using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Grid : MonoBehaviour

{

public float gridSize = 1.28f;

Node[,] nodeGrid;

public Node goalNode;

public List<Node> spawnNodeList = new List<Node>();

public List<Node> finalPath;

public Node barnNode;

public int gridSizeX, gridSizeY;

public bool initialized = false;

public Pathfinding ref\_Pathfinding;

public Spawner ref\_Spawner;

GameManagerBehavior gameManager;

private GameObject ref\_Player;

private MiniMap ref\_MiniMap;

public GameObject level\_PF;

public GameObject healthWheat\_PF;

public GameObject eatenWheat\_PF;

private GameObject[] healthWheatArray;

public GameObject healthWheatParticle;

public GameObject backgroundTile;

private int BGTileCount = 15;

private float playerBoundsExtension = 4;

public int[,] drawArray;

public int startX, startY;

public GameObject[] colors;

private ObjectPool pool;

void Start()

{

ref\_Pathfinding = GetComponent<Pathfinding>();

ref\_Spawner = GetComponent<Spawner>();

ref\_Player = GameObject.FindGameObjectWithTag("Player");

ref\_MiniMap = GameObject.FindGameObjectWithTag("MiniMap").GetComponent<MiniMap>();

createMap();

creatHealthWheat();

setPlayerBounds();

ref\_MiniMap.nodeGrid = nodeGrid;

ref\_MiniMap.drawInitialMap();

initialized = true;

gameManager = GameObject.Find("GameManager").GetComponent<GameManagerBehavior>();

gameManager.spawnPointCount = spawnNodeList.Count;

ref\_Spawner.initialStart = true;

}

void createMap()

{

GameObject drawDot;

drawArray = flipArray(drawArray);

gridSizeX = drawArray.GetLength(0);

gridSizeY = drawArray.GetLength(1);

createBackground();

nodeGrid = new Node[drawArray.GetLength(0), drawArray.GetLength(1)];

for (int row = 0; row < drawArray.GetLength(0); row++)

{

for (int col = 0; col < drawArray.GetLength(1); col++)

{

bool temp\_Isoccupied = true;

bool IsSpawnPoint = false;

bool IsGoal = false;

bool IsBarn = false;

switch (drawArray[row, col])

{

case 0: // empty

drawDot = colors[0];

temp\_Isoccupied = false;

break;

case 1: // natural obstacles

drawDot = colors[1];

temp\_Isoccupied = true;

break;

case 3: // Emu Goal, player spawn

drawDot = colors[0];

temp\_Isoccupied = false;

IsGoal = true;

break;

case 4: // barn

drawDot = colors[4];

temp\_Isoccupied = false;

IsBarn = true;

break;

case 5: // wheat tile

drawDot = colors[3];

temp\_Isoccupied = true;

break;

default:

if (drawArray[row, col] > 20) // spawn point

{

drawDot = colors[2];

temp\_Isoccupied = false;

IsSpawnPoint = true;

}

else

{

drawDot = colors[1];

}

break;

}

var temp\_drawDot = Instantiate(drawDot, new Vector2(col \* gridSize, row \* gridSize), Quaternion.identity);

nodeGrid[row, col] = new Node(temp\_Isoccupied, new Vector2(col \* gridSize, row \* gridSize), row, col);

nodeGrid[row, col].status = drawArray[row, col];

if (temp\_drawDot.GetComponent<PlaceTurret\_L>() != null) //giving the tile the pathfinding script

{

temp\_drawDot.GetComponent<PlaceTurret\_L>().parentNode = nodeGrid[row, col];

temp\_drawDot.GetComponent<PlaceTurret\_L>().ref\_Pathfinding = ref\_Pathfinding;

temp\_drawDot.GetComponent<PlaceTurret\_L>().ref\_Spawner = ref\_Spawner;

}

if (IsSpawnPoint) //set spawn point for pathfinding

{

spawnNodeList.Add(nodeGrid[row, col]);

nodeGrid[row, col].spawnNumber = drawArray[row, col] - 20;

IsSpawnPoint = false;

}

if (IsGoal) //set end goal for pathfinding

{

goalNode = nodeGrid[row, col];

//barnNodes.Add(nodeGrid[row + 1, col]);

ref\_Player.transform.position = temp\_drawDot.transform.position; // set player spawn position

ref\_Player.GetComponent<PointNShootControl>().spawnPosition = temp\_drawDot.transform.position;

IsGoal = false;

}

if (IsBarn)

{

barnNode = nodeGrid[row, col];

IsBarn = false;

}

}

}

}

void createBackground()

{

for (int row = -BGTileCount; row < gridSizeX + BGTileCount; row++)

{

for (int col = -BGTileCount; col < gridSizeY + BGTileCount; col++)

{

var temp\_drawDot = Instantiate(backgroundTile, new Vector2(col \* gridSize, row \* gridSize), Quaternion.identity);

}

}

}

void setPlayerBounds()

{

ref\_Player.GetComponent<PointNShootControl>().XBounds = (gridSizeY + playerBoundsExtension) \* gridSize;

ref\_Player.GetComponent<PointNShootControl>().YBounds = (gridSizeX + playerBoundsExtension) \* gridSize;

ref\_Player.GetComponent<PointNShootControl>().BoundsExtension = playerBoundsExtension \* gridSize;

}

public List<Node> getNeighboringNodes(Node t\_Node)

{

List<Node> neighboringNodes = new List<Node>();

int xCheck;

int yCheck;

//Right side

xCheck = t\_Node.gridX + 1;

yCheck = t\_Node.gridY;

//Make sure the node is within bounds

if (xCheck >= 0 && xCheck < gridSizeX)

{

if (yCheck >= 0 && yCheck < gridSizeY)

{

neighboringNodes.Add(nodeGrid[xCheck, yCheck]);

}

}

//Left side

xCheck = t\_Node.gridX - 1;

yCheck = t\_Node.gridY;

//Make sure the node is within bounds

if (xCheck >= 0 && xCheck < gridSizeX)

{

if (yCheck >= 0 && yCheck < gridSizeY)

{

neighboringNodes.Add(nodeGrid[xCheck, yCheck]);

}

}

//Top side

xCheck = t\_Node.gridX;

yCheck = t\_Node.gridY + 1;

//Make sure the node is within bounds

if (xCheck >= 0 && xCheck < gridSizeX)

{

if (yCheck >= 0 && yCheck < gridSizeY)

{

neighboringNodes.Add(nodeGrid[xCheck, yCheck]);

}

}

//Bottom side

xCheck = t\_Node.gridX;

yCheck = t\_Node.gridY - 1;

//Make sure the node is within bounds

if (xCheck >= 0 && xCheck < gridSizeX)

{

if (yCheck >= 0 && yCheck < gridSizeY)

{

neighboringNodes.Add(nodeGrid[xCheck, yCheck]);

}

}

return neighboringNodes;

}

int[,] flipArray(int[,] drawArray)

{

int temp;

int rows = drawArray.GetLength(0);

int columns = drawArray.GetLength(1);

for (int i = 0; i < rows; i++)

{

if (i < ((rows - 1) - i))

{

for (int j = 0; j < columns; j++)

{

temp = drawArray[i, j];

drawArray[i, j] = drawArray[(rows - 1) - i, j];

drawArray[(rows - 1) - i, j] = temp;

}

}

else

{

break;

}

}

return drawArray;

}

void creatHealthWheat()

{

healthWheatArray = new GameObject[5];

Instantiate(eatenWheat\_PF, nodeGrid[goalNode.gridX, goalNode.gridY + 1].position, Quaternion.identity);

var temp\_HealthWheat = Instantiate(healthWheat\_PF, nodeGrid[goalNode.gridX, goalNode.gridY + 1].position, Quaternion.identity);

nodeGrid[goalNode.gridX, goalNode.gridY + 1].IsOccupied = true;

nodeGrid[goalNode.gridX, goalNode.gridY + 1].status = 7;

healthWheatArray[0] = temp\_HealthWheat;

Instantiate(eatenWheat\_PF, nodeGrid[goalNode.gridX + 2, goalNode.gridY].position, Quaternion.identity);

temp\_HealthWheat = Instantiate(healthWheat\_PF, nodeGrid[goalNode.gridX + 2, goalNode.gridY].position, Quaternion.identity);

nodeGrid[goalNode.gridX + 2, goalNode.gridY].IsOccupied = true;

nodeGrid[goalNode.gridX + 2, goalNode.gridY].status = 7;

healthWheatArray[1] = temp\_HealthWheat;

Instantiate(eatenWheat\_PF, nodeGrid[goalNode.gridX + 2, goalNode.gridY - 1].position, Quaternion.identity);

temp\_HealthWheat = Instantiate(healthWheat\_PF, nodeGrid[goalNode.gridX + 2, goalNode.gridY - 1].position, Quaternion.identity);

nodeGrid[goalNode.gridX + 2, goalNode.gridY - 1].IsOccupied = true;

nodeGrid[goalNode.gridX + 2, goalNode.gridY - 1].status = 7;

healthWheatArray[2] = temp\_HealthWheat;

Instantiate(eatenWheat\_PF, nodeGrid[goalNode.gridX, goalNode.gridY - 2].position, Quaternion.identity);

temp\_HealthWheat = Instantiate(healthWheat\_PF, nodeGrid[goalNode.gridX, goalNode.gridY - 2].position, Quaternion.identity);

nodeGrid[goalNode.gridX, goalNode.gridY - 2].IsOccupied = true;

nodeGrid[goalNode.gridX, goalNode.gridY - 2].status = 7;

healthWheatArray[3] = temp\_HealthWheat;

Instantiate(eatenWheat\_PF, nodeGrid[goalNode.gridX + 1, goalNode.gridY - 2].position, Quaternion.identity);

temp\_HealthWheat = Instantiate(healthWheat\_PF, nodeGrid[goalNode.gridX + 1, goalNode.gridY - 2].position, Quaternion.identity);

nodeGrid[goalNode.gridX + 1, goalNode.gridY - 2].IsOccupied = true;

nodeGrid[goalNode.gridX + 1, goalNode.gridY - 2].status = 7;

healthWheatArray[4] = temp\_HealthWheat;

}

public void deductHealthWheat(int health)

{

if (initialized && healthWheatArray[health] != null)

{

Instantiate(healthWheatParticle, healthWheatArray[health].transform.position, Quaternion.identity);

Destroy(healthWheatArray[health]);

}

}

}