Programowanie dla fizyków 2

Raport

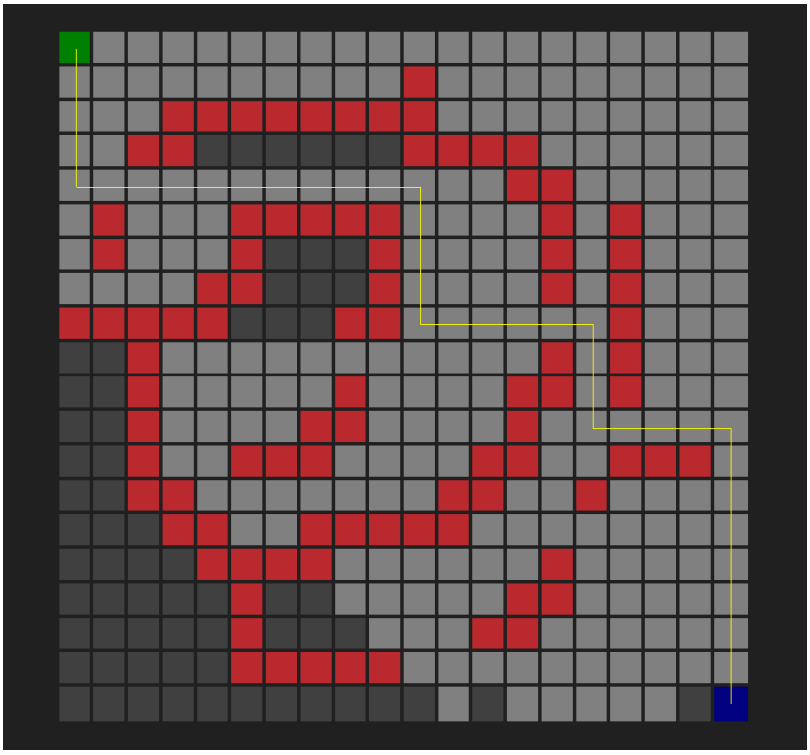
Algorytm A*

Izabela Czarny
Michał Kreft

11 czerwca 2021

Wstęp

W naszym projekcie obraliśmy temat algorytmu A*, odpowiada za znalezienie najkrótszej drogi z punktu A do punktu B, priorytetyzuje potencjalnie najlepszą drogę jako pierwszą do sprawdzenia. Ścieżka zostanie zawsze znaleziona pod warunkiem że istnieje. Jest stosowany głównie w dziedzienie sztucznej inteligencji i w grach komputerowych do imitowania inteligentnego zachowania.



Obsługa

Trzymając/klikając lewy przycisk myszy tworzymy lub usuwamy ściany. Dodając do tego przycisk 'D' ustawiamy początek, a dodając 'F' ustawiamy koniec. Przyciskiem 'R' resetujemy ściany. Po każdej takiej zmianie szukamy nowej drogi. Kółkiem myszy możemy powiększyć lub pomniejszyć obszar widzenia.

Heurestyka

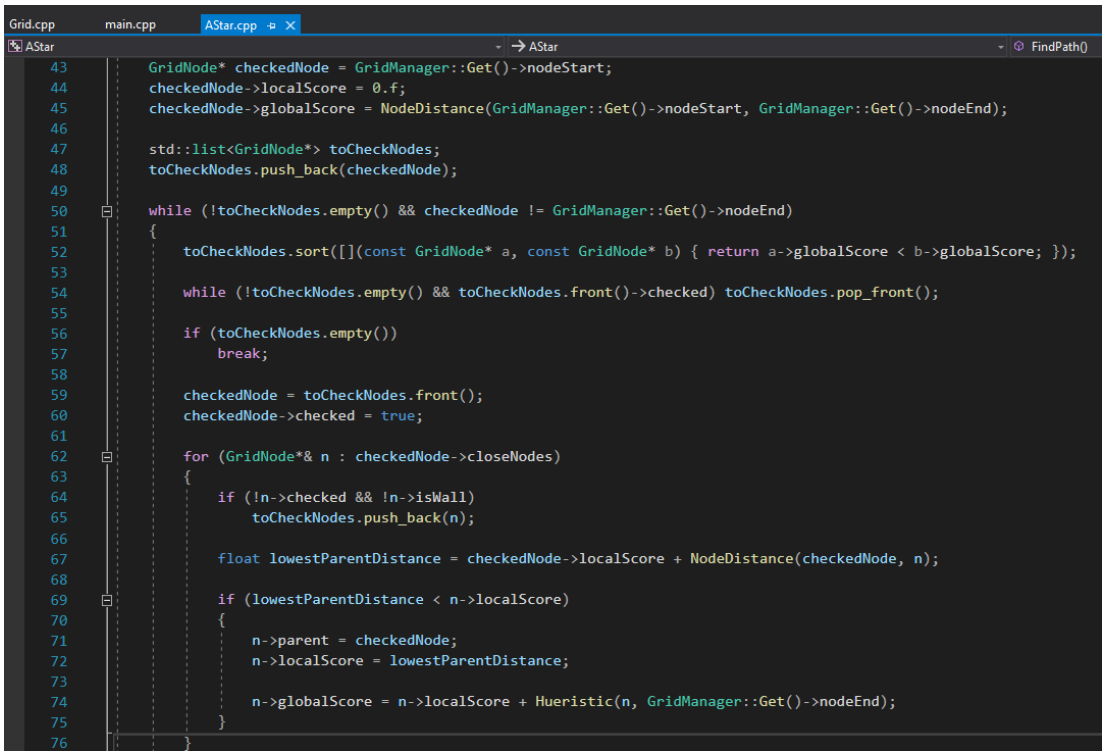
Aby algorytm wiedział jak dobry jest kierunek poszukiwań, potrzebujemy funkcji określającej odległość stanu do rozwiązania. Taką funkcje nazywamy funkcją huerestyczną, ideał takiej funkcji zwraca wartość dokładnego kosztu przejścia z aktualnego punktu do rozwiązania. Dzięki takiemu rozwiązaniu praca algorytmu była by bezbłędna, lecz uzyskanie dokładnej wartości może być skomplikowane i czasochłonne, a w praktyce niemożliwe do zastosowania. Do znalezienia rozwiązania wystarczy aby funkcja nie zwracała wartości większej niż jej faktyczny koszt przejścia z danego punktu do końca.

Sąsiedzi

Podczas tworzenia nodów przypisujemy każdemu jego sąsiadów, u nas są to horyzontalne przypisy, ale nic nie stoi na drodze aby przypisać sąsiadów diagonalnie. W takim wypadku program sprawdzał by drogę również na skos.

Sposób działania algorytmu

Cały algorytm jest zawarty w osobnych plikach(AStar.h, AStar.cpp). Algorytm sortuje pola wyznaczając im punkty za jakość rozwiązania(localScore, globalScore). Upewniamy się że wyniki jakości drogi są zresetowane. Tworzymy pustą liste nodów, będziemy w niej trzymać wszystkie nody do sprawdzenia. Dodajemy pierwszą, czyli start. Tworzymy pętle sprawdzającą czy nie znaleźliśmy końca, przy każdym powtórzeniu sortujemy nody względem wyniku globalnego. Łączymy pola i wszystkich sąsiadów kolejno sprawdzanych pół dodajemy do listy. Do każdego noda przypisujemy najlepsze pole ruchu. Na końcu się cofamy i od końca rysujemy trasę uwzględniając przypisane pola ruchu.




```
43      GridNode* checkedNode = GridManager::Get()->nodeStart;
44      checkedNode->localScore = 0.f;
45      checkedNode->globalScore = NodeDistance(GridManager::Get()->nodeStart, GridManager::Get()->nodeEnd);
46
47      std::list<GridNode*> toCheckNodes;
48      toCheckNodes.push_back(checkedNode);
49
50      while (!toCheckNodes.empty() && checkedNode != GridManager::Get()->nodeEnd)
51      {
52          toCheckNodes.sort([](const GridNode* a, const GridNode* b) { return a->globalScore < b->globalScore; });
53
54          while (!toCheckNodes.empty() && toCheckNodes.front()->checked) toCheckNodes.pop_front();
55
56          if (toCheckNodes.empty())
57              break;
58
59          checkedNode = toCheckNodes.front();
60          checkedNode->checked = true;
61
62          for (GridNode*& n : checkedNode->closeNodes)
63          {
64              if (!n->checked && !n->isWall)
65                  toCheckNodes.push_back(n);
66
67              float lowestParentDistance = checkedNode->localScore + NodeDistance(checkedNode, n);
68
69              if (lowestParentDistance < n->localScore)
70              {
71                  n->parent = checkedNode;
72                  n->localScore = lowestParentDistance;
73
74                  n->globalScore = n->localScore + Hueristic(n, GridManager::Get()->nodeEnd);
75              }
76          }
```

Sposób wykoniania

Do stworzenia okna, renderowania grafiki używamy API SFML dla języka C++.

Bibliografia

[1]www.pl.wikipedia.org/wiki/Algorytm_A*

[2]www.pl.wikipedia.org/wiki/Graf_(matematyka)

[3]www.pl.wikipedia.org/wiki/Algorytm_Dijkstry

[4]www.101computing.net/a-star-search-algorithm/

[5]www.xevaquor.wordpress.com/2015/03/09/gwiazda-wieczoru-algorytm-a-a-star/