Документация по проекту

Крысиный лабиринт

В создание игры мы использовали такой язык программирования как C++ и использовали в помощь книгу как ‘’ Бонус к самоучителю уроки по C++’’.

Суть игры заключается в том то что, крыса, пытается найти в просторах лабиринта сыр. Работу выполняли Лев и Дмитрий, Лев делал движение крысы, а Дмитрий делал генерацию лабиринта .

Мы использовали такие библиотеки как: (#include "pch.h" #include "Board.h")*

#include <cstdlib> #include <ctime> #include <stack> #include <algorithm> #include <vector>

* - наши созданные библиотеки

Пояснение кода:

```
#pragma once
#include "Board.h"
class CRatMazeApp : public CWinApp {
public:
    virtual BOOL InitInstance();
};
class CRatMazeWnd : public CFrameWnd {
public:
    CRatMazeWnd();
    afx_msg void OnPaint();
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
    DECLARE_MESSAGE_MAP()
private:
    void RestartGame();
    Board m_board;
```

```
    int m_ratX;
    int m_ratY;
};
```

Объяснение:

```
BEGIN_MESSAGE_MAP(CRatMazeWnd, CFrameWnd)
    ON_WM_PAINT()
    ON_WM_KEYDOWN()
END_MESSAGE_MAP()
```

*__BEGIN_MESSAGE_MAP__ и __END_MESSAGE_MAP__: Макросы, объявляющие таблицу сообщений MFC для класса CRatMazeWnd.

```
CRatMazeApp theApp;
BOOL CRatMazeApp::InitInstance() {
    m_pMainWnd = new CRatMazeWnd();
    m_pMainWnd->ShowWindow(m_nCmdShow);
    m_pMainWnd->UpdateWindow();
    return TRUE;\
}
```

*Объект theApp: Глобальный объект приложения.

InitInstance(): Инициализирует приложение, создавая главное окно (CRatMazeWnd).

```
void CRatMazeWnd::OnPaint() {
```

```
    CPaintDC dc(this);
    for (int i = 0; i < m_board.GetRows(); ++i) {
        for (int j = 0; j < m_board.GetColumns(); ++j) {
            CBrush brush(m_board.GetColor(i, j));
            dc.FillRect(CRect(j * 20, i * 20, (j + 1) * 20, (i + 1) * 20), &brush);
```

*OnPaint(): Рисует текущую доску на экране. Каждая клетка рисуется в зависимости от значения в массиве m_arrBoard.

```
void CRatMazeWnd::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags) {
    int newX = m_ratX;
    int newY = m_ratY;
    switch (nChar) {
        case VK_LEFT: newX--; break;
        case VK_RIGHT: newX++; break;
        case VK_UP: newY--; break;
        case VK_DOWN: newY++; break;
    }
```

***OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)**: Обрабатывает нажатия клавиш. Изменяет координаты крысы в зависимости от нажатой клавиши и проверяет, достигла ли крыса сыра. Если достигла, вызывается метод `RestartGame`:

```
void CRatMazeWnd::RestartGame() {
    m_board.SetupBoard();
    m_ratX = 1;
    m_ratY = 1;
    Invalidate(); // Перерисовка окна
}
```

Также основные команды в Board.cpp:

Конструктор Board(): Инициализирует параметры доски (размеры, цвета) и создает доску.

Деструктор ~Board(): Очищает память, выделенную под доску.

CreateBoard(): Создает новую доску, заполняя ее стенами и генерируя лабиринт. Устанавливает начальную позицию крысы и сыра.

DeleteBoard(): Удаляет текущую доску, освобождая память.

SetupBoard(): Настраивает доску, вызывая метод CreateBoard.

GetColor(int row, int col) const: Возвращает цвет клетки по заданным координатам.

GetCell(int row, int col) const: Возвращает значение клетки по заданным координатам.

SetCell(int row, int col, int value): Устанавливает значение клетки по заданным координатам.

GenerateMaze(int startX, int startY): Генерирует лабиринт с использованием алгоритма поиска в глубину (DFS). Начинается с заданной стартовой позиции.

Основные компоненты:

CRatMazeApp: Главный класс приложения MFC.

CRatMazeWnd: Класс главного окна, обрабатывающий отрисовку и ввод с клавиатуры.

Board: Класс, представляющий игровую доску, включая лабиринт, крысу и сыр.

Также хочется выделить особое внимание генерации лабиринта:

для генерации мы использовали метод DFC (Depth-First Search, DFS).Этот метод включает в себя использование стека и случайного выбора направления для создания лабиринта. Давайте подробно рассмотрим, как он работает:

Инициализация генератора случайных чисел:

```
std::srand(static_cast<unsigned>(std::time(nullptr)));
```

Здесь инициализируется генератор случайных чисел на основе текущего времени, чтобы лабиринт каждый раз был уникальным.

Вектор направлений:

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

Этот вектор хранит направления для перемещения: вправо, влево, вниз и вверх.

Стек для хранения пути:

```
std::stack<std::pair<int, int>> stack;
stack.push({ startX, startY });
m_arrBoard[startY][startX] = 0; // Начальная позиция крысы свободна
```

Стек используется для хранения текущего пути в процессе генерации лабиринта. Начальная позиция добавляется в стек и отмечается как свободная.

Основной цикл:

```
while (!stack.empty()) {
    int x, y;
    std::tie(x, y) = stack.top();
    std::shuffle(directions.begin(), directions.end(), std::mt19937{
std::random_device{}() });

    bool moved = false;
    for (auto [dx, dy] : directions) {
        int nx = x + 2 * dx;
        int ny = y + 2 * dy;
        if (nx > 0 && nx < m_nColumns && ny > 0 && ny < m_nRows &&
m_arrBoard[ny][nx] == 3) {
```

```
            m_arrBoard[ny][nx] = 0;
            m_arrBoard[y + dy][x + dx] = 0;
            stack.push({ nx, ny });
            moved = true;
            break;
        }
    }
    if (!moved) {
        stack.pop();
    }
}
```

Цикл продолжается до тех пор, пока стек не пуст. Внутри цикла текущие координаты извлекаются из вершины стека, и направления перемешиваются для случайного выбора.

Перемешивание направлений:

```
std::shuffle(directions.begin(), directions.end(), std::mt19937{ std::random_device{}() });
```

Направления перемешиваются, чтобы каждый раз алгоритм выбирал случайный путь.

Проход по направлениям:

```
for (auto [dx, dy] : directions) {
    int nx = x + 2 * dx;
    int ny = y + 2 * dy;
    if (nx > 0 && nx < m_nColumns && ny > 0 && ny < m_nRows && m_arrBoard[ny][nx] == 3) {
        m_arrBoard[ny][nx] = 0;
        m_arrBoard[y + dy][x + dx] = 0;
```

```
        stack.push({ nx, ny });
        moved = true;
        break;
    }
}
```

Алгоритм проверяет каждое направление на возможность движения. Новая позиция вычисляется с шагом 2, чтобы пропускать одну клетку, тем самым создавая стены между проходами.

Если новая позиция не выходит за границы и является стеной (не посещена), то:

Убирается стена в новой позиции.

Убирается стена между текущей и новой позицией.

Новая позиция добавляется в стек.

Устанавливается флаг moved, чтобы продолжить генерацию из новой позиции.

Возврат по стеку:

```
if (!moved) {
    stack.pop();
}
```

Если движение невозможно (все направления ведут к уже посещенным или выходящим за границы клеткам), текущая позиция удаляется из стека, возвращаясь к предыдущей позиции.

В итоге этот алгоритм создает случайный лабиринт с единственным проходом, начиная с заданной начальной позиции. Алгоритм гарантирует, что лабиринт будет проходимым, так как каждая новая позиция добавляется в стек и обрабатывается до тех пор, пока возможные ходы не будут исчерпаны.