# Algorytm wyszukiwania A\*

Autor: Konrad Obal

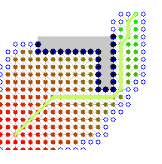
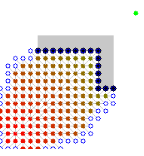
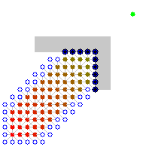
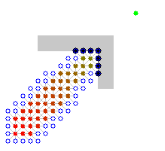
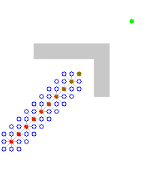
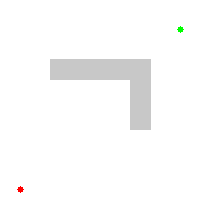
## Opis algorytmu

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) D:\Downloads\A-star.png

## 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

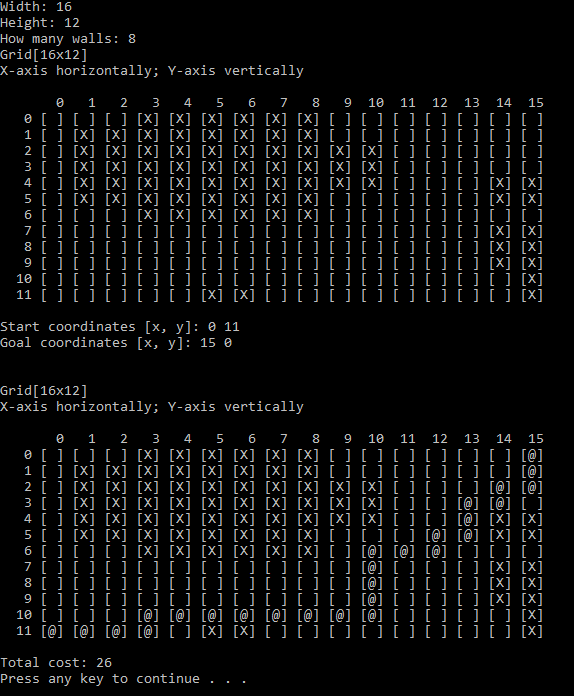
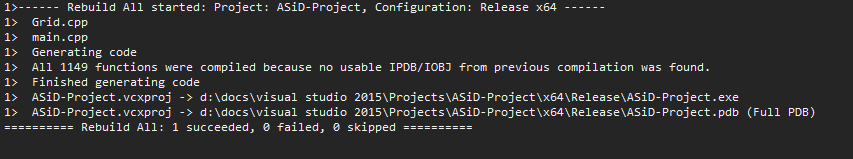
**costs**[**neighbour**] := **cost**

priority := **cost**+ hx(**goal**, **neighbour)**

**min\_queue**.enque(**neighbour**, priority)

**previous\_location**[**neighbour**] := **current** end if  
  
 end foreach  
  
end while

## Demonstracja załączonej implementacji

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


*Źródła*:  
  
- Działanie oraz pseudokod algorytmu   
[https://en.wikipedia.org/wiki/A\*\_search\_algorithm](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>