**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«Российский университет дружбы народов имени Патриса Лумумбы»**

Инженерная академия

*Кафедра механики и процессов управления*

**Курсовая работа**

По информатике и программированию

**Направление:** Прикладная математика и информатика

**Профиль:** Математические методы механики космического полёта и анализ геоинформационных данных

**Тема:** Игровые алгоритмы: генерация лабиринтов и поиск пути для NPC **на С++**

Выполнено студентами: Кирюхин Антон Павлович,  
Жуков Марк Владимирович

Группа: ИПМбд-01-23

№ студенческого: 1132233512,  
1132233508

**Москва, 2025**

Генерация лабиринта — **DFS с возвратом (Backtracking)**

Что это за алгоритм?

Это один из самых простых и популярных способов создания лабиринтов. Он гарантирует, что лабиринт будет связным — то есть между любой парой клеток будет путь.

Принцип работы:

* Начинаем с одной стартовой клетки (обычно (1, 1)).
* Отмечаем текущую клетку как посещённую и превращаем её в проход (символ ' ').
* Выбираем **случайный** порядок направлений: вверх, вниз, влево, вправо.
* Для каждого направления:
* Переходим через 2 клетки (например, вправо — (x+2, y)) — так мы оставляем «стену» между проходами.
* Если соседняя клетка в пределах лабиринта и не была посещена:
* Помечаем промежуточную клетку (стену) как проход.
* Рекурсивно переходим в новую клетку.
* Если нет доступных направлений, **возвращаемся назад** по стеку (отсюда и название backtracking).
* Повторяем, пока стек не опустеет.

Почему через 2 клетки?

Потому что мы чередуем стены и проходы?

Таким образом, проходы не слипаются, и между ними всегда есть стены.

📍 Поиск пути — **Алгоритм A\***

Что это за алгоритм?

A\* (читается «эй-стар») — один из самых популярных и быстрых алгоритмов поиска кратчайшего пути на графе или сетке. Он работает быстрее, чем простой алгоритм Дейкстры, потому что использует **эвристику** — оценку, насколько мы близки к цели.

Как работает:

* У каждой клетки считаются:
* g — расстояние от старта до текущей клетки (реальная стоимость пути).
* h — оценка расстояния до цели (эвристика, в нашем случае — **манхэттенское расстояние**).
* f = g + h — общая оценка стоимости пути через эту клетку.
* Все клетки хранятся в **приоритетной очереди** (по наименьшему f).
* На каждом шаге:
* Извлекаем клетку с минимальным f.
* Проверяем, не достигли ли мы цели.
* Если нет — добавляем всех соседей в очередь (если они проходимы).
* Если сосед уже в очереди с меньшим f, то игнорируем его.
* Когда цель достигнута, восстанавливаем путь, двигаясь назад через parent-указатели.

Почему манхэттенское расстояние?

Потому что в нашем лабиринте можно двигаться только по вертикали и горизонтали. Манхэттенская эвристика — это просто:

ini

CopyEdit

h = |x1 - x2| + |y1 - y2|

Она быстрая и эффективная для таких сеток.

🌀 Анимация в терминале

Как реализовано:

* Когда путь найден, он записывается как список узлов (от цели до старта).
* Затем:
* Поочерёдно, от начала до конца, каждая клетка пути отмечается символом ·.
* После каждого шага вызывается std::system("clear") — чтобы очистить экран.
* Лабиринт выводится заново.
* usleep(100000) делает паузу в 100 мс.
* Таким образом, создаётся иллюзия движения — будто NPC идёт по лабиринту в реальном времени.