# LAMPIRAN

## MapGenerator.cs

```
using UnityEngine;
using UnityEngine.UI;
public class MapGenerator :
MonoBehaviour
{
    [Header("World Settings")]
    public int width = 512;
    public int height = 512;
    public float
heightMultiplier = 10f;
    public int seed = 0;

    [Header("Continents
Layer")]
    public float
continentScale = 100f;
    public float
continentThreshold = 0.3f;

    [Header("Terrain Layer")]
    public float terrainScale
= 25f;
    public float
terrainMultiplier = 1f;

    [Header("Erosion Layer")]
    public float erosionScale
= 50f;
    public float
erosionIntensity = 0.5f;
    [Header("River Layer")]
    public float riverScale =
80f;
    public float
riverThreshold = 0.02f;
    [Header("Biome Settings")]
    public float
temperatureScale = 50f;
    public float humidityScale
= 60f;
    public float biomeBlend =
0.05f;
    [Header("Visualization")]
    public RawImage
biomeMapRawImage;
    private float[,]
heightMap;
    private float[,]
temperatureMap;
    private float[,]
humidityMap;
    private Biome[,] biomeMap;
    public GameObject player;
    public enum Biome : int
    {
        Ocean = 0,
        Beach = 1,
        Grassland = 2,
        Forest = 3,
        Mountain = 4,
        Snow = 5
    }
    public float[,]
GetHeightMap() => heightMap;
    public Biome[,]
GetBiomeMap() => biomeMap;
    public float
GetHeightAt(int x, int z) =>
heightMap[x, z];
    public Biome
GetBiomeAt(int x, int z) =>
biomeMap[x, z];
    public int GetMapWidth()
=> width;
    public int GetMapHeight()
=> height;

    public static MapGenerator
instance;
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
Destroy(gameObject);
    }
    void Start()
    {
        player =
GameStart.instance.player;
        InitializeMaps();
GenerateWorld();
    }
    void Update()
    {
        VisualizeBiomeMap();
    }
    public void
GenerateWorld()
    {
        InitializeMaps();

ApplyContinentsLayer();
        ApplyTerrainLayer();
        ApplyErosionLayer();
        ApplyRiverLayer();
        GenerateBiomeMaps();
        GenerateBiomeMap();
```

```
        VisualizeBiomeMap();
    }
    public void
InitializeMaps()
    {
        heightMap = new
float[width, height];
        temperatureMap = new
float[width, height];
        humidityMap = new
float[width, height];
        biomeMap = new
Biome[width, height];
    }
    private float
GetPerlinNoise(float x, float
y)
    {
        float newX = x + seed;
        float newY = y + seed;
        return
Mathf.PerlinNoise(newX, newY);
    }
    void
ApplyContinentsLayer()
    {
        float transitionRange
= 0.2f;

        for (int x = 0; x <
width; x++)
        {
            for (int y = 0; y
< height; y++)
            {
                float xCoord =
x / continentScale;
                float yCoord =
y / continentScale;
                float sample =
GetPerlinNoise(xCoord,
yCoord);
                if (sample <
continentThreshold)
                {
                    float t =
Mathf.InverseLerp(continentThr
eshold - transitionRange,
continentThreshold, sample);

heightMap[x, y] =
Mathf.SmoothStep(0f, 0.05f,
t);
                }
                else
heightMap[x, y] = sample -
continentThreshold;
            }
        }
    }
    void ApplyTerrainLayer()
    {
        for (int x = 0; x <
width; x++)
        {
            for (int y = 0; y
< height; y++)
            {
                if
(heightMap[x, y] <= 0)
continue;

                float xCoord =
x / terrainScale;
                float yCoord =
y / terrainScale;
                heightMap[x,
y] += GetPerlinNoise(xCoord,
yCoord) * terrainMultiplier;
            }
        }
    }
    void ApplyErosionLayer()
    {
        for (int x = 0; x <
width; x++)
        {
            for (int y = 0; y
< height; y++)
            {
                if
(heightMap[x, y] <= 0)
continue;

                float xCoord =
x / erosionScale;
                float yCoord =
y / erosionScale;
                heightMap[x,
y] -= GetPerlinNoise(xCoord,
yCoord) * erosionIntensity;
                heightMap[x,
y] =
Mathf.Clamp01(heightMap[x,
y]);
            }
        }
    }
    void ApplyRiverLayer()
    {
```

```
        for (int x = 0; x <
width; x++)
        {
            for (int y = 0; y
< height; y++)
            {
                float xCoord =
x / riverScale;
                float yCoord =
y / riverScale;

                float
riverNoise =
Mathf.Abs(Mathf.PerlinNoise(xC
oord + seed, yCoord + seed) -
0.5f);

                if (riverNoise
< riverThreshold &&
heightMap[x, y] > 0.1f)
                {

heightMap[x, y] -= 0.05f;

biomeMap[x, y] = Biome.Ocean;
                }
            }
        }
    }
    void GenerateBiomeMaps()
    {
        for (int x = 0; x <
width; x++)
        {
            for (int y = 0; y
< height; y++)
            {

temperatureMap[x, y] =
GetPerlinNoise(x /
temperatureScale, y /
temperatureScale);
                humidityMap[x,
y] = GetPerlinNoise(x /
humidityScale, y /
humidityScale);
            }
        }
    }
    void GenerateBiomeMap()
    {
        for (int x = 0; x <
width; x++)
        {
            for (int y = 0; y
< height; y++)
            {
                float
elevation = heightMap[x, y];
                float temp =
temperatureMap[x, y];
                float humidity
= humidityMap[x, y];

                if (elevation
< 0.1f)
                {

biomeMap[x, y] = Biome.Ocean;
                }
                else if
(elevation < 0.15f)
                {

biomeMap[x, y] = Biome.Beach;
                }
                else if
(elevation > 0.85f)
                {

biomeMap[x, y] = temp < 0.5f ?
Biome.Snow : Biome.Mountain;
                }
                else
                {

biomeMap[x, y] =
GetContinentalBiome(temp,
humidity);
                }
            }
        }
    }
    Biome
GetContinentalBiome(float
temp, float humidity)
    {
        if (temp < 0.2f)
        {
            return Biome.Snow;
        }
        else if (temp < 0.4f)
        {
            return humidity <
0.3f ? Biome.Mountain :
Biome.Forest;
        }
        else if (temp < 0.6f)
        {
            return humidity <
0.4f ? Biome.Grassland :
Biome.Forest;
```

```
        }
        else if (temp < 0.8f)
        {
            return humidity <
0.3f ? Biome.Grassland :
Biome.Forest;
        }
        else
        {
            return humidity <
0.4f ? Biome.Beach :
Biome.Forest;
        }
    }
    void VisualizeBiomeMap()
    {
        if
(PlayerController.instance.IsM
apOpen)
        {
            Texture2D texture
= new Texture2D(width,
height);
            Vector3
playerPosition =
player.transform.position;
            float
worldUnitsPerPixel = 45f;
            int playerX =
Mathf.FloorToInt(playerPositio
n.x / worldUnitsPerPixel);
            int playerZ =
Mathf.FloorToInt(playerPositio
n.z / worldUnitsPerPixel);
            int markerSize =
2;
            for (int x = 0; x
< width; x++)
            {
                for (int y =
0; y < height; y++)
                {
                    Color
color;

                    if
(biomeMap[x, y] ==
Biome.Ocean)
                    {
                        if
(heightMap[x, y] < 0.05f)
color = new Color(0f, 0f,
0.3f);
                        else
color = new Color(0.1f, 0.1f,
0.8f);
                    }
                    else color
= GetBiomeColor(biomeMap[x,
y]);
                    if
(Mathf.Abs(x - playerX) <=
markerSize && Mathf.Abs(y -
playerZ) <= markerSize)
                    {
                        color
= Color.red;
                    }

texture.SetPixel(x, y, color);
                }
            }
            texture.Apply();

biomeMapRawImage.texture =
texture;
        }
    }
    public Color
GetBiomeColor(Biome biome)
    {
        switch (biome)
        {
            case Biome.Ocean:
return new Color(0.0f, 0.4f,
0.7f);
            case Biome.Beach:
return new Color(0.98f, 0.9f,
0.6f);
            case
Biome.Grassland: return new
Color(0.44f, 0.75f, 0.3f);
            case Biome.Forest:
return new Color(0.13f, 0.55f,
0.13f);
            case
Biome.Mountain: return new
Color(0.5f, 0.5f, 0.5f);
            case Biome.Snow:
return new Color(0.95f, 0.95f,
1.0f);
            default: return
Color.magenta;
        }
    }
    public float[,]
GetChunkHeightMap(int chunkX,
int chunkZ, int chunkSize)
    {
        float[,] chunkMap =
new float[chunkSize,
chunkSize];
```

```
        for (int x = 0; x <
chunkSize; x++)
        {
            for (int z = 0; z
< chunkSize; z++)
            {
                int worldX =
chunkX * chunkSize + x;
                int worldZ =
chunkZ * chunkSize + z;

                if (worldX <
width && worldZ < height)

chunkMap[x, z] =
heightMap[worldX, worldZ];
                else

chunkMap[x, z] = 0f;
            }
        }
        return chunkMap;
    }
}
```

## ChunkGenerator.cs

```
using UnityEngine;
using
System.Collections.Generic;
using static MapGenerator;
using System.Collections;
using System;

public class ChunkManager :
MonoBehaviour
{
    public static ChunkManager
instance;
    [Header("References")]
    public MapGenerator
mapGen;
    public Transform player;

    [Header("Chunk Settings")]
    public int chunkSize = 64;
    public int viewDistance =
2;

    [Header("Spawning")]
    public ObjectSpawner
objectSpawner;
    public EnemySpawner
enemySpawner;
    public
List<BuiltStructure>
placedStructures = new
List<BuiltStructure>();
    public GameObject
craftingTablePrefab;
    public GameObject
tendaPrefab;
    public GameObject
kasurPrefab;
    public GameObject
apiPrefab;
    public GameObject
gubukPrefab;
    Dictionary<Vector2Int,
List<Matrix4x4>> grassMatrices
= new Dictionary<Vector2Int,
List<Matrix4x4>>();
    public Mesh grassMesh;
    public Material
grassMaterial;
    [Header("Water")]
    public GameObject
waterPlanePrefab;
    private GameObject
currentWaterPlane;
    public float waterHeight =
60f;
    [Header("Terrain
Settings")]
    public float scale = 50f;
    public float stepHeight =
30f;
    public Material[]
biomeMaterials;
    [Range(0f, 1f)]
    public float
stairSmoothness = 0.5f;
    private
Dictionary<Vector2Int,
GameObject> loadedChunks = new
Dictionary<Vector2Int,
GameObject>();
    private Vector2Int
currentPlayerChunk =
Vector2Int.zero;
    private Texture2D
biomeTexture;

    public System.Action
OnTerrainReady;
    private void Awake()
    {
        if (!instance)
        {
            instance = this;
        }
        else
```

```
        {
            Destroy(instance);
        }
    }
    private float[,]
heightMap;
    private Biome[,] biomeMap;
    private float
chunkCheckTimer = 0f;
    public float checkInterval
= 2f;
    void Update()
    {
        chunkCheckTimer +=
Time.deltaTime;
        if (chunkCheckTimer >=
checkInterval)
        {
            chunkCheckTimer =
0f;
            Vector2Int
newChunk =
GetChunkCoord(player.position)
;
            if (newChunk !=
currentPlayerChunk)
            {

currentPlayerChunk = newChunk;

UpdateChunks();
            }
        }
    }
    public void Start()
    {
        player =
GameStart.instance.player.tran
sform;

mapGen.GenerateWorld();
        heightMap =
mapGen.GetHeightMap();
        biomeMap =
mapGen.GetBiomeMap();

objectSpawner.Initialize();

enemySpawner.Initialize();
        currentPlayerChunk =
GetChunkCoord(player.position)
;
        UpdateChunks();
    }
    public void ResetChunks()
    {
        foreach (var chunk in
loadedChunks.Values)
        {
            Destroy(chunk);
        }
        loadedChunks.Clear();
        if (currentWaterPlane
!= null)
        {

Destroy(currentWaterPlane);
            currentWaterPlane
= null;
        }
        currentPlayerChunk =
GetChunkCoord(player.position)
;

mapGen.GenerateWorld();
        heightMap =
mapGen.GetHeightMap();
        biomeMap =
mapGen.GetBiomeMap();

objectSpawner.Initialize();

enemySpawner.Initialize();
        UpdateChunks();
    }


    void
GenerateBiomeTexture()
    {
        int width =
mapGen.GetMapWidth();
        int height =
mapGen.GetMapHeight();
        biomeTexture = new
Texture2D(width, height);

        var biomeMap =
mapGen.GetBiomeMap();
        for (int x = 0; x <
width; x++)
        {
            for (int y = 0; y
< height; y++)
            {
                Color color =
mapGen.GetBiomeColor(biomeMap[
x, y]);

biomeTexture.SetPixel(x, y,
color);
            }
```

```
        }
        biomeTexture.Apply();
    }
    Vector2Int
GetChunkCoord(Vector3
worldPos)
    {
        int x =
Mathf.FloorToInt(worldPos.x /
(chunkSize * scale));
        int z =
Mathf.FloorToInt(worldPos.z /
(chunkSize * scale));
        return new
Vector2Int(x, z);
    }
    private readonly
HashSet<Vector2Int>
wantedChunks = new
HashSet<Vector2Int>();
    private readonly
List<Vector2Int>
toRemoveChunks = new
List<Vector2Int>();

    void UpdateChunks()
    {
        wantedChunks.Clear();
        for (int dx = -
viewDistance; dx <=
viewDistance; dx++)
            for (int dz = -
viewDistance; dz <=
viewDistance; dz++)
            {
                var coord =
new
Vector2Int(currentPlayerChunk.
x + dx, currentPlayerChunk.y +
dz);

wantedChunks.Add(coord);
                if
(!loadedChunks.ContainsKey(coo
rd))

LoadChunk(coord);
            }


toRemoveChunks.Clear();
        foreach (var kv in
loadedChunks)
            if
(!wantedChunks.Contains(kv.Key
))
toRemoveChunks.Add(kv.Key);

        foreach (var coord in
toRemoveChunks)

UnloadChunk(coord);
        UpdateWaterPlane();
    }
    void LoadChunk(Vector2Int
coord)
    {
        var go = new
GameObject($"Chunk_{coord.x}_{
coord.y}");
        go.transform.parent =
transform;
        go.layer =
LayerMask.NameToLayer("Ground"
);

        var mf =
go.AddComponent<MeshFilter>();
        var mr =
go.AddComponent<MeshRenderer>(
);
        mr.materials =
biomeMaterials;
        var mc =
go.AddComponent<MeshCollider>(
);

        Mesh mesh =
GenerateChunkMesh(coord);
        mf.mesh = mesh;
        mc.sharedMesh = mesh;

        loadedChunks[coord] =
go;


StartCoroutine(SpawnDelayed(co
ord, go));
    }
    public ObjectPool
objectPool;
    void SpawnGrass(Vector2Int
coord)
    {
        float grassSpacing =
1f;
        int qualityLevel =
QualitySettings.GetQualityLeve
l();
        int density =
qualityLevel == 2 ? 75 : 50;
```

```
        float
grassHeightOffset = 0.1f;

        List<Matrix4x4>
matrices = new
List<Matrix4x4>();

        for (int z = 0; z <
chunkSize; z++)
        {
            for (int x = 0; x
< chunkSize; x++)
            {
                int worldX =
coord.x * chunkSize + x;
                int worldZ =
coord.y * chunkSize + z;
                Biome biome =
GetBiomeSafe(biomeMap, worldX,
worldZ, mapGen.GetMapWidth(),
mapGen.GetMapHeight());
                if (biome !=
Biome.Grassland && biome !=
Biome.Forest) continue;
                for (int i =
0; i < density; i++)
                {
                    float
offsetX =
UnityEngine.Random.Range(-
grassSpacing / 2f,
grassSpacing / 2f);
                    float
offsetZ =
UnityEngine.Random.Range(-
grassSpacing / 2f,
grassSpacing / 2f);
                    Vector3
potentialPos = new
Vector3((worldX + offsetX) *
scale, 0, (worldZ + offsetZ) *
scale);

                    RaycastHit
hit;
                    if
(Physics.Raycast(new
Vector3(potentialPos.x, 1000f,
potentialPos.z), Vector3.down,
out hit, 2000f,
LayerMask.GetMask("Ground")))
                    {
                        if
(hit.collider != null &&
hit.collider.gameObject.layer
==
LayerMask.NameToLayer("Ground"
))
                        {

Vector3 finalPos = hit.point +
Vector3.up *
grassHeightOffset;

Quaternion alignToNormal =
Quaternion.FromToRotation(Vect
or3.up, hit.normal);

Quaternion randomY =
Quaternion.Euler(0f,
UnityEngine.Random.Range(0f,
360f), 0f);

Quaternion finalRot =
alignToNormal * randomY;

Vector3 scl = Vector3.one *
UnityEngine.Random.Range(0.8f,
1.2f);

matrices.Add(Matrix4x4.TRS(fin
alPos, finalRot, scl));
                        }

                    }
                }
            }
        }

        grassMatrices[coord] =
matrices;
    }
    void LateUpdate()
    {
        if
(!grassMatrices.TryGetValue(cu
rrentPlayerChunk, out var
list)) return;

        Vector3 scale = new
Vector3(8f, 2.5f, 8f);
        Vector3 camPos =
Camera.main.transform.position
;
        float maxDistance =
200f;
        float sqrMaxDist =
maxDistance * maxDistance; //
biar hemat, pakai jarak
kuadrat
```

```
        for (int i = 0; i <
list.Count; i += 1023)
        {
            List<Matrix4x4>
scaledList = new
List<Matrix4x4>();

            int count =
Mathf.Min(1023, list.Count -
i);
            for (int j = 0; j
< count; j++)
            {
                Matrix4x4 m =
list[i + j];
                Vector3 pos =
m.GetColumn(3);

                // cek jarak
kuadrat
                if ((pos -
camPos).sqrMagnitude <=
sqrMaxDist)
                {
                    Quaternion
rot =
Quaternion.LookRotation(m.GetC
olumn(2), m.GetColumn(1));

scaledList.Add(Matrix4x4.TRS(p
os, rot, scale));
                }
            }

            if
(scaledList.Count > 0)

Graphics.DrawMeshInstanced(gra
ssMesh, 0, grassMaterial,
scaledList);
        }
    }
    IEnumerator
SpawnDelayed(Vector2Int coord,
GameObject go)
    {
        yield return null; //
Delay one frame

objectSpawner.SpawnObjectsInCh
unk(coord, go, chunkSize);

enemySpawner.SpawnEnemiesInChu
nk(coord, go, chunkSize);
        SpawnGrass(coord);

        // Delay beberapa
frame tambahan
        for (int i = 0; i < 5;
i++)
            yield return null;

        GameStart gameStart =
FindFirstObjectByType<GameStar
t>();
        if (gameStart != null
&& !gameStart.gameStarted)
        {

gameStart.PlacePlayerOnTerrain
();
        }
        foreach (var structure
in placedStructures)
        {
            Vector2Int
structChunk =
GetChunkCoord(structure.positi
on);
            if (structChunk ==
coord)
            {
                Quaternion rot
= structure.GetRotation();

PlaceStructure(structure.struc
tureType, structure.position,
rot);
            }
        }
    }
    void
UnloadChunk(Vector2Int coord)
    {
        GameObject chunk =
loadedChunks[coord];

        foreach (Transform
child in chunk.transform)
        {
            if (child == null)
continue;

            // Coba return ke
pool

objectPool.ReturnObjectToPool(
child.gameObject);
        }

        Destroy(chunk); //
Destroy hanya parent kosong
```

```
loadedChunks.Remove(coord);
    }
    Mesh
GenerateChunkMesh(Vector2Int
coord)
    {
        int worldW =
mapGen.GetMapWidth();
        int worldH =
mapGen.GetMapHeight();

        List<Vector3> vertices
= new List<Vector3>();
        List<Vector2> uvs =
new List<Vector2>();
        List<int> triangles =
new List<int>();
        List<int>
subMeshTriangles = new
List<int>(); // For each sub-
mesh

        // List to store each
sub-mesh (i.e., each biome)
        List<List<int>>
subMeshes = new
List<List<int>>();

        // Initialize
subMeshes for each biome
material
        for (int i = 0; i <
biomeMaterials.Length; i++)
        {
            subMeshes.Add(new
List<int>());
        }

        for (int z = 0; z <
chunkSize; z++)
        {
            for (int x = 0; x
< chunkSize; x++)
            {
                int worldX =
coord.x * chunkSize + x;
                int worldZ =
coord.y * chunkSize + z;

                // Heights
                float h00;
                float h10;
                float h01;
                float h11;

                float h00_raw
= GetHeightSafe(heightMap,
worldX, worldZ, worldW,
worldH);
                float h10_raw
= GetHeightSafe(heightMap,
worldX + 1, worldZ, worldW,
worldH);
                float h01_raw
= GetHeightSafe(heightMap,
worldX, worldZ + 1, worldW,
worldH);
                float h11_raw
= GetHeightSafe(heightMap,
worldX + 1, worldZ + 1,
worldW, worldH);

                // Membuat
height yang "semi stepped"
                h00 =
Mathf.Lerp(h00_raw,
Mathf.Round(h00_raw /
stepHeight) * stepHeight,
stairSmoothness);
                h10 =
Mathf.Lerp(h10_raw,
Mathf.Round(h10_raw /
stepHeight) * stepHeight,
stairSmoothness);
                h01 =
Mathf.Lerp(h01_raw,
Mathf.Round(h01_raw /
stepHeight) * stepHeight,
stairSmoothness);
                h11 =
Mathf.Lerp(h11_raw,
Mathf.Round(h11_raw /
stepHeight) * stepHeight,
stairSmoothness);


                //h00 =
Mathf.Round(h00 / stepHeight)
* stepHeight;
                //h10 =
Mathf.Round(h10 / stepHeight)
* stepHeight;
                //h01 =
Mathf.Round(h01 / stepHeight)
* stepHeight;
                //h11 =
Mathf.Round(h11 / stepHeight)
* stepHeight;

                // Get biome
for this quad
```

```
                Biome biome =
GetBiomeSafe(biomeMap, worldX,
worldZ, worldW, worldH);

                // Add
vertices (quad: 4 verts)
                int vertIndex
= vertices.Count;

vertices.Add(new
Vector3(worldX * scale, h00,
worldZ * scale));           //
v00

vertices.Add(new
Vector3((worldX + 1) * scale,
h10, worldZ * scale));     //
v10

vertices.Add(new
Vector3(worldX * scale, h01,
(worldZ + 1) * scale));     //
v01

vertices.Add(new
Vector3((worldX + 1) * scale,
h11, (worldZ + 1) * scale));
// v11

                // Add uvs
(use same UVs as before, no
change)
                uvs.Add(new
Vector2(0, 0));         // v00
                uvs.Add(new
Vector2(1, 0));         // v10
                uvs.Add(new
Vector2(0, 1));         // v01
                uvs.Add(new
Vector2(1, 1));         // v11

                // Add
triangles (2 triangles per
quad)

subMeshes[(int)biome].Add(vert
Index + 0);

subMeshes[(int)biome].Add(vert
Index + 2);

subMeshes[(int)biome].Add(vert
Index + 1);

subMeshes[(int)biome].Add(vert
Index + 1);

subMeshes[(int)biome].Add(vert
Index + 2);

subMeshes[(int)biome].Add(vert
Index + 3);
            }
        }

        // Now we create the
mesh with multiple sub-meshes
        Mesh m = new Mesh();
        m.vertices =
vertices.ToArray();
        m.uv = uvs.ToArray();

        m.subMeshCount =
biomeMaterials.Length; // One
sub-mesh per material/biome
        for (int i = 0; i <
biomeMaterials.Length; i++)
        {

m.SetTriangles(subMeshes[i].To
Array(), i); // Assign
triangles to each sub-mesh
        }


m.RecalculateNormals();

        // Return the mesh
        return m;
    }
    public float
GetHeightSafe(float[,]
heightMap, int x, int z, int
w, int h)
    {
        if (x >= 0 && x < w &&
z >= 0 && z < h)
        {
            float baseHeight =
heightMap[x, z] *
mapGen.heightMultiplier;
            Biome biome =
GetBiomeSafe(biomeMap, x, z,
w, h);
            switch (biome)
            {
                case
Biome.Snow:
```

```
                    return
baseHeight = heightMap[x, z] *
(1.2f *
mapGen.heightMultiplier);
                case
Biome.Mountain:
                    return
baseHeight = heightMap[x, z] *
(1.15f *
mapGen.heightMultiplier);
                case
Biome.Ocean:
                    return
baseHeight = heightMap[x, z] *
(0.4f *
mapGen.heightMultiplier);  //
Lower ocean
                case
Biome.Beach:
                    return
baseHeight = heightMap[x, z] *
(1.1f *
mapGen.heightMultiplier);  //
Lower ocean
                default:
                    return
baseHeight;
            }
        }
        return 0f;
    }
    Biome
GetBiomeSafe(Biome[,]
biomeMap, int x, int z, int w,
int h)
    {
        if (x >= 0 && x < w &&
z >= 0 && z < h)
            return biomeMap[x,
z];
        return Biome.Ocean;
    }
    void UpdateWaterPlane()
    {
        if (currentWaterPlane
!= null)

Destroy(currentWaterPlane);
        float worldX =
(currentPlayerChunk.x + 0.5f)
* chunkSize * scale;
        float worldZ =
(currentPlayerChunk.y + 0.5f)
* chunkSize * scale;
        Vector3 waterCenter =
new Vector3(worldX,
waterHeight, worldZ);
        currentWaterPlane =
Instantiate(waterPlanePrefab,
waterCenter,
Quaternion.identity);
        currentWaterPlane.name
= "WaterPlane";

currentWaterPlane.transform.lo
calScale = new Vector3(7f, 1f,
7f);
    }

    public void
PlaceStructure(string
structureType, Vector3
position, Quaternion rotation)
    {
        GameObject prefab =
GetStructurePrefab(structureTy
pe);
        if (prefab != null)
        {

Instantiate(prefab, position,
rotation);

placedStructures.Add(new
BuiltStructure(structureType,
position, rotation));
        }
    }

    public GameObject
GetStructurePrefab(string
type)
    {
        switch (type)
        {
            case "Meja
Crafting": return
craftingTablePrefab;
            case "Tenda":
return tendaPrefab;
            case "Ranjang":
return kasurPrefab;
            case "Api Unggun":
return apiPrefab;
            case "Gubuk":
return gubukPrefab;
            default: return
null;
        }
    }
```

```
}
[System.Serializable]
public class BuiltStructure
{
    public string
structureType;
    public Vector3 position;

    public float rotX;
    public float rotY;
    public float rotZ;
    public float rotW;

    public
BuiltStructure(string type,
Vector3 pos, Quaternion rot)
    {
        structureType = type;
        position = pos;
        rotX = rot.x;
        rotY = rot.y;
        rotZ = rot.z;
        rotW = rot.w;
    }

    public Quaternion
GetRotation()
    {
        return new
Quaternion(rotX, rotY, rotZ,
rotW);
    }
}
```

## ObjectSpawner.cs

```
using UnityEngine;
using
System.Collections.Generic;
using static MapGenerator;
[System.Serializable]
public class ObjectSpawnRule
{
    public GameObject
objectPrefab;
    public float spawnChance;
    public float
clusterFrequency;
    public float
clusterThreshold;
    public List<Biome> biomes;
}
public class ObjectSpawner :
MonoBehaviour
{
    public static
ObjectSpawner instance;
    [Header("Object
Spawning")]
    public
List<ObjectSpawnRule>
objectSpawnRules;
    private Dictionary<Biome,
List<ObjectSpawnRule>>
biomeRules;
    public ObjectPool
objectPool;
    private HashSet<Vector3>
takenObjectPositions = new
HashSet<Vector3>();
    private HashSet<string>
brokenObjectPositions = new
HashSet<string>();
    [Header("Terrain
Settings")]
    public float scale = 50f;
    public float stepHeight =
30f;
    [Header("Raycast
Settings")]
    public LayerMask
terrainLayerMask;
    public LayerMask
dontPlaceLayerMask;
    public MapGenerator
mapGen;
    private void Awake()
    {
        if (instance == null)
instance = this;
        else
Destroy(gameObject);
        terrainLayerMask =
LayerMask.GetMask("Ground",
"Terrain");
        dontPlaceLayerMask =
LayerMask.GetMask("Water",
"Interactable", "Resource");
        scale =
ChunkManager.instance.scale;
    }
    public void Initialize()
    {
        biomeRules = new
Dictionary<Biome,
List<ObjectSpawnRule>>();
        if (objectSpawnRules
== null)
        {
```

```
Debug.LogError("objectSpawnRul
es is not initialized!");
            return;
        }
        foreach (var rule in
objectSpawnRules)
        {
            if (rule == null
|| rule.biomes == null)
continue;
            foreach (var biome
in rule.biomes)
            {
                if
(!biomeRules.ContainsKey(biome
))

biomeRules[biome] = new
List<ObjectSpawnRule>();

biomeRules[biome].Add(rule);
            }
        }
        if (mapGen == null)
return;

        foreach (var rule in
objectSpawnRules)
        {
            if
(rule.objectPrefab != null)

objectPool.CreatePoolObject(ru
le.objectPrefab, 50);
        }
    }
    public void
SpawnObjectsInChunk(Vector2Int
chunkCoord, GameObject parent,
int chunkSize)
    {
        if (mapGen == null ||
biomeRules == null)  return;
        if (parent == null)
return;

Random.InitState(chunkCoord.x
* 73856093 ^ chunkCoord.y *
19349663);
        int startX =
chunkCoord.x * chunkSize;
        int startZ =
chunkCoord.y * chunkSize;
        int mapWidth =
mapGen.GetMapWidth();
        int mapHeight =
mapGen.GetMapHeight();
        var heightMap =
mapGen.GetHeightMap();
        var biomeMap =
mapGen.GetBiomeMap();
        if (heightMap == null
|| biomeMap == null) return;
        for (int x = 0; x <
chunkSize; x++)
        {
            for (int z = 0; z
< chunkSize; z++)
            {
                int worldX =
startX + x;
                int worldZ =
startZ + z;
                if (worldX < 0
|| worldX >= mapWidth ||
worldZ < 0 || worldZ >=
mapHeight) continue;



                Biome biome =
biomeMap[worldX, worldZ];
                if
(!biomeRules.ContainsKey(biome
)) continue;
                foreach (var
rule in biomeRules[biome])
                {
                    if (rule
== null || rule.objectPrefab
== null) continue;
                    float
clusterNoise =
Mathf.PerlinNoise(
                        worldX
* rule.clusterFrequency +
1000f,
                        worldZ
* rule.clusterFrequency +
1000f
                    );
                    if
(clusterNoise >=
rule.clusterThreshold &&
Random.value <=
rule.spawnChance)
                    {
                        int
clusterRadius = 1;
```

```
                        for
(int dx = -clusterRadius; dx
<= clusterRadius; dx++)
                        {

for (int dz = -clusterRadius;
dz <= clusterRadius; dz++)
                            {

int cx = worldX + dx;

int cz = worldZ + dz;

if (cx < 0 || cx >= mapWidth
|| cz < 0 || cz >= mapHeight)
continue;

if (biomeMap[cx, cz] != biome)
continue;

float jitterRange = scale *
0.4f;

float offsetX = Random.Range(-
jitterRange, jitterRange);

float offsetZ = Random.Range(-
jitterRange, jitterRange);

float posX = cx * scale +
offsetX;

float posZ = cz * scale +
offsetZ;

Vector3 rayOrigin = new
Vector3(posX, 1000f, posZ);

float rawHeight =
GetInterpolatedHeight(heightMa
p, posX, posZ);

float steppedHeight =
Mathf.Round(rawHeight /
stepHeight) * stepHeight;

float adjustedHeight =
Mathf.Lerp(rawHeight,
steppedHeight,
ChunkManager.instance.stairSmo
othness);

LayerMask combinedMask =
terrainLayerMask |
dontPlaceLayerMask;

Ray ray = new Ray(rayOrigin,
Vector3.down);

if (Physics.Raycast(ray, out
RaycastHit hitInfo, 2000f,
combinedMask))

{

if (((1 <<
hitInfo.collider.gameObject.la
yer) & dontPlaceLayerMask) !=
0) continue;

Vector3 pos = hitInfo.point;

string objectIdentifier =
rule.objectPrefab.name + "_" +
pos.x + "_" + pos.z;

if
(brokenObjectPositions.Contain
s(objectIdentifier)) continue;

if
(takenObjectPositions.Contains
(pos)) continue;

Quaternion rotation;

if
(rule.objectPrefab.GetComponen
t<ResourceStone>() != null)
rotation = Random.rotation;

else rotation =
Quaternion.Euler(0,
Random.Range(0f, 360f), 0);

GameObject obj =
objectPool.GetObjectFromPool(r
ule.objectPrefab);

if (obj != null)

{

obj.transform.position = pos;

obj.transform.rotation =
rotation;

obj.transform.SetParent(parent
.transform);
```

```
}

int seed = cx * 73856093 ^ cz
* 19349663 ^
rule.objectPrefab.name.GetHash
Code();

System.Random prng = new
System.Random(seed);

float scaleFactor =
Mathf.Lerp(1f, 2f,
(float)prng.NextDouble());

obj.transform.localScale =
rule.objectPrefab.transform.lo
calScale * scaleFactor;

if (obj == null) continue;

obj.name = objectIdentifier;

takenObjectPositions.Add(pos);

}
                            }
                        }
                    }
                }
            }
        }
    }

    float
GetInterpolatedHeight(float[,]
heightMap, float worldX, float
worldZ)
    {
        if (mapGen == null ||
heightMap == null) return 0f;
        int mapWidth =
mapGen.GetMapWidth();
        int mapHeight =
mapGen.GetMapHeight();
        float scaledX = worldX
/ scale;
        float scaledZ = worldZ
/ scale;
        int x0 =
Mathf.FloorToInt(scaledX);
        int x1 = x0 + 1;
        int z0 =
Mathf.FloorToInt(scaledZ);
        int z1 = z0 + 1;
        x0 = Mathf.Clamp(x0,
0, mapWidth - 1);
        x1 = Mathf.Clamp(x1,
0, mapWidth - 1);
        z0 = Mathf.Clamp(z0,
0, mapHeight - 1);
        z1 = Mathf.Clamp(z1,
0, mapHeight - 1);
        float h00 =
heightMap[x0, z0];
        float h10 =
heightMap[x1, z0];
        float h01 =
heightMap[x0, z1];
        float h11 =
heightMap[x1, z1];
        float tx = scaledX -
x0;
        float tz = scaledZ -
z0;
        float h0 =
Mathf.Lerp(h00, h10, tx);
        float h1 =
Mathf.Lerp(h01, h11, tx);
        float height =
Mathf.Lerp(h0, h1, tz);
        return height *
mapGen.heightMultiplier;
    }
    public void
MarkObjectAsTaken(string
objectPosition)
    {
        try
        {

brokenObjectPositions.Add(obje
ctPosition);

Debug.Log(objectPosition);
        }
        catch
(System.Exception e)
        {

Debug.LogError($"Error in
ObjectSpawner.MarkObjectAsTake
n(): {e.Message}");
        }
    }
    public void
SetObjectTaken(HashSet<string>
killedEnemies)
    {
        brokenObjectPositions
= new
```

```
HashSet<string>(killedEnemies)
;
    }

    public HashSet<string>
GetObjectTaken()
    {
        return new
HashSet<string>(brokenObjectPo
sitions);
    }
}
```

## EnemySpawner.cs

```
using UnityEngine;
using
System.Collections.Generic;
using UnityEngine.UI;
using static MapGenerator;

[System.Serializable]
public class EnemySpawnRule
{
    public GameObject
enemyPrefab;
    public float spawnChance;
    public float
clusterFrequency;
    public float
clusterThreshold;
    public List<Biome> biomes;
    public int maxPerCluster =
3;
}
public class EnemySpawner :
MonoBehaviour
{
    public static EnemySpawner
instance;

    [Header("Enemy Spawning")]
    public
List<EnemySpawnRule>
enemySpawnRules;
    private Dictionary<Biome,
List<EnemySpawnRule>>
biomeRules;
    private HashSet<Vector3>
takenEnemyPositions = new
HashSet<Vector3>();
    public ObjectPool
objectPool;

    [Header("Terrain
Settings")]
    public float scale = 50f;
    public float stepHeight =
30f;
    [Header("Raycast
Settings")]
    public LayerMask
terrainLayerMask;
    public LayerMask
dontPlaceLayerMask;
    public MapGenerator
mapGen;
    private HashSet<string>
deadEnemies = new
HashSet<string>();
    private void Awake()
    {
        if (instance == null)
instance = this;
        else
Destroy(gameObject);
        terrainLayerMask =
LayerMask.GetMask("Ground",
"Terrain");
        dontPlaceLayerMask =
LayerMask.GetMask("Water",
"Interactable", "Resource");
    }
    public void Initialize()
    {
        biomeRules = new
Dictionary<Biome,
List<EnemySpawnRule>>();
        foreach (var rule in
enemySpawnRules)
        {
            if (rule == null
|| rule.biomes == null)
continue;
            foreach (var biome
in rule.biomes)
            {
                if
(!biomeRules.ContainsKey(biome
))

biomeRules[biome] = new
List<EnemySpawnRule>();

biomeRules[biome].Add(rule);
            }
        }
        foreach (var rule in
enemySpawnRules)
        {
            if
(rule.enemyPrefab != null)
```

```
objectPool.CreatePoolObject(ru
le.enemyPrefab, 50);
        }

    }
    public void
SpawnEnemiesInChunk(Vector2Int
chunkCoord, GameObject parent,
int chunkSize)
    {
        if (mapGen == null ||
biomeRules == null || parent
== null) return;

Random.InitState(chunkCoord.x
* 73856093 ^ chunkCoord.y *
19349663);
        int startX =
chunkCoord.x * chunkSize;
        int startZ =
chunkCoord.y * chunkSize;
        int mapWidth =
mapGen.GetMapWidth();
        int mapHeight =
mapGen.GetMapHeight();
        var heightMap =
mapGen.GetHeightMap();
        var biomeMap =
mapGen.GetBiomeMap();
        for (int x = 0; x <
chunkSize; x++)
        {
            for (int z = 0; z
< chunkSize; z++)
            {
                int worldX =
startX + x;
                int worldZ =
startZ + z;
                if (worldX < 0
|| worldX >= mapWidth ||
worldZ < 0 || worldZ >=
mapHeight)
                    continue;
                Biome biome =
biomeMap[worldX, worldZ];
                if
(!biomeRules.ContainsKey(biome
)) continue;
                foreach (var
rule in biomeRules[biome])
                {
                    float
clusterNoise =
Mathf.PerlinNoise(worldX *
rule.clusterFrequency + 1000f,
worldZ * rule.clusterFrequency
+ 1000f);
                    if
(clusterNoise <
rule.clusterThreshold ||
Random.value >
rule.spawnChance) continue;
                    int
clusterRadius = 1;
                    float
multiplier =
GetSpawnMultiplierByDay();
                    int
baseMax = rule.maxPerCluster;
                    int
scaledMax =
Mathf.CeilToInt(baseMax *
multiplier);
                    int
targetSpawnCount =
Random.Range(1, Mathf.Max(2,
scaledMax + 1));
                    int
spawnedCount = 0;
                    for (int
dx = -clusterRadius; dx <=
clusterRadius; dx++)
                    {
                        for
(int dz = -clusterRadius; dz
<= clusterRadius; dz++)
                        {
                            if
(spawnedCount >=
targetSpawnCount) break;

int cx = worldX + dx;

int cz = worldZ + dz;
                            if
(cx < 0 || cx >= mapWidth ||
cz < 0 || cz >= mapHeight)
continue;
                            if
(biomeMap[cx, cz] != biome)
continue;

float jitterRange = scale *
0.4f;

float offsetX = Random.Range(-
jitterRange, jitterRange);

float offsetZ = Random.Range(-
jitterRange, jitterRange);
```

```
float posX = cx * scale +
offsetX;

float posZ = cz * scale +
offsetZ;

Vector3 rayOrigin = new
Vector3(posX, 1000f, posZ);

float rawHeight =
GetInterpolatedHeight(heightMa
p, posX, posZ);

float steppedHeight =
Mathf.Round(rawHeight /
stepHeight) * stepHeight;

float adjustedHeight =
Mathf.Lerp(rawHeight,
steppedHeight,
ChunkManager.instance.stairSmo
othness);

LayerMask combinedMask =
terrainLayerMask |
dontPlaceLayerMask;

Ray ray = new Ray(rayOrigin,
Vector3.down);
                            if
(Physics.Raycast(ray, out
RaycastHit hitInfo, 2000f,
combinedMask))
                            {

if (((1 <<
hitInfo.collider.gameObject.la
yer) & dontPlaceLayerMask) !=
0)

continue;

Vector3 pos = hitInfo.point;

string enemyIdentifier =
rule.enemyPrefab.name + "_" +
pos.x + "_" + pos.z;

if
(deadEnemies.Contains(enemyIde
ntifier)) continue;

if
(takenEnemyPositions.Contains(
pos)) continue;

Quaternion randomRot =
Quaternion.Euler(0,
Random.Range(0f, 360f), 0);

GameObject enemy =
objectPool.GetObjectFromPool(r
ule.enemyPrefab);

if (enemy != null)

{

enemy.transform.position =
pos;

enemy.transform.rotation =
randomRot;

enemy.transform.SetParent(pare
nt.transform);

}

if (enemy != null)

{

enemy.name = enemyIdentifier;

takenEnemyPositions.Add(pos);

spawnedCount++;

}
                                    }
                                }
                                if
(spawnedCount >=
rule.maxPerCluster) break;
                            }

                        }
                    }
                }
            }

    public void
MarkEnemyAsDead(string
enemyIdentifier)
    {
        if
(!string.IsNullOrEmpty(enemyId
entifier))
```

```
deadEnemies.Add(enemyIdentifie
r);
    }
    public void
SetKilledEnemy(HashSet<string>
killedEnemies)
    {
        deadEnemies = new
HashSet<string>(killedEnemies)
;
    }
    public HashSet<string>
GetKilledEnemy()
    {
        return new
HashSet<string>(deadEnemies);
    }
    float
GetInterpolatedHeight(float[,]
heightMap, float worldX, float
worldZ)
    {
        float scaledX = worldX
/ scale;
        float scaledZ = worldZ
/ scale;
        int mapWidth =
heightMap.GetLength(0);
        int mapHeight =
heightMap.GetLength(1);
        int x0 =
Mathf.FloorToInt(scaledX);
        int z0 =
Mathf.FloorToInt(scaledZ);
        int x1 =
Mathf.Clamp(x0 + 1, 0,
mapWidth - 1);
        int z1 =
Mathf.Clamp(z0 + 1, 0,
mapHeight - 1);
        if (x0 < 0 || z0 < 0
|| x0 >= mapWidth || z0 >=
mapHeight)
        {
            return 0f;
        }

        float h00 =
heightMap[x0, z0];
        float h10 =
heightMap[x1, z0];
        float h01 =
heightMap[x0, z1];
        float h11 =
heightMap[x1, z1];
        float tx = scaledX -
x0;
        float tz = scaledZ -
z0;
        float h0 =
Mathf.Lerp(h00, h10, tx);
        float h1 =
Mathf.Lerp(h01, h11, tx);
        float height =
Mathf.Lerp(h0, h1, tz);
        return height *
mapGen.heightMultiplier;
    }
    private float
GetSpawnMultiplierByDay()
    {
        int difficulty =
PlayerPrefs.GetInt("Difficulty
");
        int currentDay =
Mathf.Max(ScoreManager.Days,
1);
        float dayMultiplier;
        if (currentDay <= 5)
        {
            dayMultiplier =
Mathf.Lerp(0.25f, 1f,
(currentDay - 1) / 4f);
        }
        else if (currentDay <=
10)
        {
            dayMultiplier =
Mathf.Lerp(1f, 1.5f,
(currentDay - 5) / 5f);
        }
        else
        {
            dayMultiplier =
1.5f;
        }
        float
difficultyMultiplier =
(difficulty == 1) ? 1.5f : 1f;
        return dayMultiplier *
difficultyMultiplier;
    }
}
```