Algorytm wyszukiwania A*

Autor: Konrad Obal

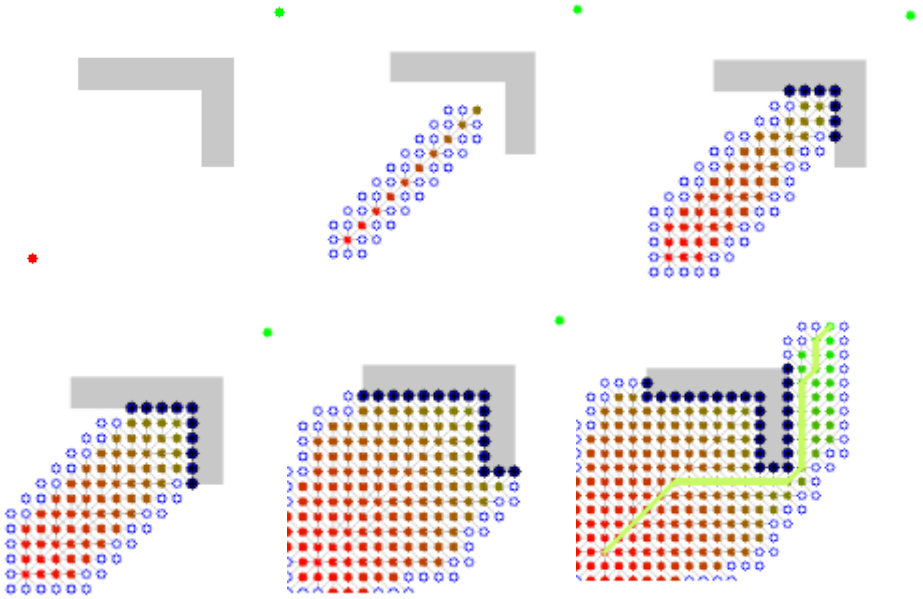
Opis algorytmu

Jest to algorytm heurystyczny znajdowania najkrótszej ścieżki w grafie ważonym z dowolnego wierzchołka do wierzchołka spełniającego określony warunek zwany testem celu. Algorytm jest zupełny i optymalny, w tym sensie, że znajduje ścieżkę, jeśli tylko taka istnieje, i przy tym jest to ścieżka najkrótsza. Stosowany głównie w dziedzinie sztucznej inteligencji do rozwiązywania problemów i w grach komputerowych do imitowania inteligentnego zachowania.

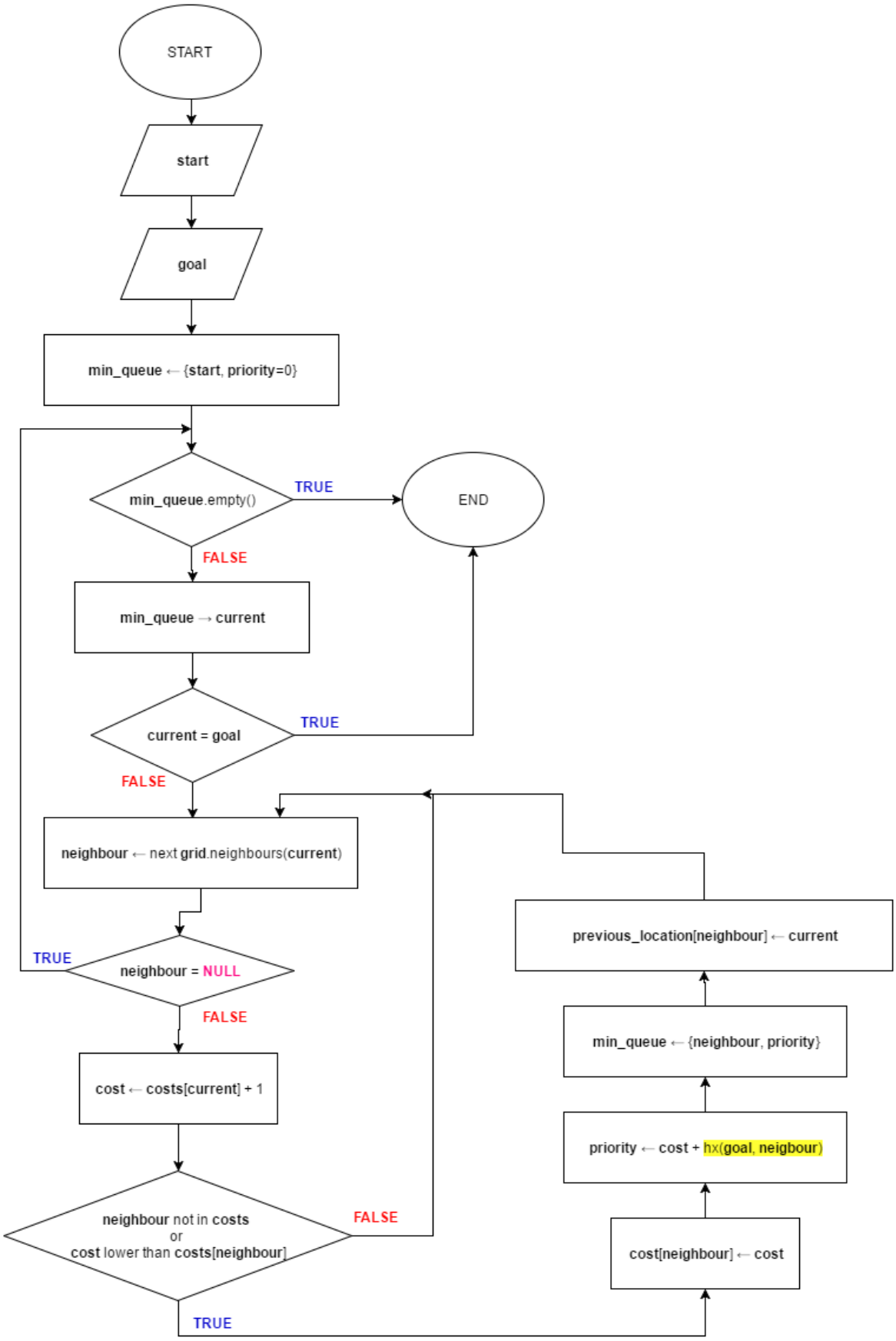
Opis implementacji

Implementacja algorytmu A* w języku C++ dotyczy siatki punktów (przestrzeni girdowej), której zastosowanie znacząco ułatwia napisanie funkcji heurystycznej, potrzebnej do prawidłowej pracy algorytmu. W prostych (i nie tylko) grach komputerowych bardzo często właśnie w takiej postaci jest przedstawiana mapa obszaru gry. W implementacji założono, że przestrzeń składa się tylko z dwóch typów „operacyjnej" – czyli takiej po której można się swobodnie poruszać, a koszt ruchu z jednego punktu do sąsiedniego wynosi 1, oraz „ścian" czyli punktów wyłączonych z ruchu. Każdy punkt w przestrzeni jest reprezentowany przez składowe X i Y, będące odpowiednio składową osi poziomej oraz pionowej.

Ilustracja pracy algorytmu szukającego najkrótszej ścieżki na siatce punktów



Schemat blokowy algorytmu A* dla siatki punktów (grid)



Pseudokod algorytmu A*

```
grid – siatka punktów, ma zdefiniowane metodę:
       .neighbours(punkt) zwraca tablice punktów sąsiadujących
start – punkt startowy na siatce punktów
goal – punkt końcowy na siatce punktów
min_queue – kolejka priorytetowa typu min
previous_location – mapa przechowująca dla kluczy poprzedni punkt
                    jako wartość
costs – mapa przechowująca minimalny koszt dojścia z punktu startowego
        do tego podanego jako klucz

function hx(a, b) – optymalna funkcja heurystyczna od niej zależy
                    optymalność algorytmu A-star

   return difference between a and b

end function

min_queue.enque(start, priority := 0)

while not min_queue.empty()
then

   current := min_queue.dequeue()

   if current = goal then break;

   foreach grid.neighbours(current) as neighbour

      // Zakładamy, że koszt przejścia do następnego punktu na siatce
      // punktów wynosi „1”
      cost := 1 + costs[current]

      if neighbour not in costs or cost lower than costs[neighbour]
      then

         costs[neighbour] := cost
         priority := cost + hx(goal, neighbour)
         min_queue.enque(neighbour, priority)
         previous_location[neighbour] := current

      end if

    end foreach

end while
```

Demonstracja załączonej implementacji

```
Width: 16
Height: 12
How many walls: 8
Grid[16x12]
X-axis horizontally; Y-axis vertically

      0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
  0 [ ] [ ] [ ] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
  1 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
  2 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [ ] [ ]
  3 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [ ] [ ]
  4 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [X] [X]
  5 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [ ] [ ] [X] [X]
  6 [ ] [ ] [ ] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
  7 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [X] [X]
  8 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [X] [X]
  9 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [X] [X]
 10 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [X]
 11 [ ] [ ] [ ] [ ] [ ] [X] [X] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [X]

Start coordinates [x, y]: 0 11
Goal coordinates [x, y]: 15 0


Grid[16x12]
X-axis horizontally; Y-axis vertically

      0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
  0 [ ] [ ] [ ] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [ ] [ ] [ ] [@]
  1 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [ ] [ ] [ ] [@]
  2 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [@] [@]
  3 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [@] [@] [ ]
  4 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [@] [X] [X]
  5 [ ] [X] [X] [X] [X] [X] [X] [X] [X] [ ] [ ] [ ] [@] [@] [X] [X]
  6 [ ] [ ] [ ] [X] [X] [X] [X] [X] [X] [ ] [@] [@] [@] [ ] [ ] [ ]
  7 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [@] [ ] [ ] [X] [X]
  8 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [@] [ ] [ ] [X] [X]
  9 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [@] [ ] [ ] [X] [X]
 10 [ ] [ ] [ ] [@] [@] [@] [@] [@] [@] [@] [@] [ ] [ ] [ ] [ ] [X]
 11 [@] [@] [@] [@] [ ] [X] [X] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [X]

Total cost: 26
Press any key to continue . . .
```

Info: Kod źródłowy został skompilowany w Viusal Studio 2015 wersja Release/x64.

```
1>------ Rebuild All started: Project: ASiD-Project, Configuration: Release x64 ------
1>  Grid.cpp
1>  main.cpp
1>  Generating code
1>  All 1149 functions were compiled because no usable IPDB/IOBJ from previous compilation was found.
1>  Finished generating code
1>  ASiD-Project.vcxproj -> d:\docs\visual studio 2015\Projects\ASiD-Project\x64\Release\ASiD-Project.exe
1>  ASiD-Project.vcxproj -> d:\docs\visual studio 2015\Projects\ASiD-Project\x64\Release\ASiD-Project.pdb (Full PDB)
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Źródła:

- Działanie oraz pseudokod algorytmu
https://en.wikipedia.org/wiki/A*_search_algorithm
http://www.redblobgames.com/pathfinding/a-star/introduction.html

- Implementacja
http://en.cppreference.com/w/cpp/container/unordered_map
http://en.cppreference.com/w/cpp/container/unordered_set
http://en.cppreference.com/w/cpp/utility/hash
http://stackoverflow.com/questions/2439283/how-can-i-create-min-stl-priority-queue