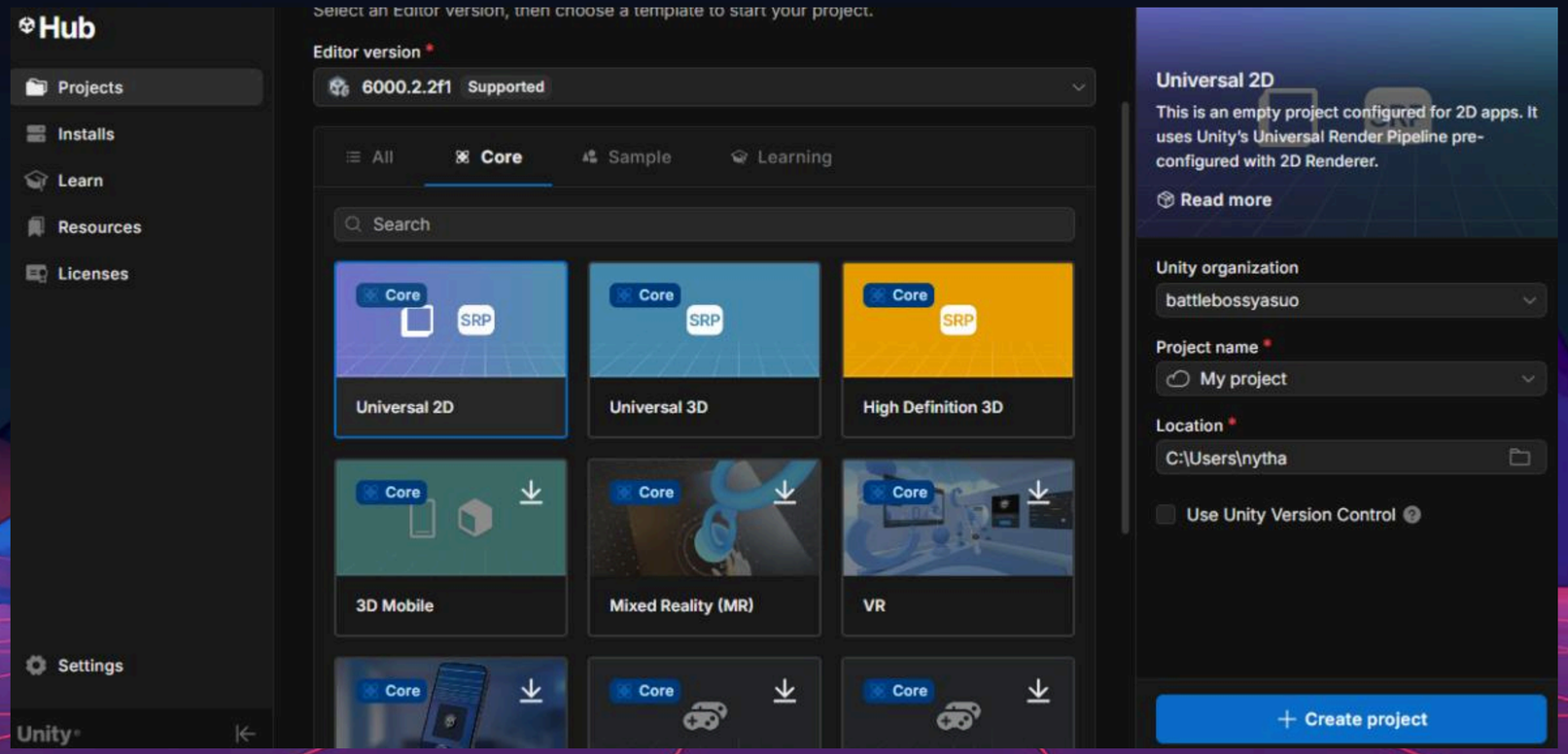
GNOME WELL STEP-BY-STEP

By: Abogado, Nythan
Abara, Jezreel
Cezar, Davidson

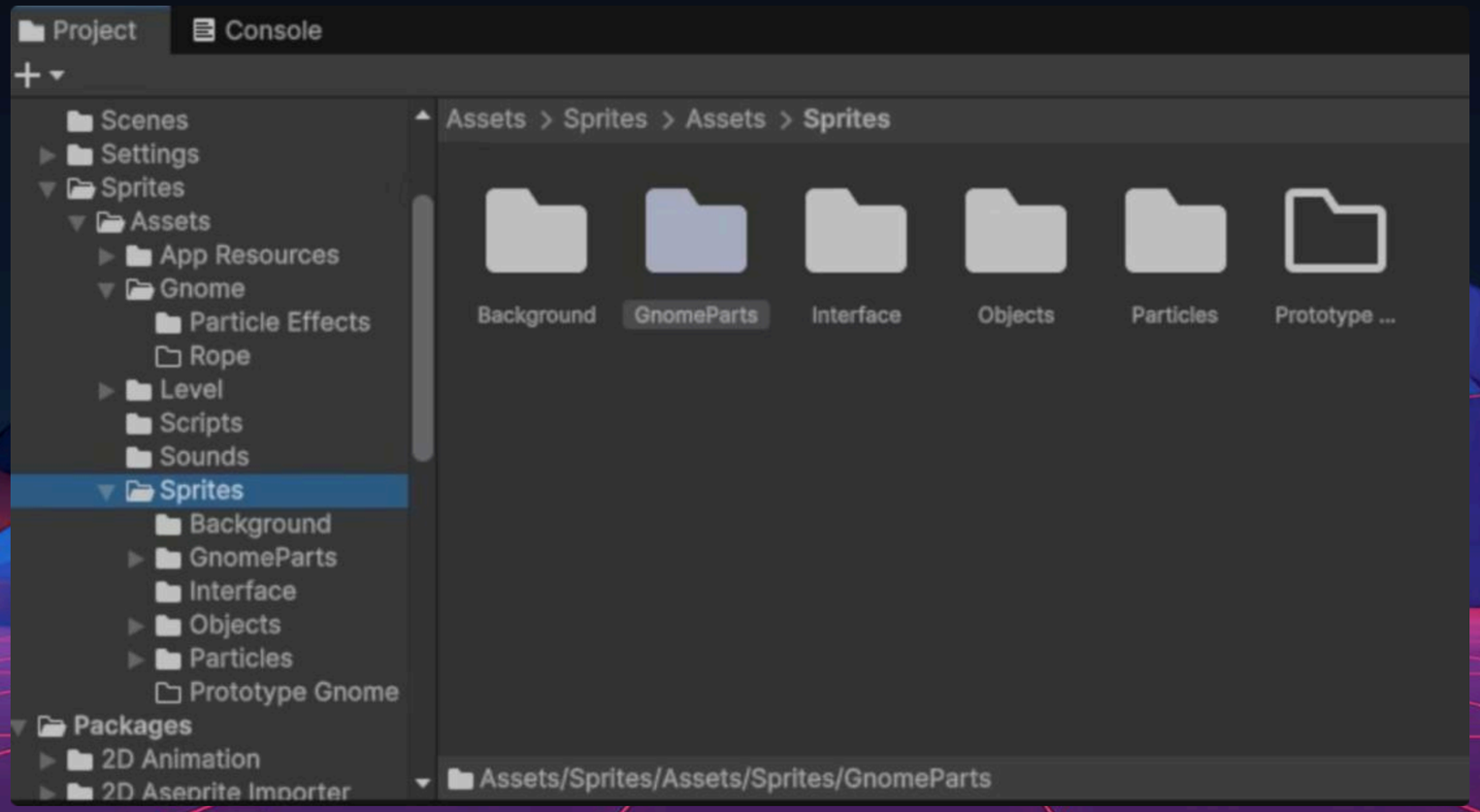
STEP 1

First create a new project in unity



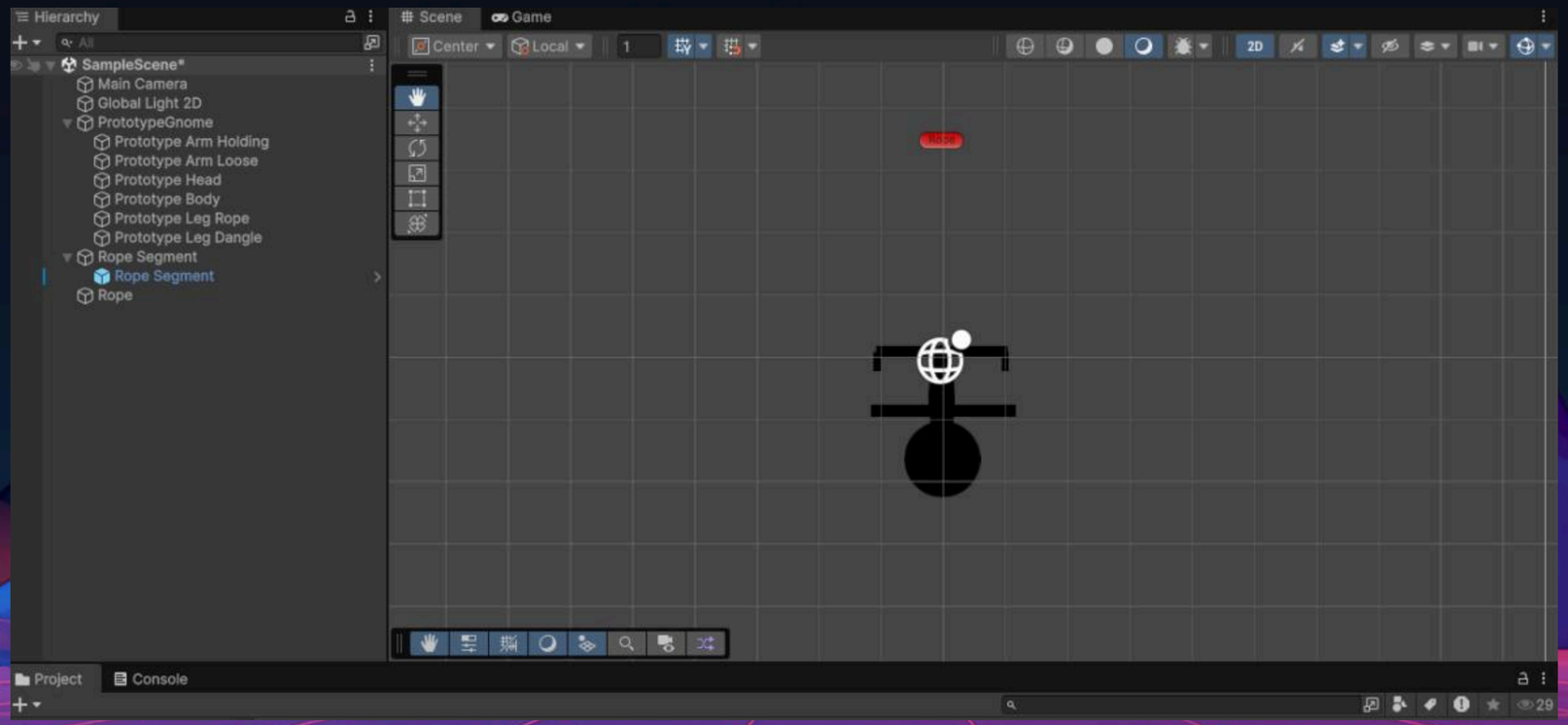
STEP 2

Import the files you downloaded in the book



STEP 3

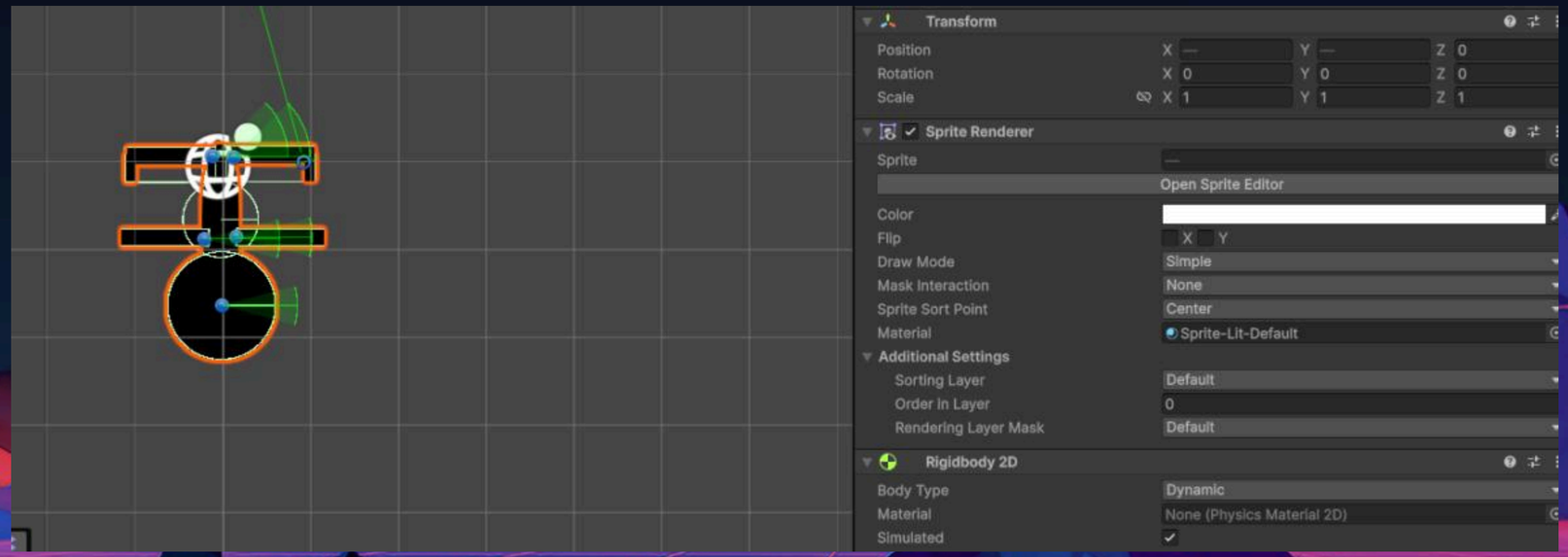
Add the assets, and make it like this



STEP 4

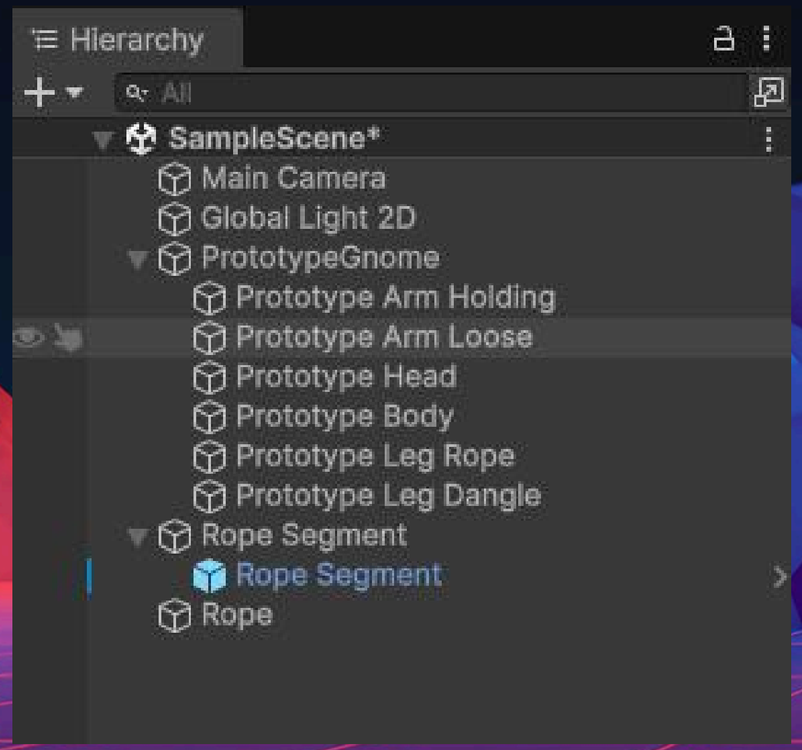
Add a component to each body parts

Add Ridigbody2d, Springjoint2d, Hingejoint2d, and add to the head circlecollider2d



STEP 5

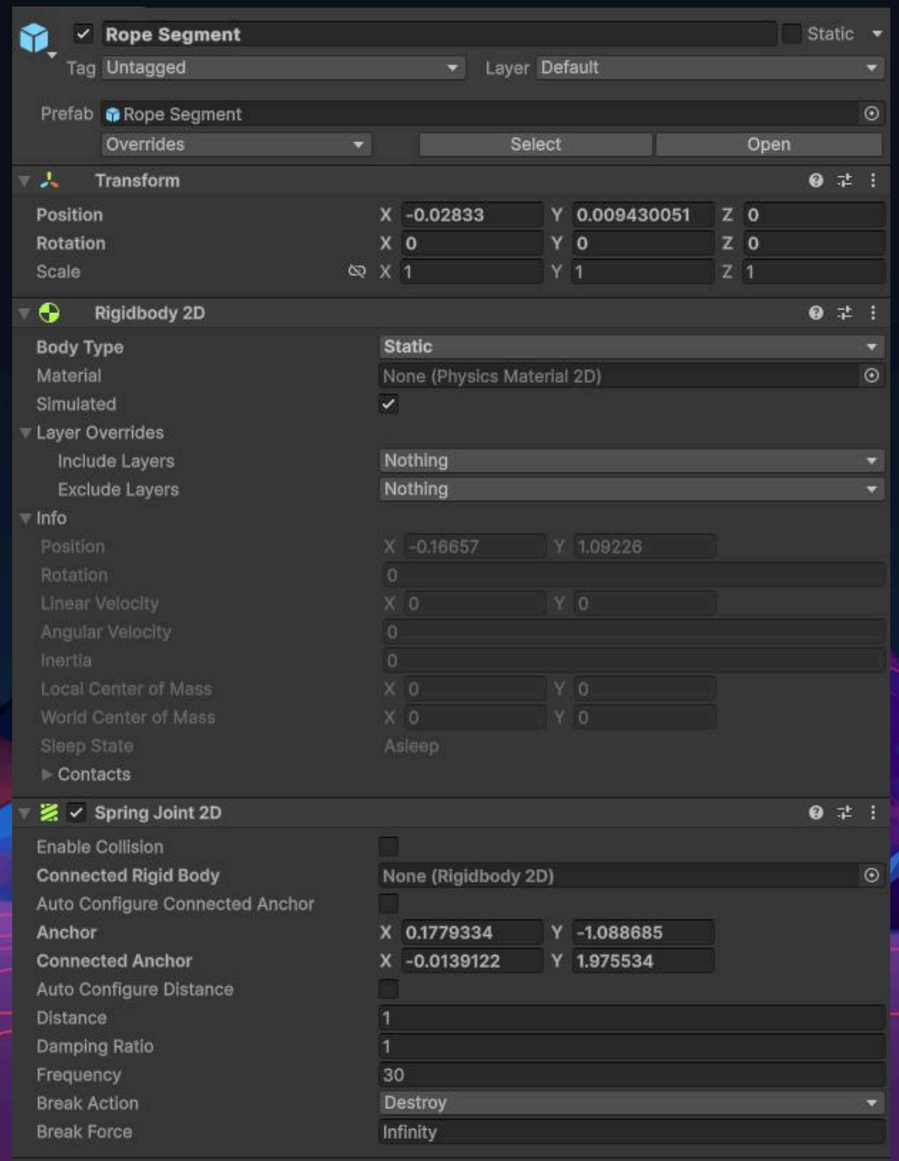
Create a empty game object name it ROPE SEGMENT, and add the Rope Segment prefab in the ROPE SEGMENT you've created



STEP 6

Config the Rope Segment prefab, change the value of the Damping Ratio 1, and Frequency to 30 of the SPRING JOINT2D

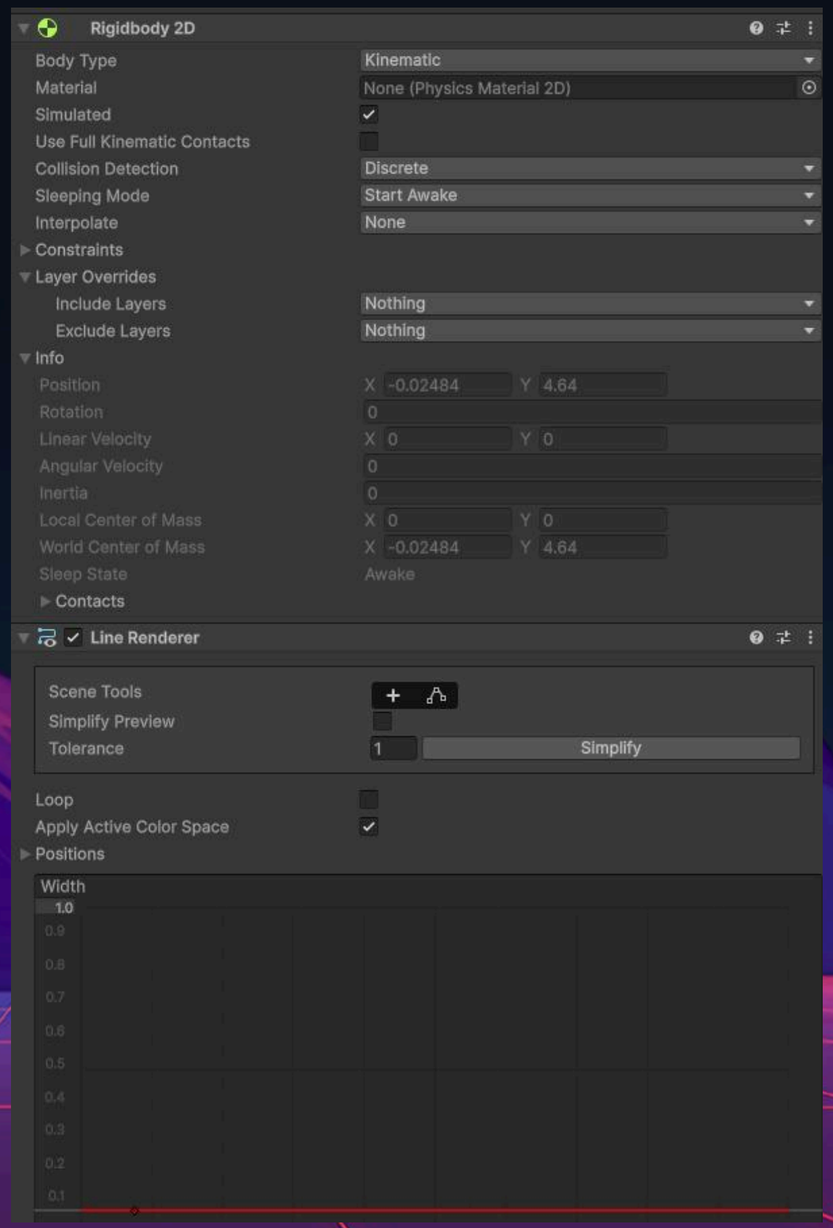
Change the body type of the RigidBody2D to STATIC so that the rope segment will stay put



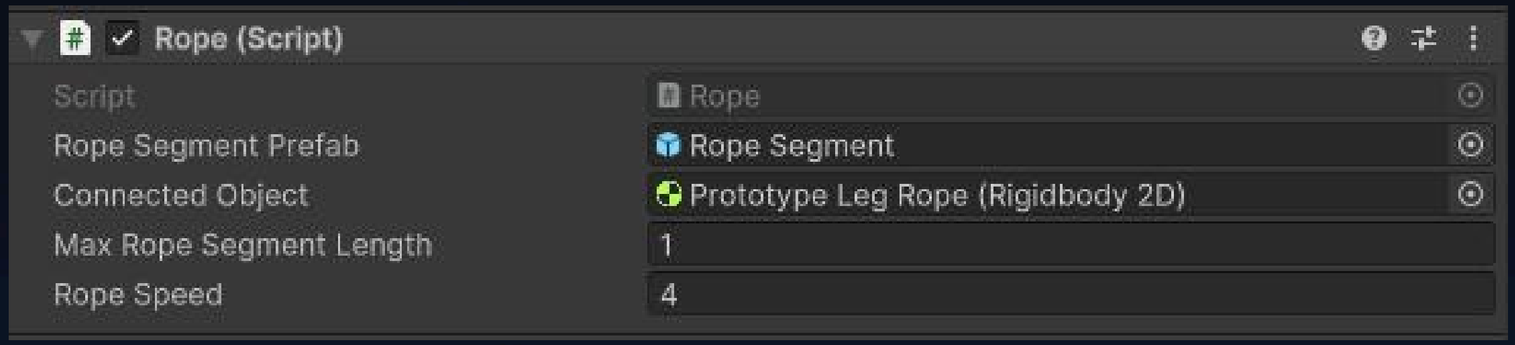
STEP 7

Add a component Rigidbody2d, change the body type to Kinematic so that it will move and stay the rope like a physics

Add a Line renderer which will create a block line to have a frame to the material to be added later, Change the width value to 0.0075 as the book recommended

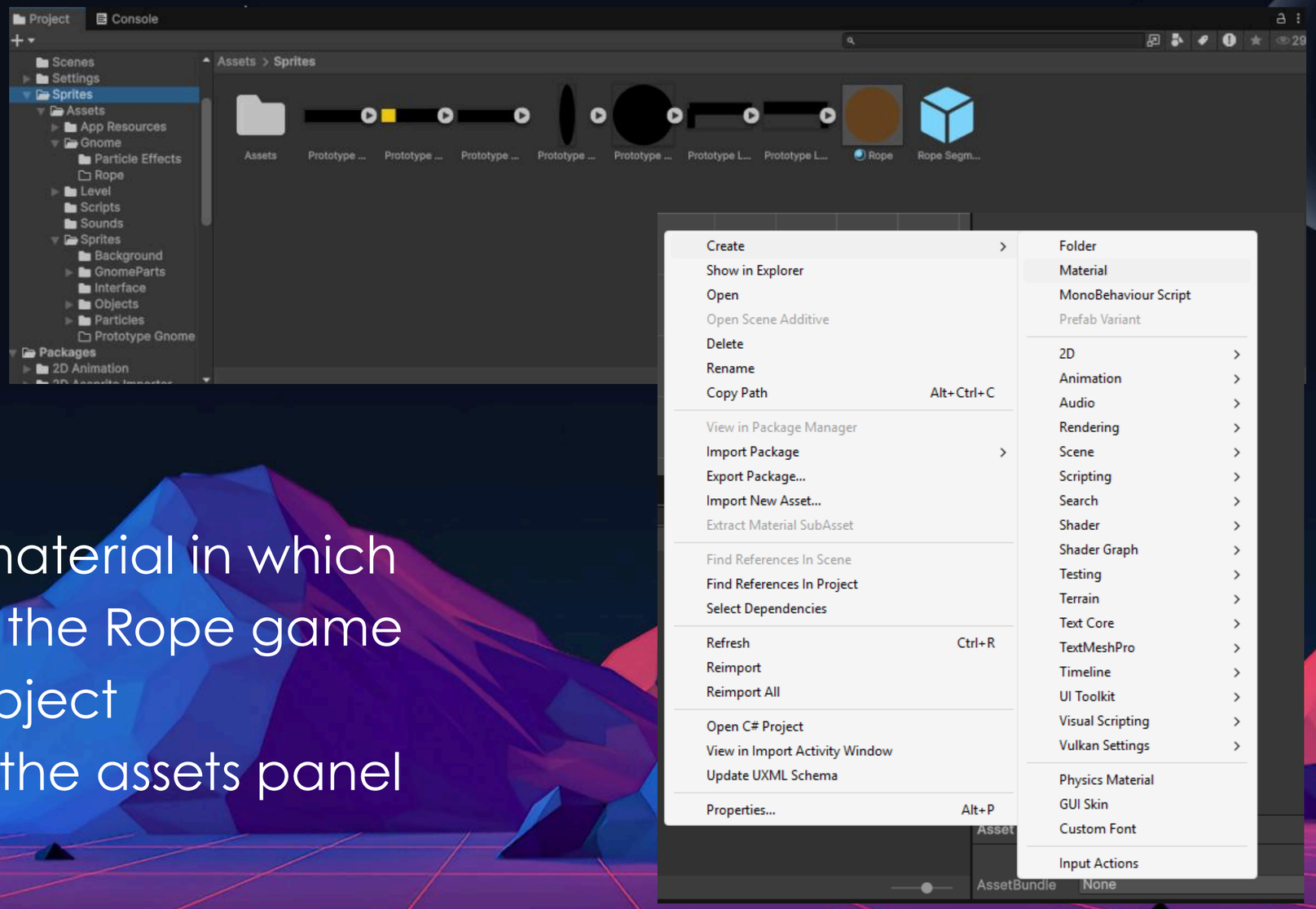


STEP 8



Add a new component called Rope.cs
note copy paste the code sa book

STEP 9

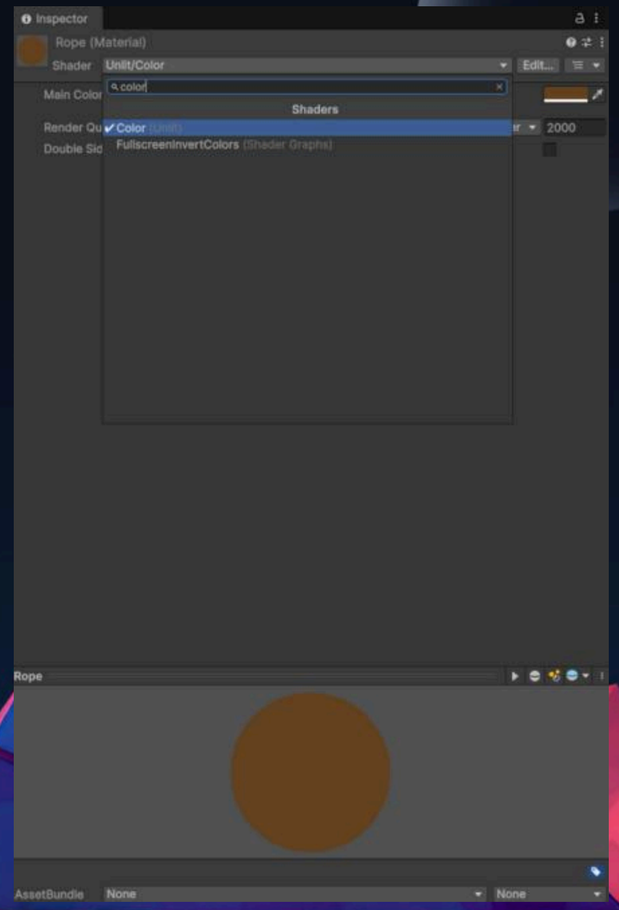


Add a new material in which will be use to the Rope game object

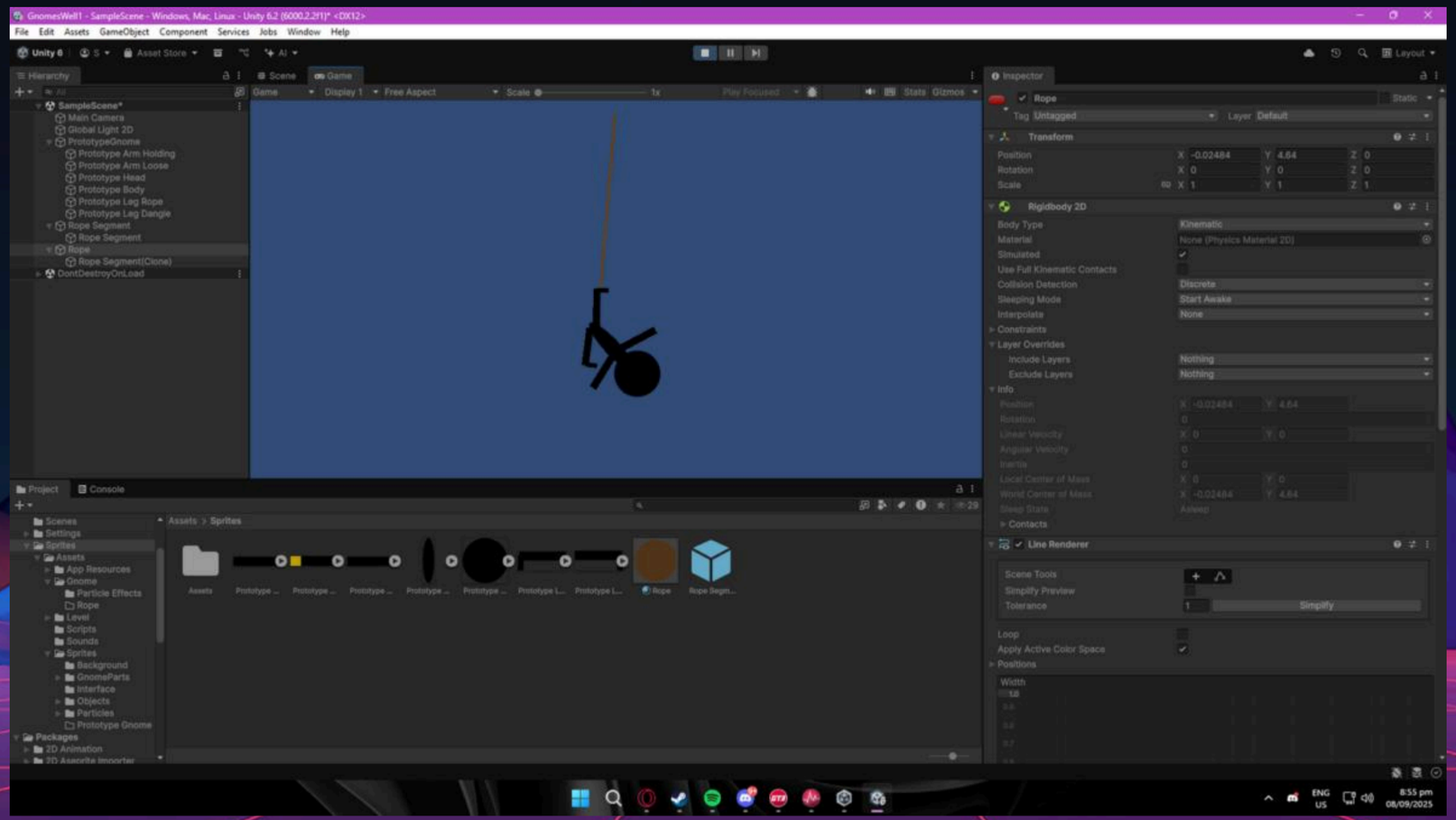
by right click the assets panel

STEP 10

Set the Shader to Color and change the color to dark brown same the color of the rope we usually see



Game Prototype



Setting Up Buttons

Create another cs file names singleton

Singleton (Mono Script)

Assembly Information

Filename Assembly-CSharp.dll

```
using UnityEngine;
using System.Collections;
// This class allows other objects to refer to a single
// shared object. The GameManager and InputManager
classes
// use this.
// To use this, subclass like so:
// public class MyManager : Singleton<MyManager> { }
// You can then access the single shared instance of the
// class like so:
// MyManager.instance.DoSomething();
public class Singleton<T> : MonoBehaviour
where T : MonoBehaviour
{
    // The single instance of this class.
    private static T _instance;
    // The accessor. The first time this is called, _instance
    // will be set up. If an appropriate object can't be
found,
    // an error will be logged.
    public static T instance
    {
        get
        {
            // If we haven't already set up _instance...
            if (_instance == null)
            {
                // Try to find the object.
                _instance = FindObjectOfType<T>();
                // Log if we can't find it.
                if (_instance == null)
                {

                Debug.LogError("Can't find " +
                typeof(T) + "!");
                }
            }
            // Return the instance so that it can be used!
            return _instance;
        }
    }
}
```

and add this code

Setting Up Buttons

Create an object named input manager, add the script

Select all the asset body, and create new cs paste the code

Input Manager (Mono Script) Import Settin

Open | Execution Order...

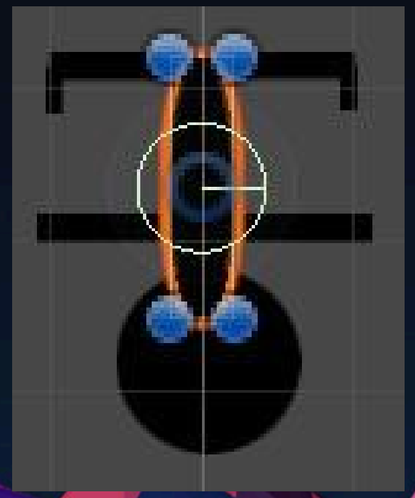
Imported Object

Input Manager (Mono Script)

Assembly Information

Filename Assembly-CSharp.dll

```
using UnityEngine;
using System.Collections;
// Translates the accelerometer data into sideways motion
// info.
public class InputManager : Singleton<InputManager>
{
   // How much we're moving. -1.0 = full left, +1.0 = full
   // right
   private float _sidewaysMotion = 0.0f;
// This property is declared as read-only, so that other
// classes can't change it

public float sidewaysMotion
   {
      get
      {
         return _sidewaysMotion;
      }
   }
   // Every frame, store the tilt
   void Update()
   {
      Vector3 accel = Input.acceleration;
      _sidewaysMotion = accel.x;
   }
}
```



Imported Object

Swinging (Mono Script)

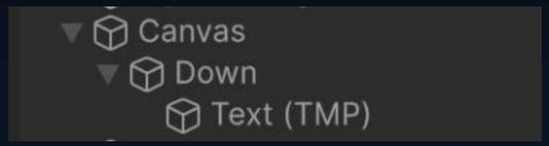
Assembly Information

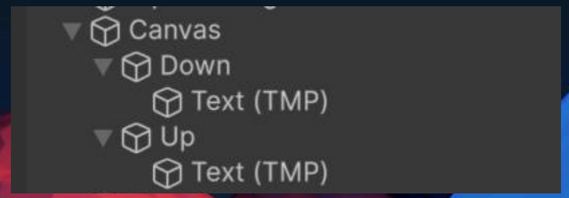
Filename Assembly-CSharp.dll

```
using UnityEngine;
using System.Collections;
// Uses the input manager to apply sideways forces to an
// object. Used to make the gnome swing side-to-side.
public class Swinging : MonoBehaviour
{
   // How much should we swing by? Bigger numbers =
more
   // swing
   public float swingSensitivity = 100.0f;

// Use FixedUpdate instead of Update, in order to play
// better with the physics engine
void FixedUpdate()
   {
      // If we have no ridigbody (anymore), remove this
      // component
      if (GetComponent<Rigidbody2D>() == null)
      {
         Destroy(this);
         return;
      }
      // Get the tilt amount from the InputManager
      float swing =
InputManager.instance.sidewaysMotion;
      // Calculate a force to apply
      Vector2 force =
      new Vector2(swing * swingSensitivity, 0);
      // Apply the force
      GetComponent<Rigidbody2D>().AddForce(force);
   }
}
```

Setting Up Buttons

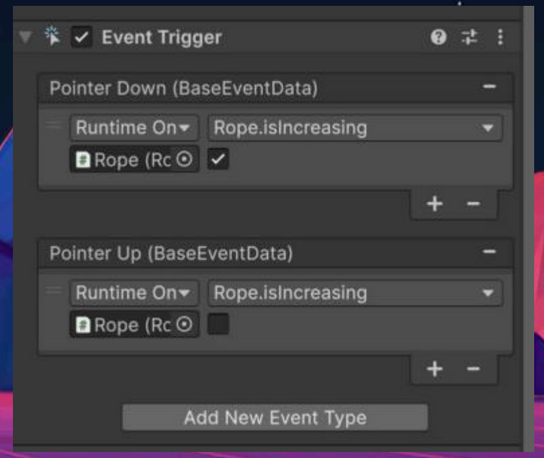
Add new game obeject and name it DOWN, Up





configyure gameobject (down,up)

Add Event Trigger component



Creating Camera Follow

Add the code cs
camerafollow.cs

```
using UnityEngine;

// Adjusts the camera to always match the Y-position of a
// target object, within certain limits.
public class CameraFollow : MonoBehaviour
{
    // The object we want to match the Y position of.
    public Transform target;
    // The highest point the camera can go.
    public float topLimit = 10.0f;
    // The lowest point the camera can go.
    public float bottomLimit = -10.0f;
    // How quickly we should move toward the target.
    public float followSpeed = 0.5f;
    // After all objects have updated position, work out where
    // this camera should be
    void LateUpdate()
    {
        // If we have a target...
        if (target != null)
        {
            // Get its position
            Vector3 newPosition = this.transform.position;
            // Work out where this camera should be
            newPosition.y = Mathf.Lerp(newPosition.y,

            target.position.y, followSpeed) ;
            // Clamp this new location to within our
            // limits
            newPosition.y =
            Mathf.Min(newPosition.y, topLimit);
            newPosition.y =
            Mathf.Max(newPosition.y, bottomLimit);
            // Update our location
            transform.position = newPosition;
        }
    }
    // When selected in the editor, draw a line from the top
    // limit to the bottom.
    void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.yellow;
        Vector3 topPoint =
        new Vector3(this.transform.position.x,
        topLimit, this.transform.position.z);
        Vector3 bottomPoint =
        new Vector3(this.transform.position.x,
        bottomLimit, this.transform.position.z);
        Gizmos.DrawLine(topPoint, bottomPoint);
    }
}
```

Creating Camera Follow

Add code cs Gnome.cs

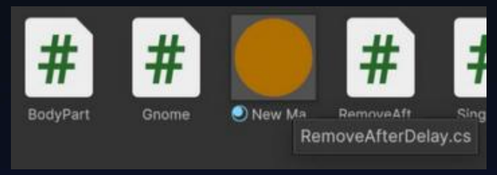
Gnome ▸ ReleaseGhost()

```
using System.Collections;
using UnityEngine;
public class Gnome : MonoBehaviour
{
    // The object that the camera should follow.
    public Transform cameraFollowTarget;
    public Rigidbody2D ropeBody;
    public Sprite armHoldingEmpty;
    public Sprite armHoldingTreasure;
    public SpriteRenderer holdingArm;
    public GameObject deathPrefab;
    public GameObject flameDeathPrefab;
    public GameObject ghostPrefab;
    public float delayBeforeRemoving = 3.0f;
    public float delayBeforeReleasingGhost = 0.25f;
    public GameObject bloodFountainPrefab;

bool dead = false;
    bool _holdingTreasure = false;
    public bool holdingTreasure
    {
        get
        {
            return _holdingTreasure;
        }
        set
        {
            if (dead == true)
            {
                return;
            }
            _holdingTreasure = value;
            if (holdingArm != null)
            {
                if (_holdingTreasure)
                {
                    holdingArm.sprite =
                    armHoldingTreasure;
                }
                else
                {
                    holdingArm.sprite =
                    armHoldingEmpty;
                }
            }
        }
    }
    public enum DamageType
    {
        Slicing,
        Burning
    }
    public void ShowDamageEffect(DamageType type)
    {
        switch (type)
        {
            case DamageType.Burning:
                if (flameDeathPrefab != null)
                {
                    Instantiate(
                    flameDeathPrefab, cameraFollowTarget.position,
                    cameraFollowTarget.rotation
                    );
                }
                break;
            case DamageType.Slicing:
                if (deathPrefab != null)
                {
                    Instantiate(
                    deathPrefab,
                    cameraFollowTarget.position,
                    cameraFollowTarget.rotation);


                }
                break;
        }
    }
    public void DestroyGnome(DamageType type)
```

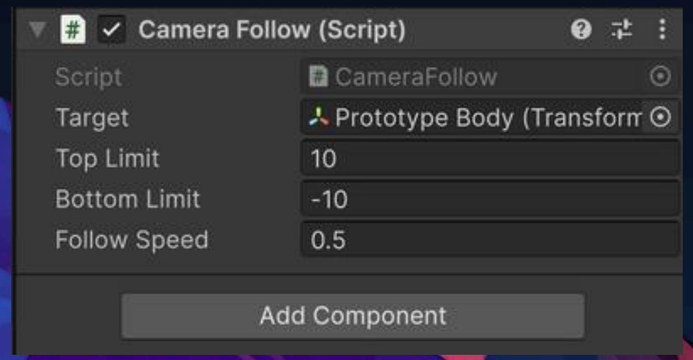
```
    }
    public void DestroyGnome(DamageType type)
    {
        holdingTreasure = false;
        dead = true;
        // Find all child objects, and randomly disconnect
        // their joints
        foreach (BodyPart part in
        GetComponentsInChildren<BodyPart>())
        {
            switch (type)
            {
                case DamageType.Burning:
                    // 1 in 3 chance of burning
                    bool shouldBurn = Random.Range(0, 3) == 0;
                    if (shouldBurn)
                    {
                        part.ApplyDamageSprite(type);
                    }
                    break;
                case DamageType.Slicing:
                    // Slice damage always applies a damage sprite
                    part.ApplyDamageSprite(type);
                    break;
            }
            // 1 in 8 chance of separating from body
            bool shouldDetach = Random.Range(0, 2) == 0;
            if (shouldDetach)
            {
                // Make this object remove its rigidbody and
                // collider after it comes to rest
                part.Detach();
                // If we're separating, and the damage type was
                // Slicing, add a blood fountain
                if (type == DamageType.Slicing)
                {
                    if (part.bloodFountainOrigin != null &&
                    bloodFountainPrefab != null)
                    {

                        // Attach a blood fountain for
                        // this detached part
                        GameObject fountain = Instantiate(
                        bloodFountainPrefab,
                        part.bloodFountainOrigin.position,
                        part.bloodFountainOrigin.rotation
                        ) as GameObject;
                        fountain.transform.SetParent(
                        this.cameraFollowTarget,
                        false
                        );
                    }
                }
                // Disconnect this object
                var allJoints = part.GetComponentsInChildren<Joint2D>();
                foreach (Joint2D joint in allJoints)
                {
                    Destroy(joint);
                }
            }
        }
        // Add a RemoveAfterDelay component to this object
        var remove = gameObject.AddComponent<RemoveAfterDelay>();
        remove.delay = delayBeforeRemoving;
        StartCoroutine(ReleaseGhost());
    }
    IEnumerator ReleaseGhost()
    {

        // No ghost prefab? Bail out.
        if (ghostPrefab == null)
        {
            yield break;
        }
        // Wait for delayBeforeReleasingGhost seconds
        yield return new WaitForSeconds(delayBeforeReleasingGhost);
        // Add the ghost
        Instantiate(
        ghostPrefab,
        transform.position,
        Quaternion.Identity
        );
    }
}
```

Creating Camera Follow

create 3 cs files BodyPart,Gnome and Removeafter delay



Add Camera Follow script on the main camera object and Drag prototype body in the target



Add to the BodyPart

```
using UnityEngine;
[RequireComponent(typeof(SpriteRenderer))]
public class BodyPart : MonoBehaviour
{
    // The sprite to use when ApplyDamageSprite is called with
    // damage type 'slicing'
    public Sprite detachedSprite;
    // The sprite to use when ApplyDamageSprite is called with
    // damage type 'burning'
    public Sprite burnedSprite;
    // Represents the position and rotation that a blood
    // fountain will appear at on the main body
    public Transform bloodFountainOrigin;
    // If true, this object will remove its collision, joints,
    // and rigidbody when it comes to rest
    bool detached = false;
    // Decouple this object from the parent, and flag it as
    // needing physics removal
    public void Detach()
    {
        detached = true;
        this.tag = "Untagged";
        transform.SetParent(null, true);

    }
    // Every frame, if we're detached, remove physics if the
    // rigidbody is sleeping. This means this detached body
    // part will never get in the way of the gnome.
    public void Update()
    {
        // If we're not detached, do nothing
        if (detached == false)
        {
            return;
        }
        // Is our rigidbody sleeping?
        var rigidbody = GetComponent<Rigidbody2D>();
        if (rigidbody.IsSleeping())
        {
            // If so, destroy all joints..
            foreach (Joint2D joint in
            GetComponentsInChildren<Joint2D>())
            {
                Destroy(joint);
            }
            // ...rigidbodies...
            foreach (Rigidbody2D body in
            GetComponentsInChildren<Rigidbody2D>())
            {
                Destroy(body);
            }
            // ...and the collider.
            foreach (Collider2D collider in
            GetComponentsInChildren<Collider2D>())
            {
                Destroy(collider);
            }
            // Finally, remove this script.
            Destroy(this);
        }
    }
    // Swaps out the sprite for this part based on what kind
    // of damage was received
    public void ApplyDamageSprite(
    Gnome.DamageType damageType)
    {
        Sprite spriteToUse = null;
        switch (damageType)
        {
            case Gnome.DamageType.Burning:
                spriteToUse = burnedSprite;

                break;
            case Gnome.DamageType.Slicing:
                spriteToUse = detachedSprite;
                break;
        }
        if (spriteToUse != null)
        {
            GetComponent<SpriteRenderer>().sprite =
            spriteToUse;
        }
    }
}
```

Adjusting game parameters

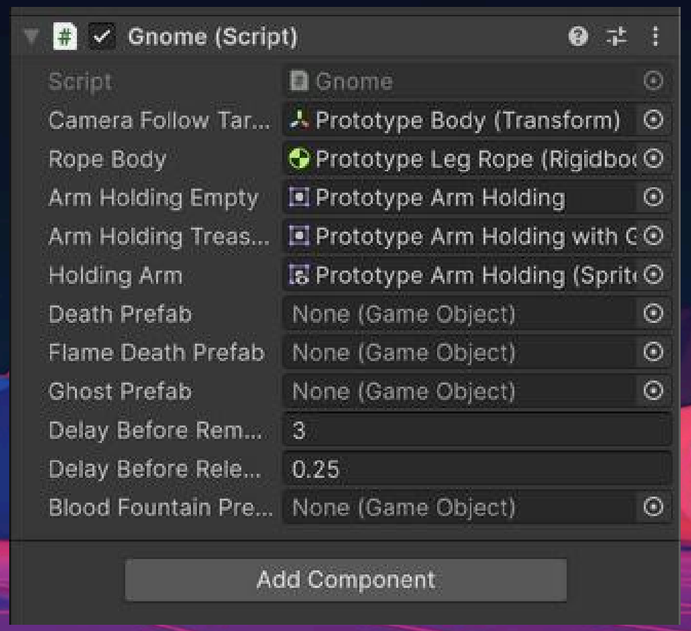
Add this to the RemoveAfterDelay.cs

RemoveAfterDelay ▸ Remove()

```
// Removes an object after a certain delay.
using UnityEngine;
using System.Collections;
public class RemoveAfterDelay : MonoBehaviour
{
    // How many seconds to wait before removing.
    public float delay = 1.0f;
    void Start()
    {
        // Kick off the 'Remove' coroutine.
        StartCoroutine("Remove");
    }
    IEnumerator Remove()
    {
        // Wait 'delay' seconds, and then destroy the
        // gameObject attached to this object.
        yield return new WaitForSeconds(delay);
        Destroy(gameObject);
        // Don't say Destroy(this) - that just destroys this
        // RemoveAfterDelay script.
    }
}
```

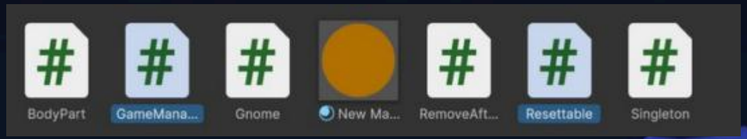
Select all the body parts and after that Prototype gnomewell too put the BodyPart.cs

Configured Gnome.cs looks like



Adjusting game parameters

create new 2 file cs named gamemanager and resettable



copy paste the code from the book

```
using UnityEngine;
using UnityEngine.Events;
// Contains a UnityEvent that can be used to reset the state
// of this object.
public class Resettable : MonoBehaviour
{
    // In the editor, connect this event to methods that should
    // run when the game resets.
    public UnityEvent onReset;
    // Called by the GameManager when the game resets.
    public void Reset()
    {
        // Kicks off the event, which calls all of the
        // connected methods.
        onReset.Invoke();
    }
}
```

Adjusting game parameters

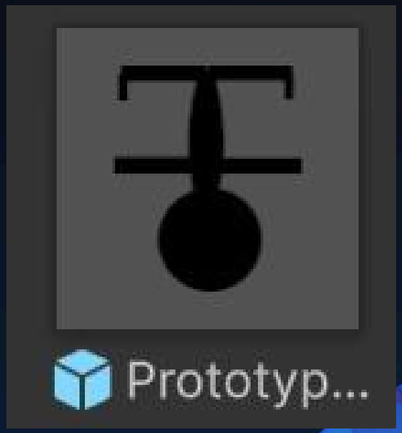
Add this code to the game manager from the book

```
using System.Collections;
using UnityEngine;

// Manages the game state.
public class GameManager : Singleton<GameManager>
{
    // The location where the gnome should appear.
    public GameObject startingPoint;
// The rope object, which lowers and raises the gnome.

public Rope rope;
    // The follow script, which will follow the gnome
    public CameraFollow cameraFollow;
    // The 'current' gnome (as opposed to all those dead ones)
    Gnome currentGnome;
    // The prefab to instantiate when we need a new gnome
    public GameObject gnomePrefab;
    // The UI component that contains the 'restart' and 'resume'
    // buttons
    public RectTransform mainMenu;
    // The UI component that contains the 'up', 'down' and
    // 'menu' buttons
    public RectTransform gameplayMenu;
    // The UI component that contains the 'you win!' screen
    public RectTransform gameOverMenu;
    // If true, ignore all damage (but still show damage
    // effects) The 'get; set;' make this a property, to make
    // it show up in the list of methods in the Inspector for
    // Unity Events
    public bool gnomeInvincible { get; set; }
    // How long to wait after dying before creating a new gnome
    public float delayAfterDeath = 1.0f;
    // The sound to play when the gnome dies
    public AudioClip gnomeDiedSound;
    // The sound to play when the game is won
    public AudioClip gameOverSound;
    void Start()
    {
        // When the game starts, call Reset to set up the
        // gnome.
        Reset();
    }
    // Reset the entire game.
    public void Reset()
    {
        // Turn off the menus, turn on the gameplay UI
        if (gameOverMenu)
            gameOverMenu.gameObject.SetActive(false);
        if (mainMenu)
            mainMenu.gameObject.SetActive(false);
        if (gameplayMenu)
            gameplayMenu.gameObject.SetActive(true);
        // Find all Resettable components and tell them to
```

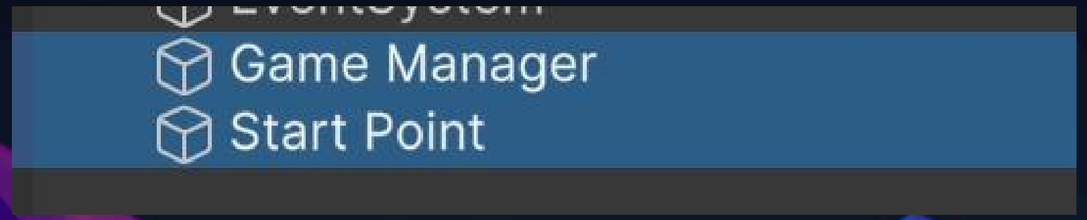
```
GameManager ▸ DelayAfterDeath
        // Un-pause the game
        Time.timeScale = 1.0f;
    }
    void CreateNewGnome()
    {
        // Remove the current gnome, if there is one
        RemoveGnome();
        // Create a new Gnome object, and make it be our
        // currentGnome
        GameObject newGnome =
        (GameObject)Instantiate(gnomePrefab,
        startingPoint.transform.position,
        Quaternion.identity);
        currentGnome = newGnome.GetComponent<Gnome>();
        // Make the rope visible
        rope.gameObject.SetActive(true);
        // Connect the rope's trailing end to whichever
        // rigidbody the Gnome object wants (e.g., his foot)
        rope.connectedObject = currentGnome.ropeBody;
        // Reset the rope's length to the default
        rope.ResetLength();
        // Tell the cameraFollow to start tracking the new
        // Gnome object
        cameraFollow.target = currentGnome.cameraFollowTarget;
    }
    void RemoveGnome()
    {

        // Don't actually do anything if the gnome is invincible
if (gnomeInvincible)
            return;
        // Hide the rope
        rope.gameObject.SetActive(false);
        // Stop tracking the gnome
        cameraFollow.target = null;
        // If we have a current gnome, make that no longer be
        // the player
        if (currentGnome != null)
        {
            // This gnome is no longer holding the treasure
            currentGnome.holdingTreasure = false;
            // Mark this object as not the player (so that
            // colliders won't report when the object
            // hits them)
            currentGnome.gameObject.tag = "Untagged";
            // Find everything that's currently tagged
            // "Player" and remove that tag
            foreach (Transform child in
            currentGnome.transform)
            {
                child.gameObject.tag = "Untagged";
            }
```

Modifying the Gnome

make the gnome prototype into prefab

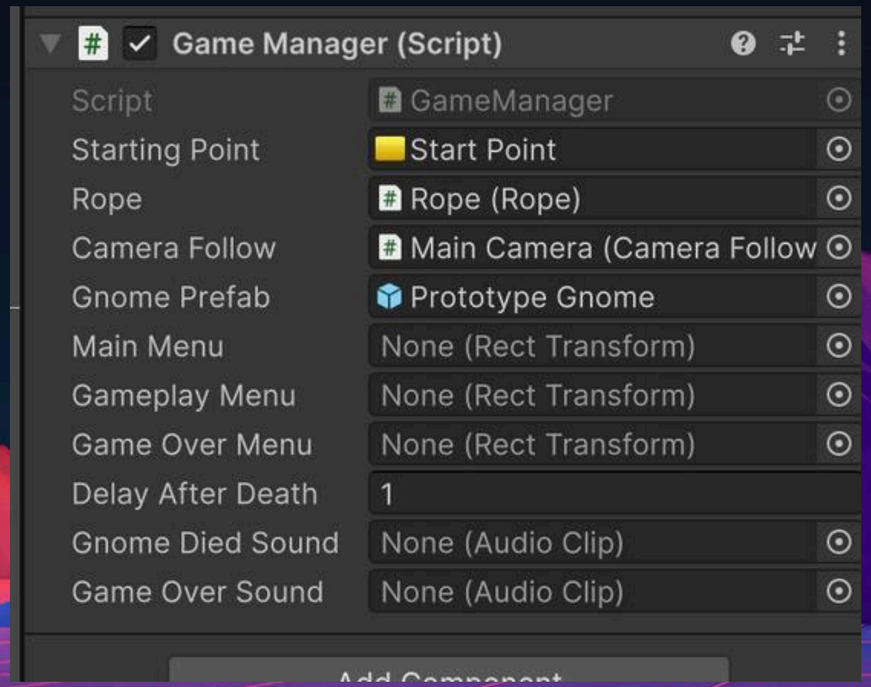


Create new gameobject and name it
Gmane Manager and Start point



Modifying the Gnome

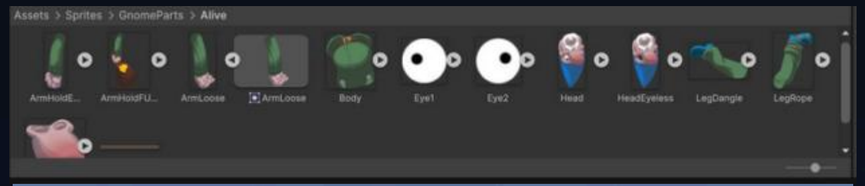
Put the Gamemanager.cs into the game manager object and configure this setting given



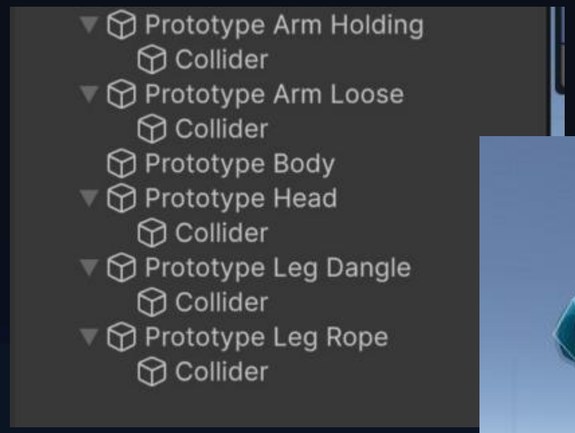
Align Start point object to the gnome axis

Modifying the Gnome

add the sprites and replace the prefab witht the actual sprites



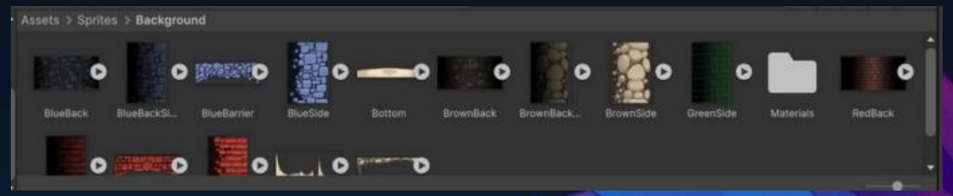
Add new component collider object to each part except the body and inside the colider add polygon collider will look like this

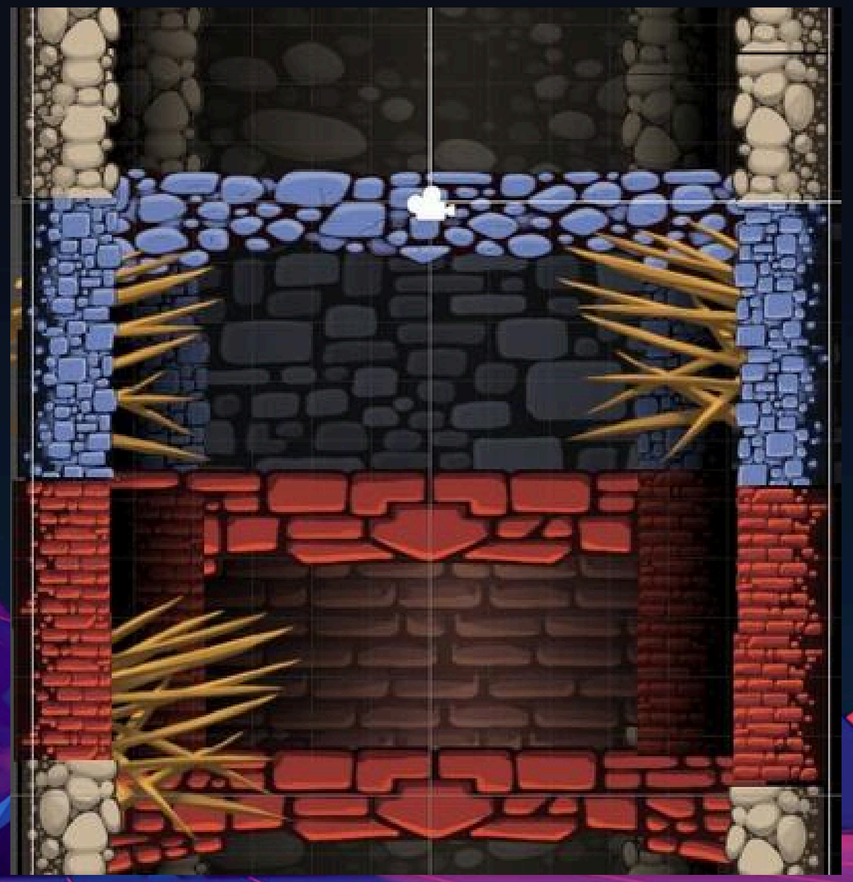


Designing the Level

Do want you want, design the level on your own

Add the asset for the level

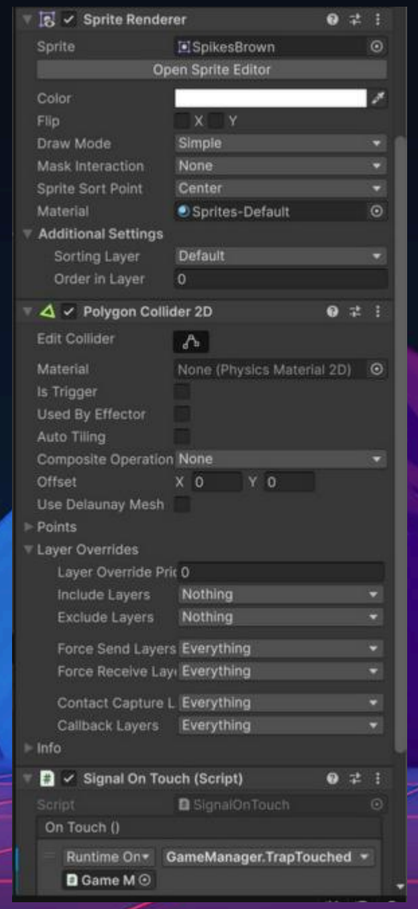




Designing the Level

create new cs again paste the code and name it signalontouch.cs

Add this components to the spikes

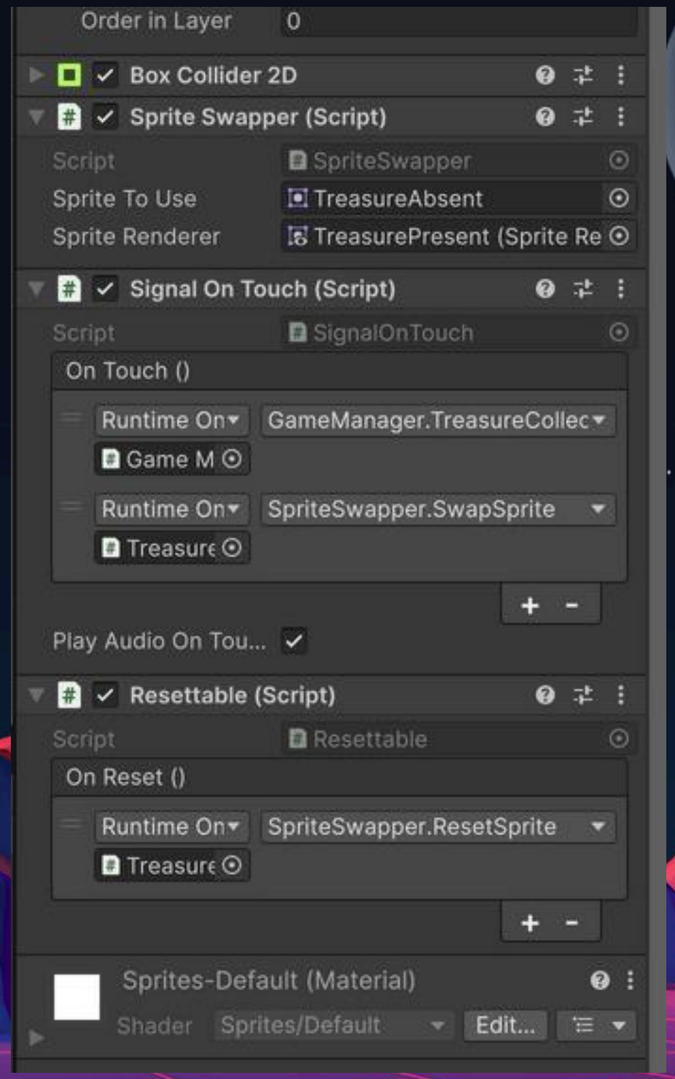



```
using UnityEngine;
using UnityEngine.Events;
// Invokes a UnityEvent when the Player collides with this
// object.
[RequireComponent(typeof(Collider2D))]
public class SignalOnTouch : MonoBehaviour
{

// The UnityEvent to run when we collide.
// Attach methods to run in the editor.
public UnityEvent onTouch;
    // If true, attempt to play an AudioSource when we collide.
    public bool playAudioOnTouch = true;
    // When we enter a trigger area, call SendSignal.
    void OnTriggerEnter2D(Collider2D collider)
    {
        SendSignal(collider.gameObject);
    }
    // When we collide with this object, call SendSignal.
    void OnCollisionEnter2D(Collision2D collision)
    {
        SendSignal(collision.gameObject);
    }
    // Checks to see if this object was tagged as Player, and
    // invoke the UnityEvent if it was.
    void SendSignal(GameObject objectThatHit)
    {
        // Was this object tagged Player?
        if (objectThatHit.CompareTag("Player"))
        {
            // If we should play a sound, attempt to play it
            if (playAudioOnTouch)
            {
                var audio = GetComponent<AudioSource>();
                // If we have an audio component,
                // and this component's parents
                // are active, then play
                if (audio &&
                audio.gameObject.activeInHierarchy)
                    audio.Play();
            }
            // Invoke the event
            onTouch.Invoke();
        }
    }
}
```

Designing the Level

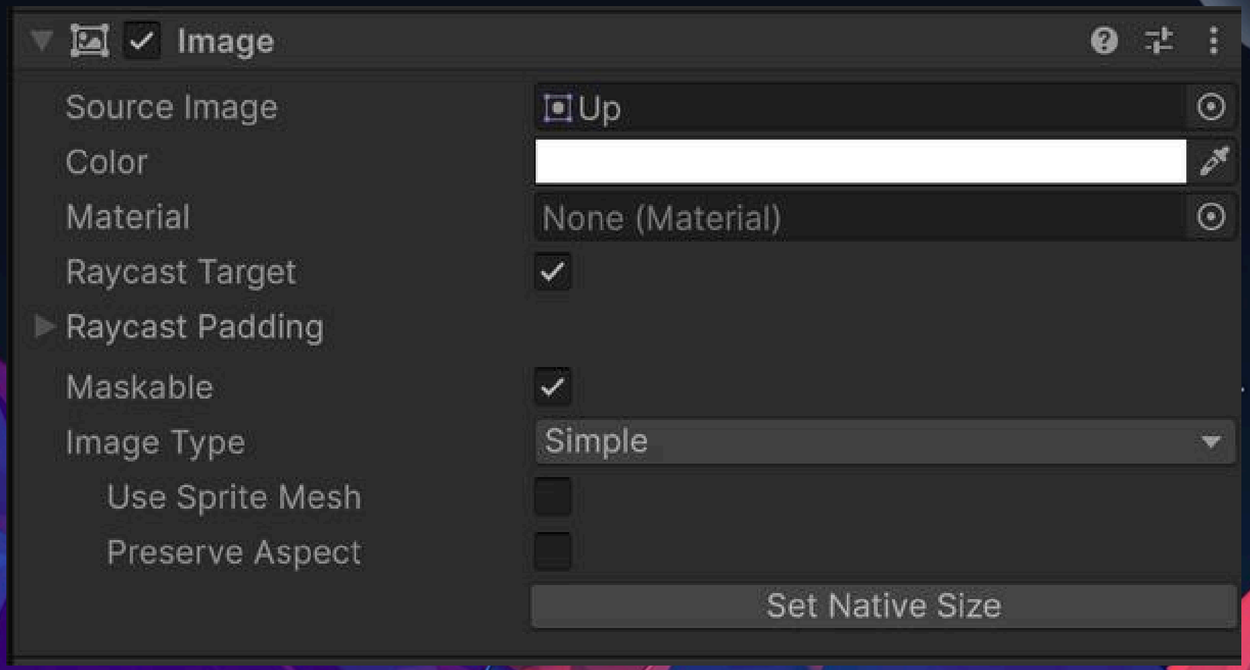
and create the treasure add
from the asset

add component to the treasure



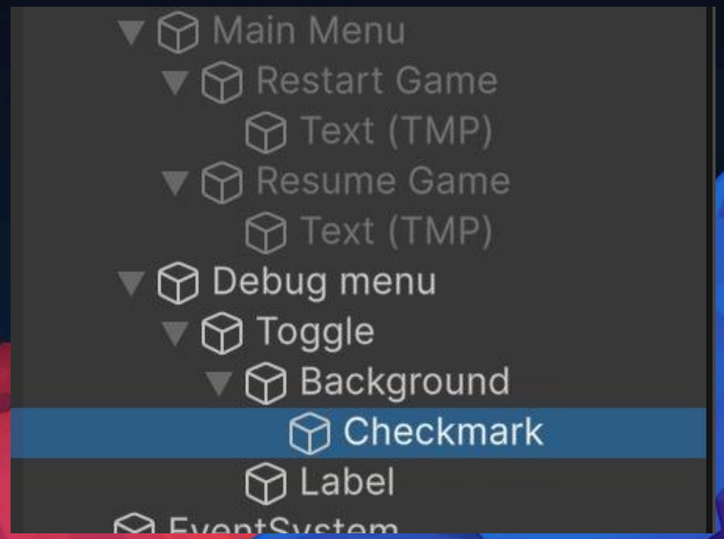
Setting Up Buttons

Click the 2 up, down gameobject button and remove the text and add this component and add the image from the asset folder

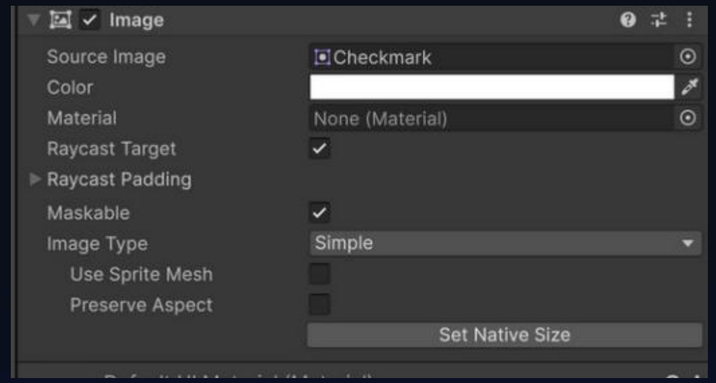


Game Mechanics

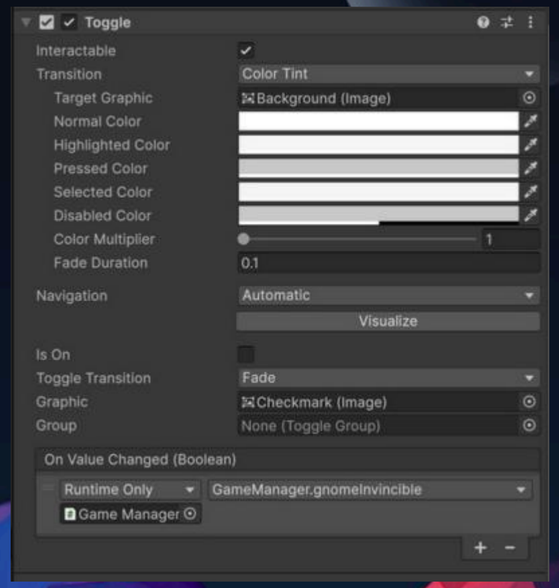
Add menu, invincible button and make the objects with parent and child objects



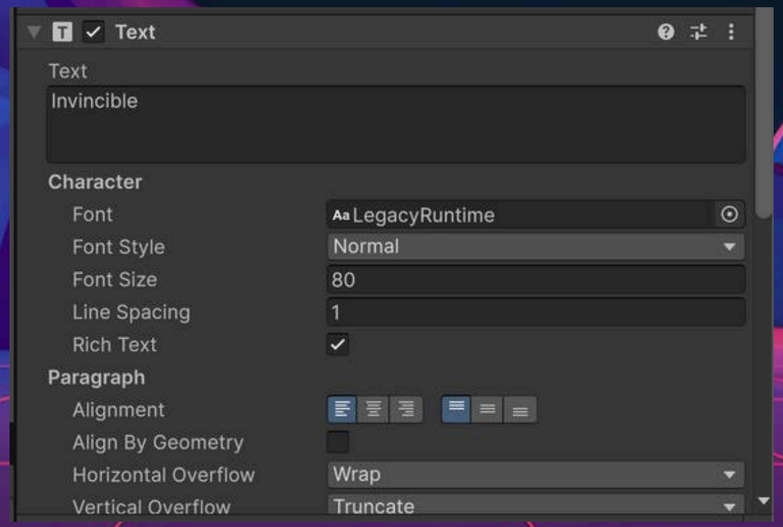
Add tis on the checkmark object



And put this on the toggle object

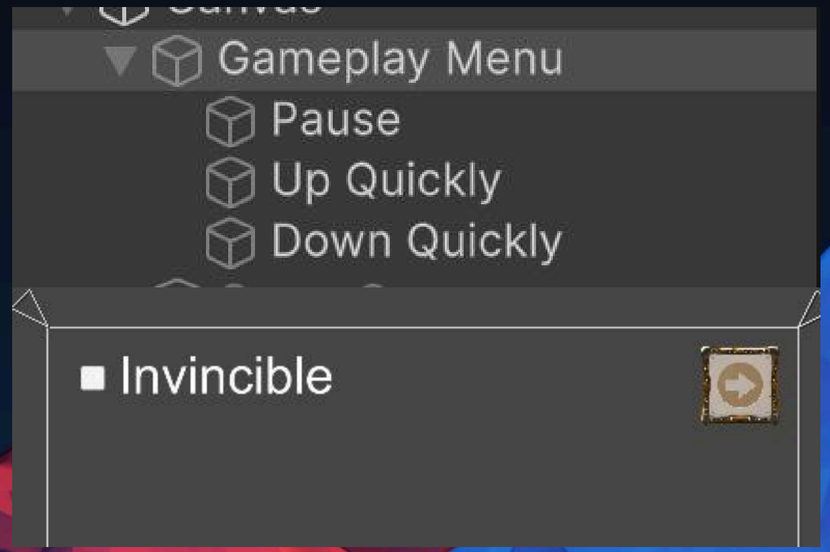


And modify level to this

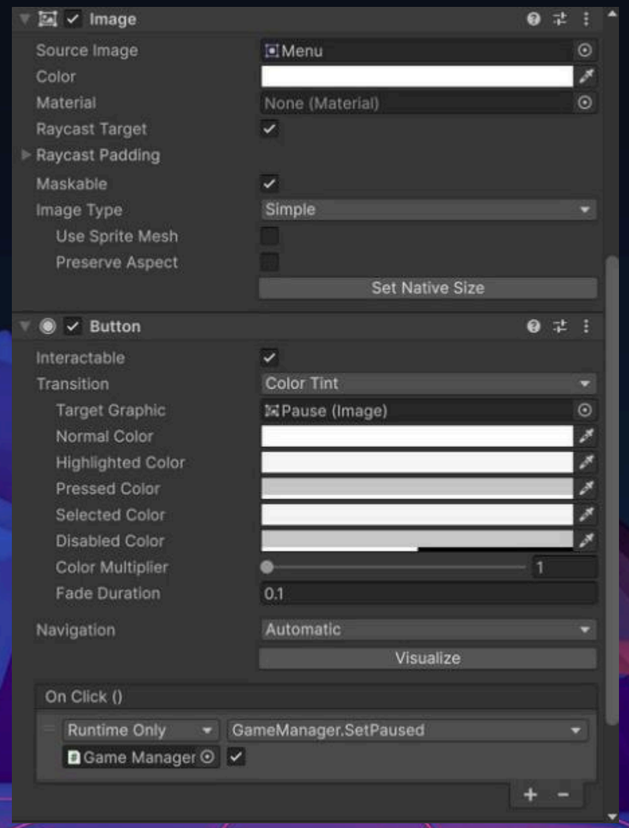


Game Mechanics

Add pause to the gameplay menu and move it to the top left



Add this components to the gameobject pause

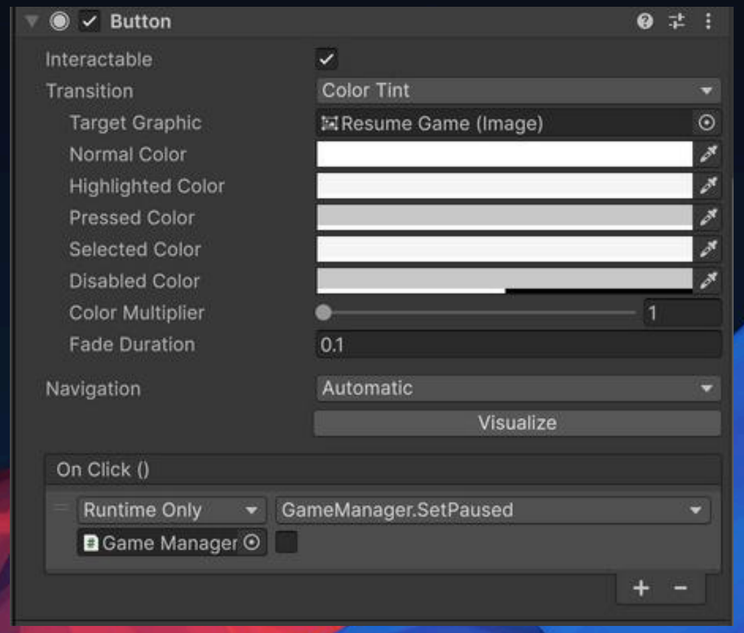


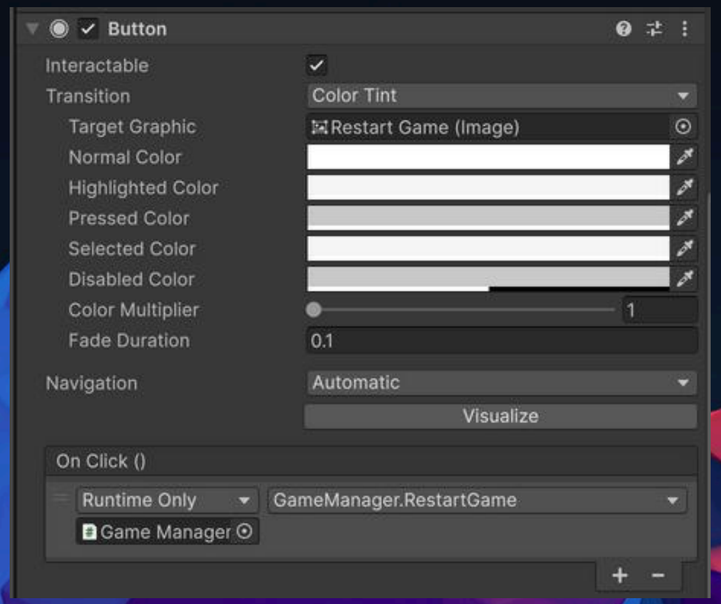
craete 2 objects named main menu and add this gameobjects will look like this

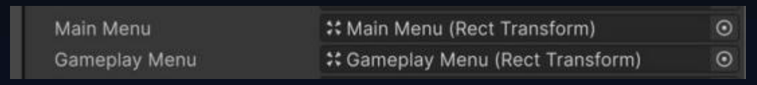
Creating End Screen

Add this 2 componnets

And set the game manager to main menu and gameplay menu

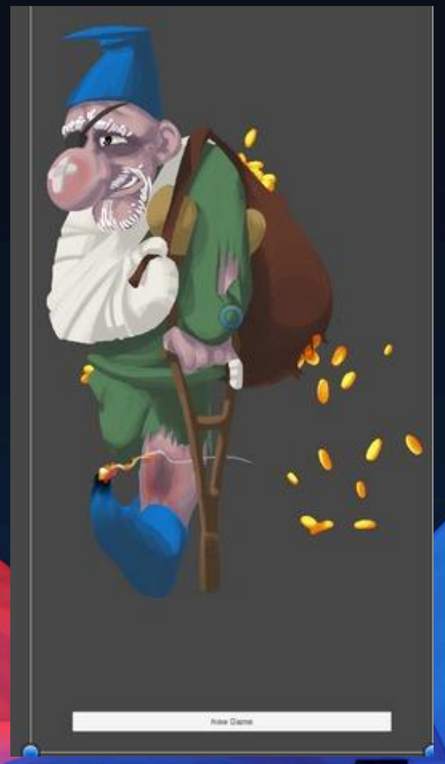




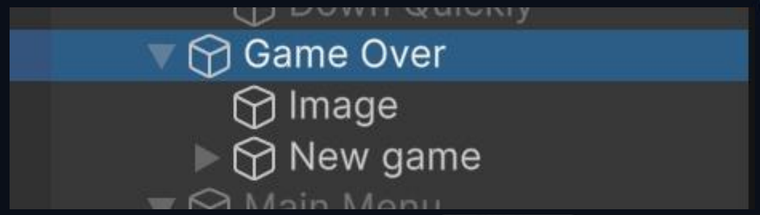


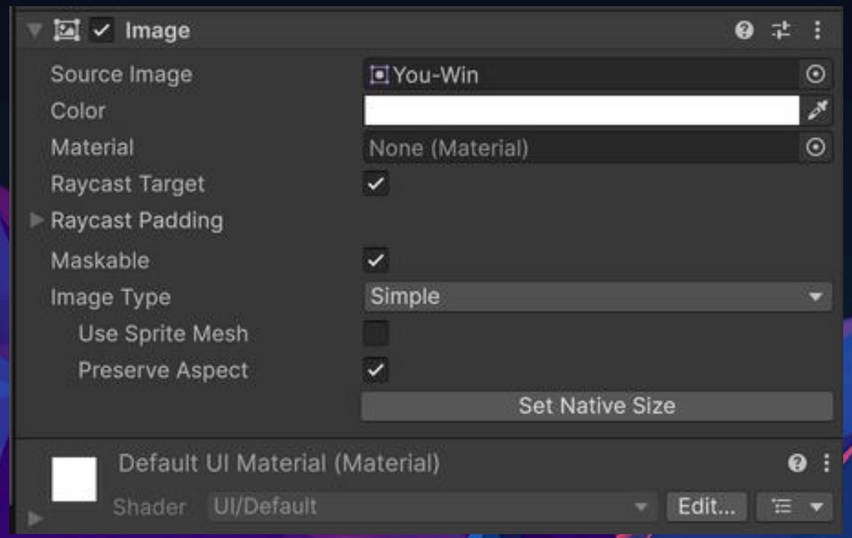
Creating End Screen

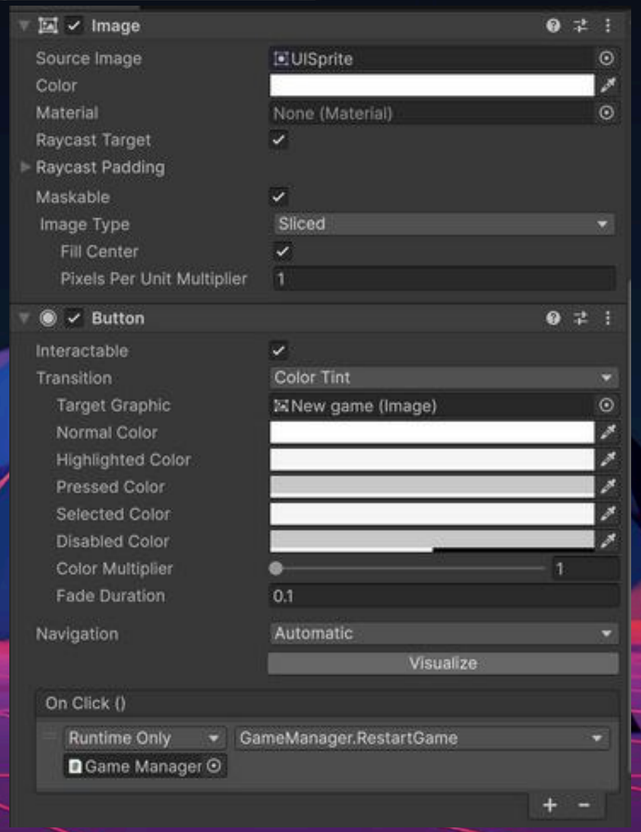
Create the game over screen



Create this 3 gameobjects

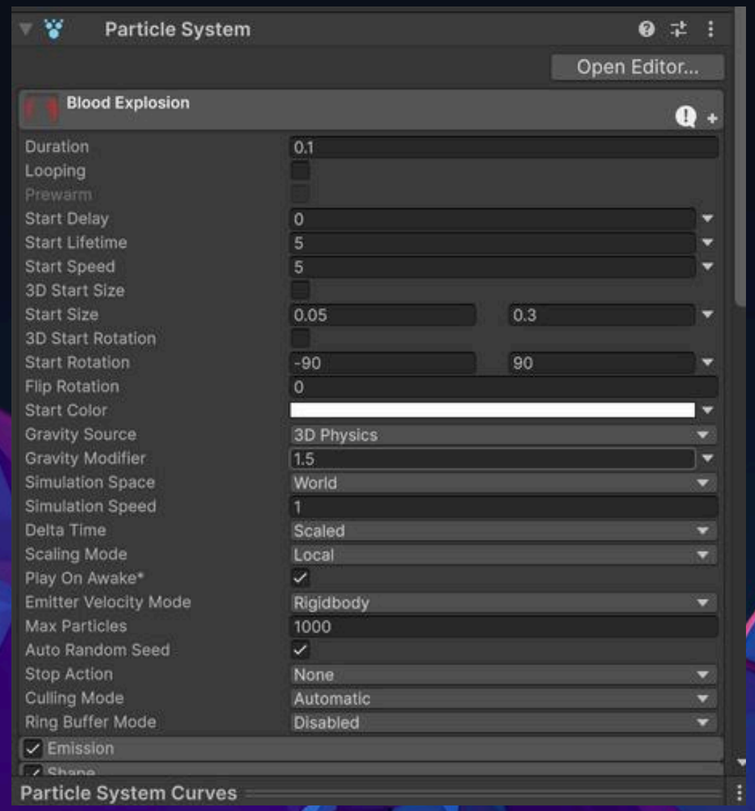


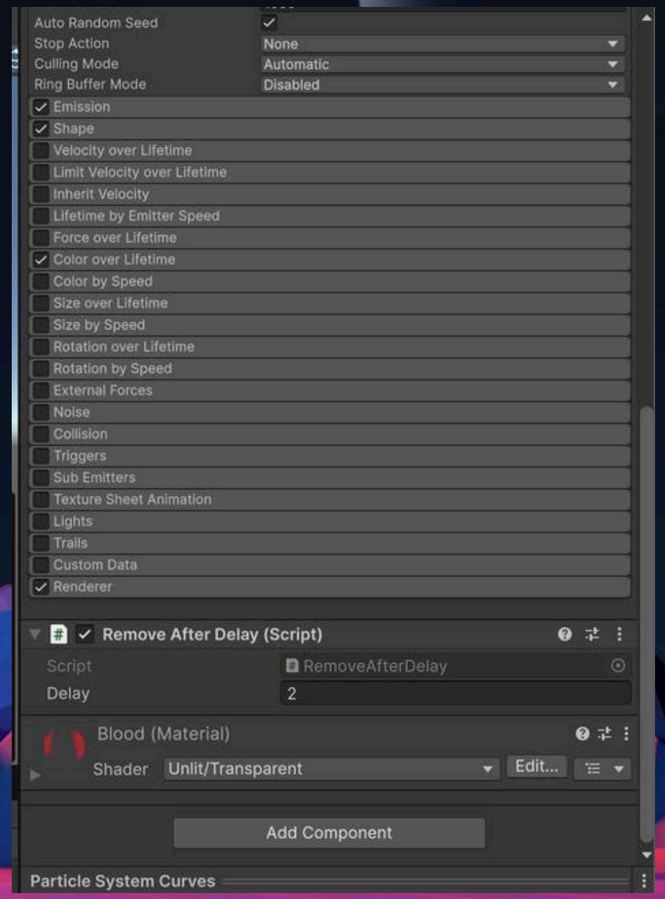




Setting Up Buttons

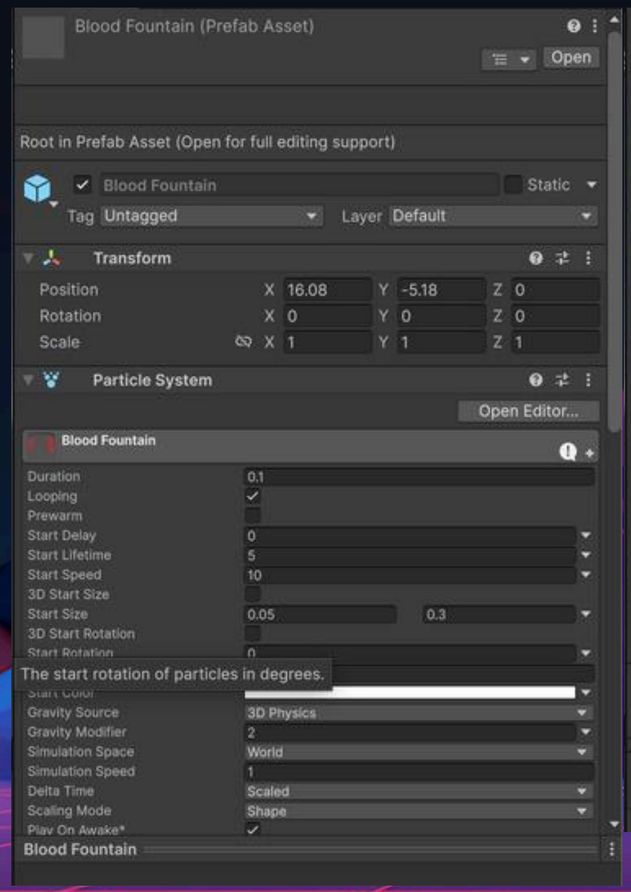
Create this prefabs called blood explosion and blood fountain into a particle system and copy this settings to blood explosion

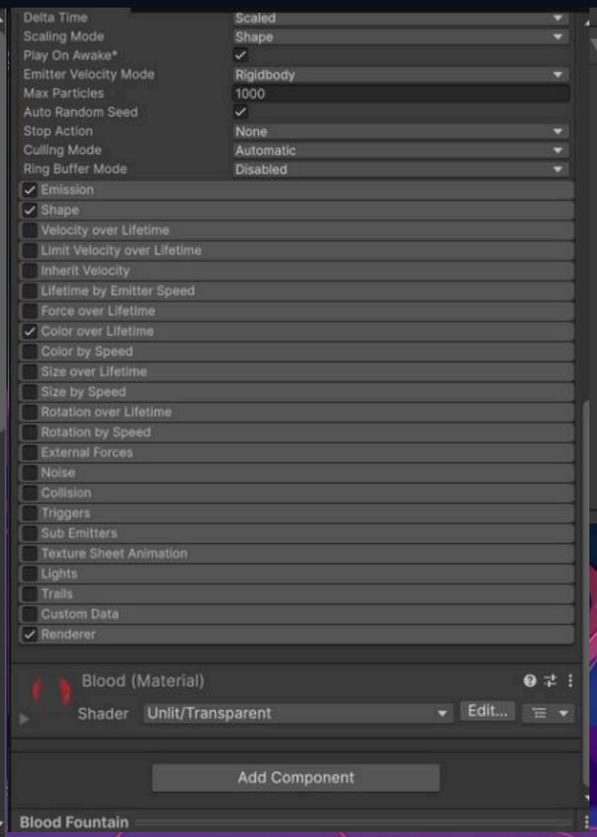


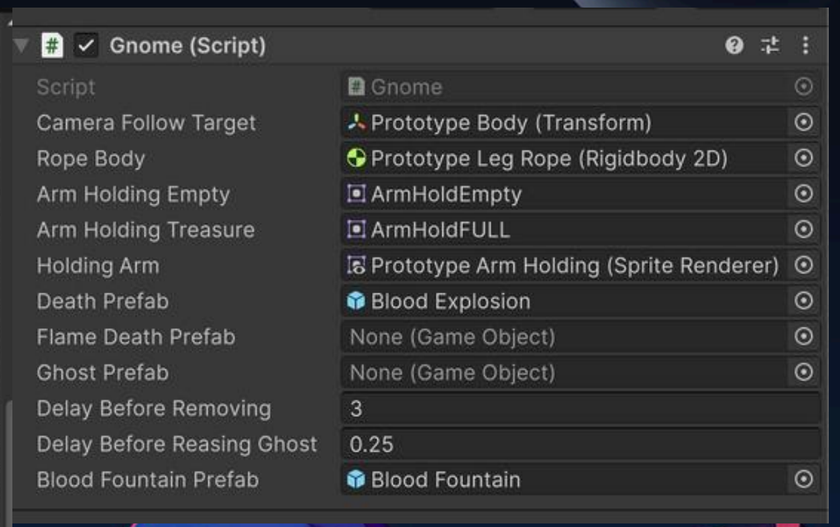


Setting Up Buttons

Copy this settings to blood fountain and update the gnome script in the Gnome prefab

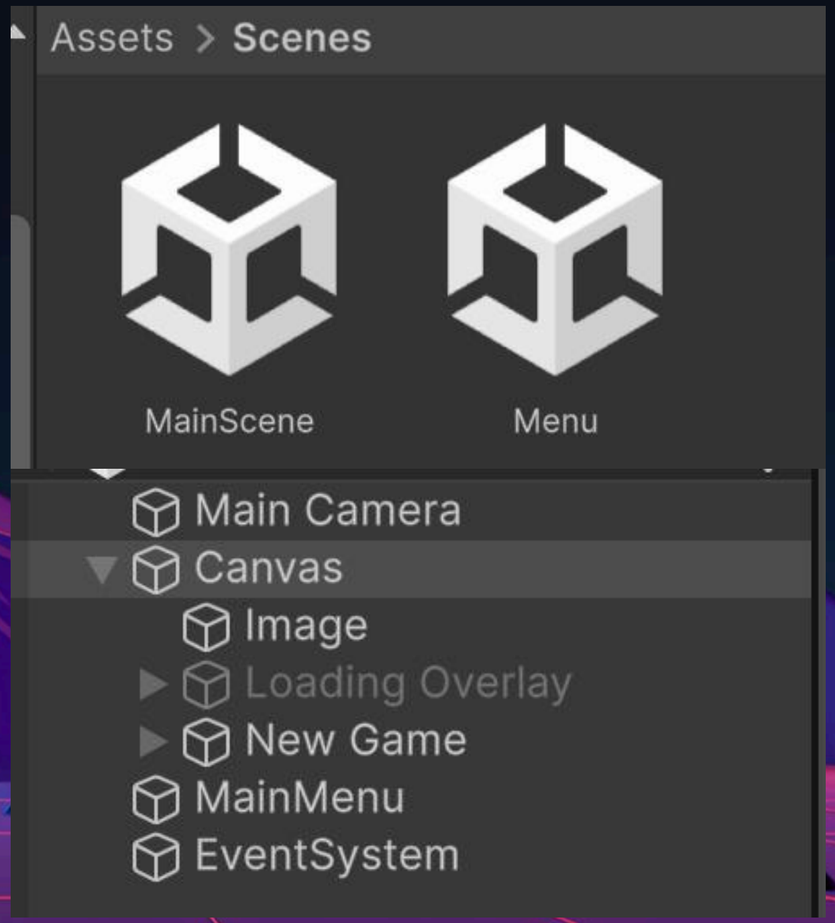






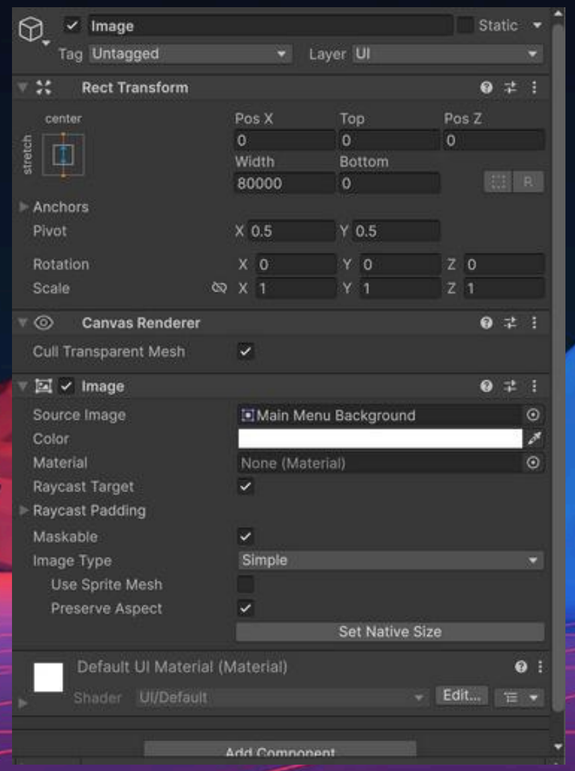
Game Main Menu

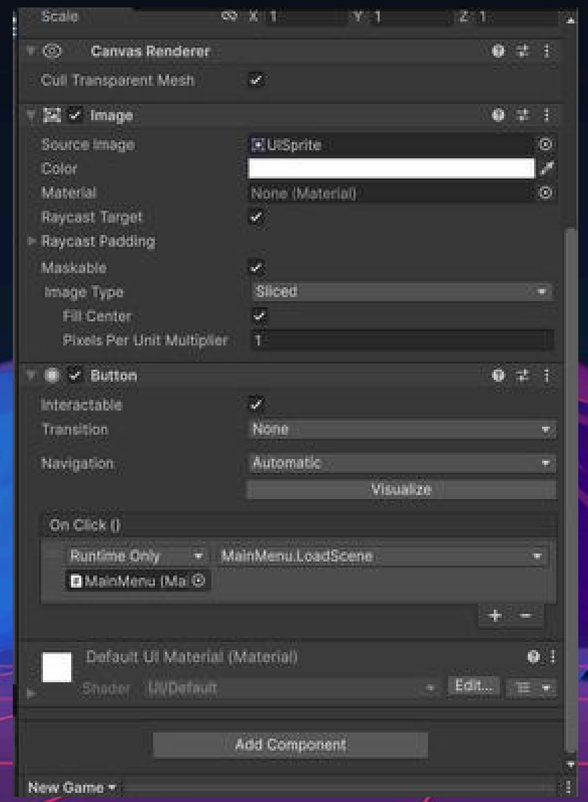
Create new scene named menu and make this objects

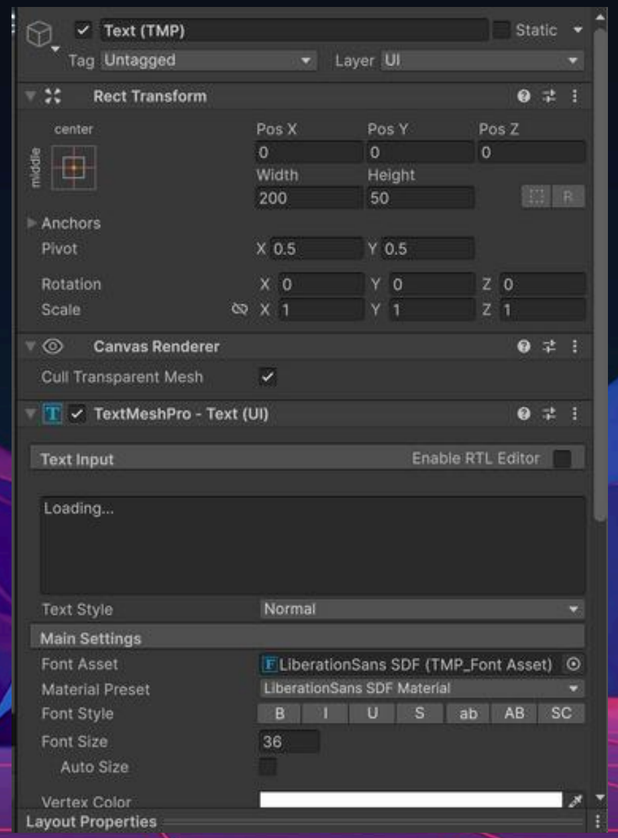


Game Main Menu

Make this image settings and in the pause game add a new game button and add this buttons



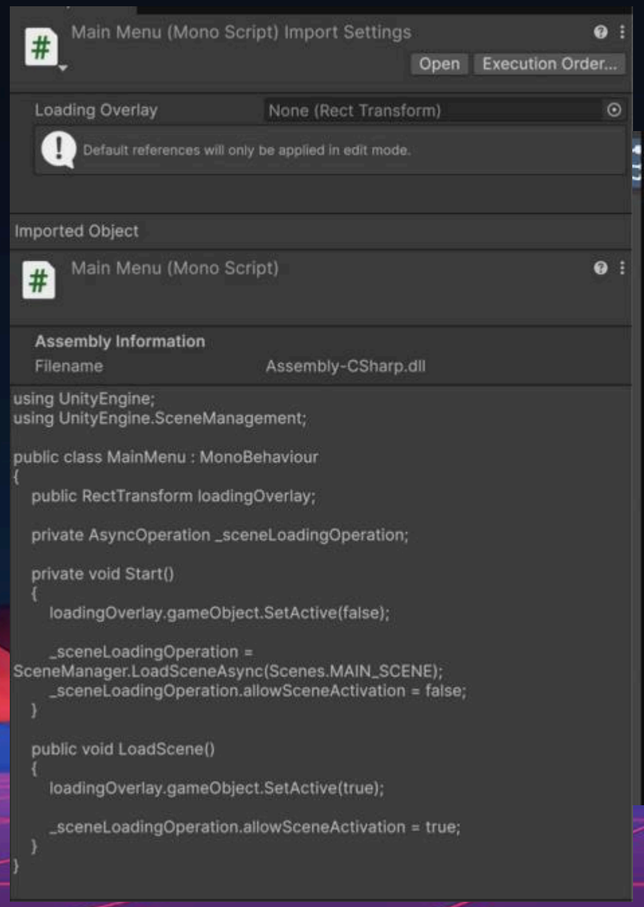


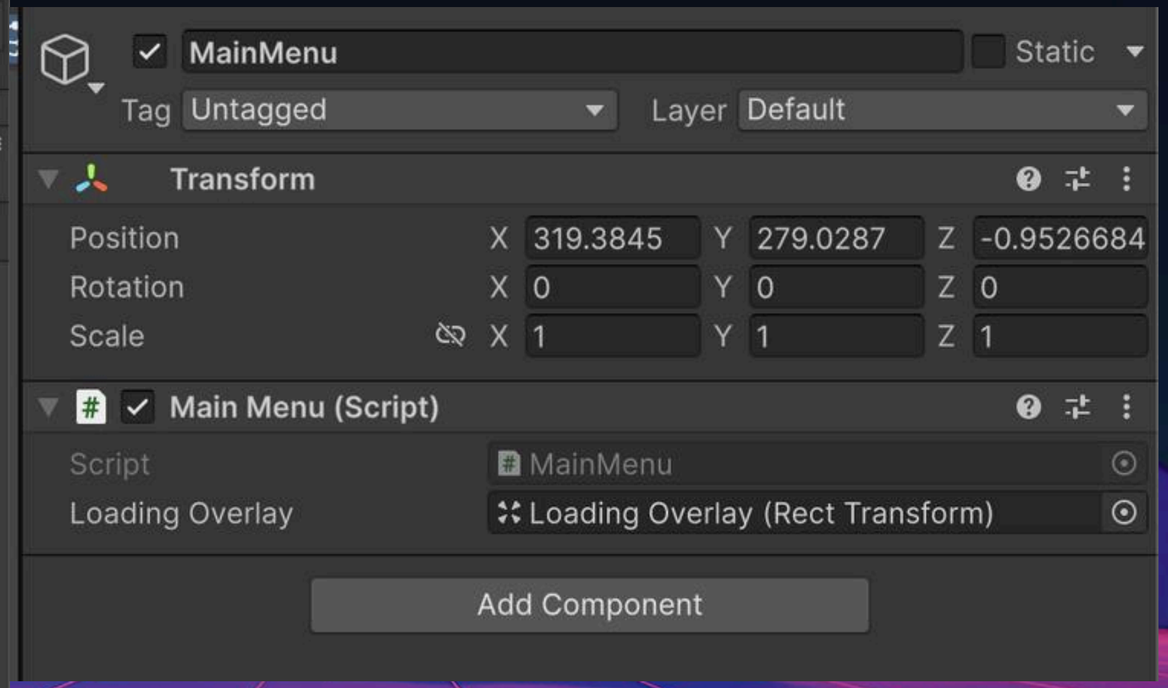


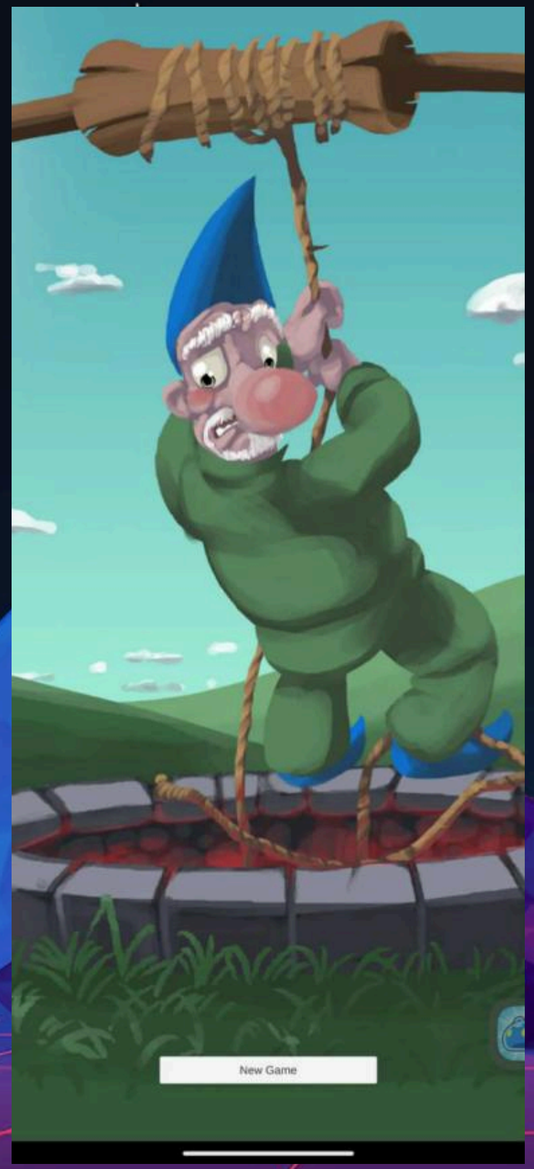
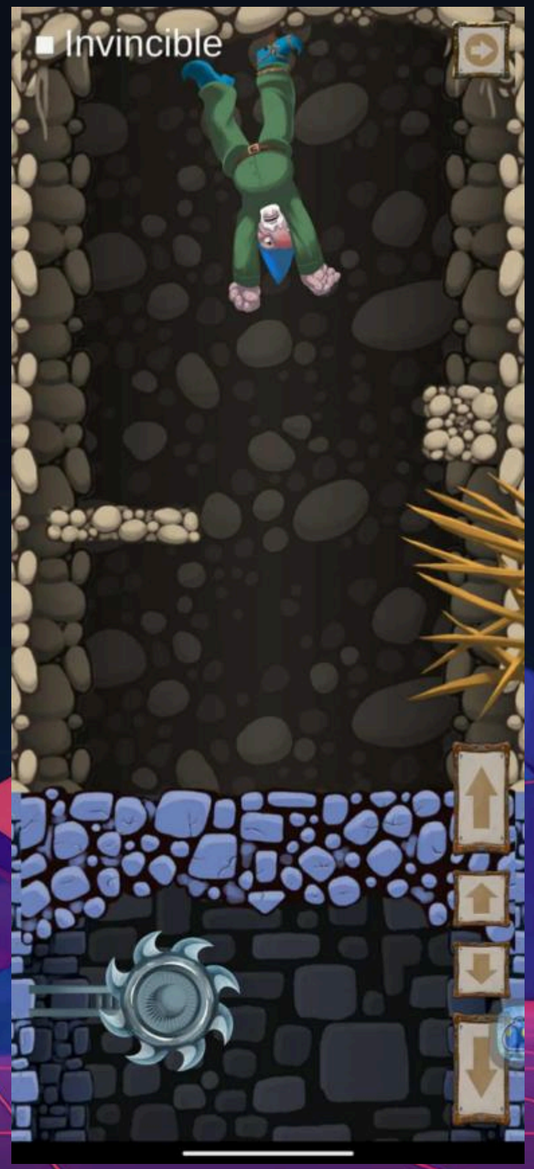
make this in the loading overlay text will look like this

Game Main Menu

copy paste main menu code and
move it on the main menu object





Final Output

There you go, now you've created a game