Eigenes Minispiel in Unity 3D erstellen

Thierry Bratschi
OSS AESCHI

Inhalt

Vorwort

Unity ist eine kostenlos nutzbare Umgebung um Spiele und Applikationen zu entwickeln. Es bietet alles für Anfänger aber auch für Profis. Viele beliebte Games wie Pokemon Go, Among us aber auch graphisch anspruchsvollere Spiele wurden in Unity entwickelt.

Für komplette Anfänger ist die Software sehr umfassend und es muss erst viel Wissen aufgebaut werden, deshalb soll diese Anleitung die Ersten Schritte erleichtern.

Die Anleitung wird dir helfen eine einfache Spielumgebung mit einem Player in der «First Person View» zu erschaffen, den du mit «wasd» und der Maus steuern kannst. Weiter schauen wir an, wie du 3D Objekte importieren und als physikalische Körper im Spielfeld einbauen kannst.

Die benötigten Scripts lassen wir uns mit ChatGPT schreiben. Hierzu wirst du auch einige Tipps erhalten. Zum bearbeiten der Codes in Unity solltest du ein Coding-Programm haben (ich verwende z.B. Visual Studio oder Geany, es gehen aber auch viele andere, teils evtl. bessere Programme).

Für die Einzelnen Arbeitsschritte sollten dir auch die Videos im Teamsordner helfen.

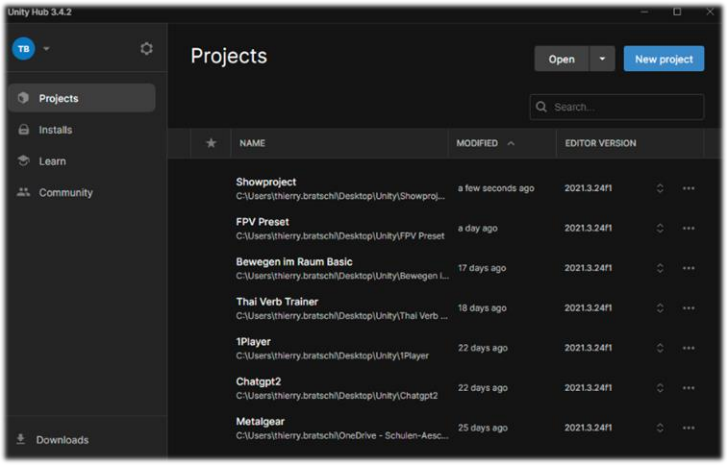
Gehe die Anleitung Schritt für Schritt durch und halte dich genau daran, verstehst du den Text nicht, schaue erst die Videos an, bevor du nachfragtst.

Installation von Unity

Als erstes musst du die Unity Homepage aufrufen und dir das Programm herunterladen und installieren (https://unity.com/de/download). Dies kann eine ganze Weile dauern. Die exe Datei findest du auch in unserem Teams-Ordner «Programme» und kannst sie direkt von dort aus installieren.

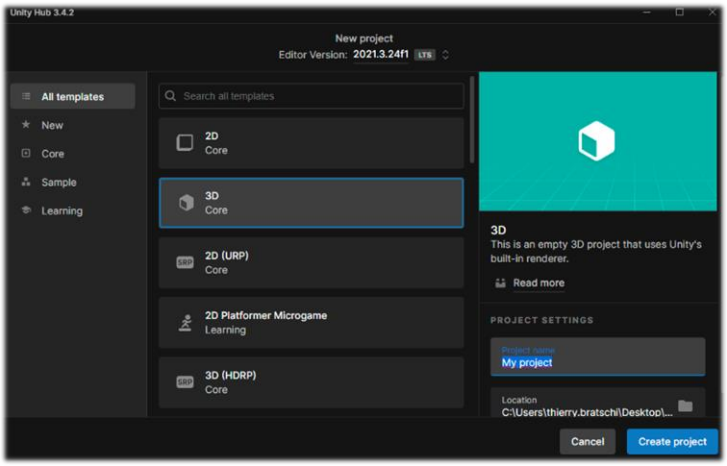
Lies die Installationsschritte, bevor du weiter klickst! Erstelle ein Desktopsymbol.

Ist das Programm auf dem Computer Installiert öffnest du die Unity Engine. Unter «Neues Projekt - **All Templates**» findest du alle möglichen Projektoptionen Wähle 3D Core aus und erstelle ein neues Projekt und benenne es. Nun fällt wieder eine mehrminütige Ladezeit an (Stand 2023).



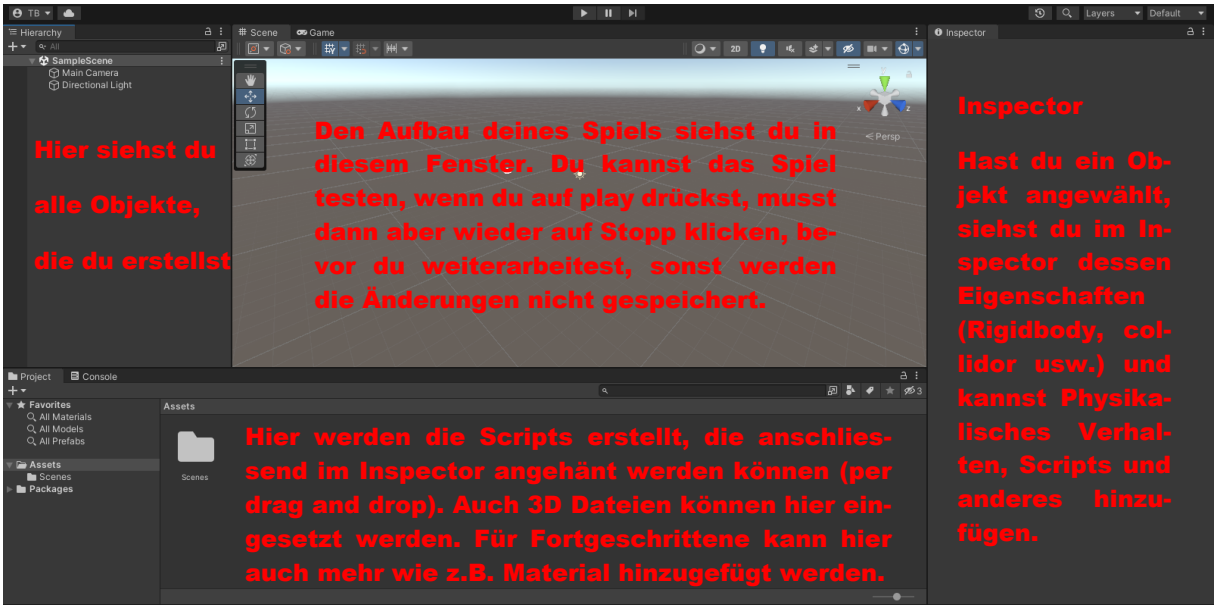
Unter Projects solltest du nun dein neues Projekt finden und kannst dieses öffnen (erneute lange Ladezeit vor allem beim Ersten öffnen).

Benenne Das Projekt gleich, damit du es später auch wiederfindest.



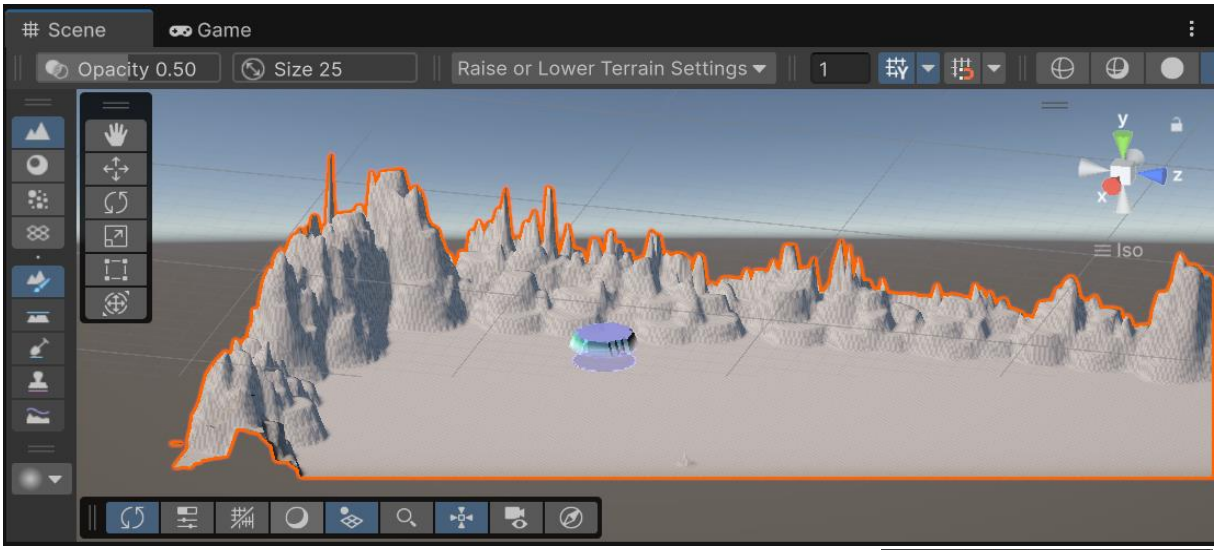
Eine Ebene und einen Player erstellen

In diesem Screenshot siehst du, Wozu die unterschiedlichen Bereiche im Fenster dienen.

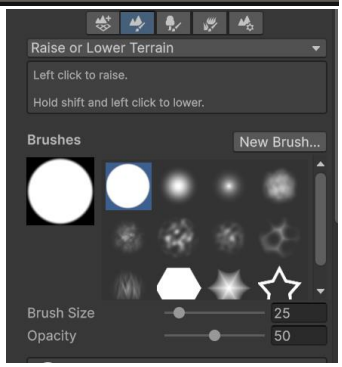


Kreiere als erstes eine Ebene, indem du einen Rechtsklick in das Fenster mit den Objekten machst und dort unter 3D Objekten die Ebene (Terrain) auswählst. Benenne sie mit «Ground».

Du kannst dann z.B. eine «natürliche» Grenze um dein Level zeichnen (Berge). Am einfachsten kannst du die Ansicht ändern, wenn du die Alt Taste und die Maustaste gedrückt hast und die Maus bewegst.

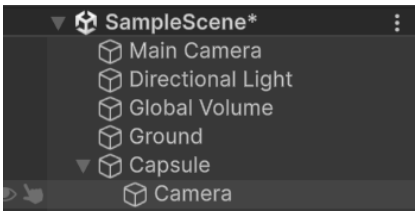


Mit dem Hand Symbol schiebst du die Karte hin und her, wenn du auf die Pfeilchen klichst, solltest du terrain malen können, das geht am besten mit dem Stift.



Den Pinsel für das Terrain kannst du rechts einstellen (Art des Pinsels «Brushes», Grösse «Brush Size» und Stärke «Opacity»)

Anschliessend erstellen wir einen vereinfachten Player, dazu fügen wir eine «Capsule» hinzu (oben links beim Pluszeichen bei den Objekten) und benenne sie «Player». Diese muss nun richtig platziert werden. Wenn du die Kapsel anklickst siehst du oben rechts die Koordinaten auf dem Feld. Passe diese an, um den Startpunkt des Players zu versetzen. Probiere aus und positioniere ihn z.B. mitten im Feld. Füge bei den Objekten «Camera» hinzu und ziehe sie auf das Objekt «Player» und lass los, so dass sie ihm angeheftet wird (sollte wie im Bild aussehen).



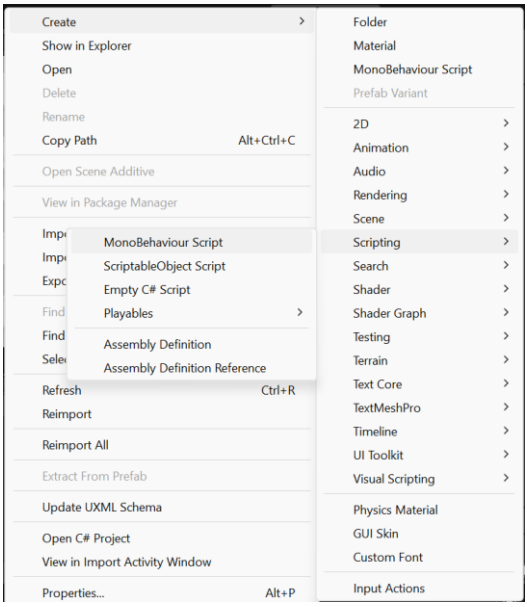
Dem Player ein Skript anfügen

Dein Player soll nun die Fähigkeit haben, sich im Feld zu bewegen. Dazu musst du ein C++ (C-Sharp) Script schreiben, das seine Fähigkeiten beschreibt. Nutze dazu ChatGPT, äussere dich aber ganz genau und ausführlich! Die Steuerung ist im Normalfall «wasd» und die Leertaste zum Springen.

Damit sich die Capsule nicht dreht, musst due die entsprechenden Achsen fixieren. Sie soll sich nur um die Y-Achse drehen, dies solltest du auch im Code erwähnen.

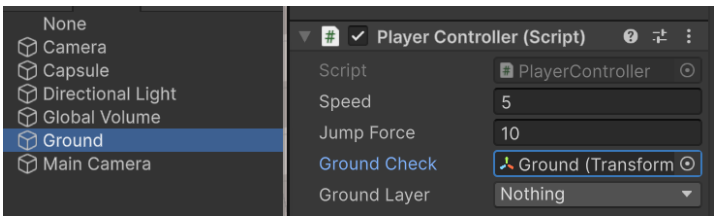
Klicke nun die Kapsel an und wähle rechts im «inspector» Fenster «add component». Nun musst du nach «rigidbody» suchen (gib es ein) und füge diesen hinzu (nicht «rigidbody2D»). Somit besitzt die Kapsel Physikalische Eigenschaften wie ein Gewicht und kann sich bewegen.

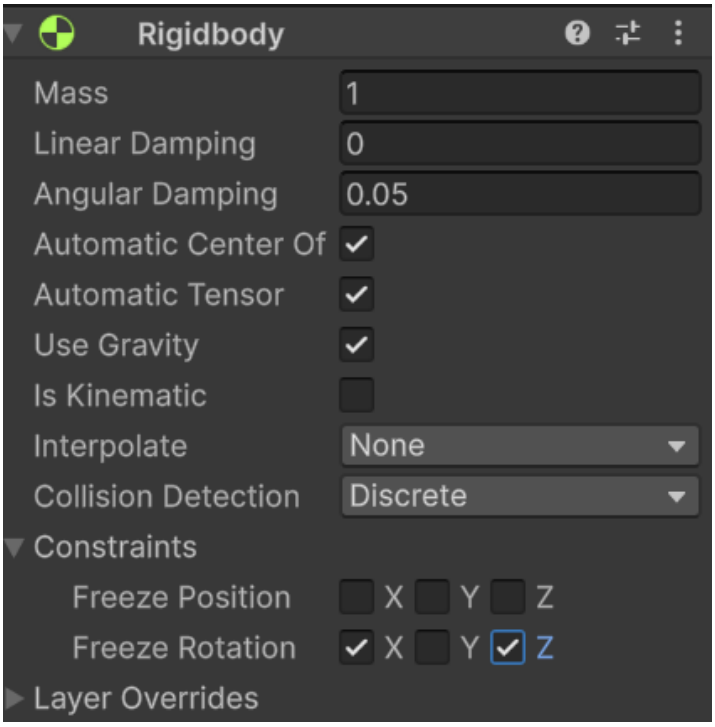
Das Script erstellst du mit Rechtsklick im Bereich «Assets», Create - Scripting - Empty C# Script.



Dieses öffnest du nun in Visual Studio (oder anderem Code-Progamm, mit doppelklick) und fügst dort den Code von Chatgpt ein. Lösche alles, was vorher schon im Fenster steht, es darf kein Restcode mehr übrig sein! Beim ersten Verwenden musst du noch C# add ons installieren, das wird automatisch gefragt.

Wenn du abgespeichert hast, kannst du das Script in das Inspectorfenster deines Players ziehen. Achtung: **Das Script muss den gleichen Namen haben, wie es am Anfang des Codes <u>im Code</u> genannt wird** (in dem Fall wohl PlayerController), sonst funktioniert es nicht!





Im Inspector kannst du dann noch beim Script den Groundcheck aktivieren. Somit erkennt dein Player den Ground als Boden. Die Geschwindigkeit und auch die Sprunghöhe lassen sich hier auch einstellen.

Nun musst du noch im Inspector unter Rigydbody die Achsenrotationen um die X- und die Z-Achse blockieren (siehe Bild).

Dein Player kann sich nun nur noch um seine Y Achse drehen (ähnlich wie eine Stecknadel durch Kopf und Körper bis in den Boden. Somit kann er nicht umfallen und nicht mehr aufstehen, was ohne diese Einstellung passieren würde (versuchs mal indem du auf Play klickst).

Die Kamera am Player fixieren und scripten

Auch die Kamera erhält einen Code, damit sie dem Player immer folgt. Dazu muss sie auf die richtige Höhe eingestellt werden und kann an den Player angeheftet werden. Die genauen Schritte und das Kamerascript versuchst du mit Chatgpt herauszufinden. **Manchmal reicht auch das blosse anheften an den Player (Kamera mit Drag and Drop auf den Player ziehen, danach die Position im Inspector gewünscht einstellen, 1. Person oder 3. Person Ansicht, Position der Kamera anhand der Achsen verschieben).**

Weitere Objekte einfügen, die sich bewegen

Du kannst nun beliebig weitere Objekte einfügen, versuche es erst mit Kugeln, die sich bewegen sollen. Füge der Kugel einen Rigidbody hinzu und ein Script, das du «RandomMovement» nennst. Generiere einen Code oder du findest einen im Klassenteams (MI, Unity). Mit dem Code soll sich das Objekt in zufällige Richtungen bewegen und die Richtungen ständig ändern.

Funktioniert der Code und die Kugel bewegt sich, kannst du das Objekt im Objektmenu kopieren und einfügen, um mehrere davon zu erhalten.

3D Objekte importieren und in das Level einbinden

Obj Dateien, lassen sich in Unity einfügen. Du kannst sie per drag and drop oder über das Dateimenu einfügen. Du musst aber, damit du nicht durch wände gehen kannst, einen Mesh Collidor (im Inspector zu finden) hinzufügen. Dann kannst du deine eigenen Gebäude besteigen und betreten. Meist sind sie noch zu klein, wenn du sie einfügst, du kannst sie bei der Position auch skalieren (die Grösse verändern).

Export für Windows (exe)

Export für Web

Quellen

Titelbild: unity.com (25.5.2023)

Bild Unity1: https://create-learn-prod.s3.us-west-2.amazonaws.com/uploads/unity/thumbnail/2LHF-MqFDQSs4Y1RV3-CCAlQCHNBdGP4pYuaopomp0-Unity.jpg (25.5.2023)

Scripts

PlayerController Beispielsscript

```
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    public float speed = 5f;
    public float jumpForce = 10f;
    public Transform groundCheck;
    public LayerMask groundLayer;

    private bool isGrounded;
    private Rigidbody rb;
    private float horizontalInput;

    void Start()
    {
        rb = GetComponent<Rigidbody>();

        // Freeze rotation around the x and z axes
        rb.constraints = RigidbodyConstraints.FreezeRotationX | RigidbodyConstraints.FreezeRotationZ;
    }

    void Update()
    {
        horizontalInput = Input.GetAxisRaw("Horizontal");
```

```
    // Jumping logic
    if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
    {
        Jump();
    }
}

void FixedUpdate()
{
    Move();
    CheckGrounded();
}

private void Move()
{
    // Move forward/backward
    float verticalInput = Input.GetAxisRaw("Vertical");
    Vector3 movement = transform.forward * verticalInput * speed;
    if (verticalInput < 0f) // moving backwards
    {
        movement *= 0.5f; // Optional: reduce speed when moving backwards
    }
    rb.MovePosition(rb.position + movement * Time.fixedDeltaTime);

    // Rotate left/right around the y-axis only
    float rotation = horizontalInput * 90f * Time.fixedDeltaTime; // Adjust rotation speed as needed
    Quaternion deltaRotation = Quaternion.Euler(0f, rotation, 0f);
    rb.MoveRotation(rb.rotation * deltaRotation);
}

private void Jump()
{
    rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
}

private void CheckGrounded()
```

```
    {
        isGrounded = Physics.CheckSphere(groundCheck.position, 0.1f, groundLayer);
    }
}
```

RandomMovement Beispielsscript

Ein Script das, angeheftet an ein Objekt, dieses in zufällige Richtungen bewegt.

```
using UnityEngine;

public class RandomMovement : MonoBehaviour
{
    public float speed = 5f;
    public float directionChangeInterval = 1f;

    private Rigidbody rb;
    private float timeUntilNextDirectionChange = 0f;
    private Vector3 movementDirection;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        ChooseNewDirection();
    }

    void Update()
    {
        timeUntilNextDirectionChange -= Time.deltaTime;
        if (timeUntilNextDirectionChange <= 0f)
        {
            ChooseNewDirection();
        }
    }

    void FixedUpdate()
    {
        // Move the object in the chosen direction
```

```
        Vector3 movement = movementDirection.normalized * speed * Time.fixedDeltaTime;
        rb.MovePosition(rb.position + movement);
    }

    void ChooseNewDirection()
    {
        timeUntilNextDirectionChange = directionChangeInterval;
        movementDirection = Random.onUnitSphere;
        movementDirection.y = 0f; // Ensure movement is in the x-z plane
        movementDirection.Normalize(); // Normalize to ensure consistent speed
    }
}
```

ChasePlayer

Ein Script, das das Objekt ab einer Bestimmten nähe dazu veranlasst, dich zu verfolgen.

```
using UnityEngine;

public class ChasePlayer : MonoBehaviour
{
    public Transform player;
    public float moveSpeed = 5f;
    public float rotationSpeed = 10f;
    public float chaseRange = 50f;
    public float wanderRange = 30f;
    public float bumpForce = 10f; // Force applied to the player when bumped

    private Rigidbody rb;
    private bool isChasingPlayer = false;
    private Vector3 wanderDirection;
    private float wanderTime = 0f;
    private float wanderInterval = 2f;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
```

```
    ChooseNewWanderDirection();
  }

  void Update()
  {
    float distanceToPlayer = Vector3.Distance(transform.position, player.position);

    if (!isChasingPlayer)
    {
      wanderTime -= Time.deltaTime;
      if (wanderTime <= 0f)
      {
        ChooseNewWanderDirection();
      }

      // Move in random direction
      Vector3 movement = wanderDirection * moveSpeed * Time.deltaTime;
      rb.MovePosition(rb.position + movement);
      transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(wanderDirection), rotationSpeed * Time.deltaTime);

      // Check if player is close
      if (distanceToPlayer < chaseRange)
      {
        isChasingPlayer = true;
      }
    }
    else
    {
      // Move towards player
      Vector3 directionToPlayer = player.position - transform.position;
      directionToPlayer.y = 0f; // Keep movement in the x-z plane
      if (directionToPlayer.magnitude > 1f) // Optional: Stop chasing if too close
      {
        directionToPlayer.Normalize();
        Vector3 movement = directionToPlayer * moveSpeed * Time.deltaTime;
```

```
            rb.MovePosition(rb.position + movement);
            transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(di-
rectionToPlayer), rotationSpeed * Time.deltaTime);
        }

        // Check if player is close enough to bump
        if (distanceToPlayer < 1.5f) // Adjust this value as needed
        {
            Rigidbody playerRb = player.GetComponent<Rigidbody>();
            if (playerRb != null)
            {
                // Apply a force to the player in the direction away from the chaser
                Vector3 bumpDirection = (player.position - transform.position).normalized;
                playerRb.AddForce(bumpDirection * bumpForce, ForceMode.Impulse);
            }
        }

        // Check if player is out of range
        if (distanceToPlayer > chaseRange)
        {
            isChasingPlayer = false;
            ChooseNewWanderDirection(); // Resume wandering
        }
    }
  }

  void ChooseNewWanderDirection()
  {
    wanderTime = wanderInterval;
    wanderDirection = Random.insideUnitSphere.normalized;
    wanderDirection.y = 0f; // Ensure movement is in the x-z plane
  }
}
```