Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

ОТЧЕТ

по лабораторной работе

**«АЛГОРИТМ ПОИСКА КРАТЧАЙШЕГО ПУТИ»**

Студент гр. 924403 К.Д. САНДАРОВИЧ

Проверил С.О. ЛУКАШОВ

Минск 2022

# Общая постановка задачи

## Краткие теоретические сведения

Благодаря простоте и эффективности, а также широким возможностям оптимизации алгоритм А\* чаще других используется для поиска пути. Не случайно существует несколько алгоритмов, основанных на нем. Поскольку алгоритм А\* имеет общие корни с алгоритмом Дейкстры, в их реализациях просматривается сходство.

## Системные подходы к решению задачи

Так же как алгоритм Дейкстры, этот рецепт использует бинарную кучу с сайта GPWiki. Кроме того, важно понимать, что такое делегаты(обработчики) и для чего они предназначены. И наконец, в этом рецепте мы входим в мир осознанного поиска, а это означает, что вы должны понимать, что такое эвристика и для чего она нужна.

## Разбор решения поставленной задачи

Алгоритм основан на эвристической функции вычисления приблизительной стоимости перехода между двумя вершинами, которая позволяет сравнивать альтернативы и выбирать те, что характеризуются минимальной стоимостью.

Внесем небольшие изменения в класс Graph:

1. Определим ссылку на делегата:

public delegate float Heuristic(Vertex а, Vertex b);

2. Реализуем функцию вычисления эвклидова расстояния для иcпользования в качестве эвристики по умолчанию:

public float EuclidDist(Vertex а, Vertex Ь)

{

VectorЗ posA = a.transform.position;

VectorЗ posB = b.transform.position;

return VectorЗ.Distance(posA, posB);

}

3. Реализуем функцию манхэттенского расстояния для использования в качестве другой эвристики. Это позволит сравнить результаты с использованием различных эвристик:

public float ManhattanDist(Vertex а, Vertex b)

{

VectorЗ posA = a.transform. position;

VectorЗ posB = b.transform.position;

return мathf .AЬs(posA.x - posB .x) + мathf .AЬs(posA.y - posB.y);

}

**Как это работает**

Алгоритм A\* работает аналогично алгоритму Дейкстры. Однако вместо узлов с действительно низкой стоимостью он выбирает наиболее перспективные, основываясь на заданной эвристической функции. В нашем случае в качестве эвристики по умолчанию используется эвклидово расстояние между вершинами и дополнительно имеется возможность использовать манхэттенское расстояние.

# Блок-схема алгоритма

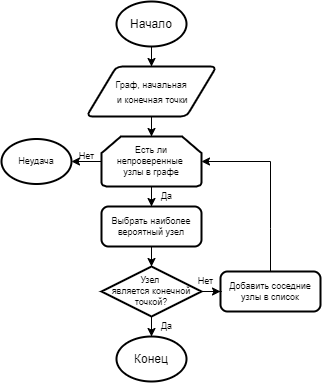
****

Рисунок1 – Блок схема алгоритма А\*

# Листинг

public List<Vertex> GetPathAstar(GameObject srcObj, GameObject dstObj, Heuristic h = null)

{

if(srcObj == null || dstObj == null) return new List<Vertex>();

if(ReferenceEquals(h, null)) h = EuclidDist;

Vertex src = GetNearestVertex(srcObj.transform.position);

Vertex dst = GetNearestVertex(dstObj.transform.position);

GPWiki.BinaryHeap<Edge> frontier = new GPWiki.BinaryHeap<Edge>()

Edge[] edges; Edge node, child;

int size = vertices.Count;

float[] distValue = new float[size];

int[] previous = new int[size];

node = new Edge(src, 0);

frontier.Add(node);

distValue[src.idJ = 0;

previous[src.id] = src.id;

for(int i = 0; i < size; i++)

{

if(i == src.id) continue;

distValue[i] = Mathf.Infinity;

previous[i] = -1;

}

while(frontier.Count ! = 0)

{

node = frontier.Remove();

int nodeId = node.vertex.id;

if(ReferenceEquals(node.vertex, dst))

return BuildPath(src.id, node.vertex.id, ref previous);

edges = GetEdges {node .vertex);

foreach(Edge е in edges)

{

int eId = e.vertex.id;

if(previous[eld] != -1) continue;

float cost = distValue[nodeld) + e.cost;

cost += h(node.vertex, e.vertex);

if(cost < distValue[e.vertex.id] )

{

distValue[eId) = cost;

previous[eId] = nodeId;

frontier.Remove(e);

child = new Edge(e.vertex, cost);

frontier .Add(child);

}

}

}

return new List<Vertex>();

}

**Вывод**

В процессе выполнения лабораторной работы был изучен и реализован алгоритм поиска пути А\*.