AISDI zadanie grafowe
Eksploracja labiryntu

Waldemar Grabski waldemar.grabski@pw.edu.pl

Opis

Dany jest labirynt o rozmiarze *n* na *m* pól gdzie 2 <= n 1024 i 2 <= m < 1024. Wyjściem z labiryntu są 4 środkowe pola. Jeśli liczba pól w wierszu/kolumnie jest nieparzysta, to wyjściem jest środkowe pole oraz pole po lewej/powyżej środkowego pola.

W labiryncie znajduje się robot umieszczony w dowolnym polu labiryntu ustawiony w podanej orientacji (N S W E). Zadaniem jest opracowanie i zaimplementowanie algorytmu sterowania robotem w taki sposób żeby robot doszedł do wyjścia z labiryntu przy jak najmniejszym koszcie - liczbie ruchów.

Jako ruch robota liczy się:

- Przesunięcie robota o jedno pole do przodu – koszt 1,
- Przesunięcie robota o jedno pole do tyłu – koszt 1,
- Przesunięcie – teleportacja - robota z pola A do pola B na planszy, gdzie pole B musiało być wcześniej widziane przez robota – koszt jest równy sumie odległości między polami w wierszach i kolumnach (teleportacja z pola (2,4) do pola (8,1) - koszt $|2-8|+|4-1|=9$).

Pole, które widział robot, to pola które: robot odwiedził oraz wszystkie pola które przylegają do pól, które robot odwiedził i nie są od nich oddzielone ścianką.

Należy zaimplementować:

- klasę pochodną od *RobotBehaviourBase*, która ma implementować algorytm zachowania robota,
- definicję metody *create* klasy *RobotBehaviourFactory*.

Klasa *RobotBehaviourBase*

Klasa bazowa dla klas implementujących algorytm zachowania robota.

```
class RobotBehaviourBase
{
public:
  RobotBehaviourBase(size_t const xSize, size_t const ySize, Robot& robot);
  virtual void reinit()=0;
  virtual void explore(Position const& startPosition,Direction const& startDirection)=0;
};
```

Konstruktor

parametry:

- *xSize*, *ySize* – rozmiar labiryntu, odpowiednio liczba kolumn i wierszy,
- *robot* – referencja do obiektu reprezentującego robota umożliwiająca sterowanie robotem i rozpoznawanie otoczenia.

reinit

Metoda wywoływana przed rozpoczęciem nowego badania labiryntu.

explore

Metoda w której należy sterować robotem za pośrednictwem przekazanego w konstruktorze obiektu typu Robot aż do osiągnięcia wyjścia z labiryntu.

Parametry:

- *startPosition* – pozycja początkowa robota,
- *startDirection* – początkowe ustawienie robota (N S W E).

Klasa *RobotBehaviourFactory*

Fabryka tworząca obiekty odpowiadające za zachowanie robota – pochodne od klasy *RobotBehaviourBase*.

```
class RobotBehaviourFactory
{
public:
  std::unique_ptr<RobotBehaviourBase> create(size_t const xSize,size_t const ySize,
                                             Robot& robot);
};
```

create

Metoda zwraca unique_pointer na obiekt odpowiadające za zachowanie robota – pochodny od klasy *RobotBehaviourBase*.

Parametry:

- *xSize*, *ySize* – rozmiar labiryntu, odpowiednio liczba kolumn i wierszy,
- *robot* – referencja do obiektu reprezentującego robota.

Klasa *Robot*

Klasa reprezentująca robota umożliwiająca sterowanie robotem i rozpoznawanie otoczenia. Jest dostarczana do obiektu implementującego algorytm zachowania robota (klasy pochodnej od *RobotBehaviourBase*). Umożliwia sterowanie robotem oraz badanie jego otoczenia.

```
class Robot
{
public:
	static size_t const minMazeXSize = 2;
	static size_t const minMazeYSize = 2;
	static size_t const maxMazeXSize = 256;
	static size_t const maxMazeYSize = 256;

	virtual void teleport(Position const& position) = 0;

	virtual void forward() = 0;
	virtual void backward() = 0;
	virtual void turnLeft() = 0;
	virtual void turnRight() = 0;

	virtual bool isWallFront() const = 0;
	virtual bool isWallBack() const = 0;
	virtual bool isWallLeft() const = 0;
	virtual bool isWallRight() const = 0;
```

```
        virtual bool isInExit() const = 0;
};
```

minMazeXSize, minMazeYSize, maxMazeXSize, maxMazeYSize

Stałe określające minimalną i maksymalną wielkość labiryntu.

teleport

Przeniesienie robota do podanego pola na planszy. Pole musiało być wcześniej zobaczone przez robota.

Parametry:

- *position* – współrzędne pola do którego ma zostać przeniesiony robot.

forward

Przesunięcie robota o jedno pole do przodu.

backward

Przesunięcie robota o jedno pole do tyłu.

turnLeft

Obrót robota w lewo o 90°.

turnRight

Obrót robota w prawo o 90°.

isWallFront

Metoda sprawdza czy z przodu (przed robotem) jest ściana labiryntu.

isWallBack

Metoda sprawdza czy z tyłu (za robotem) jest ściana labiryntu.

isWallLeft

Metoda sprawdza czy z lewej (z lewej strony robota) jest ściana labiryntu.

isWallRight

Metoda sprawdza czy z prawej (z prawej strony robota) jest ściana labiryntu.

Aplikacja

Parametry

Aplikacja może być uruchomiona z jednym lub dwoma parametrami.

Pierwszy parametr „-d” jest opcjonalny i jeśli zostanie użyty aplikacja wyświetla stan planszy po każdym ruchu robota.

Drugi parametr jest obowiązkowy i jest to nazwa pliku z definicją labiryntu.

Wynik

Aplikacja wyświetla koszt (liczba ruchów) dojścia przez robota do wyjścia z labiryntu lub -1 jeśli nie udało się dojść do wyjścia.

Format pliku z definicją labiryntu

Labirynt jest definiowany w pliku tekstowym w którym w kolejnych liniach są zapisane:

- rozmiar labiryntu, dwie liczby określające liczbę kolumn i liczbę wierszy,
- początkowe położenie i kierunek robota, dwie liczby (numer kolumny i numer wiersza – od 0) określające położenie i litera (N S W E) określająca kierunek ustawienia robota,
- informacje o kolejnych ścianach pionowych w labiryncie w kolejności od lewej do prawej i od góry do dołu, 0 oznacza brak ściany a 1 obecność ściany,
- informacje o kolejnych ścianach poziomych w labiryncie w kolejności od lewej do prawej i od góry do dołu, 0 oznacza brak ściany a 1 obecność ściany,

Przykład:

```
6 6
0 0 S
0 0 0 0 1
1 0 0 0 1
1 0 0 1 0
0 1 0 0 0
0 0 1 1 0
0 0 0 1 0
0 0 1 1 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 1 0
0 1 1 0 0 0
```