|  |
| --- |
| https://lh6.googleusercontent.com/QcftzNtI05T0Y6fjdSh1Rr2rt8oqZ1IvnLvbn1jLJ7CCyteVir3k-xBLv4SL1wAgWJsRhmmJSR0UW-RP63_GQenE4vVWv05BRoZTsmIcBccVTnfxwmsnNMvjg599x9SqZd8E3dkd |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА** - **Российский технологический университет»**  **РТУ МИРЭА** |

Институт искусственного интеллекта (ИИИ)

Кафедра проблем управления

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1** | |
| **по дисциплине** | |
| «Агентно-ориентированные системы автономного управления» | |
|  | |
| Выполнили студенты группы КРМО-01-23 | Галанин В.А.  Пичугин Д.Ю. |
| Принял | Голубов В.В. |

Практические работы выполнены «\_\_» \_\_\_\_\_ 2023 г.

(подпись студента)

«Зачтено» «\_\_» \_\_\_\_\_ 2023 г.

(подпись преподавателя)

Москва 2023

Код программы, написанный на языке C++, представлен в Листинге 1. Также представлен на GitHub: <https://github.com/vitoscape/AOS>.

В строке 129 объявляется карта в виде вектора векторов типа int. 0 — клетка свободна, 1 — клетка занята. В строках 201-204 в векторах startX, startY, targetX, targetY задаются стартовые и конечные точки агентов.

Для вычисления путей агентов используется класс AStar.

Для визуализации используется библиотека SFML.

Результат выполнения кода показан на Рисунке 1.

Листинг 1 — Алгоритм A\*

#include <SFML/Graphics.hpp>

#include <vector>

#include <iostream>

#include <queue>

#include <cmath>

const int mapWidth = 10; *// Ширина карты*

const int mapHeight = 10; *// Высота карты*

const int agentCount = 3; *// Кол-во агентов*

const int moveCost = 1; *// Стоимость передвижения*

class Map {

private:

int mapWidth;

int mapHeight;

std::vector<std::vector<int>> data;

public:

Map(int width, int height, std::vector<std::vector<int>> mapData) : mapWidth(width), mapHeight(height), data(mapData) {}

int getWidth() const { return mapWidth; }

int getHeight() const { return mapHeight; }

std::vector<std::vector<int>> getData() const { return data; }

bool isObstacle(int x, int y) const { return data[x][y] == 1; }

};

class Node {

private:

int x;

int y;

Node \*parent;

Продолжение Листинга 1

int gCost;

int hCost;

public:

Node(int xPos, int yPos, Node \*p, int g, int h) : x(xPos), y(yPos), parent(p), gCost(g), hCost(h) {}

int getX() const { return x; }

int getY() const { return y; }

Node \*getParent() const { return parent; }

int getGCost() const { return gCost; }

int getHCost() const { return hCost; }

int getFCost() const { return gCost + hCost; }

};

class AStar {

private:

struct CompareNodes {

bool operator()(const Node \*lhs, const Node \*rhs) const {

return lhs->getFCost() > rhs->getFCost();

}

};

public:

static int manhattanDistance(int x1, int y1, int x2, int y2) {

return std::abs(x1 - x2) + std::abs(y1 - y2);

}

static std::vector<std::vector<std::pair<int, int>>> findPaths(const std::vector<int>& start\_X, const std::vector<int>& start\_Y, const std::vector<int>& target\_X, const std::vector<int>& target\_Y, const Map& map, std::vector<std::vector<bool>>& occupied) {

std::vector<std::vector<std::pair<int, int>>> allPaths;

for (size\_t i = 0; i < start\_X.size(); ++i) {

std::vector<std::pair<int, int>> path;

std::priority\_queue<Node\*, std::vector<Node\*>, CompareNodes> openSet;

int mapWidth = map.getWidth();

int mapHeight = map.getHeight();

std::vector<std::vector<bool>> closedSet(mapWidth, std::vector<bool>(mapHeight, false));

Node\* startNode = new Node(start\_X[i], start\_Y[i], nullptr, 0, manhattanDistance(start\_X[i], start\_Y[i], target\_X[i], target\_Y[i]));

openSet.push(startNode);

while (!openSet.empty()) {

Продолжение Листинга 1

Node\* currentNode = openSet.top();

openSet.pop();

if (currentNode->getX() == target\_X[i] && currentNode->getY() == target\_Y[i]) {

while (currentNode != nullptr) {

path.emplace\_back(std::make\_pair(currentNode->getX(), currentNode->getY()));

currentNode = currentNode->getParent();

}

break;

}

closedSet[currentNode->getX()][currentNode->getY()] = true;

std::vector<std::pair<int, int>> neighbors = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

for (const auto& neighbor : neighbors) {

int neighborX = currentNode->getX() + neighbor.first;

int neighborY = currentNode->getY() + neighbor.second;

if (neighborX >= 0 && neighborX < mapWidth && neighborY >= 0 && neighborY < mapHeight

&& map.getData()[neighborX][neighborY] == 0 && !closedSet[neighborX][neighborY] && !occupied[neighborX][neighborY]) {

int gCost = currentNode->getGCost() + 1;

int hCost = manhattanDistance(neighborX, neighborY, target\_X[i], target\_Y[i]);

Node\* neighborNode = new Node(neighborX, neighborY, currentNode, gCost, hCost);

openSet.push(neighborNode);

}

}

}

while (!openSet.empty()) {

delete openSet.top();

openSet.pop();

}

allPaths.push\_back(path);

*// Обновление массива occupied для занятых клеток пути*

for (auto& node : path) {

occupied[node.first][node.second] = true;

}

}

return allPaths;

}

};

Продолжение Листинга 1

int main() {

sf::RenderWindow window(sf::VideoMode(400, 400), "Multi-Agent Pathfinding");

std::vector<std::vector<int>> mapData = {

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 1, 0, 0, 0, 0},

{0, 0, 1, 0, 0, 1, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 1, 0, 0, 0, 0},

{0, 0, 0, 0, 1, 0, 0, 0, 0, 0},

{0, 0, 1, 1, 1, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 1, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 1, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 1, 0}

};

Map map(10, 10, mapData);

*// Инициализация SFML фигур и цветов агентов*

sf::Color darkRed(255, 139, 139); *// Темно-красный*

sf::Color darkGreen(139, 255, 139); *// Темно-зеленый*

sf::Color darkBlue(139, 139,255); *// Темно-синий*

sf::Color gray(170, 170, 170); *// Серый*

sf::CircleShape agentShapes[agentCount]; *// Массив точек начальных координат агентов*

sf::CircleShape targetShapes[agentCount]; *// Массив точек конечных координат агентов*

sf::Color agentColors[agentCount] = {sf::Color::Red, sf::Color::Green, sf::Color::Blue}; *// Массив цветов начальных точке агентов*

sf::Color targetColors[agentCount] = {sf::Color::Red, sf::Color::Green, sf::Color::Blue}; *// Массив цветов конечных точекагентов*

sf::Color pathColors[agentCount] = {darkRed, darkGreen, darkBlue}; *// Массив цветов путей агентов*

for (int i = 0; i < agentCount; ++i) {

*// Назаначение фигур начальных координат агентов*

agentShapes[i] = sf::CircleShape(6); *// Изменение формы на круг радиусом 2*

agentShapes[i].setFillColor(agentColors[i]);

*// Назаначение фигур конечных координат агентов*

targetShapes[i] = sf::CircleShape(4); *// Изменение формы на круг радиусом 2*

targetShapes[i].setFillColor(sf::Color::Yellow);

targetShapes[i].setOutlineThickness(2);

targetShapes[i].setOutlineColor(agentColors[i]);

}

*//std::vector<std::vector<bool>> occupied(map.getWidth(), std::vector<bool>(map.getHeight(), false));*

*// Цикл визуализации*

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

Продолжение Листинга 1

if (event.type == sf::Event::Closed) {

window.close();

}

}

window.clear();

*// Визуализация карты*

sf::RectangleShape tile(sf::Vector2f(40, 40));

sf::CircleShape dot(2);

dot.setFillColor(gray);

for (int i = 0; i < mapWidth; ++i) {

for (int j = 0; j < mapHeight; ++j) {

if (map.getData()[i][j] == 1) {

tile.setFillColor(sf::Color::Black);

} else {

tile.setFillColor(sf::Color::White);

dot.setPosition(i \* 40 + 20, j \* 40 + 20);

}

tile.setPosition(i \* 40, j \* 40);

window.draw(tile);

window.draw(dot);

}

}

*////////////////////////////////////////////////////////////////////////////////////////// Начальные и конечные координаты*

*// Визуализация путей агентов*

std::vector<int> startX = {2, 5, 9};

std::vector<int> startY = {2, 5, 9};

std::vector<int> targetX = {3, 2, 6};

std::vector<int> targetY = {6, 8, 1};

std::vector<std::vector<bool>> occupied(map.getWidth(), std::vector<bool>(map.getHeight(), false));

auto allPaths = AStar::findPaths(startX, startY, targetX, targetY, map, occupied);

*// Обновление массива occupied*

for (auto &path : allPaths) {

for (auto &node : path) {

occupied[node.first][node.second] = true;

}

}

for (size\_t i = 0; i < allPaths.size(); ++i) {

for (auto &node : allPaths[i]) {

sf::CircleShape pathTile(4);

pathTile.setFillColor(pathColors[i]);

pathTile.setPosition(node.first \* 40 + 19, node.second \* 40 + 19);

Окончание Листинга 1

window.draw(pathTile);

}

*// Визуализация начальных координат агентов*

agentShapes[i].setPosition(startX[i] \* 40 + 18, startY[i] \* 40 + 18);

window.draw(agentShapes[i]);

*// Визуализация конечных координат агентов*

targetShapes[i].setPosition(targetX[i] \* 40 + 18, targetY[i] \* 40 + 18);

window.draw(targetShapes[i]);

}

window.display();

}

return 0;

}

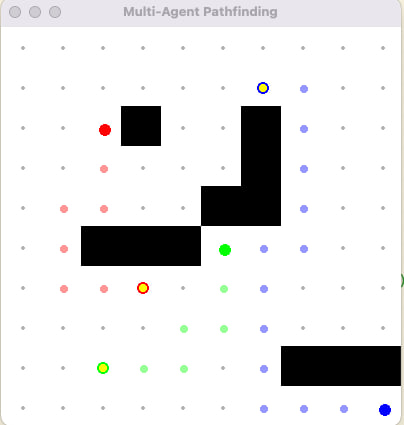


Рисунок 1 — Визуализация путей, алгоритм A\*

Красными точками обозначен путь агента 1, зелеными — агента 2, синими — агента 3. Черными квадратами обозначены препятствия, которые при объявлении карты обозначались единицами. Как мы можем заметить, алгоритм успешно нашел пути алгоритма, а также обошел препятствия.

Алгоритм потенциальных полей представлен в Листинге 2.

Для вычислений используется класс PotentialFields.

Результат выполнения показан на Рисунке 2.

Листинг 2 — Алгоритм потенциальных полей

#include <SFML/Graphics.hpp>

#include <vector>

#include <iostream>

#include <string>

#include <cmath>

#include <unistd.h>

const int mapWidth = 10; *// Ширина карты*

const int mapHeight = 10; *// Высота карты*

const int agentCount = 3; *// Количество агентов*

struct Path {

std::vector<sf::Vector2i> points;

};

std::vector<std::vector<int>> mapData = {

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 1, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 1, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 1, 1, 1, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

};

class Agent {

public:

int x, y;

sf::CircleShape shape;

int targetX, targetY;

bool reachedTarget;

Path path;

Agent(int startX, int startY, int targetX, int targetY, sf::Color color) : x(startX), y(startY), targetX(targetX), targetY(targetY), reachedTarget(false) {

shape.setRadius(10);

shape.setFillColor(color);

Продолжение Листинга 2

shape.setPosition(x \* 40 + 10, y \* 40 + 10);

}

void addPathPoint(int x, int y) {

path.points.emplace\_back(x, y);

}

void setPath(const Path& newPath) {

path = newPath;

}

void move(int newX, int newY) {

x = newX;

y = newY;

shape.setPosition(x \* 40 + 10, y \* 40 + 10);

}

void drawPath(sf::RenderWindow& window) const {

sf::VertexArray lines(sf::LinesStrip);

lines.resize(path.points.size());

for (size\_t i = 0; i < path.points.size(); ++i) {

lines[i].position = sf::Vector2f(path.points[i].x \* 40 + 30, path.points[i].y \* 40 + 30);

lines[i].color = shape.getFillColor();

}

window.draw(lines);

}

void updatePosition() {

*// Вычисляем градиент потенциального поля и перемещаем агента в направлении увеличения потенциала.*

int dx = targetX - x;

int dy = targetY - y;

*// Нормализуем вектор направления.*

float length = std::sqrt(dx \* dx + dy \* dy);

if (length > 0) {

dx /= length;

dy /= length;

}

*// Перемещаем агента.*

int newX = x + dx;

int newY = y + dy;

*// Проверяем, не находится ли новая позиция агента внутри препятствий.*

if (isValidPosition(newX, newY)) {

move(newX, newY);

}

}

bool isAtTarget() const {

Продолжение Листинга 2

return x == targetX && y == targetY;

}

*// Добавим метод для проверки достижения цели.*

void checkReachedTarget() {

if (isAtTarget()) {

reachedTarget = true;

}

}

bool isValidPosition(int newX, int newY) const {

*// Проверяем, находится ли новая позиция агента в пределах карты и не внутри препятствий.*

return newX >= 0 && newX < mapWidth && newY >= 0 && newY < mapHeight && mapData[newX][newY] != 1;

}

};

class PotentialFields {

public:

static int manhattanDistance(int x1, int y1, int x2, int y2) {

return std::abs(x1 - x2) + std::abs(y1 - y2);

}

static int calculateAgentPotential(int x, int y, int agentX, int agentY) {

int distance = manhattanDistance(x, y, agentX, agentY);

return distance;

}

static int calculateGoalPotential(int x, int y, int goalX, int goalY) {

int distance = manhattanDistance(x, y, goalX, goalY);

return -distance;

}

static int calculateTotalPotential(int x, int y, const std::vector<int>& start\_X, const std::vector<int>& start\_Y, const std::vector<int>& target\_X, const std::vector<int>& target\_Y, const std::vector<std::vector<int>>& mapData, const std::vector<Agent>& agents) {

int totalPotential = 0;

for (size\_t i = 0; i < start\_X.size(); ++i) {

int agentPotential = calculateAgentPotential(x, y, start\_X[i], start\_Y[i]);

totalPotential += agentPotential;

}

for (size\_t i = 0; i < target\_X.size(); ++i) {

int goalPotential = calculateGoalPotential(x, y, target\_X[i], target\_Y[i]);

totalPotential += goalPotential;

}

Продолжение Листинга 2

*// Потенциал для препятствий*

if (mapData[x][y] == 1) {

totalPotential += 100; *// Значение для избегания препятствий*

}

*// Потенциал для клеток, которые заняты другими агентами*

for (const Agent& agent : agents) {

if (agent.x == x && agent.y == y) {

totalPotential += 50; *// Значение для клеток, занятых агентами*

}

}

return totalPotential;

}

};

int main() {

sf::RenderWindow window(sf::VideoMode(400, 400), "Multi-Agent Pathfinding");

*// std::vector<std::vector<int>> mapData = {*

*// {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},*

*// {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},*

*// {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},*

*// {0, 0, 0, 0, 0, 1, 0, 0, 0, 0},*

*// {0, 0, 0, 0, 0, 1, 0, 0, 0, 0},*

*// {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},*

*// {0, 0, 1, 1, 1, 0, 0, 0, 0, 0},*

*// {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},*

*// {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},*

*// {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}*

*// };*

sf::Color darkRed(255, 139, 139); *// Темно-красный*

sf::Color darkGreen(139, 255, 139); *// Темно-зеленый*

sf::Color darkBlue(139, 139, 255); *// Темно-синий*

sf::Color gray(170, 170, 170); *// Серый*

sf::CircleShape agentShapes[agentCount]; *// Массив точек начальных координат агентов*

sf::Color agentColors[agentCount] = {sf::Color::Red, sf::Color::Green, sf::Color::Blue}; *// Массив цветов начальных точке агентов*

sf::CircleShape targetShapes[agentCount]; *// Массив точек конечных координат агентов*

sf::Color targetColors[agentCount] = {sf::Color::Red, sf::Color::Green, sf::Color::Blue}; *// Массив цветов конечных точек агентов*

sf::Color pathColors[agentCount] = {darkRed, darkGreen, darkBlue}; *// Массив цветов путей агентов*

*// Создаем агентов и сохраняем их в векторе*

std::vector<Agent> agents;

std::vector<int> agentStartX = {0, 4, 9};

std::vector<int> agentStartY = {0, 4, 0};

std::vector<int> agentTargetX = {0, 4, 7};

std::vector<int> agentTargetY = {0, 4, 7};

for (int i = 0; i < agentCount; ++i) {

Продолжение Листинга 2

agents.push\_back(Agent(agentStartX[i], agentStartY[i], agentTargetX[i], agentTargetY[i], agentColors[i]));

}

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

window.close();

}

}

window.clear();

sf::RectangleShape tile(sf::Vector2f(40, 40));

sf::CircleShape dot(2);

dot.setFillColor(gray);

for (int col = 0; col < mapWidth; ++col) {

for (int row = 0; row < mapHeight; ++row) {

if (mapData[row][col] == 1) {

tile.setFillColor(gray);

} else {

tile.setFillColor(sf::Color::White);

dot.setPosition(col \* 40 + 20, row \* 40 + 20);

}

tile.setPosition(col \* 40, row \* 40);

window.draw(tile);

window.draw(dot);

}

}

*// Двигаем агентов к целям, если они еще не достигли их.*

for (Agent& agent : agents) {

if (!agent.reachedTarget) {

agent.updatePosition();

agent.checkReachedTarget();

*// Добавляем текущую позицию агента в путь.*

agent.addPathPoint(agent.x, agent.y);

}

}

*// for (int i = 0; i < agentCount; ++i) {*

*// Path agentPath = PotentialFields::findPath(mapData, {agentStartX[i], agentStartY[i]}, {agentTargetX[i], agentTargetY[i]});*

*// agents[i].setPath(agentPath);*

*// }*

*// Отрисовка агентов*

for (const Agent& agent : agents) {

window.draw(agent.shape);

agent.drawPath(window); *// Отрисовываем путь агента.*

}

Окончание Листинга 2

*// Отрисовка агентов*

*// for (const Agent& agent : agents) {*

*// window.draw(agent.shape);*

*// }*

*// Вывод значений потенциалов над каждой клеткой*

sf::Font font;

font.loadFromFile("/Users/vitaliy/C projects/Agent oriented systems/PR1/arial.ttf"); *// Укажите путь к файлу шрифта*

for (int x = 0; x < mapWidth; ++x) {

for (int y = 0; y < mapHeight; ++y) {

int totalPotential = PotentialFields::calculateTotalPotential(y, x, agentStartX, agentStartY, agentTargetX, agentTargetY, mapData, agents);

sf::Text text;

text.setFont(font);

text.setCharacterSize(10);

text.setFillColor(sf::Color::Black);

text.setString(std::to\_string(totalPotential));

text.setPosition(x \* 40 + 10, y \* 40 + 5);

window.draw(text);

}

}

window.display();

sleep (1);

}

return 0;

}

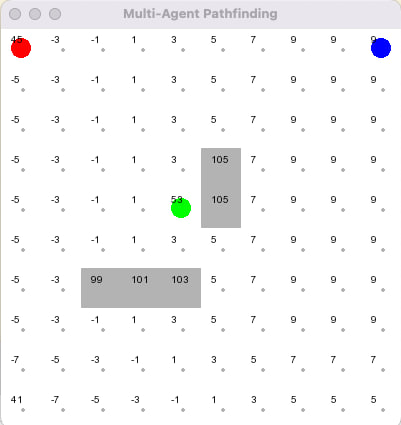


Рисунок 2 — Результат выполнения алгоритма потенциальных полей

Как можем увидеть, алгоритм просчитывает потенциал в каждой точке.

На GitHub по ранее представленной ссылке есть ветка main с алгоритмом A\*, ветка potentialField с алгоритмом потенциальных полей и ветка Python с выполненным в первую неделю заданием на языке Python.