**АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ**
**ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ**
**МОСКОВСКИЙ МЕЖДУНАРОДНЫЙ КОЛЛЕДЖ ЦИФРОВЫХ ТЕХНОЛОГИЙ**
**«АКАДЕМИЯ ТОП»**

# ИНДИВИДУАЛЬНЫЙ ПРОЕКТ

Уровень профессионального образования:
Среднее профессиональное образование

Программа подготовки специалистов среднего звена
по специальности:
09.02.07 Информационные системы и программирование

Квалификация: Программист

Учебный предмет: Основы алгоритмизации и программирования

Тема: Игра-лабиринт со случайной генерацией лабиринта и его автопрохождением игроком или алгоритмом

Преподаватель:

Рычихин Ярослав _____

Обучающиеся:

Селиванов Матвей_____

**Санкт-Петербург, 2025**

# Содержание

# Введение

Maze Game — это кроссплатформенное игровое приложение, разработанное на языке C# с использованием фреймворка Avalonia UI. Приложение представляет собой интерактивную игру-лабиринт, в которой игрок должен пройти от стартовой позиции до финиша, преодолевая препятствия в виде стен.

Приложение поддерживает генерацию случайных лабиринтов различных размеров, предоставляет визуальные подсказки для навигации, автоматическое решение лабиринта с помощью искусственного интеллекта, а также ведет статистику прохождения и таблицу лидеров.

# Постановка задачи

1. **Цели создания приложения**

1.1. Основными целями разработки приложения Maze Game являются:

1.1.1. Образовательная цель: Демонстрация применения алгоритмов генерации лабиринтов (рекурсивный обход в глубину) и поиска пути (алгоритм следования по стене).

1.1.2. Развлекательная цель: Создание увлекательной игровой механики, которая развивает пространственное мышление и логику игрока.

1.1.3. Техническая цель: Реализация кроссплатформенного приложения с использованием современных технологий .NET и Avalonia UI, демонстрирующее возможности объектно-ориентированного программирования.

1.1.4. Практическая цель: Создание полнофункционального приложения с пользовательским интерфейсом, обработкой ввода, визуализацией данных и системой статистики.

2. **Функции, которые могут быть реализованы с помощью приложения**

Приложение реализует следующие функции:

2.1. Генерация лабиринтов:

2.1.1. Создание случайных лабиринтов заданного размера

2.1.2. Использование seed для воспроизводимости результатов

2.1.3. Гарантированная связность всех клеток лабиринта

2.2. Игровой процесс:

2.2.1. Управление персонажем с помощью клавиатуры (стрелки или WASD)

2.2.2. Управление мышью (перетаскивание для перемещения)

2.2.3. Отображение текущей позиции игрока

2.2.4. Визуальная индикация старта и финиша

2.3.Подсказки и навигация:

2.3.1. Визуализация пути до выбранной точки (при клике мышью)

2.3.2. Автоматическое решение лабиринта с помощью ИИ

2.3.3. Анимированное прохождение лабиринта ИИ

2.4.Статистика и лидерборд:

2.4.1. Подсчет количества шагов

2.4.2. Измерение времени прохождения

2.4.3. Таблица лидеров с сохранением лучших результатов

2.4.4. Сортировка результатов по времени и количеству шагов

2.5.Настройки:

2.5.1. Настройка размеров лабиринта (ширина и высота от 5 до 60 клеток)

2.5.2. Опциональный параметр seed для генерации

2.5.3. Включение/выключение подсказок

2.5.4. Включение/выключение управления мышью

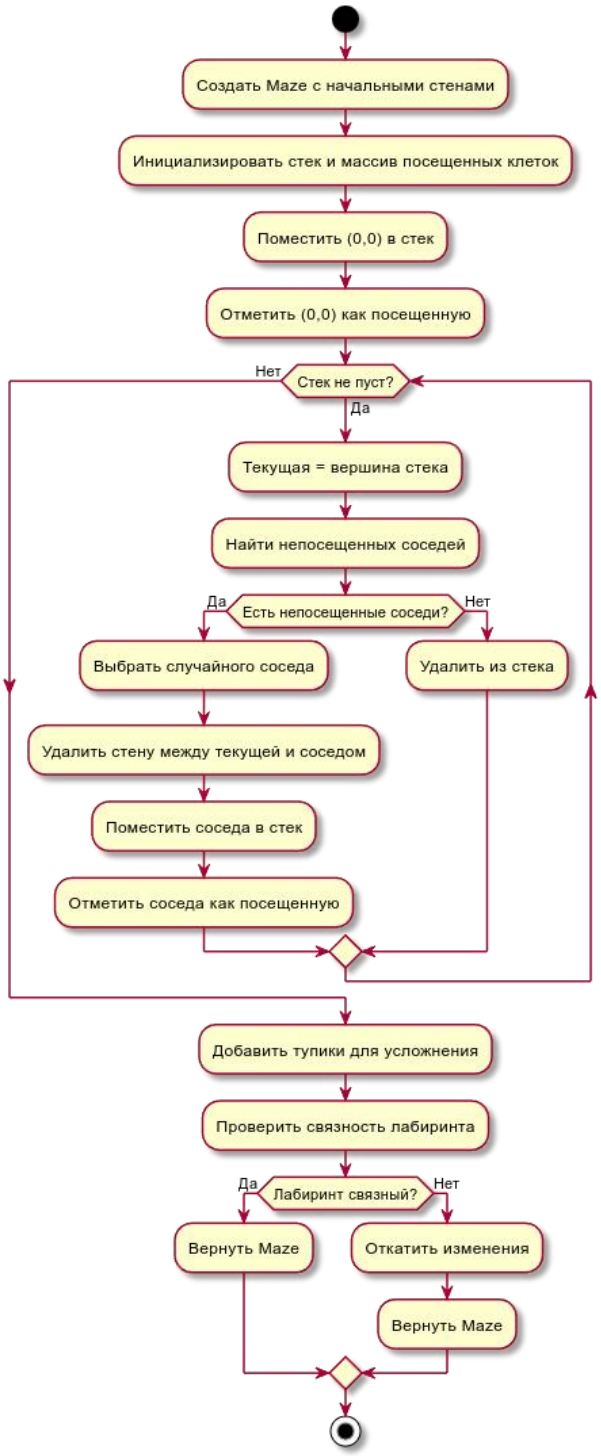
# Проектирование игрового приложения

## Блок схема

Блок-схемы алгоритмов приложения созданы с использованием PlantUML и находятся в директории “./MazeGame/DOC/diagrams”.

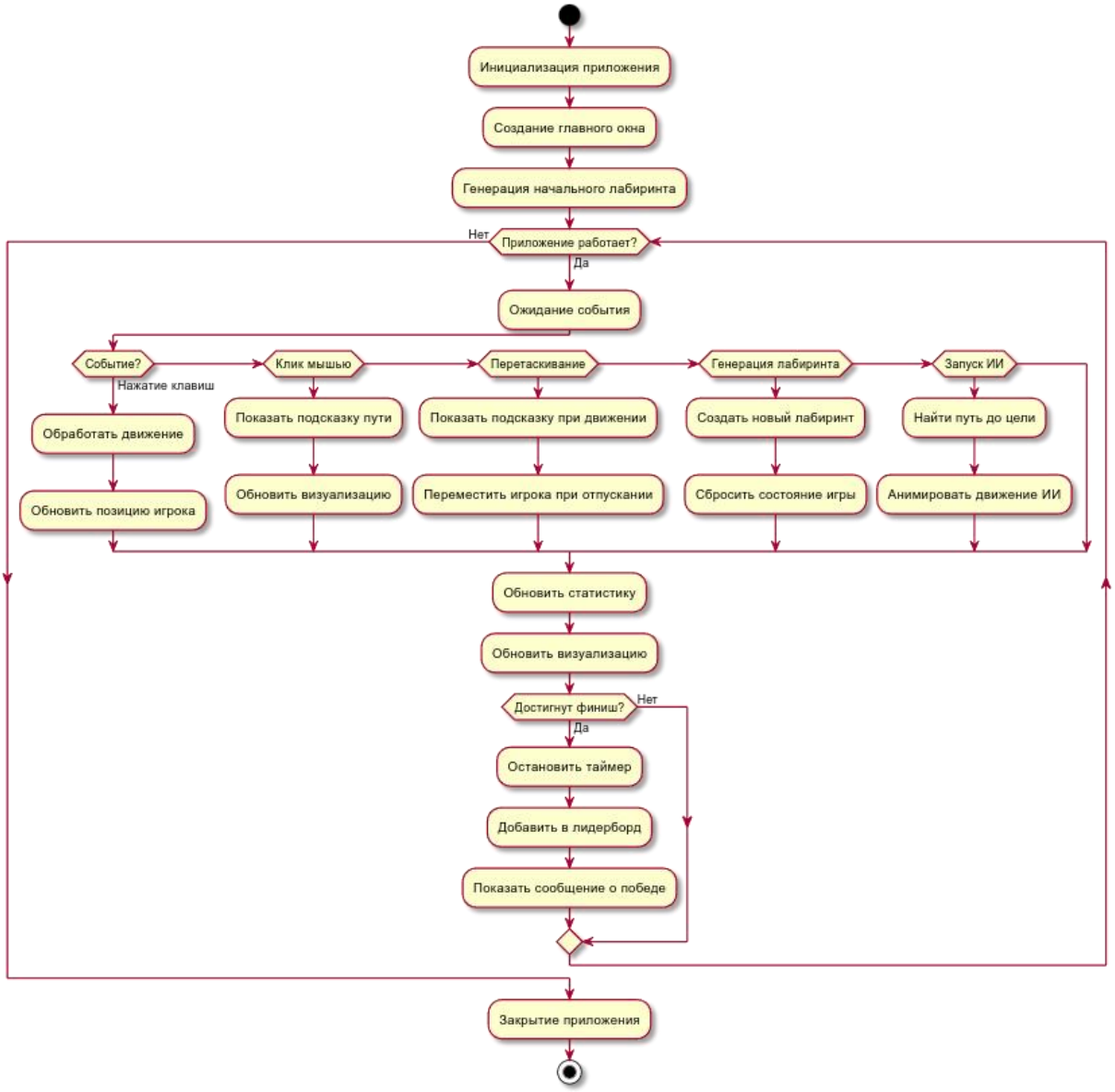
### Блок-схема генерации лабиринта

Блок-схема алгоритма генерации лабиринта с использованием рекурсивного обхода в глубину (DFS)**.**



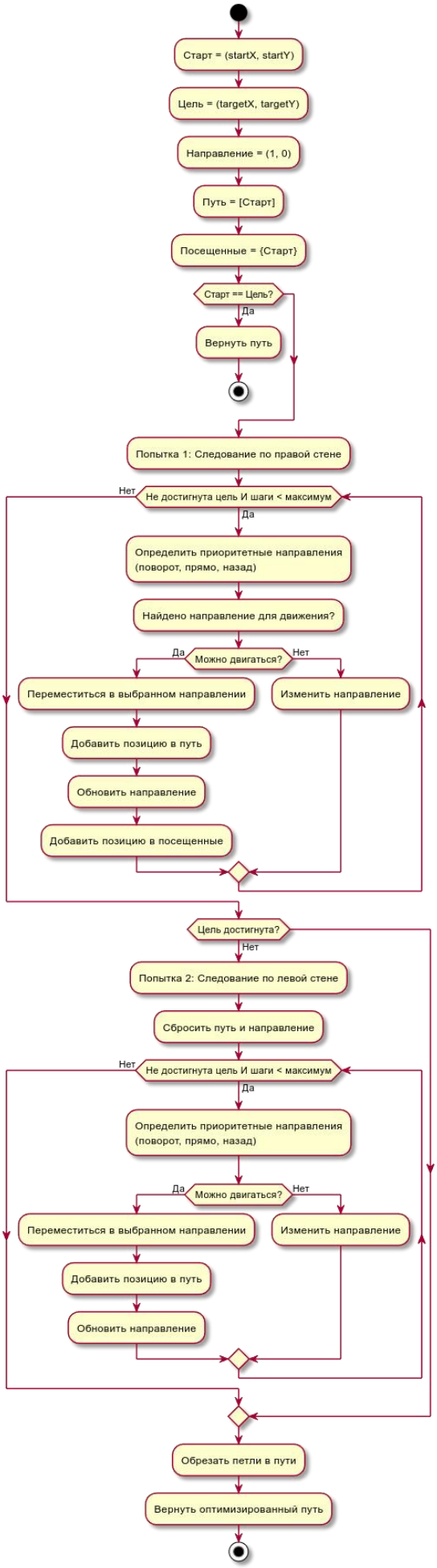

**Блок-схема игрового цикла**

Блок-схема основного игрового цикла приложения, включающая обработку событий и обновление состояния.




**Блок-схема алгоритма поиска пути**

Блок-схема алгоритма поиска пути с использованием метода следования по стене.


Старт = (startX, startY)
Цель = (targetX, targetY)
Направление = (1, 0)
Путь = [Старт]
Посещенные = {Старт}
Старт == Цель?
Да
Вернуть путь
Попытка 1: Следование по правой стене
Нет
Не достигнута цель И шаги < максимум
Да
Определить приоритетные направления (поворот, прямо, назад)
Найдено направление для движения?
Да
Можно двигаться?
Нет
Переместиться в выбранном направлении
Изменить направление
Добавить позицию в путь
Обновить направление
Добавить позицию в посещенные
Цель достигнута?
Нет
Попытка 2: Следование по левой стене
Сбросить путь и направление
Нет
Не достигнута цель И шаги < максимум
Да
Определить приоритетные направления (поворот, прямо, назад)
Да
Можно двигаться?
Нет
Переместиться в выбранном направлении
Изменить направление
Добавить позицию в путь
Обновить направление
Обрезать петли в пути
Вернуть оптимизированный путь

(Примечание: для создания таких диаграмм, я использовал плагин в vscode “PlantUML”, после вставка кода из директории, нужно забилдить проект java’ой, а после уже появится картинка)

## Используемые изображения спрайтов и фонового слоя

Приложение использует программную отрисовку графических элементов без использования внешних изображений. Все визуальные элементы создаются с помощью векторной графики Avalonia:

1. **Стены лабиринта:**
   1.1. Тип: Линии (Pen)
   1.2. Цвет: Белый (#FFFFFF) или настраиваемый через свойство `PathBrush`
   1.3. Толщина: 2 пикселя (настраивается через `WallThickness`)

2. **Стартовая позиция:**
   2.1. Тип: Прямоугольник (Rectangle)
   2.2. Цвет: Зеленый (#4CAF50) - настраивается через `StartBrush`
   2.3. Размер: 70% от размера клетки

3. **Финишная позиция:**
   3.1. Тип: Прямоугольник (Rectangle)
   3.2. Цвет: Красный (#E53935) - настраивается через `FinishBrush`
   3.3. Размер: 70% от размера клетки

4. **Игрок:**
   4.1. Тип: Прямоугольник (Rectangle)
   4.2. Цвет: Голубой (DeepSkyBlue)
   4.3. Размер: 40% от размера клетки
   4.4. Позиция: Центр текущей клетки

5. **Путь (подсказка):**
   5.1. Тип: Линии и точки

5.2. Цвет линии: Золотой (#FFD700) - настраивается через `PathLineBrush`
5.3. Толщина линии: 15% от размера клетки
5.4. Точки: Полупрозрачные квадраты (60% непрозрачности)

6. **Фон:**
6.1. Цвет: Черный (#000000) для области лабиринта
6.2. Цвет панели: Темно-синий (#1A1F2E, #252B3B)
6.3. Общий фон окна: Темно-синий (#0F1419)

# Разработка игрового приложения

## Технологии и инструменты

1. .NET 9.0: Современная платформа разработки от Microsoft
2. C#: Объектно-ориентированный язык программирования
3. Avalonia UI 11.2.7: Кроссплатформенный UI-фреймворк для создания нативных приложений

## Основные библиотеки и пакеты

4. Avalonia: Базовый пакет UI-фреймворка
5. Avalonia.Desktop: Поддержка десктопных приложений
6. Avalonia.Themes.Fluent: Тема оформления Fluent Design
7. Avalonia.Fonts.Inter: Шрифт Inter для интерфейса
8. Avalonia.Diagnostics: Инструменты диагностики (только в Debug)

## Инструменты разработки

1. xUnit: Фреймворк для unit-тестирования
2. Visual Studio / JetBrains Rider: IDE для разработки
3. Git: Система контроля версий

## Алгоритмы

1. Рекурсивный обход в глубину (DFS): Для генерации лабиринта
2. Алгоритм следования по стене (Wall Following): Для поиска пути
3. Обрезка петель: Для оптимизации найденного пути

## Описание экранов

### Главный экран (MainWindow)

Главный экран приложения разделен на две основные области:

1. Левая панель (300px шириной):
    a) Секция настроек лабиринта:
        i. Поле ввода ширины (NumericUpDown, диапазон 5-60, значение по умолчанию: 20)

ii. Поле ввода высоты (NumericUpDown, диапазон 5-60, значение по умолчанию: 20)
iii. Поле ввода seed (TextBox, опционально)
iv. Кнопка "Сгенерировать" для создания нового лабиринта
v. Чекбокс "Показывать подсказки" (включен по умолчанию)
vi. Чекбокс "Управление мышкой (перетаскивание)" (выключен по умолчанию)
vii. Кнопка "Запустить ИИ до цели" (активна только при наличии выбранной цели)

b) Секция статистики:
   i. Текстовое поле статуса игры
   ii. Текстовое поле с количеством шагов и временем
c) Секция лидерборда:
   i. Прокручиваемый список из 10 лучших результатов
   ii. Для каждой записи отображается: имя игрока, время, количество шагов
d) Правая область (остальное пространство):
   i. Заголовок: "Лабиринт"
   ii. Область отрисовки лабиринта: Кастомный контрол MazeView, занимающий всю доступную область

**Взаимодействие с экраном**

Управление клавиатурой:

1. Стрелки вверх/вниз/влево/вправо или WASD для перемещения игрока
2. Движение возможно только если нет стены в выбранном направлении

Управление мышью:

1. Одиночный клик: Показ подсказки (пути до кликнутой точки), если включены подсказки
2. Двойной клик: Автоматическое перемещение на один шаг по направлению к цели (если включено управление мышью)
3. Перетаскивание: Показ подсказки при движении мыши, перемещение при отпускании (если включено управление мышью)

Визуальная обратная связь:

1. Зеленый квадрат: Стартовая позиция (левый верхний угол)
2. Красный квадрат: Финишная позиция (правый нижний угол)
3. Голубой квадрат: Текущая позиция игрока
4. Золотая линия: Путь до выбранной точки (подсказка)

**Отображение хитбоксов**

В приложении Maze Game понятие "хитбоксов" реализовано через проверку коллизий со стенами лабиринта. Каждая клетка лабиринта имеет границы, определяемые стенами:

Структура хитбокса клетки:

1. Верхняя стена: HorizontalWalls[y, x] - блокирует движение вверх (dy = -1)
2. Нижняя стена: HorizontalWalls[y + 1, x] - блокирует движение вниз (dy = 1)
3. Левая стена: VerticalWalls[y, x] - блокирует движение влево (dx = -1)
4. Правая стена: VerticalWalls[y, x + 1] - блокирует движение вправо (dx = 1)
5. Метод проверки коллизий: Maze.CanMove(int x, int y, int dx, int dy)
6. Проверяет границы лабиринта
7. Проверяет наличие стены в направлении движения
8. Возвращает true если движение возможно, false если заблокировано

Визуализация хитбоксов:

Хитбоксы не отображаются визуально, но их можно представить как сетку из клеток размером cellSize × cellSize, где каждая клетка может иметь стены на любой из четырех сторон.

# Описание файлов проекта

## Структура проекта

```
MazeGame/
├── MazeGame/                          # Основной проект приложения
│   ├── Models/                        # Модели данных
│   │   ├── Maze.cs                    # Класс лабиринта
│   │   └── LeaderboardEntry.cs        # Запись лидерборда
│   ├── Services/                      # Бизнес-логика
│   │   ├── MazeGenerator.cs           # Генератор лабиринтов
│   │   └── MazeSolver.cs              # Решатель лабиринтов
│   ├── Controls/                      # Пользовательские контролы
│   │   └── MazeView.cs                # Контрол для отрисовки лабиринта
│   ├── App.axaml                      # Ресурсы приложения
│   ├── App.axaml.cs                   # Класс приложения
│   ├── MainWindow.axaml               # XAML главного окна
│   ├── MainWindow.axaml.cs            # Логика главного окна
│   ├── Program.cs                     # Точка входа
│   ├── MazeGame.csproj                # Файл проекта
│   └── app.manifest                   # Манифест приложения
├── MazeGame.Tests/                    # Тестовый проект
│   └── UnitTest1.cs                   # Unit-тесты
├── MazeGame.sln                       # Решение Visual Studio
└── README.md                          # Описание проекта
```

## Описание основных файлов

1. Program.cs:
    a) Точка входа приложения
    b) Инициализация Avalonia приложения
    c) Настройка платформы и шрифтов
2. App.axaml.cs:
    a) Класс приложения, наследующий Application
    b) Инициализация XAML ресурсов
    c) Создание главного окна при запуске
3. MainWindow.axaml:

    a) XAML разметка главного окна
    b) Определение структуры интерфейса (Grid с двумя колонками)
    c) Привязка элементов управления к свойствам
4. MainWindow.axaml.cs:
    a) Логика главного окна
    b) Обработка событий клавиатуры и мыши
    c) Управление игровым состоянием
    d) Интеграция генератора и решателя лабиринтов
5. Maze.cs (Models):
    a) Модель данных лабиринта
    b) Хранение горизонтальных и вертикальных стен
    c) Методы для работы со стенами и проверки движения
6. LeaderboardEntry.cs (Models):
    a) Модель записи лидерборда
    b) Свойства: имя игрока, время, шаги, размеры лабиринта, дата
    c) Форматирование времени для отображения
7. MazeGenerator.cs (Services):
    a) Генерация лабиринтов с помощью DFS
    b) Добавление тупиков для усложнения
    c) Проверка связности лабиринта
8. MazeSolver.cs (Services):
    a) Поиск пути с помощью алгоритма следования по стене
    b) Обрезка петель для оптимизации пути
9. MazeView.cs (Controls):
    a) Кастомный контрол для отрисовки лабиринта
    b) Отрисовка стен, игрока, старта, финиша, пути
    c) Обработка координат мыши для определения клетки
10. UnitTest1.cs (Tests):
    a) Unit-тесты для проверки логики генерации
    b) Тесты проверки движения и связности лабиринта

## Описание классов, методов игрового приложения

Класс Maze (Models/Maze.cs)

Назначение: Представляет структуру данных лабиринта с его стенами.

1. Свойства:
    a) Width (int): Ширина лабиринта в клетках
    b) Height (int): Высота лабиринта в клетках
    c) HorizontalWalls (bool[,]): Двумерный массив горизонтальных стен
    d) VerticalWalls (bool[,]): Двумерный массив вертикальных стен
2. Методы:
    a) Maze(int width, int height) - Конструктор, создает лабиринт с начальными стенами
    b) RemoveWallBetween(int x1, int y1, int x2, int y2) - Удаляет стену между двумя соседними клетками
    c) CanMove(int x, int y, int dx, int dy) - Проверяет возможность движения из клетки (x,y) в направлении (dx,dy)

Класс LeaderboardEntry (Models/LeaderboardEntry.cs)

Назначение: Представляет запись в таблице лидеров.

1. Свойства:
    a) PlayerName (string): Имя игрока
    b) Time (TimeSpan): Время прохождения
    c) Steps (int): Количество шагов
    d) Width (int): Ширина лабиринта
    e) Height (int): Высота лабиринта
    f) Date (DateTime): Дата прохождения
    g) FormattedTime (string): Отформатированное время (MM:SS.mmm)

Класс MazeGenerator (Services/MazeGenerator.cs)

Назначение: Генерирует случайные связные лабиринты.

1. Методы:
    a) Generate(int width, int height, int? seed = null) - Основной метод генерации лабиринта
        i. Использует DFS алгоритм для создания базового связного лабиринта

ii. Добавляет тупики для усложнения
iii. Возвращает объект Maze

2. Приватные методы:
   a) AddDeadEnds(Maze maze, Random random, int width, int height) - Добавляет тупики в лабиринт
   b) IsConnected(Maze maze, int width, int height) - Проверяет связность лабиринта (BFS)
   c) GetUnvisitedNeighbors(int x, int y, int width, int height, bool[,] visited) - Получает список непосещенных соседей

Класс MazeSolver (Services/MazeSolver.cs)

Назначение: Находит путь от старта до цели в лабиринте.

1. Методы:
   a) FindPath(Maze maze, int startX, int startY, int targetX, int targetY) - Основной метод поиска пути
      i. Пытается найти путь, следуя по правой стене
      ii. Если не находит, пробует следовать по левой стене
      iii. Обрезает петли для получения кратчайшего пути
      iv. Возвращает список координат пути
2. Приватные методы:
   a) FollowWall(Maze maze, int startX, int startY, int targetX, int targetY, bool followRight) - Алгоритм следования по стене
   b) GetWallFollowingDirections((int dx, int dy) currentDir, bool followRight) - Определяет приоритетные направления для следования по стене
   c) RemoveLoops(List<(int X, int Y)> path) - Удаляет петли из пути для оптимизации

Класс MazeView (Controls/MazeView.cs)

Назначение: Кастомный контрол для отрисовки лабиринта и игровых элементов.

1. Свойства (StyledProperty):
    a) Maze (Maze?): Текущий лабиринт для отрисовки
    b) WallThickness (double): Толщина линий стен (по умолчанию: 2)
    c) PathBrush (IBrush?): Кисть для отрисовки стен
    d) StartBrush (IBrush?): Кисть для стартовой позиции
    e) FinishBrush (IBrush?): Кисть для финишной позиции
    f) PlayerPosition ((int X, int Y)): Текущая позиция игрока
    g) Path (List<(int X, int Y)>): Путь для отображения (подсказка)
    h) PathLineBrush (IBrush?): Кисть для отрисовки пути
2. Методы:
    a) GetCellFromPoint(Point point) - Преобразует координаты мыши в координаты клетки лабиринта
    b) Render(DrawingContext context) - Отрисовывает лабиринт и все элементы
    c) Приватные методы отрисовки:
        i. CalculateCellSize(Maze maze) - Вычисляет размер клетки для отрисовки
        ii. DrawHorizontalWalls(...) - Отрисовывает горизонтальные стены
        iii. DrawVerticalWalls(...) - Отрисовывает вертикальные стены
        iv. DrawStartAndFinish(...) - Отрисовывает старт и финиш
        v. DrawPath(...) - Отрисовывает путь (подсказку)
        vi. DrawPlayer(...) - Отрисовывает игрока

Класс MainWindow (MainWindow.axaml.cs)

Назначение: Главное окно приложения, управляет игровым процессом и интерфейсом.

1. Поля:
    a) _generator (MazeGenerator): Генератор лабиринтов
    b) _solver (MazeSolver): Решатель лабиринтов
    c) _leaderboard (List<LeaderboardEntry>): Список записей лидерборда
    d) _playerPosition ((int X, int Y)): Текущая позиция игрока
    e) _isFinished (bool): Флаг завершения игры
    f) _steps (int): Счетчик шагов

g) _gameTimer (Stopwatch): Таймер игры
h) _isAiRunning (bool): Флаг работы ИИ
i) _targetPosition ((int X, int Y)?): Целевая позиция для ИИ
j) _lastClickedCell ((int X, int Y)?): Последняя кликнутая клетка
k) _isDragging (bool): Флаг перетаскивания мыши

2. Методы:
   a) MainWindow() - Конструктор, инициализирует окно и обработчики событий
   b) OnGenerateMaze(object? sender, RoutedEventArgs e) - Обработчик кнопки генерации
   c) GenerateMaze() - Генерирует новый лабиринт и сбрасывает состояние игры
   d) OnWindowKeyDown(object? sender, KeyEventArgs e) - Обработчик нажатий клавиш
   e) TryHandleMovement(Key key) - Обрабатывает движение игрока по клавишам
   f) OnMazeCanvasPointerPressed(...) - Обработчик нажатия мыши на лабиринт
   g) OnMazeCanvasPointerMoved(...) - Обработчик движения мыши
   h) OnMazeCanvasPointerReleased(...) - Обработчик отпускания мыши
   i) ShowHintPath(MazeView view, int targetX, int targetY) - Показывает подсказку пути
   j) RunAiToTarget(MazeView view, int targetX, int targetY) - Запускает ИИ для прохождения пути
   k) OnRunAiToTarget(object? sender, RoutedEventArgs e) - Обработчик кнопки запуска ИИ
   l) UpdateStatusText() - Обновляет текст статуса и статистики
   m) UpdateLeaderboard() - Обновляет список лидерборда

Класс App (App.axaml.cs)

Назначение: Класс приложения Avalonia.

1. Методы:
   a) Initialize() - Инициализирует XAML ресурсы

b) OnFrameworkInitializationCompleted() - Создает главное окно после инициализации

Класс Program (Program.cs)

Назначение: Точка входа приложения.

1. Методы:
   a) Main(string[] args) - Главный метод, запускает приложение
   b) BuildAvaloniaApp() - Настраивает и возвращает конфигурацию Avalonia приложения

# Результаты работы программы

## Функциональность

Приложение успешно реализует все заявленные функции:

1. Генерация лабиринтов:
    a) Лабиринты генерируются корректно с гарантированной связностью
    b) Поддержка настраиваемых размеров (5×5 до 60×60)
    c) Воспроизводимость через seed
2. Игровой процесс:
    a) Плавное управление с клавиатуры и мыши
    b) Корректная проверка коллизий со стенами
    c) Визуальная обратная связь для всех элементов
3. Подсказки и навигация:
    a) Визуализация пути работает корректно
    b) ИИ успешно находит и проходит путь
    c) Анимация движения ИИ плавная
4. Статистика:
    a) Подсчет шагов и времени работает точно
    b) Лидерборд сохраняет и сортирует результаты
    c) Отображение статистики в реальном времени

## Производительность

1. Генерация лабиринта: Мгновенная для размеров до 30×30, менее 1 секунды для 60×60
2. Поиск пути: Мгновенный для лабиринтов до 40×40, менее 100мс для 60×60
3. Отрисовка: Плавная (60 FPS) даже для больших лабиринтов
4. Потребление памяти: Низкое, менее 100 МБ для лабиринта 60×60

## Тестирование

1. Unit-тесты покрывают:
2. Генерацию лабиринта (проверка связности)
3. Логику движения (проверка стен и границ)
4. Все тесты проходят успешно.

**Пользовательский интерфейс**

1. Современный темный дизайн
2. Интуитивно понятное управление
3. Четкая визуальная индикация всех элементов
4. Адаптивный интерфейс, масштабируемый под размер окна

**Кроссплатформенность**

1. Приложение успешно работает на:
2. Windows 10/11
3. Linux (протестировано на Arch Linux)
4. macOS (теоретически поддерживается через Avalonia)

# Заключение

В ходе разработки игрового приложения Maze Game была успешно реализована полнофункциональная игра-лабиринт с использованием современных технологий .NET и Avalonia UI.

**Достигнутые результаты:**

1. Техническая реализация: Приложение демонстрирует применение алгоритмов генерации лабиринтов (DFS) и поиска пути (wall-following), а также правильное использование объектно-ориентированного программирования и паттернов проектирования.
2. Пользовательский опыт: Создан интуитивно понятный интерфейс с поддержкой различных способов управления (клавиатура, мышь), визуальными подсказками и системой статистики.
3. Качество кода: Код структурирован, разделен на логические модули (Models, Services, Controls), покрыт unit-тестами и следует лучшим практикам разработки.
4. Кроссплатформенность: Приложение работает на всех основных операционных системах благодаря использованию Avalonia UI и .NET.

**Возможности для дальнейшего развития:**

1. Добавление различных алгоритмов генерации лабиринтов (Kruskal, Prim, Eller)
2. Реализация алгоритма A* для более оптимального поиска пути
3. Добавление уровней сложности с различными типами препятствий
4. Реализация сохранения прогресса и загрузки сохраненных игр
5. Добавление звуковых эффектов и музыки
6. Реализация многопользовательского режима
7. Добавление редактора лабиринтов для создания пользовательских уровней

Приложение успешно демонстрирует возможности современной разработки на C# и может служить основой для более сложных игровых проектов.

## Список используемых источников

Официальная документация .NET:

Microsoft. (2024). Документация .NET. [Электронный ресурс]. URL: https://learn.microsoft.com/dotnet/

Документация Avalonia UI:

Avalonia UI. (2024). Документация Avalonia. [Электронный ресурс]. URL: https://docs.avaloniaui.net/

Алгоритмы генерации лабиринтов:

Jamis Buck. (2011). "Maze Algorithms". [Электронный ресурс]. URL: https://www.jamisbuck.org/mazes/

Алгоритмы поиска пути:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). "Introduction to Algorithms" (3rd ed.). MIT Press.

Рекурсивный обход в глубину (DFS):

Sedgewick, R., & Wayne, K. (2011). "Algorithms" (4th ed.). Addison-Wesley Professional.

xUnit Framework:

xUnit.net. (2024). Документация xUnit. [Электронный ресурс]. URL: https://xunit.net/

C# Programming Language:

Microsoft. (2024). C# Guide. [Электронный ресурс]. URL: https://learn.microsoft.com/dotnet/csharp/

Объектно-ориентированное программирование:

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley.

# Приложение А листинг кода

**MainWindow.axaml**

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:controls="clr-namespace:MazeGame.Controls"
        xmlns:models="clr-namespace:MazeGame.Models"
        mc:Ignorable="d"
        d:DesignWidth="1200"
        d:DesignHeight="800"
        x:Class="MazeGame.MainWindow"
        Title="Maze Game"
        Background="#0F1419"
        Padding="0">
    <Grid ColumnDefinitions="300,*" RowDefinitions="Auto,*">
        <!-- Левая панель с настройками и лидербордом -->
        <Border Grid.Column="0"
                Background="#1A1F2E"
                BorderBrush="#2A3441"
                BorderThickness="0,0,1,0">
            <Grid RowDefinitions="Auto,Auto,*">
                <!-- Настройки генерации -->
                <Border Grid.Row="0"
                        Background="#252B3B"
                        Padding="16"
                        Margin="12,12,12,8">
                    <StackPanel Spacing="12">
                        <TextBlock Text="Настройки лабиринта"
                                   Foreground="#E0E6ED"
                                   FontSize="16"
                                   FontWeight="Bold"/>

                        <StackPanel Spacing="8">
                            <TextBlock Text="Ширина" Foreground="#B8C5D1" FontSize="12"/>
                            <NumericUpDown Name="WidthInput"
                                           Minimum="5"
```

```
                Maximum="60"
                Value="20"
                Background="#1A1F2E"
                Foreground="#E0E6ED"
                BorderBrush="#3A4551"/>
    </StackPanel>

    <StackPanel Spacing="8">
      <TextBlock Text="Высота" Foreground="#B8C5D1" FontSize="12"/>
      <NumericUpDown x:Name="HeightInput"
                Minimum="5"
                Maximum="60"
                Value="20"
                Background="#1A1F2E"
                Foreground="#E0E6ED"
                BorderBrush="#3A4551"/>
    </StackPanel>

    <StackPanel Spacing="8">
      <TextBlock Text="Seed (опционально)" Foreground="#B8C5D1" FontSize="12"/>
      <TextBox x:Name="SeedInput"
            Watermark="Случайно"
            Background="#1A1F2E"
            Foreground="#E0E6ED"
            BorderBrush="#3A4551"/>
    </StackPanel>

    <Button Content="ã Сгенерировать"
          Click="OnGenerateMaze"
          Padding="12,8"
          Background="#4A90E2"
          Foreground="White"
          FontWeight="Bold"
          Cursor="Hand"/>

    <Separator Background="#3A4551" Margin="0,8"/>

    <StackPanel Spacing="8">
```

```
        <CheckBox Name="ShowHintsCheckBox"
                  Content="` Показывать подсказки"
                  Foreground="#B8C5D1"
                  IsChecked="True"/>
        <CheckBox Name="MouseControlCheckBox"
                  Content="χ Управление мышкой (перетаскивание)"
                  Foreground="#B8C5D1"
                  IsChecked="False"/>
        <Button Name="RunAiButton"
                Content="γ Запустить ИИ до цели"
                Click="OnRunAiToTarget"
                Padding="12,8"
                Background="#6A4C93"
                Foreground="White"
                FontWeight="Bold"
                Cursor="Hand"
                IsEnabled="False"/>
      </StackPanel>
    </StackPanel>
  </Border>

  <!-- Статистика -->
  <Border Grid.Row="1"
      Background="#252B3B"
      Padding="16"
      Margin="12,0,12,8">
    <StackPanel Spacing="8">
      <TextBlock Text="Статистика"
                 Foreground="#E0E6ED"
                 FontSize="16"
                 FontWeight="Bold"/>
      <TextBlock Name="StatusText"
                 Text="Используйте стрелки или WASD"
                 Foreground="#B8C5D1"
                 TextWrapping="Wrap"/>
      <TextBlock Name="StatsText"
                 Text="Шагов: 0"
                 Foreground="#7A8FA3"
```

```
            FontSize="12"/>
    </StackPanel>
</Border>

<!-- Лидерборд -->
<Border Grid.Row="2"
      Background="#252B3B"
      Padding="16"
      Margin="12,0,12,12">
    <StackPanel Spacing="8">
      <TextBlock Text="dz Лидерборд"
            Foreground="#E0E6ED"
            FontSize="16"
            FontWeight="Bold"/>
      <ScrollViewer MaxHeight="300">
        <ItemsControl Name="LeaderboardList">
          <ItemsControl.ItemTemplate>
            <DataTemplate x:DataType="models:LeaderboardEntry">
              <Border Background="#1A1F2E"
                  Padding="8"
                  Margin="0,0,0,4"
                  CornerRadius="4">
                <StackPanel>
                  <TextBlock Text="{Binding PlayerName}"
                        Foreground="#E0E6ED"
                        FontWeight="SemiBold"/>
                  <TextBlock Text="{Binding FormattedTime}"
                        Foreground="#4A90E2"
                        FontSize="14"/>
                  <TextBlock Text="{Binding Steps, StringFormat='{}Шагов: {0}'}"
                        Foreground="#7A8FA3"
                        FontSize="11"/>
                </StackPanel>
              </Border>
            </DataTemplate>
          </ItemsControl.ItemTemplate>
        </ItemsControl>
      </ScrollViewer>
```

```
            </StackPanel>
          </Border>
        </Grid>
      </Border>

      <!-- Основная область с лабиринтом -->
      <Grid Grid.Column="1" RowDefinitions="Auto,*">
        <Border Grid.Row="0"
                Background="#1A1F2E"
                Padding="16,12"
                BorderBrush="#2A3441"
                BorderThickness="0,0,0,1">
          <TextBlock Text="Ù Лабиринт"
                     Foreground="#E0E6ED"
                     FontSize="18"
                     FontWeight="Bold"
                     HorizontalAlignment="Center"/>
        </Border>

        <Border Grid.Row="1"
                Background="#0F1419"
                Padding="20">
          <controls:MazeView Name="MazeCanvas"
                             PathBrush="#2A3441"
                             StartBrush="#4CAF50"
                             FinishBrush="#E53935"
                             PathLineBrush="#FFD700"
                             Cursor="Hand"/>
        </Border>
      </Grid>
    </Grid>
</Window>
```

**MainWindow.axaml.cs**

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
```

```
using System.Threading.Tasks;
using Avalonia;
using Avalonia.Controls;
using Avalonia.Input;
using Avalonia.Interactivity;
using Avalonia.Media;
using MazeGame.Controls;
using MazeGame.Models;
using MazeGame.Services;

namespace MazeGame;

public partial class MainWindow : Window
{
    private readonly MazeGenerator _generator = new();
    private readonly MazeSolver _solver = new();
    private readonly List<LeaderboardEntry> _leaderboard = new();

    private (int X, int Y) _playerPosition;
    private bool _isFinished;
    private int _steps;
    private Stopwatch _gameTimer = new();
    private bool _isAiRunning;
    private (int X, int Y)? _targetPosition;
    private readonly List<(int X, int Y)> _playerPath = new(); // История пути игрока

    public MainWindow()
    {
        InitializeComponent();
        AddHandler(KeyDownEvent, OnWindowKeyDown, RoutingStrategies.Tunnel | RoutingStrategies.Bubble);

        if (MazeCanvas is not null)
        {
            MazeCanvas.PointerPressed += OnMazeCanvasPointerPressed;
            MazeCanvas.PointerMoved += OnMazeCanvasPointerMoved;
            MazeCanvas.PointerReleased += OnMazeCanvasPointerReleased;
        }
```

```
    // Обработчик изменения состояния чекбокса "Показывать подсказки"
    if (ShowHintsCheckBox is not null)
    {
        ShowHintsCheckBox.IsCheckedChanged += OnShowHintsChanged;
    }

    GenerateMaze();
}

private void OnGenerateMaze(object? sender, RoutedEventArgs e)
{
    GenerateMaze();
}

private void GenerateMaze()
{
    _isFinished = false;
    _steps = 0;
    _gameTimer.Reset();
    _gameTimer.Start();
    _isAiRunning = false;
    _targetPosition = null;
    _lastDragTarget = null;
    _playerPath.Clear(); // Очищаем путь игрока

    if (MazeCanvas is not MazeView view || WidthInput is null || HeightInput is null)
        return;

    var width = (int)Math.Clamp(WidthInput.Value ?? 10, WidthInput.Minimum, WidthInput.Maximum);
    var height = (int)Math.Clamp(HeightInput.Value ?? 10, HeightInput.Minimum, HeightInput.Maximum);

    int? seed = null;
    if (!string.IsNullOrWhiteSpace(SeedInput?.Text) && int.TryParse(SeedInput.Text, out var parsedSeed))
        seed = parsedSeed;

    view.Maze = _generator.Generate(width, height, seed);
```

```
    view.Path = new List<(int X, int Y)>();
    _playerPosition = (0, 0);
    _playerPath.Add(_playerPosition); // Добавляем стартовую позицию в путь
    view.PlayerPosition = _playerPosition;
    _lastClickedCell = null;
    _targetPosition = null;

    // Деактивируем кнопку ИИ
    if (RunAiButton is not null)
    {
        RunAiButton.IsEnabled = false;
    }

    UpdateStatusText();
}

private void OnShowHintsChanged(object? sender, RoutedEventArgs e)
{
    if (MazeCanvas is null)
        return;

    // Если подсказки отключены, очищаем текущий путь подсказок
    if (ShowHintsCheckBox?.IsChecked == false)
    {
        // Очищаем только подсказки, но не путь игрока (если игра завершена)
        if (!_isFinished)
        {
            MazeCanvas.Path = new List<(int X, int Y)>();
        }
        // Если игра завершена, путь игрока остается видимым
    }
    // Если подсказки включены, но игра завершена, показываем путь игрока
    else if (_isFinished && _playerPath.Count > 0)
    {
        MazeCanvas.Path = new List<(int X, int Y)>(_playerPath);
    }
}
```

```
private (int X, int Y)? _lastClickedCell;
private bool _isDragging;
private (int X, int Y)? _lastDragTarget;

private void OnMazeCanvasPointerPressed(object? sender, PointerPressedEventArgs e)
{
    if (MazeCanvas is not MazeView view || view.Maze is null || _isFinished || _isAiRunning)
        return;

    var point = e.GetPosition(MazeCanvas);
    var cell = view.GetCellFromPoint(point);

    if (cell is null)
        return;

    var (targetX, targetY) = cell.Value;

    // Двойной клик для перемещения (если включено управление мышкой)
    if (MouseControlCheckBox?.IsChecked == true && e.ClickCount == 2)
    {
        // Проверяем, что целевая клетка отличается от текущей позиции
        if (targetX == _playerPosition.X && targetY == _playerPosition.Y)
            return;

        // Пытаемся найти путь и переместиться по нему
        var path = _solver.FindPath(view.Maze, _playerPosition.X, _playerPosition.Y, targetX, targetY);

        // Проверяем, что путь найден и валиден
        if (path.Count < 2 || path[0] != (_playerPosition.X, _playerPosition.Y) || path.Last() != (targetX, targetY))
            return;

        // Перемещаемся по первому шагу пути
        var nextStep = path[1];

        // Проверяем, что следующий шаг отличается от текущей позиции
        if (nextStep.X == _playerPosition.X && nextStep.Y == _playerPosition.Y)
```

```
            return;

        // Вычисляем направление движения
        var dx = nextStep.X - _playerPosition.X;
        var dy = nextStep.Y - _playerPosition.Y;

        // Проверяем, что движение возможно
        if (view.Maze.CanMove(_playerPosition.X, _playerPosition.Y, dx, dy))
        {
            _playerPosition = nextStep;
            _playerPath.Add(_playerPosition); // Добавляем новую позицию в путь
            view.PlayerPosition = _playerPosition;
            _steps++;

            // Очищаем подсказку при движении
            if (ShowHintsCheckBox?.IsChecked == true)
            {
                view.Path = new List<(int X, int Y)>();
            }
            UpdateStatusText();
        }
        return;
    }

    // Обычный клик: показываем подсказку (путь до кликнутой точки)
    if (ShowHintsCheckBox?.IsChecked == true)
    {
        ShowHintPath(view, targetX, targetY);
        _lastClickedCell = (targetX, targetY);
    }

    // Начинаем перетаскивание для управления мышкой
    if (MouseControlCheckBox?.IsChecked == true)
    {
        _isDragging = true;
        _lastDragTarget = null; // Сбрасываем последнюю цель при начале перетаскивания
    }
}
```

```
private void OnMazeCanvasPointerMoved(object? sender, PointerEventArgs e)
{
    if (!_isDragging || MazeCanvas is not MazeView view || view.Maze is null || _isFinished || _isAiRunning)
        return;

    var point = e.GetPosition(MazeCanvas);
    var cell = view.GetCellFromPoint(point);

    if (cell is null)
        return;

    var (targetX, targetY) = cell.Value;

    // Обновляем подсказку только если целевая клетка изменилась
    if (!_lastDragTarget.HasValue || _lastDragTarget.Value != (targetX, targetY))
    {
        _lastDragTarget = (targetX, targetY);

        // Показываем подсказку при перетаскивании
        if (ShowHintsCheckBox?.IsChecked == true)
        {
            ShowHintPath(view, targetX, targetY);
        }
    }
}

private void OnMazeCanvasPointerReleased(object? sender, PointerReleasedEventArgs e)
{
    if (!_isDragging || MazeCanvas is not MazeView view || view.Maze is null || _isFinished || _isAiRunning)
    {
        _isDragging = false;
        _lastDragTarget = null;
        return;
    }

    var point = e.GetPosition(MazeCanvas);
```

```
var cell = view.GetCellFromPoint(point);

if (cell is null)
{
    _isDragging = false;
    _lastDragTarget = null;
    return;
}

var (targetX, targetY) = cell.Value;

// Проверяем, что целевая клетка отличается от текущей позиции
if (targetX == _playerPosition.X && targetY == _playerPosition.Y)
{
    _isDragging = false;
    _lastDragTarget = null;
    return;
}

// Перемещаемся по первому шагу пути
var path = _solver.FindPath(view.Maze, _playerPosition.X, _playerPosition.Y, targetX, targetY);

// Проверяем, что путь найден и валиден
if (path.Count < 2 || path[0] != (_playerPosition.X, _playerPosition.Y) || path.Last() != (targetX, targetY))
{
    _isDragging = false;
    _lastDragTarget = null;
    return;
}

// Берем следующий шаг из пути
var nextStep = path[1];

// Проверяем, что следующий шаг отличается от текущей позиции
if (nextStep.X == _playerPosition.X && nextStep.Y == _playerPosition.Y)
{
    _isDragging = false;
```

```
            _lastDragTarget = null;
            return;
        }

        // Вычисляем направление движения
        var dx = nextStep.X - _playerPosition.X;
        var dy = nextStep.Y - _playerPosition.Y;

        // Проверяем, что движение возможно
        if (view.Maze.CanMove(_playerPosition.X, _playerPosition.Y, dx, dy))
        {
            _playerPosition = nextStep;
            _playerPath.Add(_playerPosition); // Добавляем новую позицию в путь
            view.PlayerPosition = _playerPosition;
            _steps++;

            // Очищаем подсказку при движении
            if (ShowHintsCheckBox?.IsChecked == true)
            {
                view.Path = new List<(int X, int Y)>();
            }
            UpdateStatusText();
        }

        _isDragging = false;
        _lastDragTarget = null;
    }

    private async Task RunAiToTarget(MazeView view, int targetX, int targetY)
    {
        if (view.Maze is null || _isAiRunning)
            return;

        _isAiRunning = true;
        _targetPosition = (targetX, targetY);

        var path = _solver.FindPath(view.Maze, _playerPosition.X, _playerPosition.Y, targetX, targetY);
```

```
    if (path.Count > 0 && path.Last() == (targetX, targetY))
    {
        view.Path = path;

        // Анимируем движение ИИ
        foreach (var (x, y) in path.Skip(1))
        {
            _playerPosition = (x, y);
            _playerPath.Add(_playerPosition); // Добавляем новую позицию в путь
            view.PlayerPosition = _playerPosition;
            _steps++;
            await Task.Delay(50); // Задержка для анимации
        }

        UpdateStatusText();
    }

    _isAiRunning = false;
}

private void ShowHintPath(MazeView view, int targetX, int targetY)
{
    if (view.Maze is null)
        return;

    var path = _solver.FindPath(view.Maze, _playerPosition.X, _playerPosition.Y, targetX, targetY);

    if (path.Count > 0 && path.Last() == (targetX, targetY))
    {
        view.Path = path;
        _targetPosition = (targetX, targetY);
        _lastClickedCell = (targetX, targetY);

        // Активируем кнопку ИИ
        if (RunAiButton is not null)
        {
```

```
            RunAiButton.IsEnabled = true;
        }
    }
}

private async void OnRunAiToTarget(object? sender, RoutedEventArgs e)
{
    if (MazeCanvas is not MazeView view || view.Maze is null || _isFinished || _isAiRunning)
        return;

    // Используем последнюю кликнутую точку или финиш
    var (targetX, targetY) = _lastClickedCell ?? (view.Maze.Width - 1, view.Maze.Height - 1);

    await RunAiToTarget(view, targetX, targetY);
}

private void OnWindowKeyDown(object? sender, KeyEventArgs e)
{
    if (TryHandleMovement(e.Key))
        e.Handled = true;
}

private bool TryHandleMovement(Key key)
{
    if (_isFinished || _isAiRunning)
        return false;

    (int dx, int dy)? direction = key switch
    {
        Key.Left or Key.A => (-1, 0),
        Key.Right or Key.D => (1, 0),
        Key.Up or Key.W => (0, -1),
        Key.Down or Key.S => (0, 1),
        _ => null
    };

    if (direction is null || MazeCanvas?.Maze is not { } maze)
        return false;
```

```
    var (dx, dy) = direction.Value;
    if (!maze.CanMove(_playerPosition.X, _playerPosition.Y, dx, dy))
        return false;

    _playerPosition = (_playerPosition.X + dx, _playerPosition.Y + dy);
    _playerPath.Add(_playerPosition); // Добавляем новую позицию в путь
    MazeCanvas.PlayerPosition = _playerPosition;
    _steps++;

    // Очищаем подсказку при движении
    if (ShowHintsCheckBox?.IsChecked == true && MazeCanvas.Path.Count > 0)
    {
        MazeCanvas.Path = new List<(int X, int Y)>();
        // Деактивируем кнопку ИИ при движении
        if (RunAiButton is not null)
        {
            RunAiButton.IsEnabled = false;
        }
    }

    UpdateStatusText();
    return true;
}

private void UpdateStatusText()
{
    if (StatusText is null || StatsText is null)
        return;

    if (MazeCanvas?.Maze is not { } maze)
    {
        StatusText.Text = "Сначала сгенерируйте лабиринт";
        StatsText.Text = "";
        return;
    }

    var elapsed = _gameTimer.Elapsed;
```

```
StatsText.Text = $"Шагов: {_steps} | Время: {elapsed.Minutes:D2}:{elapsed.Seconds:D2}";

if (_playerPosition.X == maze.Width - 1 && _playerPosition.Y == maze.Height - 1)
{
    if (!_isFinished)
    {
        _isFinished = true;
        _gameTimer.Stop();

        StatusText.Text = "2 Финиш! Поздравляем!";

        // Показываем путь, который прошел игрок
        if (MazeCanvas is not null && _playerPath.Count > 0)
        {
            MazeCanvas.Path = new List<(int X, int Y)>(_playerPath);
        }

        // Добавляем в лидерборд
        var entry = new LeaderboardEntry
        {
            PlayerName = $"Игрок {_leaderboard.Count + 1}",
            Time = _gameTimer.Elapsed,
            Steps = _steps,
            Width = maze.Width,
            Height = maze.Height
        };

        _leaderboard.Add(entry);
        _leaderboard.Sort((a, b) =>
        {
            var timeCompare = a.Time.CompareTo(b.Time);
            return timeCompare != 0 ? timeCompare : a.Steps.CompareTo(b.Steps);
        });

        UpdateLeaderboard();
    }
}
```

```
        else
        {
            if (_isAiRunning)
                StatusText.Text = "γ ИИ проходит лабиринт...";
            else if (_lastClickedCell.HasValue)
                StatusText.Text = $"Кликните на место в лабиринте, чтобы увидеть путь. Нажмите кнопку ИИ для автоматического прохождения.";
            else
                StatusText.Text = "Управление: стрелки/WASD. Кликните на место в лабиринте, чтобы увидеть путь до него.";
        }
    }

    private void UpdateLeaderboard()
    {
        if (LeaderboardList is null)
            return;

        var topEntries = _leaderboard.Take(10).ToList();
        LeaderboardList.ItemsSource = topEntries;
    }
}
```

**Controls/MazeView.cs**

```
using System;
using System.Collections.Generic;
using Avalonia;
using Avalonia.Controls;
using Avalonia.Media;
using MazeGame.Models;

namespace MazeGame.Controls;

public class MazeView : Control
{
    public static readonly StyledProperty<Maze?> MazeProperty =
        AvaloniaProperty.Register<MazeView, Maze?>(nameof(Maze));

    public static readonly StyledProperty<double> WallThicknessProperty =
        AvaloniaProperty.Register<MazeView, double>(nameof(WallThickness), 2);
```

```
public static readonly StyledProperty<IBrush?> PathBrushProperty =
    AvaloniaProperty.Register<MazeView, IBrush?>(nameof(PathBrush), Brushes.White);

public static readonly StyledProperty<IBrush?> StartBrushProperty =
    AvaloniaProperty.Register<MazeView, IBrush?>(nameof(StartBrush), Brushes.LightGreen);

public static readonly StyledProperty<IBrush?> FinishBrushProperty =
    AvaloniaProperty.Register<MazeView, IBrush?>(nameof(FinishBrush), Brushes.IndianRed);

public static readonly StyledProperty<(int X, int Y)> PlayerPositionProperty =
    AvaloniaProperty.Register<MazeView, (int X, int Y)>(nameof(PlayerPosition), (0, 0));

public static readonly StyledProperty<List<(int X, int Y)>> PathProperty =
    AvaloniaProperty.Register<MazeView, List<(int X, int Y)>>(nameof(Path), new List<(int X, int Y)>());

public static readonly StyledProperty<IBrush?> PathLineBrushProperty =
    AvaloniaProperty.Register<MazeView, IBrush?>(nameof(PathLineBrush), Brushes.Yellow);

static MazeView()
{
    AffectsRender<MazeView>(
        MazeProperty,
        WallThicknessProperty,
        PathBrushProperty,
        StartBrushProperty,
        FinishBrushProperty,
        PlayerPositionProperty,
        PathProperty,
        PathLineBrushProperty);
}

public Maze? Maze
{
    get => GetValue(MazeProperty);
    set => SetValue(MazeProperty, value);
}
```

```
public double WallThickness
{
    get => GetValue(WallThicknessProperty);
    set => SetValue(WallThicknessProperty, value);
}

public IBrush? PathBrush
{
    get => GetValue(PathBrushProperty);
    set => SetValue(PathBrushProperty, value);
}

public IBrush? StartBrush
{
    get => GetValue(StartBrushProperty);
    set => SetValue(StartBrushProperty, value);
}

public IBrush? FinishBrush
{
    get => GetValue(FinishBrushProperty);
    set => SetValue(FinishBrushProperty, value);
}

public (int X, int Y) PlayerPosition
{
    get => GetValue(PlayerPositionProperty);
    set => SetValue(PlayerPositionProperty, value);
}

public List<(int X, int Y)> Path
{
    get => GetValue(PathProperty);
    set => SetValue(PathProperty, value);
}

public IBrush? PathLineBrush
{
```

```
    get => GetValue(PathLineBrushProperty);
    set => SetValue(PathLineBrushProperty, value);
}

public (int X, int Y)? GetCellFromPoint(Point point)
{
    if (Maze is null || Bounds.Width <= 0 || Bounds.Height <= 0)
        return null;

    var cellSize = CalculateCellSize(Maze);
    var offsetX = (Bounds.Width - Maze.Width * cellSize) / 2;
    var offsetY = (Bounds.Height - Maze.Height * cellSize) / 2;

    var relativeX = point.X - offsetX;
    var relativeY = point.Y - offsetY;

    if (relativeX < 0 || relativeY < 0)
        return null;

    var cellX = (int)(relativeX / cellSize);
    var cellY = (int)(relativeY / cellSize);

    if (cellX >= 0 && cellX < Maze.Width && cellY >= 0 && cellY < Maze.Height)
        return (cellX, cellY);

    return null;
}

public override void Render(DrawingContext context)
{
    base.Render(context);

    if (Maze is null || Bounds.Width <= 0 || Bounds.Height <= 0)
        return;

    var maze = Maze;
    var pen = new Pen(PathBrush ?? Brushes.White, WallThickness, lineCap: PenLineCap.Round);
```

```
        var cellSize = CalculateCellSize(maze);
        var offsetX = (Bounds.Width - maze.Width * cellSize) / 2;
        var offsetY = (Bounds.Height - maze.Height * cellSize) / 2;
        var drawingRect = new Rect(offsetX, offsetY, maze.Width * cellSize, maze.Height * cellSize);

        context.FillRectangle(Brushes.Black, drawingRect);

        DrawStartAndFinish(context, cellSize, offsetX, offsetY, maze);
        DrawPath(context, cellSize, offsetX, offsetY, maze);
        DrawPlayer(context, cellSize, offsetX, offsetY, maze);
        DrawHorizontalWalls(context, cellSize, offsetX, offsetY, maze, pen);
        DrawVerticalWalls(context, cellSize, offsetX, offsetY, maze, pen);
    }

    private double CalculateCellSize(Maze maze)
        => Math.Min(Bounds.Width / maze.Width, Bounds.Height / maze.Height);

    private void DrawHorizontalWalls(DrawingContext context, double cell, double offsetX, double offsetY, Maze
maze, Pen pen)
    {
        for (var i = 0; i <= maze.Height; i++)
            for (var j = 0; j < maze.Width; j++)
            {
                if (!maze.HorizontalWalls[i, j])
                    continue;

                var start = new Point(offsetX + j * cell, offsetY + i * cell);
                var end = new Point(start.X + cell, start.Y);
                context.DrawLine(pen, start, end);
            }
    }

    private void DrawVerticalWalls(DrawingContext context, double cell, double offsetX, double offsetY, Maze maze,
Pen pen)
    {
        for (var i = 0; i < maze.Height; i++)
            for (var j = 0; j <= maze.Width; j++)
            {
                if (!maze.VerticalWalls[i, j])
```

```
                continue;

            var start = new Point(offsetX + j * cell, offsetY + i * cell);
            var end = new Point(start.X, start.Y + cell);
            context.DrawLine(pen, start, end);
        }
    }

    private void DrawStartAndFinish(DrawingContext context, double cell, double offsetX, double offsetY, Maze maze)
    {
        var startRect = new Rect(offsetX + 0.15 * cell, offsetY + 0.15 * cell, cell * 0.7, cell * 0.7);
        var finishRect = new Rect(offsetX + (maze.Width - 1 + 0.15) * cell, offsetY + (maze.Height - 1 + 0.15) * cell,
cell * 0.7, cell * 0.7);

        if (StartBrush is not null)
            context.FillRectangle(StartBrush, startRect);

        if (FinishBrush is not null)
            context.FillRectangle(FinishBrush, finishRect);
    }

    private void DrawPath(DrawingContext context, double cell, double offsetX, double offsetY, Maze maze)
    {
        if (Path is null || Path.Count < 2 || PathLineBrush is null)
            return;

        var pen = new Pen(PathLineBrush, cell * 0.15, lineCap: PenLineCap.Round);
        var points = new List<Point>();

        foreach (var (x, y) in Path)
        {
            if (x < 0 || y < 0 || x >= maze.Width || y >= maze.Height)
                continue;

            var center = new Point(
                offsetX + (x + 0.5) * cell,
                offsetY + (y + 0.5) * cell);
            points.Add(center);
```

```
    }

    if (points.Count < 2)
        return;

    // Рисуем линии между точками пути
    for (var i = 0; i < points.Count - 1; i++)
    {
        context.DrawLine(pen, points[i], points[i + 1]);
    }

    // Рисуем точки на пути (полупрозрачные квадраты)
    IBrush? pointBrush = null;
    if (PathLineBrush is ISolidColorBrush solidBrush)
    {
        var color = solidBrush.Color;
        var semiTransparentColor = new Color(153, color.R, color.G, color.B); // 60% opacity
        pointBrush = new SolidColorBrush(semiTransparentColor);
    }
    else
    {
        pointBrush = PathLineBrush;
    }

    var pointSize = cell * 0.15;
    foreach (var point in points)
    {
        if (pointBrush is not null)
        {
            var rect = new Rect(point.X - pointSize / 2, point.Y - pointSize / 2, pointSize, pointSize);
            context.FillRectangle(pointBrush, rect);
        }
    }
}

private void DrawPlayer(DrawingContext context, double cell, double offsetX, double offsetY, Maze maze)
{
    var (px, py) = PlayerPosition;
```

```
        if (px < 0 || py < 0 || px >= maze.Width || py >= maze.Height)
            return;

        var center = new Point(
            offsetX + (px + 0.5) * cell,
            offsetY + (py + 0.5) * cell);

        var size = cell * 0.4;
        var rect = new Rect(center.X - size / 2, center.Y - size / 2, size, size);
        context.FillRectangle(Brushes.DeepSkyBlue, rect);
    }
}
```

**Models/LeaderboardEntry.cs**

```
using System;

namespace MazeGame.Models;

public class LeaderboardEntry
{
    public string PlayerName { get; set; } = string.Empty;
    public TimeSpan Time { get; set; }
    public int Steps { get; set; }
    public int Width { get; set; }
    public int Height { get; set; }
    public DateTime Date { get; set; } = DateTime.Now;

    public string FormattedTime => $"{Time.Minutes:D2}:{Time.Seconds:D2}.{Time.Milliseconds:D3}";
}
```

Models/Maze.cs

```
using System;

namespace MazeGame.Models;

public class Maze
{
    public int Width { get; }
    public int Height { get; }
```

```
public bool[,] HorizontalWalls { get; }
public bool[,] VerticalWalls { get; }

public Maze(int width, int height)
{
    if (width <= 0) throw new ArgumentOutOfRangeException(nameof(width));
    if (height <= 0) throw new ArgumentOutOfRangeException(nameof(height));

    Width = width;
    Height = height;
    HorizontalWalls = new bool[height + 1, width];
    VerticalWalls = new bool[height, width + 1];

    for (var i = 0; i <= height; i++)
        for (var j = 0; j < width; j++)
            HorizontalWalls[i, j] = true;

    for (var i = 0; i < height; i++)
        for (var j = 0; j <= width; j++)
            VerticalWalls[i, j] = true;
}


public void RemoveWallBetween(int x1, int y1, int x2, int y2)
{
    if (x1 == x2 && y1 == y2)
        return;

    var distance = Math.Abs(x1 - x2) + Math.Abs(y1 - y2);
    if (distance != 1)
        throw new ArgumentException("Cells must be adjacent to remove a wall.");

    if (x1 == x2)
    {
        var row = Math.Max(y1, y2);
        HorizontalWalls[row, x1] = false;
    }
    else
```

```
        {
            var column = Math.Max(x1, x2);
            VerticalWalls[y1, column] = false;
        }
    }

    public bool CanMove(int x, int y, int dx, int dy)
    {
        if (dx == 0 && dy == 0)
            return false;

        var targetX = x + dx;
        var targetY = y + dy;

        if (targetX < 0 || targetY < 0 || targetX >= Width || targetY >= Height)
            return false;

        if (dx == -1 && VerticalWalls[y, x])
            return false;

        if (dx == 1 && VerticalWalls[y, x + 1])
            return false;

        if (dy == -1 && HorizontalWalls[y, x])
            return false;

        if (dy == 1 && HorizontalWalls[y + 1, x])
            return false;

        return true;
    }
}
```

**Services/MazeGenerator.cs**

```
using System;
using System.Collections.Generic;
using MazeGame.Models;
```

```
namespace MazeGame.Services;

public class MazeGenerator
{
    public Maze Generate(int width, int height, int? seed = null)
    {
        var random = seed.HasValue ? new Random(seed.Value) : Random.Shared;
        var maze = new Maze(width, height);
        var visited = new bool[height, width];
        var stack = new Stack<(int X, int Y)>();

        // Базовый DFS для создания связного лабиринта
        stack.Push((0, 0));
        visited[0, 0] = true;

        while (stack.Count > 0)
        {
            var current = stack.Peek();
            var neighbors = GetUnvisitedNeighbors(current.X, current.Y, width, height, visited);

            if (neighbors.Count == 0)
            {
                stack.Pop();
                continue;
            }

            var next = neighbors[random.Next(neighbors.Count)];
            maze.RemoveWallBetween(current.X, current.Y, next.X, next.Y);
            visited[next.Y, next.X] = true;
            stack.Push(next);
        }

        // Добавляем дополнительные стены для создания тупиков и усложнения
        AddDeadEnds(maze, random, width, height);

        return maze;
    }
```

```
private static void AddDeadEnds(Maze maze, Random random, int width, int height)
{
    // Подсчитываем количество открытых проходов для каждой клетки
    var openPassages = new int[height, width];

    for (var y = 0; y < height; y++)
    for (var x = 0; x < width; x++)
    {
        if (maze.CanMove(x, y, -1, 0)) openPassages[y, x]++;
        if (maze.CanMove(x, y, 1, 0)) openPassages[y, x]++;
        if (maze.CanMove(x, y, 0, -1)) openPassages[y, x]++;
        if (maze.CanMove(x, y, 0, 1)) openPassages[y, x]++;
    }

    // Добавляем стены в клетки с 3+ открытыми проходами (создаем тупики)
    var cellsToModify = new List<(int X, int Y)>();
    for (var y = 0; y < height; y++)
    for (var x = 0; x < width; x++)
    {
        // Пропускаем старт и финиш
        if ((x == 0 && y == 0) || (x == width - 1 && y == height - 1))
            continue;

        if (openPassages[y, x] >= 3)
            cellsToModify.Add((x, y));
    }

    // Закрываем случайные проходы (20-30% от доступных клеток, более консервативно)
    var toClose = (int)(cellsToModify.Count * (0.2 + random.NextDouble() * 0.1));
    toClose = Math.Min(toClose, cellsToModify.Count);

    var attempts = 0;
    var maxAttempts = toClose * 3; // Ограничиваем попытки

    for (var i = 0; i < toClose && attempts < maxAttempts; attempts++)
    {
        if (cellsToModify.Count == 0)
            break;
```

```
var idx = random.Next(cellsToModify.Count);
var (x, y) = cellsToModify[idx];

// Выбираем случайный проход для закрытия
var directions = new List<(int dx, int dy)>();
if (maze.CanMove(x, y, -1, 0)) directions.Add((-1, 0));
if (maze.CanMove(x, y, 1, 0)) directions.Add((1, 0));
if (maze.CanMove(x, y, 0, -1)) directions.Add((0, -1));
if (maze.CanMove(x, y, 0, 1)) directions.Add((0, 1));

if (directions.Count <= 1)
{
    cellsToModify.RemoveAt(idx);
    continue; // Нельзя закрывать, оставляем как есть
}

var dir = directions[random.Next(directions.Count)];
var (dx, dy) = dir;
var nx = x + dx;
var ny = y + dy;

// Сохраняем состояние стены перед изменением
bool wallWasOpen;
if (dx == -1)
{
    wallWasOpen = !maze.VerticalWalls[y, x];
    maze.VerticalWalls[y, x] = true;
}
else if (dx == 1)
{
    wallWasOpen = !maze.VerticalWalls[y, x + 1];
    maze.VerticalWalls[y, x + 1] = true;
}
else if (dy == -1)
{
    wallWasOpen = !maze.HorizontalWalls[y, x];
    maze.HorizontalWalls[y, x] = true;
```

```
        }
        else // dy == 1
        {
            wallWasOpen = !maze.HorizontalWalls[y + 1, x];
            maze.HorizontalWalls[y + 1, x] = true;
        }

        // Проверяем связность - если нарушена, откатываем изменение
        if (!IsConnected(maze, width, height))
        {
            // Откатываем изменение
            if (dx == -1) maze.VerticalWalls[y, x] = !wallWasOpen;
            else if (dx == 1) maze.VerticalWalls[y, x + 1] = !wallWasOpen;
            else if (dy == -1) maze.HorizontalWalls[y, x] = !wallWasOpen;
            else maze.HorizontalWalls[y + 1, x] = !wallWasOpen;

            cellsToModify.RemoveAt(idx);
            continue;
        }

        // Успешно закрыли проход
        cellsToModify.RemoveAt(idx);
        i++;
    }
}

private static bool IsConnected(Maze maze, int width, int height)
{
    var visited = new bool[height, width];
    var queue = new Queue<(int X, int Y)>();
    queue.Enqueue((0, 0));
    visited[0, 0] = true;
    var visitedCount = 1;

    while (queue.Count > 0)
    {
        var (x, y) = queue.Dequeue();
```

```
            var directions = new[] { (-1, 0), (1, 0), (0, -1), (0, 1) };
            foreach (var (dx, dy) in directions)
            {
                if (maze.CanMove(x, y, dx, dy))
                {
                    var nx = x + dx;
                    var ny = y + dy;
                    if (!visited[ny, nx])
                    {
                        visited[ny, nx] = true;
                        visitedCount++;
                        queue.Enqueue((nx, ny));
                    }
                }
            }
        }

        return visitedCount == width * height;
    }

    private static List<(int X, int Y)> GetUnvisitedNeighbors(int x, int y, int width, int height, bool[,] visited)
    {
        var neighbors = new List<(int, int)>(4);

        void TryAdd(int nx, int ny)
        {
            if (nx >= 0 && ny >= 0 && nx < width && ny < height && !visited[ny, nx])
                neighbors.Add((nx, ny));
        }

        TryAdd(x - 1, y);
        TryAdd(x + 1, y);
        TryAdd(x, y - 1);
        TryAdd(x, y + 1);

        return neighbors;
    }
}
```

**Services/MazeSolver.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using MazeGame.Models;

namespace MazeGame.Services;

public class MazeSolver
{
    /// <summary>
    /// Находит кратчайший путь от старта до цели, используя алгоритм поиска в ширину (BFS)
    /// </summary>
    public List<(int X, int Y)> FindPath(Maze maze, int startX, int startY, int targetX, int targetY)
    {
        if (startX == targetX && startY == targetY)
            return new List<(int X, int Y)> { (startX, startY) };

        // Используем BFS для поиска кратчайшего пути
        return FindPathBFS(maze, startX, startY, targetX, targetY);
    }

    /// <summary>
    /// Поиск кратчайшего пути с помощью алгоритма поиска в ширину (BFS)
    /// </summary>
    private List<(int X, int Y)> FindPathBFS(Maze maze, int startX, int startY, int targetX, int targetY)
    {
        var queue = new Queue<(int X, int Y)>();
        var visited = new HashSet<(int X, int Y)>();
        var parent = new Dictionary<(int X, int Y), (int X, int Y)?>();

        queue.Enqueue((startX, startY));
        visited.Add((startX, startY));
        parent[(startX, startY)] = null;

        var directions = new[] { (-1, 0), (1, 0), (0, -1), (0, 1) };
```

```
    while (queue.Count > 0)
    {
        var current = queue.Dequeue();
        var (x, y) = current;

        // Если достигли цели, восстанавливаем путь
        if (x == targetX && y == targetY)
        {
            return ReconstructPath(parent, (startX, startY), (targetX, targetY));
        }

        // Проверяем всех соседей
        foreach (var (dx, dy) in directions)
        {
            if (maze.CanMove(x, y, dx, dy))
            {
                var nextX = x + dx;
                var nextY = y + dy;
                var next = (nextX, nextY);

                if (!visited.Contains(next))
                {
                    visited.Add(next);
                    parent[next] = current;
                    queue.Enqueue(next);
                }
            }
        }
    }

    // Путь не найден - возвращаем путь только до старта
    return new List<(int X, int Y)> { (startX, startY) };
}

/// <summary>
/// Восстанавливает путь от старта до цели по словарю parent
/// </summary>
private List<(int X, int Y)> ReconstructPath(Dictionary<(int X, int Y), (int X, int Y)?> parent,
```

```
        (int X, int Y) start, (int X, int Y) target)
    {
        var path = new List<(int X, int Y)>();
        var current = target;

        // Восстанавливаем путь от цели к старту
        while (current != start)
        {
            path.Add(current);
            if (parent.TryGetValue(current, out var prev) && prev.HasValue)
            {
                current = prev.Value;
            }
            else
            {
                // Если не можем восстановить путь, возвращаем только старт
                return new List<(int X, int Y)> { start };
            }
        }

        path.Add(start);
        path.Reverse(); // Разворачиваем, чтобы получить путь от старта к цели
        return path;
    }
}
```