Лабораторная работа №6: Генерация лабиринта

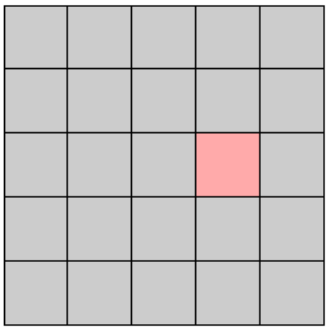
Цель работы: познакомиться с основами автоматизированной генерации уровней на примере алгоритма рекурсивного поиска с возвратом (recursive backtracker).

Подробное описание алгоритма (на английском языке) и интерактивную демонстрацию работы можно найти на сайте: https://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking
Для создания лабиринта использовался бесплатный набор ассетов доступный по ссылке: https://assetstore.unity.com/packages/3d/environments/stylized-hand-painted-dungeon-free-173934

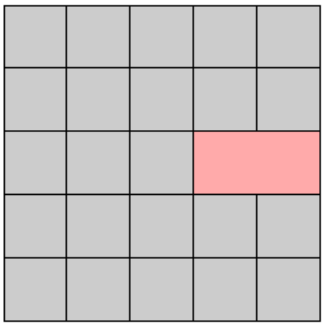
Алгоритм

Последовательность действий при генерации лабиринтов методом recursive backtracker выглядит следующим образом:

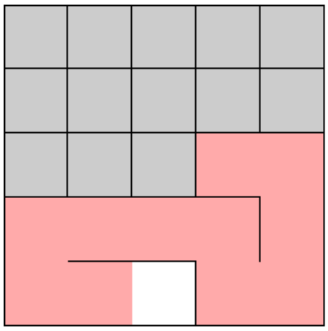
Шаг1: создаётся массив ячеек, каждая из которых имеет 4 стены, выбирается стартовая ячейка (отмечена цветом)



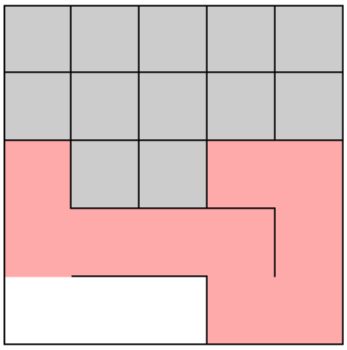
Шаг 2: на каждом шаге прямого хода, составляется список ещё не посещённых смежных ячеек, затем, случайным образом выбирается одна из них и стены между текущей и выбранной разрушаются (в данном случае, была выбрана ячейка правее стартовой)



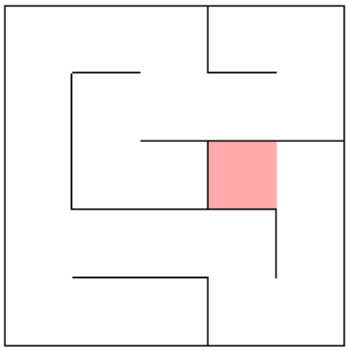
Процесс повторяется до тех пор, пока не будет достигнута ячейка, все соседи которой уже были посещены (отмечена белым цветом)



Шаг 3: происходит последовательный возврат к пройденным ячейкам до тех пор, пока не будет найдена возможность пройти к не посещённой части сетки (в данном случае, произошёл возврат на 3 ячейки назад)

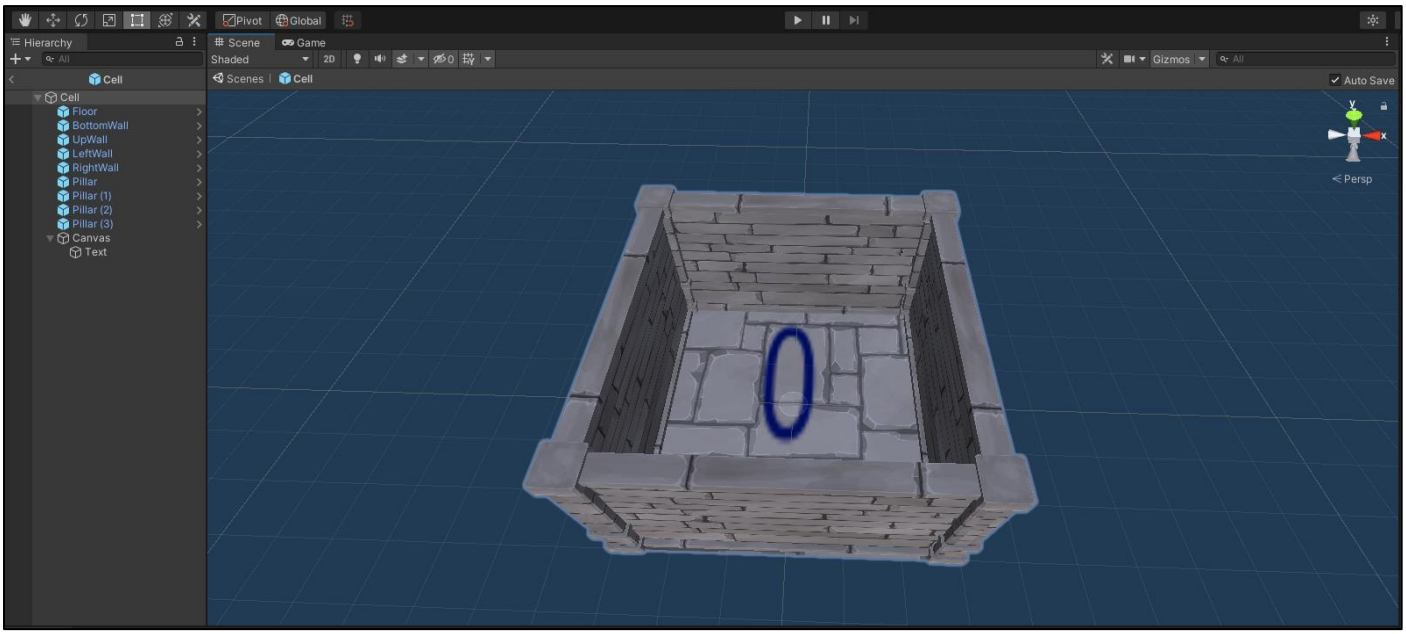


Шаг 4: повторение шагов 2 и 3 до тех пор, пока не произойдёт возврат к стартовой ячейке



Создание ячейки

Поскольку лабиринт, по сути, является набором ячеек, имеет смысл начать с создания шаблона для одной из них. Создайте пустой объект и добавьте в него 5 объектов – “пол” и 4 стены:

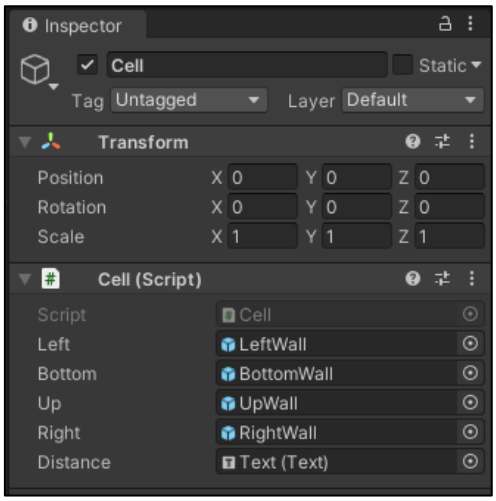


В примере на изображении, помимо пола и стен, присутствуют так же 4 колонны и холст, содержащий текст. Колонны носят чисто декоративный характер, а компонент текст может быть использован для вывода отладочной информации (вроде номера ячейки) и для выполнения задания номер 2.

Для отображения лабиринта при помощи созданного шаблона ячейки, понадобится создать скрипт, позволяющий получить доступ к её составляющим:

```
using UnityEngine;
using UnityEngine.UI;

Ссылок: 3
public class Cell: MonoBehaviour //класс, описывающий ячейку лабиринта
{
    public GameObject Left;     //наличие левой стены
    public GameObject Bottom;   //наличие левой стены
    public GameObject Up;       //наличие левой стены
    public GameObject Right;    //наличие левой стены

    public Text distance;       //ссылка на текст, находящийся в ячейке
}
```

А затем, передать в него ссылки на соответствующие объекты:



Примечание: убедитесь, что положение и название стен в шаблоне соответствует таковым в сцене.

Создание генератора лабиринта

Помимо визуального отображения ячейки, необходимо, так же, описать её логическую структуру:

```
public class MazeCell   //логическое описание одной ячейки
{
    public int X;   //координаты ячейки
    public int Y;

    public bool Left = true;    //наличие стен ячейки, по умолчанию все 4 стены есть
    public bool Bottom = true;
    public bool Up = true;
    public bool Right = true;

    public bool Visited = false;    //посещена ли ячейка в ходе работы алгоритма
}
```

Помимо указанного в примере, ячейка может содержать и прочую информацию. Например, дистанцию до “стартовой” ячейки.

Сам лабиринт может быть описан как массив ячеек:

```
public class Maze   //логическое описание лабиринта
{
    public MazeCell[,] cells;       //массив ячеек
}
```

Класс генератор, отвечающий за создание логической модели лабиринта, может быть описан следующим образом:

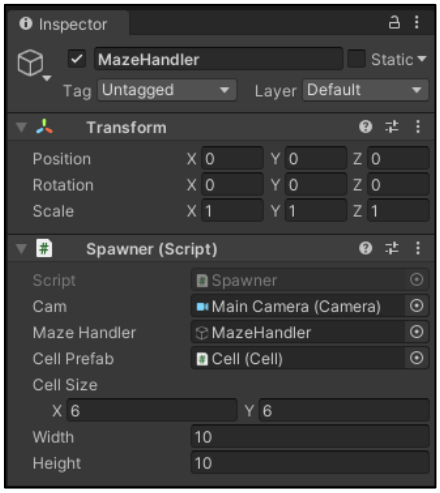
```
public class Generator //класс генератор лабиринта
{
    int Width = 10;      //размеры лабиринта
    int Height = 10;

    ссылка: 1
    public Maze GenerateMaze(int Width, int Height) //метод генерации
    {
        this.Width = Width;
        this.Height = Height;

        MazeCell[,] cells = new MazeCell[Width, Height];    //создание массива ячеек

        for (int x = 0; x < cells.GetLength(0); x++)
        {
            for (int y = 0; y < cells.GetLength(1); y++)
            {
                cells[x, y] = new MazeCell { X = x, Y = y };    //инициализация ячеек лабиринта
            }
        }

        Maze maze = new Maze(); //создание лабиринта

        maze.cells = cells;

        return maze;
    }
}
```

При этом класс, отвечающий за создание визуального отображения лабиринта может выглядеть так:

```
public class Spawner : MonoBehaviour
{
    public Camera cam;                  //ссылка на камеру
    public GameObject mazeHandler;      //ссылка на объект хранения модели лабиринта

    public Cell CellPrefab;     //шаблон ячейки
    public Vector2 CellSize = new Vector2(1, 1);    //размер ячейки

    public int Width = 10;  //размеры лабиринта
    public int Height = 10;

    Ссылок: 0
    public void GenerateMaze()  //вызов метода генерации лабиринта
    {
        foreach(Transform child in mazeHandler.transform) //очистка объекта хранения лабиринта
            GameObject.Destroy(child.gameObject);

        Generator generator = new Generator();
        Maze maze = generator.GenerateMaze(Width, Height);   //получение логической модели лабиринта

        for (int x = 0; x < maze.cells.GetLength(0); x++)
        {
            for (int z = 0; z < maze.cells.GetLength(1); z++)
            {
                //создание и размещение визуального представления ячеек лабиринта
                Cell c = Instantiate(CellPrefab, new Vector3(x * CellSize.x, 0, z * CellSize.y), Quaternion.identity);

                //удаление стен ячеек в соответствии с логической моделью
                if (maze.cells[x, z].Left == false)
                    Destroy(c.Left);
                if (maze.cells[x, z].Right == false)
                    Destroy(c.Right);
                if (maze.cells[x, z].Up == false)
                    Destroy(c.Up);
                if (maze.cells[x, z].Bottom == false)
                    Destroy(c.Bottom);

                c.transform.parent = mazeHandler.transform; //добавление ячейки в объект хранения лабиринта
            }
        }

        //установка камеры над лабиринтом
        cam.transform.position = new Vector3((Width*CellSize.x)/2, Mathf.Max(Width, Height) * 8, (Height * CellSize.y) / 2);
    }
}
```

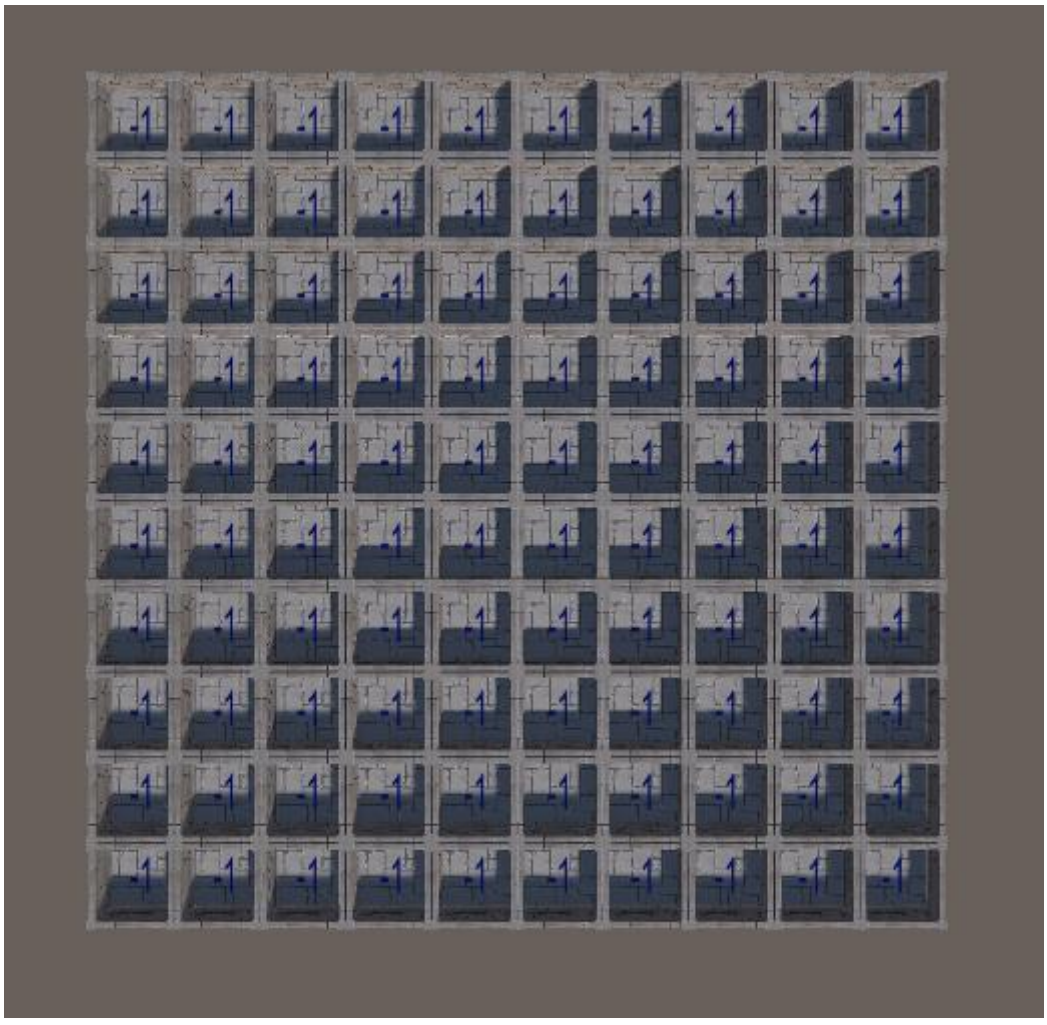
Указанные классы отвечают за создание логической модели массива ячеек и её визуальное отображение. На данном этапе, все ячейки имеют по 4 стены.

Для того, чтобы создать лабиринт, используя описанные выше классы, необходимо создать пустой объект в сцене и добавить ему скрипт создания визуального отображения лабиринта:



Вызов метода генерации можно назначить в качестве обработчика события нажатия кнопки:

Если всё было сделано правильно, то после нажатия на кнопку вызова генерации лабиринта, должно появиться изображение следующего вида:



Чтобы получить лабиринт, достаточно в класс Generator добавить метод удаления стен, реализующий алгоритм recursive backtracker:

```
private void removeWalls(MazeCell[,] maze)  //удаление стен
{
    MazeCell current = maze[0, 0];          //стартовая ячейка
    current.Visited = true;

    Stack<MazeCell> stack = new Stack<MazeCell>();  //очередь посещённых ячеек
    do
    {
        List<MazeCell> unvisitedNeighbours = new List<MazeCell>();  //список не посещённых соседей

        int x = current.X;
        int y = current.Y;

        if (x > 0 && !maze[x - 1, y].Visited) unvisitedNeighbours.Add(maze[x - 1, y]);              //добавление непосещённых соседей в список
        if (y > 0 && !maze[x, y - 1].Visited) unvisitedNeighbours.Add(maze[x, y - 1]);
        if (x < Width - 1 && !maze[x + 1, y].Visited) unvisitedNeighbours.Add(maze[x + 1, y]);
        if (y < Height - 1 && !maze[x, y + 1].Visited) unvisitedNeighbours.Add(maze[x, y + 1]);

        if (unvisitedNeighbours.Count > 0)  //если есть не посещённые соседи
        {
            MazeCell chosen = unvisitedNeighbours[UnityEngine.Random.Range(0, unvisitedNeighbours.Count)];  //выбор случайного соседа
            RemoveWall(current, chosen);         //удаление стен между текущей и выбранной ячейками

            chosen.Visited = true;  //отметка о посещении
            stack.Push(chosen);     //добавление выбранной ячейки в список посещённых

            current = chosen;   //переход к выбранной ячейке
        }
        else
        {
            current = stack.Pop();  //возврат по очереди посещений если нет не посещённых соседей
        }
    } while (stack.Count > 0);  //до тех пор пока не произошёл возврат к стартовой ячейке
}
```

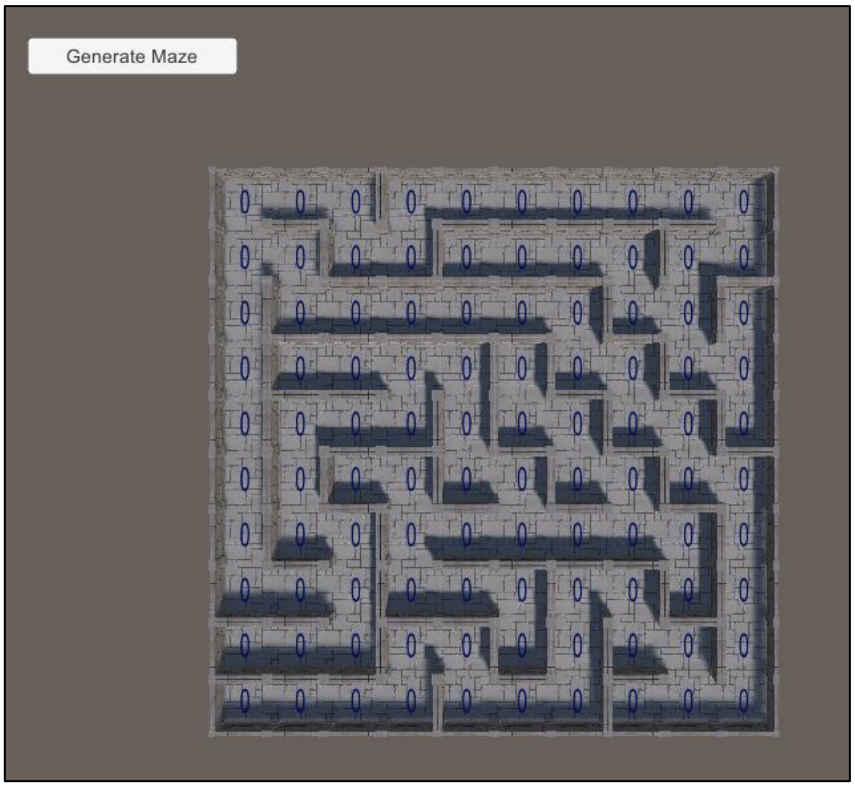
При этом метод удаления стен между смежными ячейками будет выглядеть следующим образом:

```
private void RemoveWall(MazeCell a, MazeCell b) //удаление стен
{
    if (a.X == b.X)
    {
        if (a.Y > b.Y)          //если происходит удаление стен с ячейкой выше
        {
            a.Bottom = false;
            b.Up = false;
        }
        else                    //если происходит удаление стен с ячейкой ниже
        {
            b.Bottom = false;
            a.Up = false;
        }
    }
    else
    {
        if (a.X > b.X)          //если происходит удаление стен с ячейкой левее
        {
            a.Left = false;
            b.Right = false;
        }
        else                    //если происходит удаление стен с ячейкой правее
        {
            b.Left = false;
            a.Right = false;
        }
    }
}
```

В примере представлена не рекурсивная версия алгоритма, реализующая возврат к уже пройденным ячейкам через структуру типа stack. Метод Push добавляет в структуру элемент, метод Pop получает последний добавленный элемент из структуры. На каждом шаге алгоритма, в stack добавляется текущая обрабатываемая ячейка, и в случае, если все смежные ячейки уже посещены, происходит возврат к предыдущей добавленной в структуру ячейке. Когда происходит возврат к начальной ячейке, работа алгоритма завершается.

Вызываться метод removeWalls будет после инициализации ячеек в методе GenerateMaze.

Пример работы:



Задание:

Реализовать приложение, позволяющее сгенерировать лабиринт заданного размера, отвечающий следующим условиям:

1. Лабиринт должен содержать цикличные пути. Цикличный путь образуется вокруг последовательности стен, не связанной с остальной частью лабиринта.
2. В лабиринте должна случайным образом выбираться клетка “Старт”, после чего, должно рассчитываться кратчайшее расстояние между клеткой “Старт” и остальными клетками.

Реализовать второй метод построения лабиринта, использующий алгоритм Олдоса-Бродера, либо алгоритм Уилсона.

Онлайн демонстрация: https://h-anim.github.io/AmazingMaze/

Подробное описание алгоритмов Олдоса-Бродера и Уилсона можно найти по ссылке: https://habr.com/ru/post/321210/