: 73.92
Średni czas wykonania algorytmu dla funkcji heurystycznej 1: 180.6ms
Średni czas wykonania algorytmu dla funkcji heurystycznej 2: 528.6ms
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 1: 302.94
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 2: 353.47
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 1: 425.28
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 2: 1286.73

Dla (100, 50, 1.0, 1.2, 100):
Średni dystans start->cel w heurystyce 1: 73.9262
Średni dystans start->cel w heurystyce 2: 66.24
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 1: 69.174
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 2: 67.698
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 1: 67.24
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 2: 67.28
Średni czas wykonania algorytmu dla funkcji heurystycznej 1: 181.57ms
Średni czas wykonania algorytmu dla funkcji heurystycznej 2: 416.89ms
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 1: 191.35
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 2: 304.04
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 1: 313.3
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 2: 989.58

Dla (100, 50, 1.0, 5.0, 100):
Średni dystans start->cel w heurystyce 1: 71.2244
Średni dystans start->cel w heurystyce 2: 64.73
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 1: 76.16
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 2: 76.03
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 1: 72.76
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 2: 72.47
Średni czas wykonania algorytmu dla funkcji heurystycznej 1: 275.71ms
Średni czas wykonania algorytmu dla funkcji heurystycznej 2: 536.22ms
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 1: 482.48
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 2: 477.46
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 1: 627.36
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 2: 1258.51

Dla (100, 25, 1.0, 2.0, 100):
Średni dystans start->cel w heurystyce 1: 72.7389
Średni dystans start->cel w heurystyce 2: 65.94
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 1: 67.46
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 2: 66.79
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 1: 67.86
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 2: 67.37
Średni czas wykonania algorytmu dla funkcji heurystycznej 1: 80.4ms
Średni czas wykonania algorytmu dla funkcji heurystycznej 2: 282.53ms
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 1: 225.86
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 2: 368.99
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 1: 187.02
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 2: 657.05

Dla (100, 75, 1.0, 2.0, 100):
Średni dystans start->cel w heurystyce 1: 72.9336
Średni dystans start->cel w heurystyce 2: 66.32
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 1: 88.42
Średni koszt ścieżki rozwiązania dla funkcji heurystycznej 2: 88.41
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 1: 68.94
Średnia ilość węzłów w ścieżce rozwiązania dla funkcji heurystycznej 2: 68.62
Średni czas wykonania algorytmu dla funkcji heurystycznej 1: 751.67ms
Średni czas wykonania algorytmu dla funkcji heurystycznej 2: 1029.31ms
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 1: 210.53
Średnia ilość węzłów w zbiorze open dla funkcji heurystycznej 2: 198.75
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 1: 1848.43
Średnia ilość węzłów w zbiorze closed dla funkcji heurystycznej 2: 2474.77

Pliki z powyższymi testami nie zostały zawarte w zbiorczym zipie ze względu na to, że każdy miał pojemność przynajmniej kilkadziesiąt MB

Wnioski:

Funkcja heurystyczna 2(D1 * max(dx, dy)) znajduje zawsze ścieżkę o mniejszym koszcie i jako, że dokładnie odwzorowuje przestrzeń problemu i możliwe w niej akcje można stwierdzić, że jest optymalna i dopuszczalna.
Natomiast funkcja heurystyczna 1(D1 * sqrt (dx * dx + dy * dy)), nie spełnia warunków dopuszczalności, ponieważ istnieją takie węzły dla których prawdziwy koszt do celu będzie mniejszy od wyliczonego kosztu funkcji heurystycznej, dlatego też nie jest optymalna.

Z drugiej strony jednak funkcja heurystyczna 1 jest w stanie znaleźć ścieżkę optymalną albo o niewiele gorszym koszcie wykorzystując mniej zasobów komputera(liczba węzłów w zbiorach open i closed) oraz w krótszym czasie.

Procent ilości przeszkód w tablicach komórek czy też wielkość kosztu akcji na przeszkodę ma mały wpływ na poprawę działania funkcji heurystycznej 1, który równie dobrze można uzasadnić wylosowaniem bardziej korzystnych tablic komórek w grupie testowej.