using System.Collections;

using System.Collections.Generic;

using UnityEngine;

/\*public class Block

{

public int type;

// is visible true/false

public bool vis;

// physical block in array

public GameObject block;

// constructor

public Block(int t, bool v, GameObject b)

{

// type of block

type = t;

// visibility of the block

vis = v;

// the block

block = b;

}

}\*/

// interface to use test function across several classes that use it( check for instances of the diferent public blocks)

public interface IMyInterface

{

//test function

void test();

}

// class which implements the interface

// NOTE: all interface methods must be fully implemented with code

public class MCBlock : IMyInterface

{

// type of block

public int type;

// is visible true/false

public bool vis;

// physical block in array

public GameObject block;

// constructor

// public block methods

public MCBlock (int t, bool v, float x, float y, float z)

{

// type of block

type = t;

// visibility of the block

vis = v;

// dynamically create a primitive cube

block = GameObject.CreatePrimitive(PrimitiveType.Cube);

// position the cube

block.transform.position = new Vector3(x, y, z);

// rotate the cube (Quaternion.identity is a default rotation value)

block.transform.rotation = Quaternion.identity;

test();

}

// function AddMaterial

public void AddMaterial (Material m)

{

//get the block color

block.GetComponent<Renderer>().material = m;

}

// a virtual function test

public virtual void test()

{

//display "MC Block added ..." in the console

Debug.Log("MC Block added ...");

}

}

// class used to instantiate prefabs at runtime

// note: a prefab is a collection of predefined GameObjects and Components

public class GrassBlock : MCBlock

{

// constructor method

// base () calls the base class constructor method with input parameters

public GrassBlock(int t, bool v, float x, float y, float z) : base (t, v, x, y, z)

{

// Assigns a material to the object.

Material newMat = Resources.Load("Minecraft\_Grass", typeof(Material)) as Material;

//change material

AddMaterial(newMat);

}

// override function test

public override void test()

{

//display "Grass Block added ..." in the console

Debug.Log("Grass Block added ...");

}

}

// class used to instantiate prefabs at runtime

public class CloudBlock : MCBlock

{

// constructor method

// base () calls the base class constructor method with input parameters

public CloudBlock(int t, bool v, float x, float y, float z) : base(t, v, x, y, z)

{

// Assign a material to the object.

Material newMat = Resources.Load("nocrack", typeof(Material)) as Material;

//change material

AddMaterial(newMat);

}

// override function test

public override void test()

{

//display "Cloud Block added ..." in the console

Debug.Log("Cloud Block added ...");

}

}

// class used to instantiate prefabs at runtime

public class SnowBlock : MCBlock

{

// constructor method

// base () calls the base class constructor method with input parameters

public SnowBlock(int t, bool v, float x, float y, float z) : base(t, v, x, y, z)

{

// Assigns a material to the object.

Material newMat = Resources.Load("Minecraft\_Snow", typeof(Material)) as Material;

//change material

AddMaterial(newMat);

}

}

// class used to instantiate prefabs at runtime

public class SandBlock : MCBlock

{

// constructor method

// base () calls the base class constructor method with input parameters

public SandBlock(int t, bool v, float x, float y, float z) : base(t, v, x, y, z)

{

// Assigns a material to the object.

Material newMat = Resources.Load("Minecraft\_Dirt", typeof(Material)) as Material;

//change material

AddMaterial(newMat);

}

}

// class used to instantiate prefabs at runtime

public class DiamondBlock : MCBlock

{

// constructor method

// base () calls the base class constructor method with input parameters

public DiamondBlock(int t, bool v, float x, float y, float z) : base(t, v, x, y, z)

{

// Assigns a material to the object.

Material newMat = Resources.Load("diamond", typeof(Material)) as Material;

//change material

AddMaterial(newMat);

}

}

// class used to generate the landscape, inheritance from MonoBehaviour

public class GenerateLandscape : MonoBehaviour

{

// set the variables

// define the size of the 3 dimensional matrix (landscape)

public static int width = 128;

public static int depth = 128;

public static int height = 128;

// define the height

public int heightScale = 20;

// define the offset of the height

public int heightOffset = 100;

//define the detail of the map

public float detailScale = 25.0f;

//NO LONGER NEEDED (INHERITANCE HAS BEEN USED)

// define grassBlock/sandBlock/SnowBlock

/\*

public GameObject grassBlock;

public GameObject sandBlock;

public GameObject snowBlock;

public GameObject cloudBlock;

public GameObject diamondBlock;

\*/

// public GameObject bottomBlock;

// declare the array, name thearray, create the array (matrix)

MCBlock[,,] worldBlocks = new MCBlock[width, height, depth];

// Use this for initialization

void Start()

{

//seed value on network time times 10 (get rid of decimal place)

int seed = (int)Network.time \* 10;

// nested for loop

// if z position is 0 or less add 1

for (int z = 0; z < depth; z++)

{

// if x position is 0 or less add 1

for (int x = 0; x < width; x++)

{

// set the height of the block using perlinNoise

int y = (int)(Mathf.PerlinNoise((x + seed) / detailScale, (z + seed) / detailScale) \* heightScale) + heightOffset;

// create the position of the block

Vector3 blockPos = new Vector3(x, y, z);

// create block function and make visible the top layer

CreateBlock(y, blockPos, true);

// loop y > 0

while (y > 0)

{

// the block below the top block

y--;

// find the position, create the block

blockPos = new Vector3(x, y, z);

// create block but don't instantiate

CreateBlock(y, blockPos, false);

}

}

}

// draw clouds, 20 clouds, 100 loops on the crawler

DrawClouds(20, 100);

// dig mines, 5 mines, 500 loops on the crawler

DigMines(5, 500);

}

//draw clouds function

void DrawClouds(int numClouds, int cSize)

{

// nested for loop

// loop for the number of clouds

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

{

// start position x for clouds (0 - 127)

int xpos = Random.Range(0, width);

// start position z for clouds (0 - 127)

int zpos = Random.Range(0, depth);

// loop for the size of the clouds

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

{

// set block position height

Vector3 blockPos = new Vector3(xpos, height - 1, zpos);

// create the clouds

// GameObject newBlock = (GameObject)Instantiate(cloudBlock, blockPos, Quaternion.identity);

CloudBlock newBlock = new CloudBlock (4, true, (float)blockPos.x, (float)blockPos.y, (float)blockPos.z);

// create a block for the clouds (true - visible)

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = newBlock;

// update the x position to a position close by (-1, 2)

xpos += Random.Range(-1, 2);

// update the x position to a position close by (-1, 2)

zpos += Random.Range(-1, 2);

// if the crawler goes out of the array this re centers it

if (xpos < 0 || xpos >= width) xpos = width / 2;

if (zpos < 0 || zpos >= depth) zpos = depth / 2;

}

}

}

// dig mines function

void DigMines(int numMines, int mSize)

{

// variable holesize

int holeSize = 2;

// the number of mines

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

{

//

// define the range on the x axis

int xpos = Random.Range(10, width - 10);

// define the range on the y axis

int ypos = Random.Range(10, height - 10);

// define the range on the z axis

int zpos = Random.Range(10, depth - 10);

// the size of the mines

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

{

// triple for loop

// with holeSize = 2 the hole size ends up 5\*5\*5

// dig out blocks on x axis

for(int x = -holeSize; x <=holeSize; x++)

// dig out blocks on y axis

for(int y = -holeSize; y <=holeSize; y++)

// dig out blocks on z axis

for(int z = -holeSize; z <=holeSize; z++)

{

// if x,y andz are all not 0

if(!(x == 0 && y == 0 && z == 0))

{

// define the position of the block

Vector3 blockPos = new Vector3(xpos + x, ypos + y, zpos + z);

// if there is a block in this position (not null)

try

{

if (worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] != null)

// if there is a block in this position,

if (worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].block != null)

// destroy the block

Destroy(worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].block);

// destroy the instance in the array

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = null;

}

catch

{

//

}

}

}

// crawler

// define the range on the x axis

xpos += Random.Range(-1, 2);

// define the range on the z axis, (not sure why the order is different)

zpos += Random.Range(-1, 2);

// define the range on the y axis

ypos += Random.Range(-1, 2);

// check to see it's not out of range on the x axis, if it is out relocate it back to the center

if (xpos < holeSize || xpos >= width - holeSize) xpos = width / 2;

// check to see it's not out of range on the y axis, if it is out relocate it back to the center

if (ypos < holeSize || xpos >= height - holeSize) ypos = width / 2;

// check to see it's not out of range on the z axis, if it is out relocate it back to the center

if (zpos < holeSize || xpos >= depth - holeSize) zpos = width / 2;

}

}

// for loop

// locate blocks on the z axis, single or plural???????????

for(int z = 1; z < depth-1; z++)

{

// locate blocks on the x axis, single or plural???????????

for (int x = 1; x < width-1; x++)

{

// locate blocks on the y axis, single or plural???????????

for (int y = 1; y < height-1; y++)

{

// check to see that there is no block there

if(worldBlocks[x,y,z] == null)

{

// locate the surrounding blocks on the x axis

for(int x1 = -1; x1 <=1; x1++)

// locate the surrounding blocks on the y axis

for (int y1 = -1; y1 <=1; y1++)

//locate the surrounding blocks on the z axis

for (int z1 = -1; z1 <=1; z1++)

{

// ????????????????????????????????????????

if(!(x1 == 0 && y1 == 0 && z1 == 0))

{

// define the position

Vector3 neighbour = new Vector3(x + x1, y + y1, z + z1);

// use drawBlock function

DrawBlock(neighbour);

}

}

}

}

}

}

}

// create block function

void CreateBlock(int y, Vector3 blockPos, bool create)

{

//

//GameObject newBlock = null;

// Instatiate the grass block in a position with a rotation !!!! better wording

//if height more than 15 instantiate a snowBlock

if (y > 115)

{

if(create)

{

// create a snow block

//newBlock = (GameObject) Instantiate(snowBlock, blockPos, Quaternion.identity);

SnowBlock newBlock = new SnowBlock(1, create, (float)blockPos.x, (float)blockPos.y, (float)blockPos.z);

// make the snow block visible

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = newBlock;

}

}

//if height more than 5 instantiate a grassBlock

else if (y > 105)

{

if(create)

{

// create a grass block

// newBlock = (GameObject)Instantiate(grassBlock, blockPos, Quaternion.identity);

GrassBlock newBlock = new GrassBlock(2, create, (float)blockPos.x, (float)blockPos.y, (float)blockPos.z);

// make the grass block visible

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = newBlock;

}

}

/\* else if (y > 50)

{

// water block?

if (create)

{

//

}

}

\*/

/\*

// if height is less than 1 create a bottom block

if (y < 1)

{

if (create)

// create a bottom block

newBlock = (GameObject) Instantiate(bottomBlock, blockPos, Quaternion.identity);

// make the bottom block visible

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = new Block(4???, create, newBlock); remove 4???

}

\*/

//if height less than 5 or is 5 instantiate a sandBlock

else

{

if(create)

{

// create a sand block

//newBlock = (GameObject)Instantiate(sandBlock, blockPos, Quaternion.identity);

SandBlock newBlock = new SandBlock(3, create, (float)blockPos.x, (float)blockPos.y, (float)blockPos.z);

// make the sand block visible

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = newBlock;

}

}

// if y is more than 80 and less than 85 random value of less than 10

if(y > 80 && y < 85 && Random.Range(0,100) < 10)

{

if (create)

{

// create a diamond block

//newBlock = (GameObject)Instantiate(diamondBlock, blockPos, Quaternion.identity);

DiamondBlock newBlock = new DiamondBlock(5, create, (float)blockPos.x, (float)blockPos.y, (float)blockPos.z);

// make the sand block visible

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = newBlock;

}

}

}

// drawblock function

void DrawBlock(Vector3 blockPos)

{

// to stop you digging outside the map

if (blockPos.x < 0 || blockPos.x >= width-1 ||

blockPos.y < 0 || blockPos.y >= height-1 ||

blockPos.z < 0 || blockPos.x >= depth-1) return;

// test if the block exists in this position return from this function

if (worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] == null) return;

// if the block is not visible draw it (instantiate)

if (!worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].vis)

{

//-----------

MCBlock newBlock = null;

// set the block to be visible

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].vis = true;

// find out if it is a snowblock (type 1)

if (worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 1)

// draw a snowblock

// newBlock = (GameObject) Instantiate(snowBlock, blockPos, Quaternion.identity);

newBlock = new SnowBlock(1, true, (float)blockPos.x, (float)blockPos.y, (float)blockPos.z);

// find out if it is a grassblock (type 2)

else if (worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 2)

// draw a grassblock

//newBlock = (GameObject) Instantiate(grassBlock, blockPos, Quaternion.identity);

newBlock = new GrassBlock(2, true, (float)blockPos.x, (float)blockPos.y, (float)blockPos.z);

// find out if it is a sandblock (type 3)

else if (worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 3)

// draw a sandblock

// newBlock = (GameObject) Instantiate(sandBlock, blockPos, Quaternion.identity);

newBlock = new SandBlock(3, true, (float)blockPos.x, (float)blockPos.y, (float)blockPos.z);

// find out if it is a diamondblock (type 5)

else if (worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 5)

// draw a diamondblock

// newBlock = (GameObject)Instantiate(diamondBlock, blockPos, Quaternion.identity);

newBlock = new DiamondBlock(5, true, (float)blockPos.x, (float)blockPos.y, (float)blockPos.z);

/\*

// find out if it is a bottomblock (type 4)

else if (worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 4)

// draw a bottomblock

newBlock = (GameObject) Instantiate(bottomBlock, blockPos, Quaternion.identity);

\*/

else

// if not type 1,2,3 or 4 turn the vision off

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].vis = false;

// if newBlock exists

if (newBlock != null)

//

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = newBlock;

}

}

int NeighbourCount(Vector3 blockPos)

{

int nCount = 0;

// same comments as below, check for a better explaination

//counting any neighbouring blocks for the blockpos vector

for(int x = -1; x <=1; x++)

for(int y=-1; y <=1; y++)

for(int z=-1; z <=1; z++)

{

if(!(x == 0 && y ==0 && z == 0))

{

// if these blocks exist

if (worldBlocks[(int)blockPos.x + x, (int)blockPos.y + y, (int)blockPos.z + z] != null)

nCount++;

}

}

// returns a count of 26

return (nCount);

}

//functiom

// check obscured neighbours using the new blocks position

void CheckObscuredNeighbours(Vector3 blockPos)

{

// change the x value between -1 and 1

// add to the current block to pick up the neighbouring blocks

for (int x = -1; x <=1; x++)

// change the y value between -1 and 1

// add to the current block to pick up the neighbouring blocks

for (int y = -1; y <=1; y++)

// change the z value between -1 and 1

// add to the current block to pick up the neighbouring blocks

for (int z = -1; z <=1; z++)

{

// check to see that its not the block that has been added

if (!(x == 0 && y == 0 && z == 0))

{

//new variable

// neighbour, position of new block plus changes on the x,y and z axis (from the for loop)

Vector3 neighbour = new Vector3(blockPos.x + x, blockPos.y + y, blockPos.z + z);

// if it is outside the map on the x axis

if (neighbour.x < 1 || neighbour.x > width - 2 ||

// if it is outside the map on the y axis

neighbour.y < 1 || neighbour.y > height - 2 ||

// if it is outside the map on the z axis, if it is continue

neighbour.z < 1 || neighbour.z > depth - 2) continue;

// check to see that the block exists

if(worldBlocks[(int)neighbour.x,(int)neighbour.y,(int)neighbour.z] != null)

{

// if the neighbour count is 26

if(NeighbourCount(neighbour) == 26)

{

// destroy the block

Destroy(worldBlocks[(int)neighbour.x, (int)neighbour.y, (int)neighbour.z].block);

worldBlocks[(int)neighbour.x, (int)neighbour.y, (int)neighbour.z] = null;

}

}

}

}

}

// Update is called once per frame

void Update ()

{

// test if the left mouse button is down, also used on touch screens

if (Input.GetMouseButtonDown(0))

{

// shoot a ray from the center of the screen

RaycastHit hit;

// set the ray to the camera position as a reference point to render from

Ray ray = Camera.main.ScreenPointToRay(new Vector3(Screen.width / 2.0f, Screen.height / 2.0f, 0));

// set the rays distance, detect if it hits something

if(Physics.Raycast(ray, out hit,1000.0f))

{

// define the position of the block that has been hit

Vector3 blockPos = hit.transform.position;

// this is the bottom block. Don't delete it. (return - get out of function)

if ((int)blockPos.y == 0) return;

// remove block from the array

worldBlocks[(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = null;

// delete object in the scene

Destroy(hit.transform.gameObject);

// 3 nested for loop

// find the block behind on the x axis

for(int x = -1; x<= 1; x++)

// find the block behind on the y axis

for (int y = -1; y <=1; y++)

// find the block behind on the z axis

for (int z = -1; z <= 1; z++)

{

// 46 min first video need a better explanation

if(!(x == 0 && y == 0 && z == 0))

{

// find the neighbour block

Vector3 neighbour = new Vector3(blockPos.x + x, blockPos.y + y, blockPos.z + z);

// draw blocks that Vector3 neighbour has found

DrawBlock(neighbour);

}

}

}

}

// test for right mouse button click (down)

else if (Input.GetMouseButtonDown(1))

{

// shoot a ray from the center of the screen

RaycastHit hit;

// set the ray to the camera position as a reference point to render from

Ray ray = Camera.main.ScreenPointToRay(new Vector3(Screen.width / 2.0f, Screen.height / 2.0f, 0));

// set the rays distance, detect if it hits something

if (Physics.Raycast(ray, out hit, 1000.0f))

{

// define the position of the block that has been hit

Vector3 blockPos = hit.transform.position;

// - one position along raycast vector to get the new position

Vector3 hitVector = blockPos - hit.point;

// if the distance is greatest on the x axis a new block will be created on the x axis

hitVector.x = Mathf.Abs(hitVector.x);

// if the distance is greatest on the y axis a new block will be created on the y axis

hitVector.y = Mathf.Abs(hitVector.y);

// if the distance is greatest on the z axis a new block will be created on the z axis

hitVector.z = Mathf.Abs(hitVector.z);

// if the x value is the greatest

if (hitVector.x > hitVector.z && hitVector.x > hitVector.y)

// add a block on the x axis

blockPos.x -= (int)Mathf.RoundToInt(ray.direction.x);

// if the y value is the greatest

else if (hitVector.y > hitVector.x && hitVector.y > hitVector.z)

// add a block on the y axis

blockPos.y -= (int)Mathf.RoundToInt(ray.direction.y);

else

// if not x or y axis add a block on the z axis

blockPos.z -= (int)Mathf.RoundToInt(ray.direction.z);

// add the block into the array and instantiate the block making it visible (true)

CreateBlock((int) blockPos.y, blockPos, true);

// CheckObscuredNeighbours block is in the array

}

}

}

}

// add extra block types eg - gems. last 5 mins of the 2nd video