Schlümmelberger

Von Schlümmel für Schlümmel

Stefanie Schljak, Christian Schljak, Sabrina Dev

Gliederung

1. Demovideo
2. Verantwortung
3. Anhang

Demovideo

https://www.youtube.com/watch?v=uZfsrHujkaY

Verantwortung

Spieler Steuerung

```
    //Wenn der Spieler nicht zurück gestoßen wird
    if (rueckstoßZaehler <= 0)
    {
        //Speichert den Wert von bewegungsRichtung auf der Y Achse
        float yStore = bewegungsRichtung.y;

        //Lässt den Spieler in beide Richtungen gleichzeitig laufen transform.forward * Input.GetAxis("Vertical")) +
        bewegungsRichtung = (transform.forward * Input.GetAxis("Vertical") + transform.right * Input.GetAxis("Horizontal"));

        //Verhindert das man schneller wird wenn man beide Knöpfe gleichzeitig drückt
        bewegungsRichtung = bewegungsRichtung.normalized * bewegungsGeschwindigkeit;

        //Setzt den Wert wieder zurück
        bewegungsRichtung.y = yStore;

        //Überprüft ob der Spieler auf dem Boden ist
        if (controller.isGrounded)
        {
            springen = false;
            //Zurück setung des Y Wertes. Sonst komisch wenn man von Kanten fällt wegen der Gravitation
            bewegungsRichtung.y = 0f;

            //Überprüft ob die Sprung Taste gedrückt wird
            if (Input.GetButtonDown("Jump"))
            {
              // Der Y Wert bei der bewegungsRichtung bekommt die Sprungkraft zugewiesen
              bewegungsRichtung.y = sprungKraft;
              sprung.Play();
            }
        }
    }
    else
    {
        //Der Zähler wird verringert sich nach der Zeit
        rueckstoßZaehler -= Time.unscaledDeltaTime;
    }

        // Der Y Wert der bewegungsRichtung wird nochmal durch die Game Physics verändert
        bewegungsRichtung.y = bewegungsRichtung.y + (Physics.gravity.y * gravityScale * Time.unscaledDeltaTime);

        //Spieler bewegt sich wie es die bewegungsRichtung angibt mit abhängigkeit von der Zeit
        controller.Move(bewegungsRichtung * Time.unscaledDeltaTime);

        //Bewegt den Spieler in Verschiedene Richtungen basierend darauf in welche Richtung die Kamera schaut
        if (Input.GetAxis("Horizontal") != 0 || Input.GetAxis("Vertical") != 0)
        {
            transform.rotation = Quaternion.Euler(0f, pivot.transform.rotation.eulerAngles.y, 0f);
            Quaternion neueRotation = Quaternion.LookRotation(new Vector3(bewegungsRichtung.x, 0f, bewegungsRichtung.z));
            //Slerp für die Richtige Rotations bewegung
            spielerModel.transform.rotation = Quaternion.Slerp(spielerModel.transform.rotation, neueRotation, rotationsGeschwindigkeit * Time.unscaledDeltaTime);
        }

        //Setzt den Wert für den Animator auf true solange der Spiele am Boden ist
        anim.SetBool("AmBoden", controller.isGrounded);
        //Erhöht den Speed Wert für den Animator
        anim.SetFloat("Speed", (Mathf.Abs(Input.GetAxis("Vertical")) + Mathf.Abs(Input.GetAxis("Horizontal"))));

}
```

Leben

```
public void schadenSpieler(int schaden, Vector3 richtung)
{
    //Wenn man nicht unverwundbar ist
    if (unverwundbarkeitZaehler <= 0)
    {
        momLeben -= schaden;
        FindObjectOfType<SpieleManager>().hurtHealth(schaden);
        if (momLeben <= 0)
        {
            Respawn();
        }
        else
        {
            derSpieler.RÜckstoß(richtung);
            unverwundbarkeitZaehler = unverwundbarkeitsDauer;

            //Sorgt dafür das der Spieler nicht mehr sichtbar ist
            spielerRenderer.enabled = false;

            flashZaehler = flashDauer;
        }
    }

}

public void Respawn ()
{
    if(!istRespawnt)
    {
        StartCoroutine("RespawnCo");
    }

}

public IEnumerator RespawnCo()
{
    istRespawnt = true;
    derSpieler.gameObject.SetActive(false);
    Instantiate(respawnEffect, derSpieler.transform.position, derSpieler.transform.rotation);
    yield return new WaitForSeconds(respawnDauer);
    istRespawnt = false;
    derSpieler.gameObject.SetActive(true);
    derSpieler.transform.position = respawnPunkt;
    momLeben = maxLeben;
    FindObjectOfType<SpieleManager>().AddHealth(maxLeben);


    unverwundbarkeitZaehler = unverwundbarkeitsDauer;
    //Sorgt dafür das der Spieler nicht mehr sichtbar ist
    spielerRenderer.enabled = false;

    flashZaehler = flashDauer;
}

public void heileSpieler (int heilung)
{
    momLeben += heilung;

    //Verhindert das man mehr Leben als Maximal haben kann
    if(momLeben > maxLeben)
    {
        momLeben = maxLeben;
    }
```

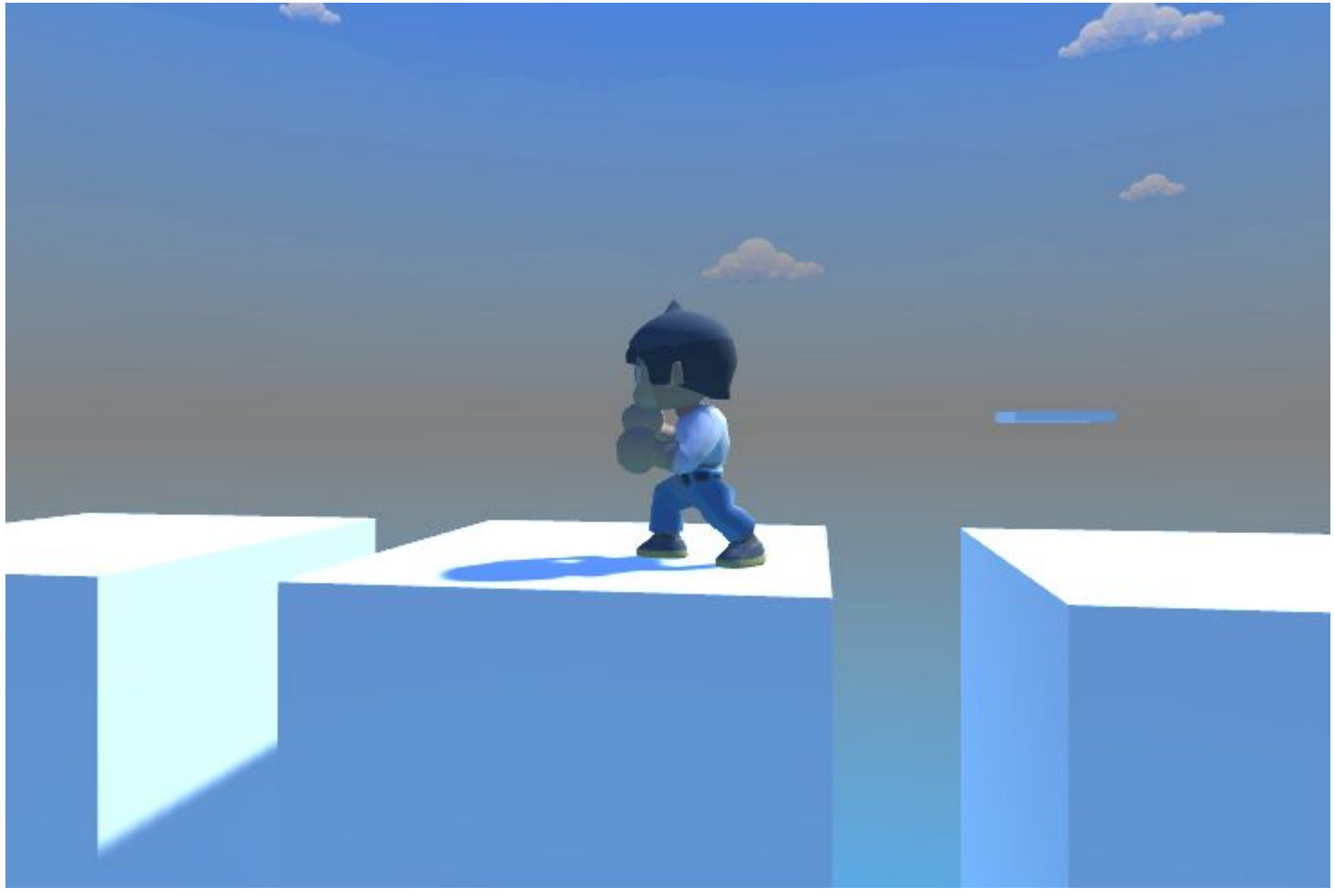
Kamera Steuerung

```
    pivot.transform.position = ziel.transform.position;

    //Verändert die Eltern Objekte nicht mit
    pivot.transform.parent = null;

    if (!useOffsetValues)
    {
        offset = ziel.transform.position - transform.position;
    }
}

// Damit es nach Spieler Update ausgeführt wird
void LateUpdate ()
{
    //Macht das der Pivot Punkt sicher immer mit dem Spieler bewegt
    pivot.transform.position = ziel.transform.position;

    //Holt die X Position der Maus und Rotiert den Pivot Punkt. Verhindert das der Spieler sich mit der Kamera dreht
    float horizontal = Input.GetAxis("Mouse X") * rotationsGeschwindigkeit;
    pivot.transform.Rotate(0, horizontal,0);

    //Y Position der Maus holen und den Pivot drehen
    float vertical = Input.GetAxis("Mouse Y") * rotationsGeschwindigkeit;
    //pivot.transform.Rotate(vertical, 0, 0);

    if(invertY)
    {
        pivot.transform.Rotate(vertical, 0, 0);
    }
    else
    {
        pivot.transform.Rotate(-vertical, 0, 0);
    }

    //Limit oben/unten für Kamera
    if(pivot.transform.rotation.eulerAngles.x > max && pivot.transform.rotation.eulerAngles.x < 180f)
    {
        pivot.transform.rotation = Quaternion.Euler(max, 0, 0);
    }

    if (pivot.transform.rotation.eulerAngles.x > 180 && pivot.transform.rotation.eulerAngles.x < 360f + min)
    {
        pivot.transform.rotation = Quaternion.Euler(360f + min, 0, 0);
    }
    //Bewegt die Kamera nach der momentanen Rotation des Pivots und des offsets
    float desiredYAngle = pivot.transform.eulerAngles.y;
    float desiredXAngle = pivot.transform.eulerAngles.x;

    Quaternion drehung = Quaternion.Euler(desiredXAngle, desiredYAngle, 0);
    transform.position = ziel.transform.position - (drehung * offset);

    //Verhinder das die Kamera durch den Boden geht
    if(transform.position.y < ziel.transform.position.y)
    {
        transform.position = new Vector3(transform.position.x, ziel.transform.position.y, transform.position.z);
    }
    //Schaut immer auf das Ziel
    transform.LookAt(ziel.transform);
```

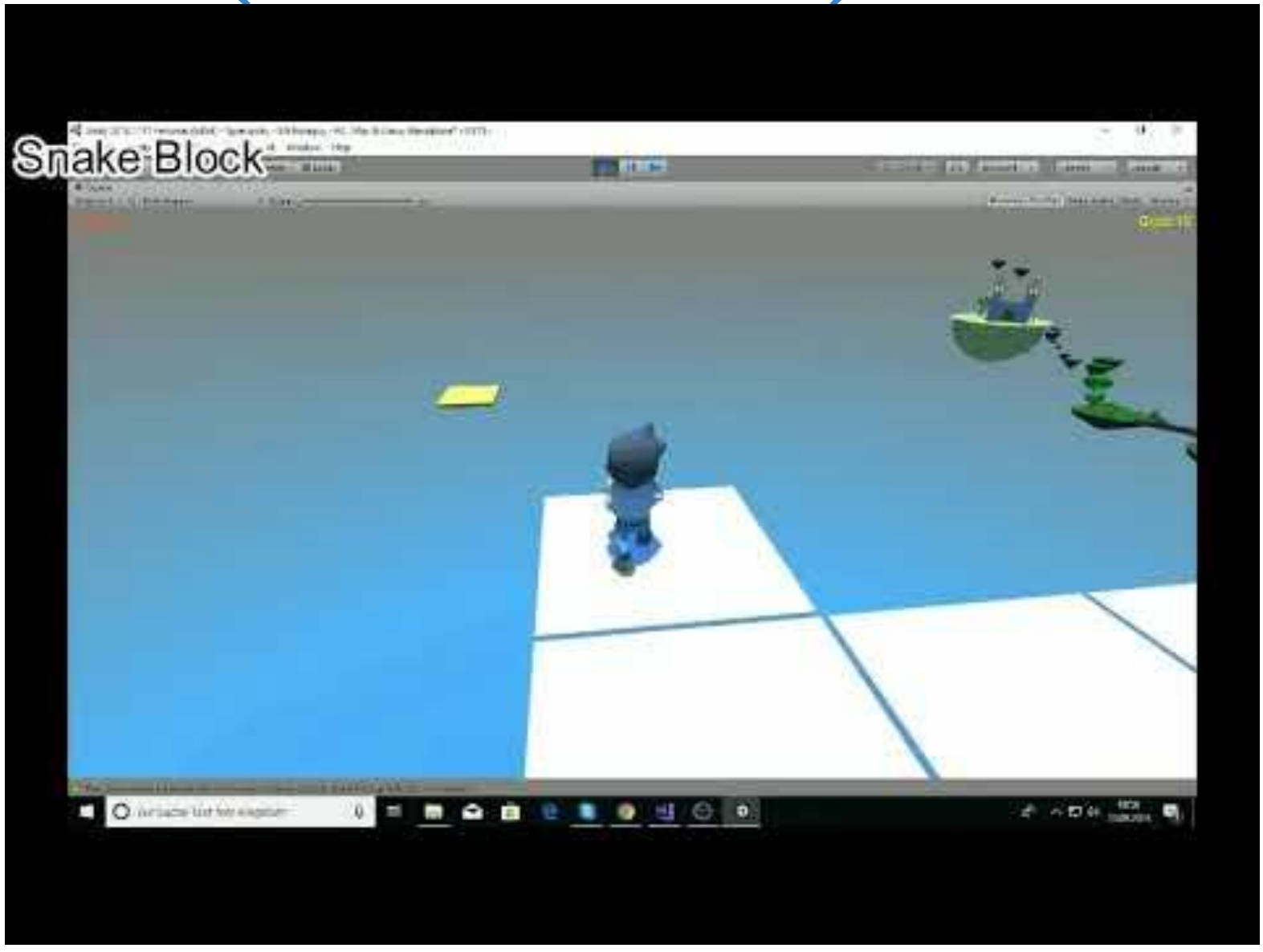
Snake (Generieren)

```
private IEnumerator GenerateTail()
{
    //Generiert nicht noch mehr.
    ready = false;
    //Snake größe hinzufügen.
    array = new GameObject[anzahl];

    //Wir starten bei 1, damit auf dem Kopf
    //kein zweiter Teil der Snake gebaut wird und der Kopf mit zieht.
    array[0] = this.gameObject;

    for (int zahl = 1; zahl < anzahl; zahl++)
    {
        //Die Generierung der Tails mit Cubes.
        GameObject hinten = GameObject.CreatePrimitive(PrimitiveType.Cube);

        //Die Größe der Snaketails festgelegt und die Position merken.
        hinten.transform.localScale = new Vector3(3.5f, 3.5f, 3.5f);
        hinten.transform.position = this.transform.position;

        // Als nächstes sollen die Tails nach vorne ausgerichtet sein und nicht aufeinander gestappelt sein.
        //Desshalb benutzen wir forward.
        //Damit diese nicht an einander geklebt sind nimmt man den Abstand mal.
        hinten.transform.Translate(this.transform.forward * space);
        array[zahl] = hinten;

        yield return new WaitForSeconds(Time.timeScale);
        // Damit die Tails sich nach einem Zeitabbstand bilden.
    }

    //Was war das letze Teil im Array derhinzugefügt wurde
    anzahlN = array.Length - 1;
    // Jetzt darf er sich bewegen.
    moveNow = true;
}
```

Snake(Generieren)



Snake(Generieren)



Snake(MoveToTarget)

```
//Richtungswechsel zum Target hin
int achse = (int)Random.Range(1, 4);
switch (achse)
{ case 1:
        if (this.transform.position.x < newTarget.x)
        {
            whichDirection = 3; // Rechts
        }else if (this.transform.position.x > newTarget.x)
        {
            whichDirection = 4;//Links
        }else {
            whichDirection = 1;//Vorwärts
        }
        break;
    case 2:
            if (this.transform.position.y < newTarget.y) {
                whichDirection = 2; //Hoch
            } else if (this.transform.position.y > newTarget.y)
            {
                whichDirection = 5;//Unten
            } else {
                whichDirection = 1;//Vorwärts
            }
            break;
        case 3:
        if (this.transform.position.z < newTarget.z){
                whichDirection = 1;//Vorwärts
        } else if (this.transform.position.z > newTarget.z)
            {
                whichDirection = 5;//Hinten
        } else {
                whichDirection = 3;//Rechts
            }
        break;
}
RotateSnake();
```

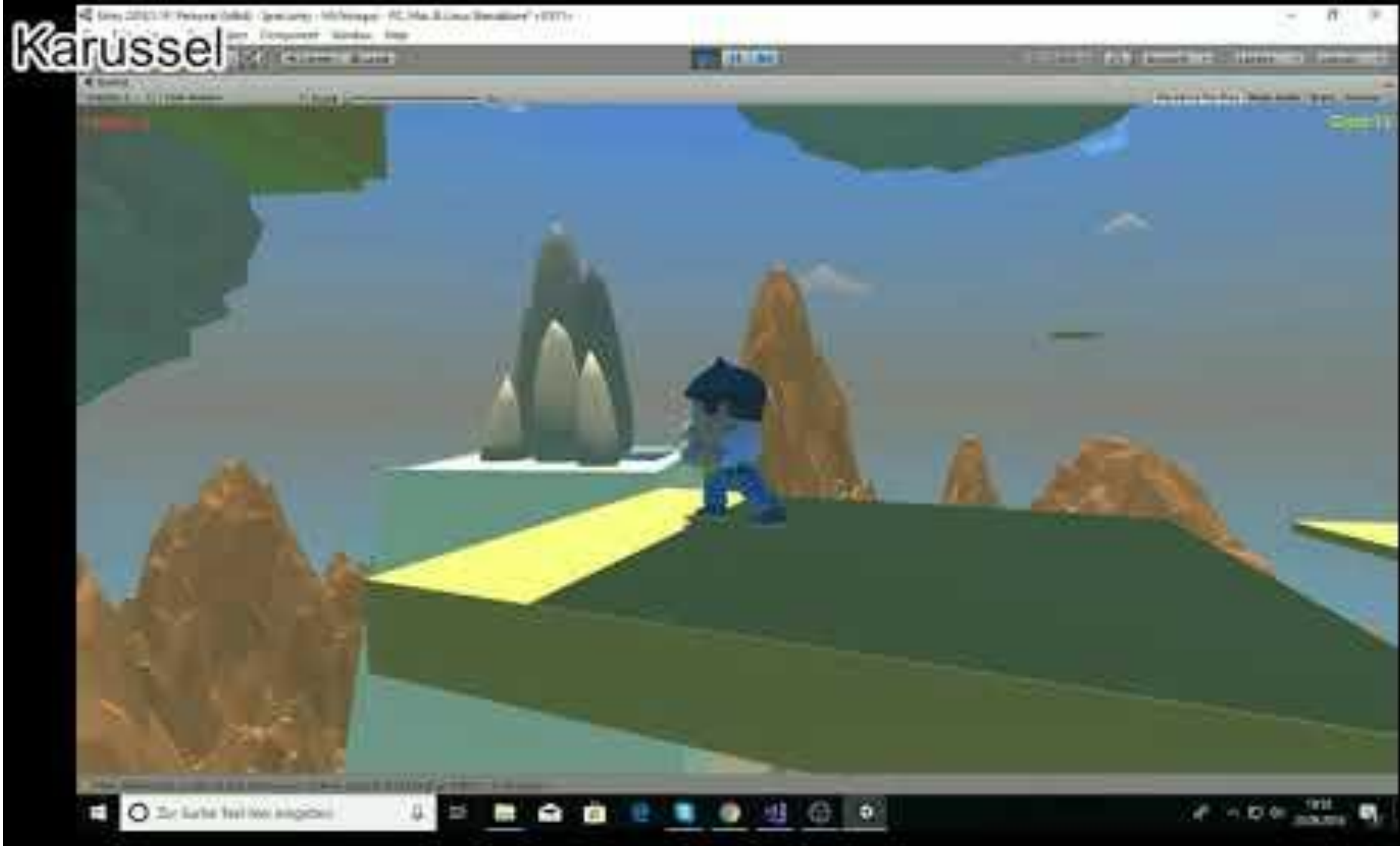
Enemy (Controller)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;


public class Enemycontroller : MonoBehaviour {

    [SerializeField]

    public float lookRadius = 10f;
    public Transform target;
    public NavMeshAgent agent;


    private void Start()
    {
        target = GameObject.Find("Spieler").transform;
    }
    // Use this for initialization
    void Update () {
        float distance = Vector3.Distance(target.position, transform.position);
        if (distance <= lookRadius)
        {
            agent = this.GetComponent<NavMeshAgent>();

            if (agent == null)
            {
                Debug.Log("Fehler" + gameObject.name);
            }
            else
            {
                SetDestination();
            }
        }
    }

    // Zeigt den Radius in dem er Reagiert
    private void SetDestination()
    {
        if(target != null)
        {
            Vector3 targetVector = target.transform.position;
            agent.SetDestination(targetVector);
        }
    }

    public void lookRadiusChange (float wert)
    {
        lookRadius = wert;
    }
}
```

Weltgenerierung & Terrain

```
    void Start () {
        StartCoroutine("SpawnWorld");
    }
    //Hier wird "SpawnWorld" definiert
    IEnumerator SpawnWorld()
    {
        groundWidth = trMarker.transform.position.x - blMarker.transform.position.x; // Die Breitenbreite ist der Abstand unser Markers

        worldObjectIncAmt = groundWidth / worldObjectRowsAndCols; //Distanz
        worldObjectRandAmt = worldObjectIncAmt / 2f; // wie weit Objekte auseinander liegen

        terrainIncAmt = groundWidth / terrainRowsAndCols; //Distanz
        terrainRandAmt = terrainIncAmt / 2f;  // wie weit Objekte auseinander liegen

        //Wie bewegen wir uns von der Startposition aus
        worldObjectsStartPos = new Vector3(blMarker.transform.position.x - (worldObjectIncAmt/2f), blMarker.transform.position.y, blMarker.transform.position.z + (worldObjectIncAmt/2f));
        terrainStartPos = new Vector3(blMarker.transform.position.x -(terrainIncAmt /2f), blMarker.transform.position.y,blMarker.transform.position.z+(terrainIncAmt / 2f));

        //Die Spawn-Schleife
        for (int rp = 0; rp <= repeatPasses; rp++)
        {
            currentPass = rp; //Variable die den aktuellen Durchlauf speichert

            if (currentPass == 0)
            {
                currentPos = terrainStartPos; //Die aktuelle Position soll der Startposition des Terains entsprechen

                //Durchlaufen Zeile und Spalte
                for (int rows = 1; rows <= terrainRowsAndCols; rows++)
                {
                    for (int cols = 1; cols <= terrainRowsAndCols; cols++)
                    {
                        // Die Aktuelle Position bekommt jedes mal ein Neuen Vektor desen X Position sich erhöht
                        currentPos = new Vector3(currentPos.x + terrainIncAmt, currentPos.y, currentPos.z);

                        //Das Objekt das gespawned werden soll
                        GameObject newSpawn = terrain[Random.Range(0, terrain.Length)]; // Random Objekt zwischen 0 und Terrain länge

                        // Sobald wir ein Objekt definiert haben soll es gespawnd werden
                        SpwanHere(currentPos, newSpawn, terrainSphereRad, true);//True bei einem Terrainobjekt

                        // Gibt Wartezeit zurück
                        // es soll ein wenig warten bevor er zum nächsten Objekt geht
                        yield return new WaitForSeconds(0.01f);
                    }
                    // Wenn wir am Ende des Gitters angekommen sind müssen wir es auf den Beginn zurück setzen und eine Gitterzeile weiter rücken
                    currentPos = new Vector3(terrainStartPos.x, currentPos.y, currentPos.z + terrainIncAmt);
                }
            }
            else if (currentPass > 0)
            {
                currentPos = worldObjectsStartPos; // Die aktuelle Position soll die Startposition der Worldobjekte bekommen

                //Durchlaufe Zeile und Spalten
                for (int rows = 1; rows <= worldObjectRowsAndCols; rows++)
                {
                    for (int cols = 1; cols <= worldObjectRowsAndCols; cols++)
                    {
                        // Die Aktuelle Position bekommt jedes mal ein Neuen Vektor desen X Position sich erhöht
                        currentPos = new Vector3(currentPos.x + worldObjectIncAmt, currentPos.y,currentPos.z);

                        //Damit Steine nicht so oft spawnen wie Bäume
                        //deswegen eine Varianble die die Reichweite zwischen 1 und 5 festgelegt ist
                        int SpawnChance = Random.Range(1, stoneChanceAmt +1);

                        if (SpawnChance == 1)
                        {
                            // Sollten wie eine SpawnChance bekommen machen wir Sie zu einem GameObject (eine Stein)
                            GameObject newSpawn = stones[Random.Range(0, stones.Length)];
                            SpwanHere(currentPos, newSpawn, worldObjectSphereRad, false);//False bei einem Weltobjekt

                            // Gibt Wartezeit zurück
                            // es soll ein wenig warten bevor er zum nächsten Objekt geht
                            yield return new WaitForSeconds(0.01f);
                        }
                        else
                        {
                            // ansonsten sollen Bäume gespawnd werden

                            GameObject newSpawn = trees[Random.Range(0, trees.Length)];

                            SpwanHere(currentPos, newSpawn, worldObjectSphereRad, false);//False bei einem Weltobjekt

                            // Gibt Wartezeit zurück
                            // es soll ein wenig warten bevor er zum nächsten Objekt geht
                            yield return new WaitForSeconds(0.01f);
                        }
                    }
                    // Wenn wir am Ende des Gitters angekommen sind müssen wir es auf den Beginn zurück setzen und eine Gitterzeile weiter rücken
                    currentPos = new Vector3(worldObjectsStartPos.x,currentPos.y, currentPos.z + worldObjectIncAmt);
                }
            }
        }
        // Soll eine Nachticht ausgeben wenn sie fertig ist
        WorldGenDone();
    }
```

```
void SpwanHere(Vector3 newSpawnPos, GameObject objectToSpawn, float radiusOfSphere, bool isObjectTerrain)
{
    //Wenn es ein Terrainobjekt ist dann...
    if (isObjectTerrain == true)
    {
        //eine Variable mit Random Punktebereichen aber darauf aufpasst das sie noch in Randbereich liegen (ob wie noch im Gitter uns befinden)
        Vector3 randPos = new Vector3(newSpawnPos.x + Random.Range(-terrainRandAmt, terrainRandAmt + 1), 0, newSpawnPos.z + Random.Range(-terrainRandAmt, terrainRandAmt + 1));

        //Wir werfen ein Raycast um zu sehen ob unten oder oben Boden ist
        //Variable für Raycast ob es hoch oder runter gehen soll
        Vector3 rayPos = new Vector3(randPos.x, 10, randPos.z);

        if (Physics.Raycast(rayPos, -Vector3.up, Mathf.Infinity, groundLayer))//Es soll nach unten gehen(also ist unter uns Boden)
        {
            // es soll nur kollidieren wen wir überhalb vom Boden sind also wenn wir über dem Rand der Welt sind
            //Es könnten mehrere Objekte sein die wir treffen könnten deswegen ein Array
            //Wir werfen ein Radius von einer Kugel zum überprüfen von Überlappungen
            Collider[] objectsHit = Physics.OverlapSphere(randPos, radiusOfSphere, terrainLayer);

            if (objectsHit.Length == 0)// wenn das Array leer sein sollte (also wenn wir nichts getroffen haben)
            {
                // wir erschaffen ein neues GameObjekt, instanziieren Objekte zum spawnen und referenzieren es als Terrainobjekt
                GameObject terrainObject = (GameObject)Instantiate(objectToSpawn, randPos, Quaternion.identity);
                //Random die Objekte routieren lassen
                terrainObject.transform.eulerAngles = new Vector3(transform.eulerAngles.x, Random.Range(0, 360), transform.eulerAngles.z);
            }
        }
    }
    else
    {
        Vector3 randPos = new Vector3(newSpawnPos.x + Random.Range(-worldObjectRandAmt, worldObjectRandAmt + 1), newSpawnPos.y, newSpawnPos.z + Random.Range(-worldObjectRandAmt, worldObjectRandAmt + 1));
        Vector3 rayPos = new Vector3(randPos.x, 20, randPos.z);

        RaycastHit hit;

        if (Physics.Raycast(rayPos, -Vector3.up, out hit, Mathf.Infinity, groundLayer))
        {
            randPos = new Vector3(randPos.x, hit.point.y, randPos.z);

            Collider[] objectsHit = Physics.OverlapSphere(randPos, radiusOfSphere, worldObjectsLayer);//Ueberprüft Kollisionen

            if (objectsHit.Length == 0)
            {
                // wir erschaffen ein neues GameObjekt, instanziieren Objekte zum spawnen und referenzieren es als Worldobjekt
                GameObject worldObject = (GameObject)Instantiate(objectToSpawn, randPos, Quaternion.identity);
                // Worldobjekte ein wenig ausrichten
                worldObject.transform.position = new Vector3(worldObject.transform.position.x, worldObject.transform.position.y + (worldObject.GetComponent<Renderer>().bounds.extents.y*0.7f),
                worldObject.transform.position.z);
                //Random die Objekte routieren lassen
                worldObject.transform.eulerAngles = new Vector3(transform.eulerAngles.x, Random.Range(0, 360), transform.eulerAngles.z);
            }
        }
    }
}
//Als Feedback um zu wissen wann die Welt sich fertig generiert hat
void WorldGenDone()
{ print("Welt ist generiert!"); }
```


Karussel

Terrain

Toolbox

InfiniteGenerate.cs | TerrainGeneratorScript.cs | Pyramide.cs | TerrainGenerator.cs

Assembly-CSharp | InfiniteGenerate

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   class Tile
6   {
7       public GameObject theTile;
8       public float creationTime;
9
10      public Tile(GameObject t, float ct)
11      {
12          theTile = t;
13          creationTime = ct;
14      }
15  }
16
17  public class InfiniteGenerate : MonoBehaviour
18  {
19
20      public GameObject plane;
21      public GameObject player;
22
23      int planeSize = 10;
24      int halfTilesX = 5;
25      int halfTilesZ = 5;
26
27      Vector3 startPos;
28
29      Hashtable tiles = new Hashtable();
30
31      // Use this for initialization
32      void Start()
33      {
34          this.gameObject.transform.position = Vector3.zero;
35          startPos = Vector3.zero;
36
37          float updateTime = Time.realtimeSinceStartup;
38
39          for (int x = -halfTilesX; x < halfTilesX; x++)
40          {
41              for (int z = -halfTilesZ; z < halfTilesZ; z++)
42              {
43                  Vector3 pos = new Vector3((x * planeSize + startPos.x), 0, (z * planeSize + startPos.z));
44                  GameObject t = (GameObject)Instantiate(plane, pos, Quaternion.identity);
45
46                  string tilename = "Tile_" + ((int)(pos.x)).ToString() + "_" + ((int)(pos.z)).ToString();
47                  t.name = tilename;
48                  Tile tile = new Tile(t, updateTime);
49                  tiles.Add(tilename, tile);
50              }
```

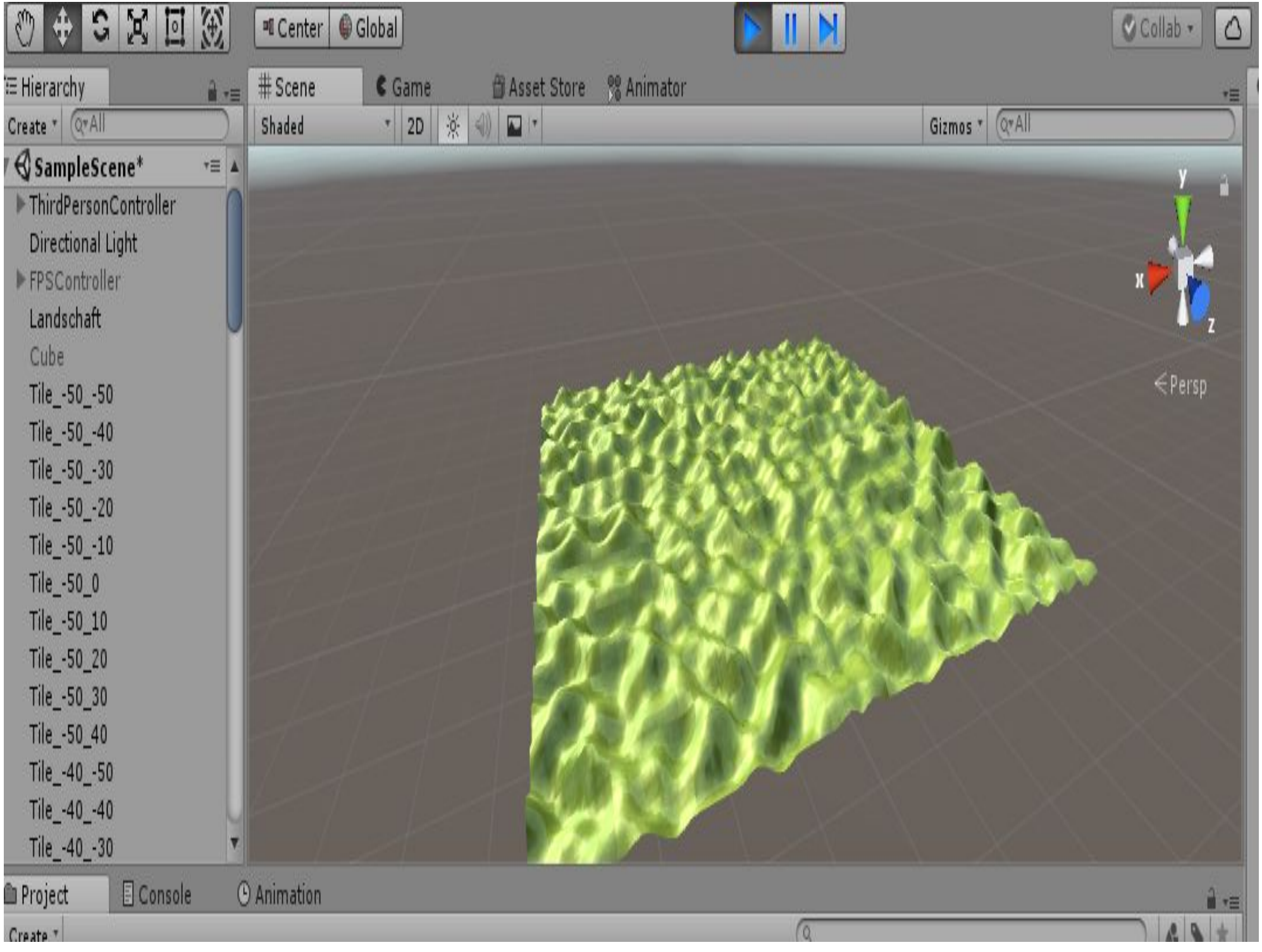
InfiniteGenerate.cs | Pyramide.cs | TerrainGenerator.cs

Assembly-CSharp | TerrainGenerator | Start()

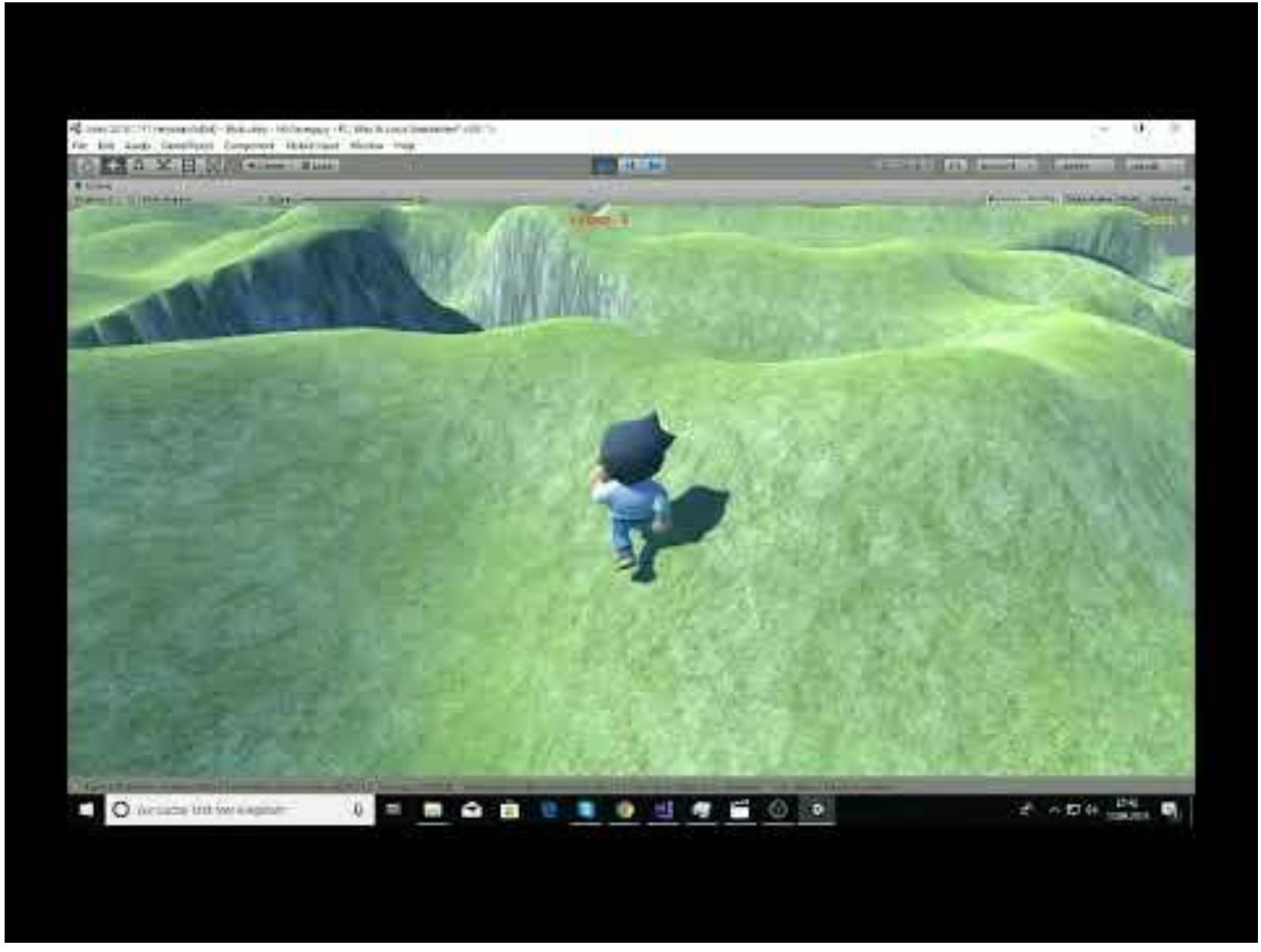
```
4
5   public class TerrainGenerator : MonoBehaviour {
6
7       int heightScale = 5;
8       float detailScale = 5.0f;
9       //List<GameObject> myTrees = new List<GameObject>();
10      //public GameObject tree;
11      // Use this for initialization
12      void Start () {
13          Mesh mesh = this.GetComponent<MeshFilter>().mesh;
14          Vector3[] vertices = mesh.vertices;
15          for (int v = 0; v < vertices.Length; v++)
16          {
17              vertices[v].y = Mathf.PerlinNoise((vertices[v].x + this.transform.position.x) / detailScale, (vertices[v].z + this.transform.position.z) / detailScale) * heightScale;
18
19                if (vertices[v].y > 2.4 && Random.Range(0,100)<10)
20                {
21                    Vector3 treePos = new Vector3(vertices[v].x + this.transform.position.x, vertices[v].y, vertices[v].z + this.transform.position.z);
22                    //Instantiate(tree, treePos, Quaternion.identity);
23                }
24
25          }
26
27          mesh.vertices = vertices;
28          mesh.RecalculateBounds();
29          mesh.RecalculateNormals();
30          this.gameObject.AddComponent<MeshCollider>();
31      }
32
33
34      // Update is called once per frame
35      void Update () {
36
37      }
38  }
```

60 %

Ausgabe


Center
Global
Collab
Hierarchy
Scene
Game
Asset Store
Animator
Create
All
Shaded
2D
Gizmos
All
SampleScene*
ThirdPersonController
Directional Light
FPSController
Landschaft
Cube
Tile_-50_-50
Tile_-50_-40
Tile_-50_-30
Tile_-50_-20
Tile_-50_-10
Tile_-50_0
Tile_-50_10
Tile_-50_20
Tile_-50_30
Tile_-50_40
Tile_-40_-50
Tile_-40_-40
Tile_-40_-30
y
x
z
Persp
Project
Console
Animation
Create

Slope

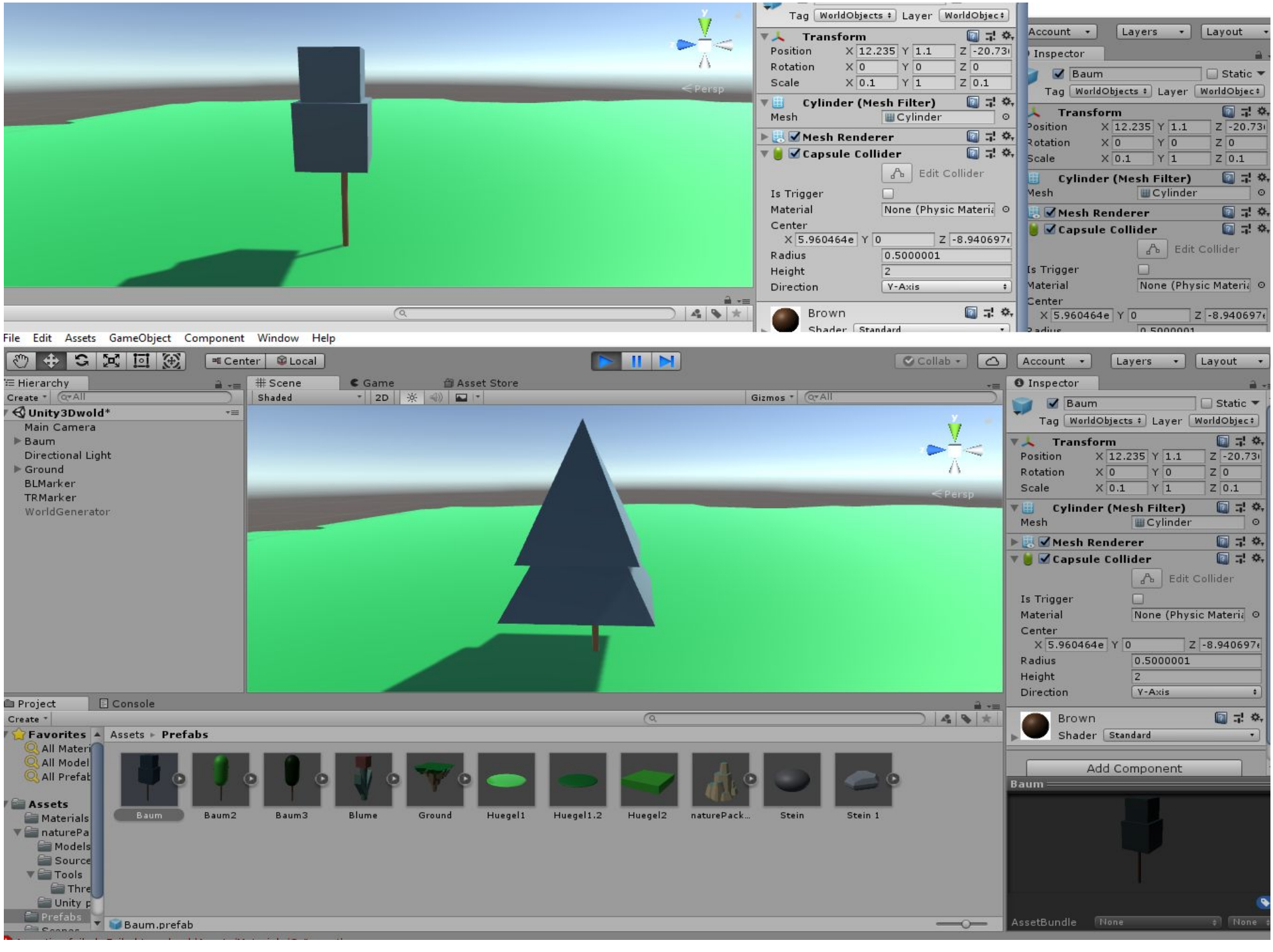


Objektverformung

Pyramide.cs | TerrainGeneratorScript.cs* | Terra

Sonstige Dateien

```
public class Pyramide : MonoBehaviour {

    public Vector3 vertLeftFront = new Vector3(-1,1,1);
    public Vector3 vertRightFront = new Vector3(1,1,1);
    public Vector3 vertRightBack = new Vector3(1,1,-1);
    public Vector3 vertLeftBack = new Vector3(-1,1,-1);

    private readonly float waitN = 3f;
    private readonly float waitD = 3f;
    public int shapeN = 0;

    // Use this for initialization
    void Start () {
        MeshFilter mf = GetComponent<MeshFilter>();
        Mesh mesh = mf.mesh;

        //Vertices
        Vector3[] vertices = new Vector3[]
        {
            //Vorderseite
            vertLeftFront, //linke vordere Front,0
            vertRightFront,//rechte vordere Front,1
            new Vector3(-1,-1,1),//linke hintere front,2
            new Vector3(1,-1,1),//rechte hintere front,3

            //Rückseite
             vertRightBack,
             vertLeftBack,
             new Vector3(1,-1,-1),
             new Vector3(-1,-1,-1),

             //Linkeseite
             vertLeftBack,
             vertLeftFront,
             new Vector3(-1,-1,-1),
             new Vector3(-1,-1,1),

             //Rechteseite
             vertRightFront,
             vertRightBack,
             new Vector3(1,-1,1),
             new Vector3(1,-1,-1),

             //Obereseite (Deckel)
             vertLeftBack,
             vertRightBack,
             vertLeftFront,
             vertRightFront,
             //Untereseite (Grund)
             new Vector3(-1,-1,1),
             new Vector3(1,-1,1),
             new Vector3(-1,-1,-1),
             new Vector3(1,-1,-1),
        };

        // Dreieck
        int[] triangles = new int[]
         {
             //Vorderseite
             0,2,3, //erstes triangles
             3,1,0, //zweites triangles

              //Rückseite
             4,6,7, //drittes triangles
             7,5,4, //viertes triangles

              //Linkeseite
             8,10,11, //fünftes triangles
             11,9,8, //sechstes triangles

              //Rechteseite
             12,14,15, //siebtes triangles
             15,13,12, //achtes triangles

              //Obereseite (Deckel)
             16,18,19, //neuntes triangles
             19,17,16, //zehntes triangles

              //Untereseite (Grund)
             20,22,23, //elftes triangles
             23,21,20 //zwelftes triangles
         };

        //UVs
        Vector2[] uvs = new Vector2[]
        {
            //Vorderseite
            new Vector2(0,1),
            new Vector2(0,0),
            new Vector2(1,1),
            new Vector2(1,0),

            //Rückseite
            new Vector2(0,1),
            new Vector2(0,0),
            new Vector2(1,1),
            new Vector2(1,0),

            //Linkeseite
            new Vector2(0,1),
            new Vector2(0,0),
            new Vector2(1,1),
            new Vector2(1,0),

            //Rechteseite
            new Vector2(0,1),
            new Vector2(0,0),
            new Vector2(1,1),
            new Vector2(1,0),

            //Obereseite (Deckel)
            new Vector2(0,1),
            new Vector2(0,0),
            new Vector2(1,1),
            new Vector2(1,0),

            //Untereseite (Grund)
            new Vector2(0,1),
            new Vector2(0,0),
            new Vector2(1,1),
            new Vector2(1,0)
        };

        mesh.Clear();
        mesh.vertices = vertices;
        mesh.triangles = triangles;
        mesh.uv = uvs;
        MeshUtility.Optimize(mesh);
        mesh.RecalculateNormals();
    }

    // Update is called once per frame
    void Update () {

    }
}
```


File Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Layout
Hierarchy
Create All
Unity3Dwold*
Main Camera
Baum
Directional Light
Ground
BLMarker
TRMarker
WorldGenerator
Scene Game Asset Store
Shaded 2D Gizmos All
Persp
Inspector
Baum Static
Tag WorldObjects Layer WorldObject
Transform
Position X 12.235 Y 1.1 Z -20.73
Rotation X 0 Y 0 Z 0
Scale X 0.1 Y 1 Z 0.1
Cylinder (Mesh Filter)
Mesh Cylinder
Mesh Renderer
Capsule Collider
Edit Collider
Is Trigger
Material None (Physic Materia
Center X 5.960464e Y 0 Z -8.940697
Radius 0.5000001
Height 2
Direction Y-Axis
Brown
Shader Standard
Add Component
Project Console
Create
Favorites
All Materi
All Model
All Prefab
Assets
Materials
naturePa
Models
Source
Tools
Thre
Unity p
Prefabs
Assets ▸ Prefabs
Baum Baum2 Baum3 Blume Ground Huegel1 Huegel1.2 Huegel2 naturePack... Stein Stein 1
Baum.prefab
AssetBundle None None

Anhang

Terminplan Alt

Zeiten	Alle	Christian	Stefanie	Sabrina
25.06-01.07		Spieler Steuerung,Springen	Objekt Generierung, Objekte verschieben	Level Generierung
02.07-08.07		kamerasteuerung,item aufheben	Feinde generieren (Ki die beim zerstören Items liegen lassen)	Feinde generieren (Ki die auf den Spieler reagieren)
09.07-15.07		Leben, Schaden , Rückstoß, Items einsammeln	-	-
16.07-22.07		dreifach Sprung	-	-
23.07-29.07		Wandsprung	Plattform Rutschen (Physik, dh bei einer gewissen Neigung der Plattform rutscht der Spieler runter)	snake block(wie bei Mario)
30.07-05.08		-	Respawn, Checkpoints	Verwundbarkeit (beim Schaden nehmen 1-3 Sekunden unverwundbar)

Terminplan Neu

Zeiten	Christian	Stefanie	Sabrina
25.06-01.07	Spieler Steuerung,Springen 9 Stunden	Checkpoints, Töterbereich, Respawn 2 Stunden	Enemy Controller (auf Spieler reagieren) 5 Stunden
02.07-08.07	kamerasteuerung,item aufheben 8 Stunden	Level Generierung (fehlerhaft) 9 Stunden	KI Logik (auf Spieler reagieren) 5 Stunden
09.07-15.07	Leben, Schaden , Rückstoß, Items einsammeln 4 Stunden	-	-
16.07-22.07	Dreifach Sprung (Funktioniert nicht) 4 Stunden Wandsprung 1 Stunden	-	-
23.07-29.07	Wandsprung 2 Stunden	Slope Controller (Fehlerhaft auf den Spieler da, die Gravitation den Spieler blockiert hat) 15 Stunden	Verwundbarkeit (beim Schaden nehmen 1-3 Sekunden unverwundbar) 2 Stunden
30.07-05.08	-	Level Generierung (3 Versionen) 5 Stunden	Snakeblock(wie bei Mario) 5 Stunden

Terminplan Alt

Zeiten	Alle	Christian	Stefanie	Sabrina
06.08-12.08		-	Fallschaden (Physik)	**Fallende Platform,** Bewegende-Plattform
13.08-19.08		Zeitmanipulation	Rotierende Plattformen	Sprung Platten,Spikes
20.08-26.08		Perspektiven Wechseln Z.B Egoperspektive	schwimmen, Tauchen	Menüs (Start, Pause, Beenden)
27.08-02.09		kämpfen	-	klettern, Festhalten
03.09-09.09	Soundeffekte bei Interaktion mit Objekten			
10.09-16.09		-	-	-
17.09-23.09	kleine Feinarbeiten Z.b flüßigere Bewegungen			

Terminplan Neu

Zeiten	Alle	Christian	Stefanie	Sabrina
06.08-12.08		Feind ausgebessert (Kanone neuer Gegener) (Schweinchen soll Charakter angreifen und nicht weg laufen) 6 Stunden	Objekt Generierung, Feind generierung(Schwein) 4 Stunden	Snakeblock(wie bei Mario) 15 Stunden
13.08-19.08		Zeitmanipulation 2Stunden	-	-
20.08-26.08		Lebens- und Goldanzeige 1 Stunden	Rotierende Plattform (fehlerhaft, Spieler fällt nicht runter wenn er von den Plattformen gedrückt wird) 2 Stunden schwimmen (fehlerhaft) 2 Stunden	Fallende Plattform, Bewegende Plattform 4 Stunden

Terminplan Neu

Zeiten	Alle	Christian	Stefanie	Sabrina
27.08-02.09		kämpfen 4 Stunden	Schwimmen & tauchen (funktioniert Leider nicht auf unseren Charakter) 5 Stunden	Spikes 3 Stunden
03.09-09.09		-	Main Menü (Titelbild gestaltet und Funktionalität der Buttons) 2 Stunden	Snakeblock(wie bei Mario) 5 Stunden
10.09-16.09	Soundeffekte bei Interaktion mit Objekten 5 Stunden			
17.09-23.09	kleine Feinarbeiten Z.b flüßigere Bewegungen 10 Stunden			
Aufwand in Stunden(ins gesam.):		**56 Stunden**	**61 Stunden**	**64 Stunden**

Vielen Dank für ihre Aufmerksamkeit.

Gruppe Schlümmelberger:

Stefanie Schljak, Christian Schljak, Sabrina Dev