**Maze Commented Document**

**// Floor generator class**

**// Responsible for creating the floor, holding the cells and rooms**

**public class FloorGenerator**

**{**

**// Floor details: Width, height, total size, minimum and maximum room size**

**// Its Index in the game logic**

**int floor\_width;**

**int floor\_height;**

**int size;**

**public int min\_room;**

**public int max\_room;**

**int index;**

**// Cells array, holding all cells within the floor**

**public Tiles[] cells;**

**// Cell size and scale within the size**

**public float tile\_size;**

**float tile\_scale;**

**// Floor and wall object which gets pushed to the cells**

**GameObject floor;**

**GameObject wall;**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//**

**// Constructor which generates the floor object**

**public FloorGenerator(float \_tile\_size, int \_grid\_width, int \_grid\_height, float \_cell\_scale, int \_index,GameObject \_floor,GameObject \_wall)**

**{**

**this.index = \_index;**

**this.floor\_height = \_grid\_height;**

**this.floor\_width = \_grid\_width;**

**this.tile\_size = \_tile\_size;**

**this.tile\_scale = \_cell\_scale;**

**this.size = \_grid\_height \* \_grid\_width;**

**this.floor = \_floor;**

**this.wall = \_wall;**

**this.min\_room = (int)(this.size \* 0.05);**

**this.max\_room = (int)(this.size \* 0.8);**

**generateGrid();**

**}**

**// Clears and destroys the whole floor**

**public void clearGrid()**

**{**

**for (int i = 0; i < cells.Length; i++)**

**{**

**cells[i].clear();**

**}**

**cells = new Tiles[0];**

**}**

**// Grid generation that fills the cells array**

**public void generateGrid()**

**{**

**if (cells != null)**

**{**

**clearGrid();**

**}**

**cells = new Tiles[size];**

**int cell\_index = 0;**

**if ((tile\_scale \* 10) / tile\_size > 1)**

**{**

**tile\_scale = tile\_size / 10;**

**}**

**for (int i = 0; i < floor\_height; i++)**

**{**

**for (int j = 0; j < floor\_width; j++)**

**{**

**Vector3 cell\_position = new Vector3(j \* tile\_size, 0, i \* tile\_size);**

**cells[cell\_index] = new Tiles(cell\_position, floor, cell\_index, tile\_scale);**

**int[] sides = new int[4];**

**System.Random rnd = new System.Random();**

**for (int k = 0; k < 4; k++)**

**{**

**sides[k]= (int)(rnd.Next(0, 2));**

**}**

**cell\_index++;**

**}**

**}**

**Debug.Log(cells.Length);**

**}**

**public void generateRoom()**

**{**

**}**

**}**

**// Tile object**

**// Stores position, floor and wall objects and materials**

**public class Tiles**

**{**

**// Gameobject array=[floor + 4 walls]**

**public GameObject[] tile;**

**// Position**

**float x;**

**float z;**

**// Wall size in scale**

**float wall\_size = 0.6f;**

**// Wall array, 0 = no wall, 1 = wall**

**public int[] side;**

**// Offset from x,z position**

**Vector3 left\_right = new Vector3(0, 0, 90);**

**Vector3 front\_back = new Vector3(90, 0, 0);**

**// The tiles index in the floors array**

**public int index;**

**// Bool to check if the tile has already been visited by the maze algorithm or part of a room**

**bool visited = false;**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//**

**// Constructor which calls the tiles generation**

**public Tiles(Vector3 position,GameObject \_floor,int \_index,float \_cell\_scale)**

**{**

**this.x = position.x;**

**this.z = position.z;**

**this.index = \_index;**

**generateCell(\_floor,\_cell\_scale);**

**}**

**// Generating and rotating the floor**

**void generateCell(GameObject \_floor,float \_cell\_scale)**

**{**

**tile = new GameObject[5];**

**tile[0] = UnityEngine.Object.Instantiate(\_floor, new Vector3(this.x,0,this.z), new Quaternion());**

**tile[0].name = "Tile " + (int)(index + 1);**

**tile[0].transform.localScale = new Vector3(\_cell\_scale, 1, \_cell\_scale);**

**}**

**// Generating walls depending on side array**

**public void generateSide(int[] \_side,GameObject \_wall,float \_tile\_size, float \_cell\_scale)**

**{**

**//left //1**

**//right //2**

**//front //3**

**//back //4**

**side = \_side;**

**float xz\_offset = \_tile\_size / 2 + wall\_size / 2;**

**float y = \_tile\_size / 2;**

**if (side[0] == 1)**

**{**

**tile[1] = UnityEngine.Object.Instantiate(\_wall, new Vector3(this.x - xz\_offset, y, this.z), new Quaternion());**

**tile[1].transform.Rotate(left\_right);**

**tile[1].name = "Wall " + (int)(index + 1) + " left";**

**tile[1].transform.localScale = new Vector3(\_cell\_scale \* 10, wall\_size, \_cell\_scale \* 10);**

**}**

**if (side[1] == 1){}**

**if (side[2] == 1) {}**

**if (side[3] == 1) {}**

**}**

**// Destroys all stored objects**

**public void clear()**

**{**

**for (int i = 0; i < tile.Length; i++)**

**{**

**UnityEngine.Object.Destroy(tile[i]);**

**}**

**}**

**}**