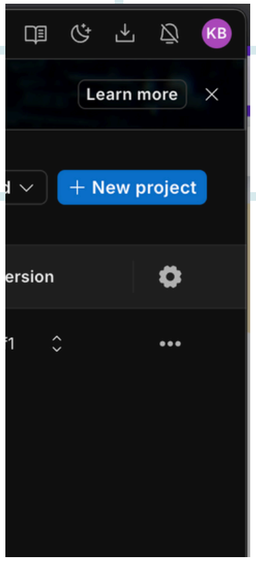
GNOME WELL STEP-BY-STEP

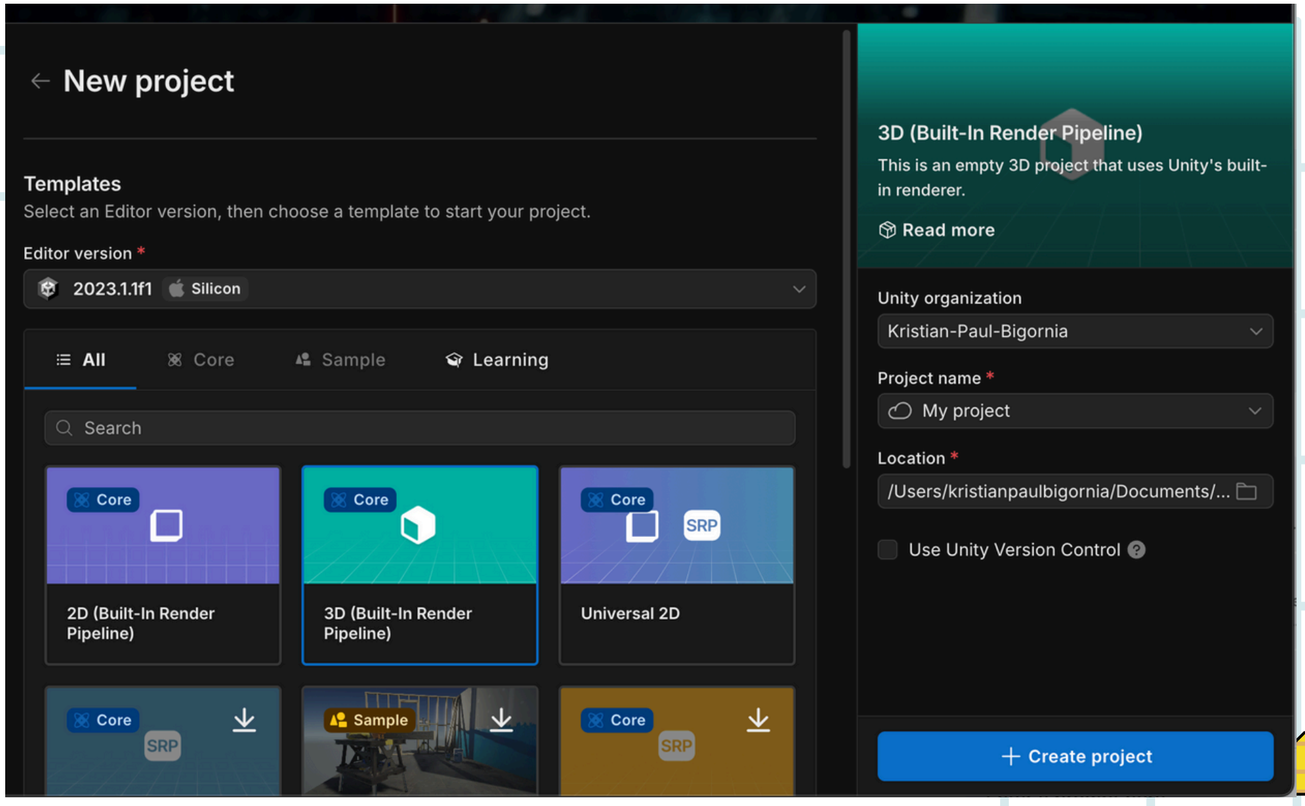
By:
Kristian Paul Bigornia
Calvin Klein Catalon
Dhann Angelo Guerero

STEP 1

First create a new project in unity

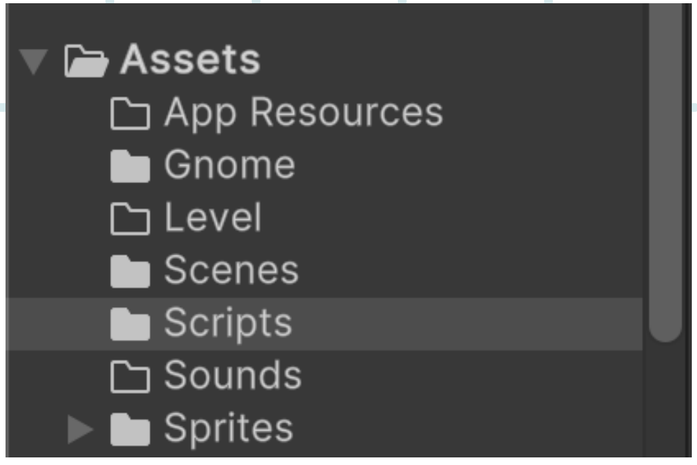
Then Select the Editor Version and The Core Render, We used the 2D.



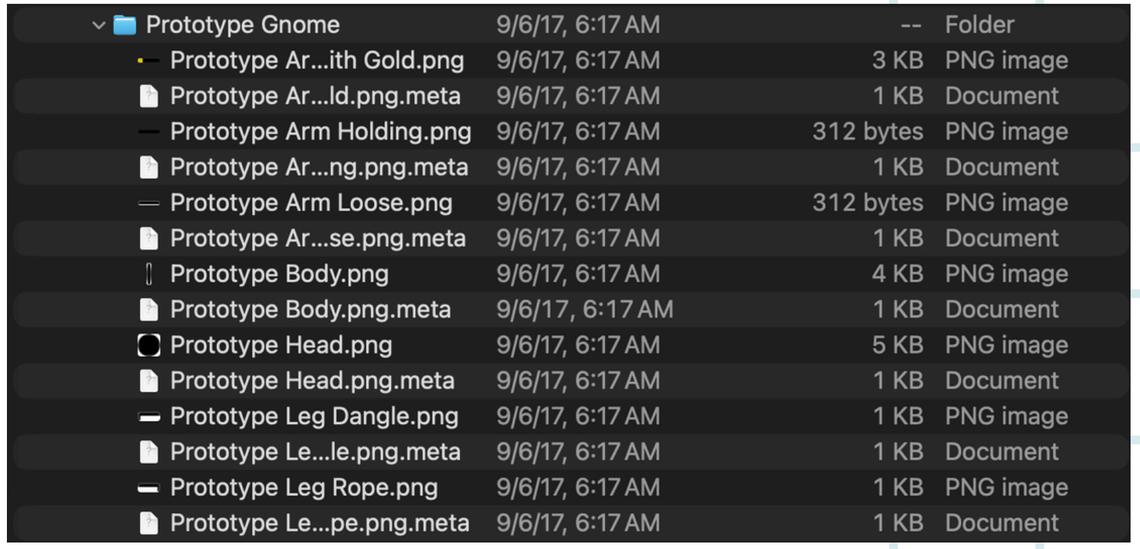


STEP 2

After Creating the Project, in the assets tab, we created this folders

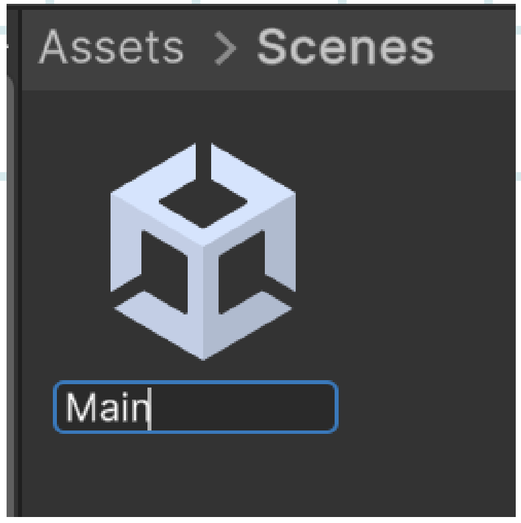


We then Downloaed the assets that came from the book

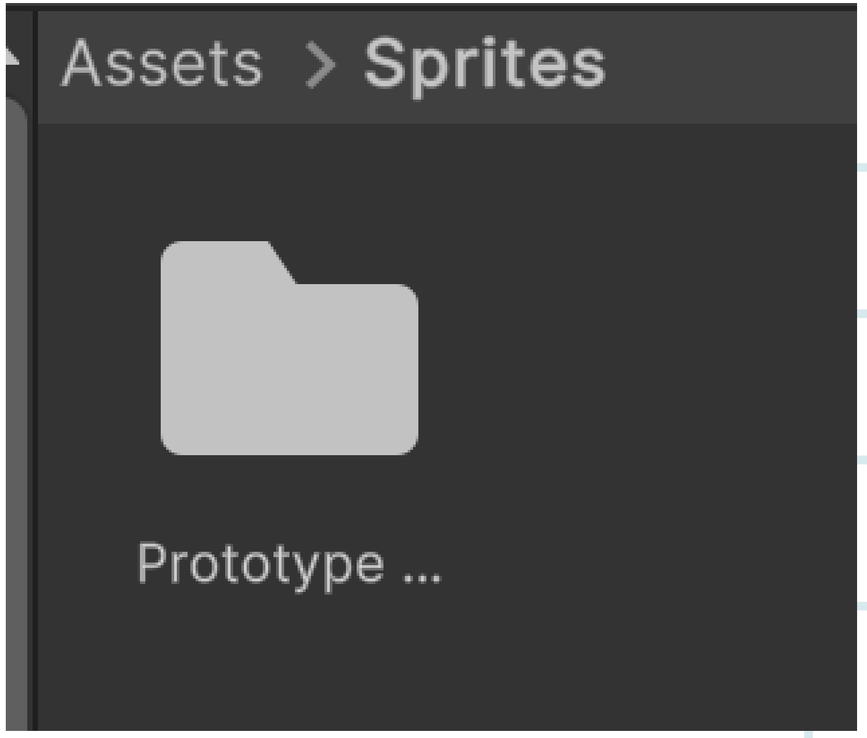


STEP 4

We then make sure that the Scene has a name which is Main

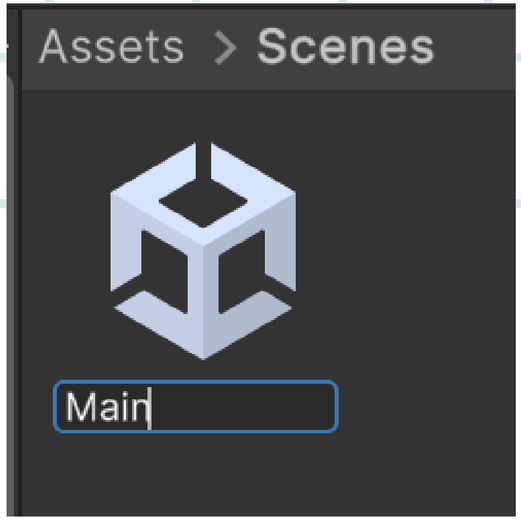


We then transfered the Prorotype Gnome Folder Asset to the Sprites folder in the asset tab

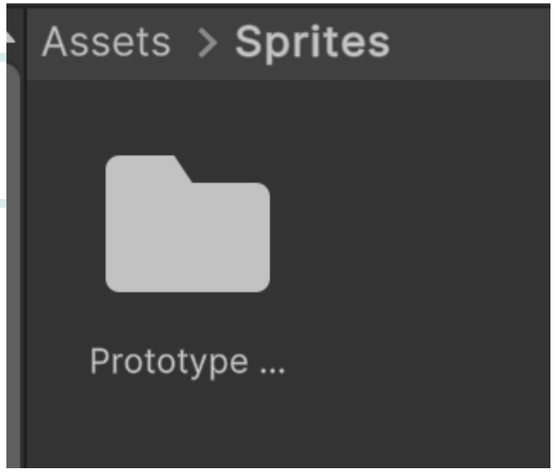


STEP 5

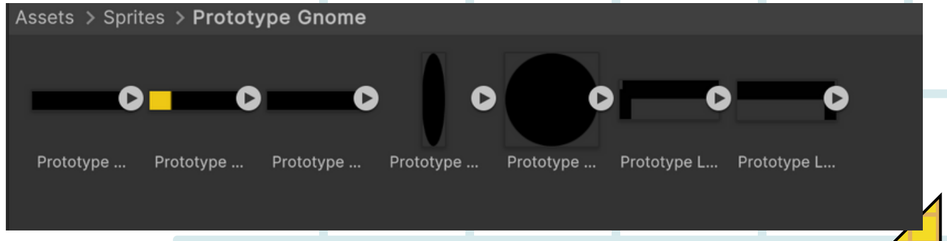
We then make sure that the Scene has a name which is Main



We then transfered the Prorotype Gnome Folder Asset to the Sprites folder in the asset tab

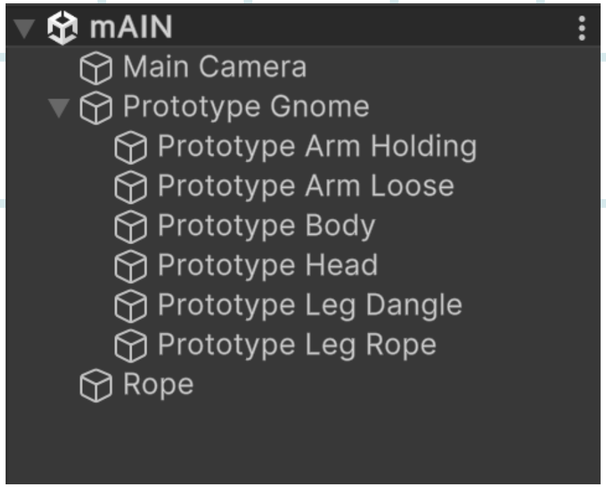


It Will look like this

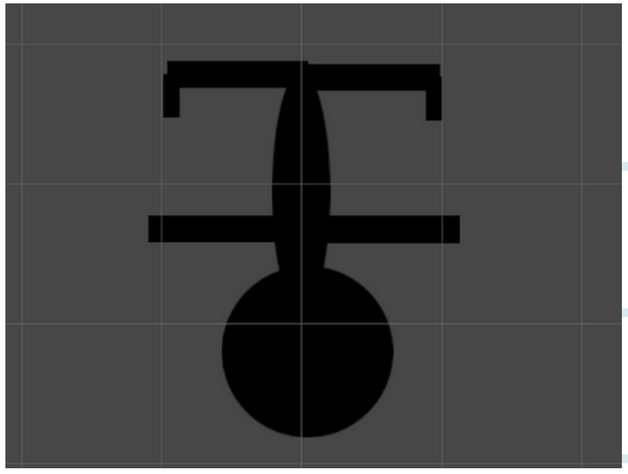


STEP 6

We then created this objects , The prototype Gnome has the prorotypes assets inside of it



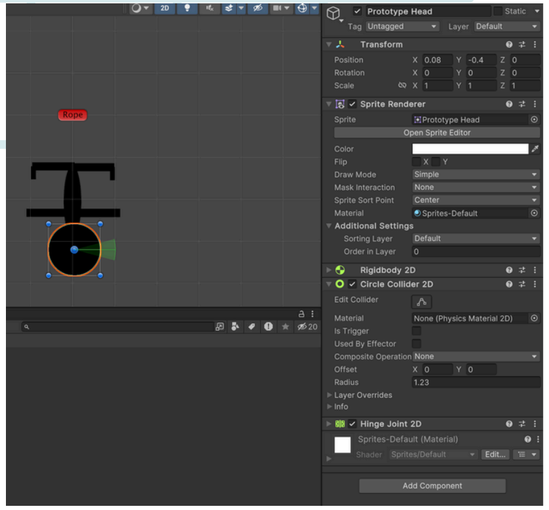
We then moved the parts to make it look like this



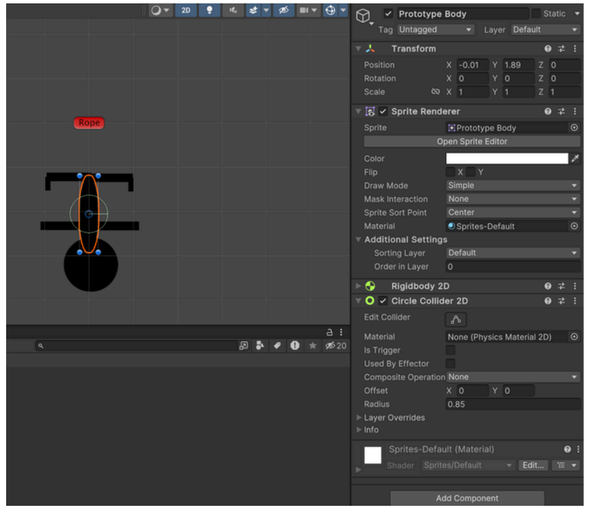
STEP 7

We clicked each part of the body and added components

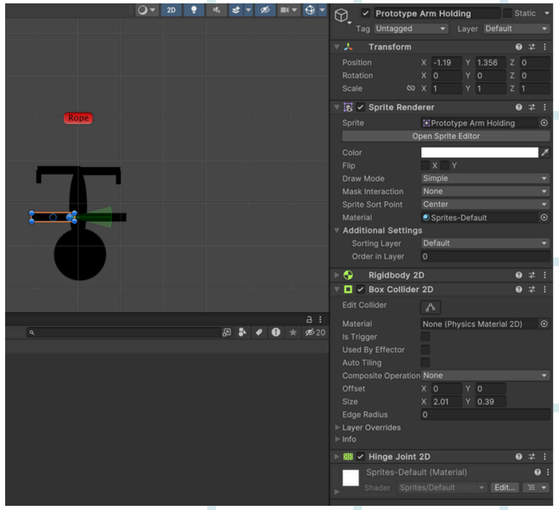
Prototype Head



Prototype Body



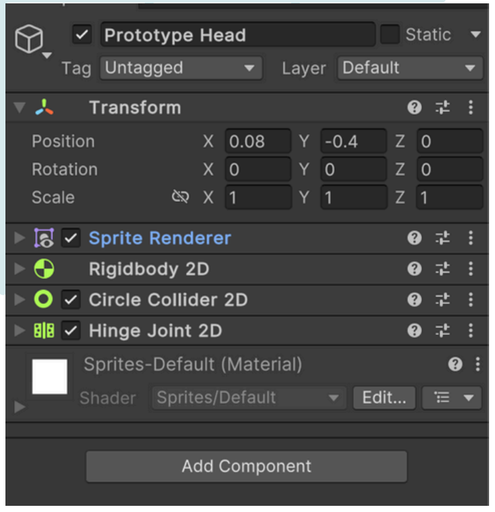
Prototype Arm Holding



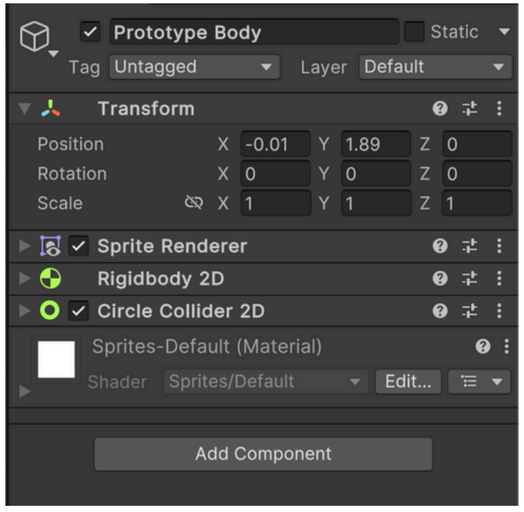
STEP 7

We clicked each part of the body and added components

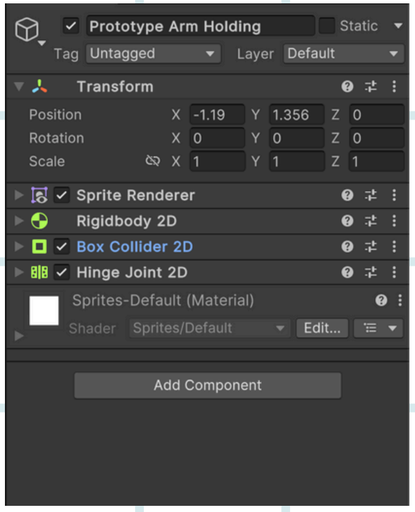
Prototype Head



Prototype Body



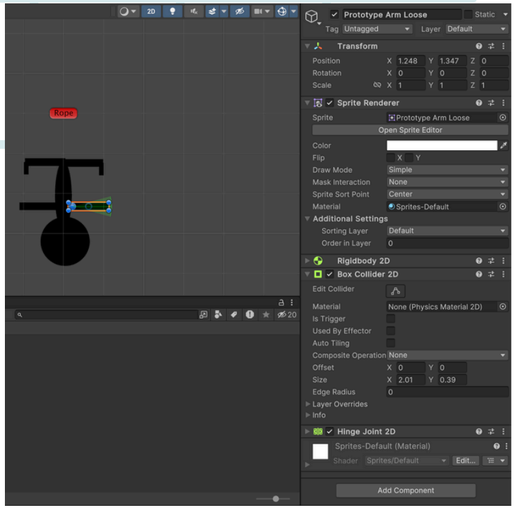
Prototype Arm Holding



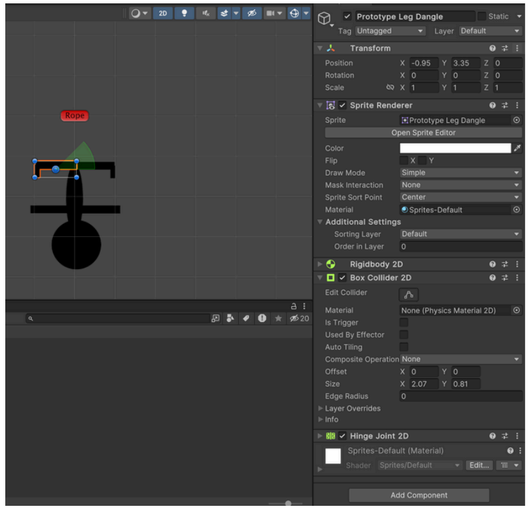
STEP 7

We clicked each part of the body and added components

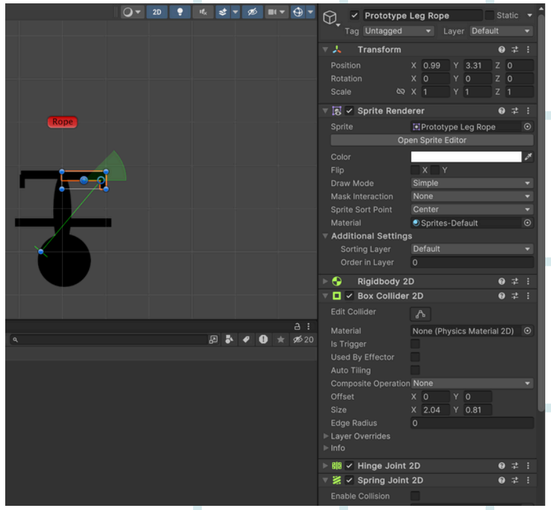
Prototype Arm Loose



Prototype Leg Dangle



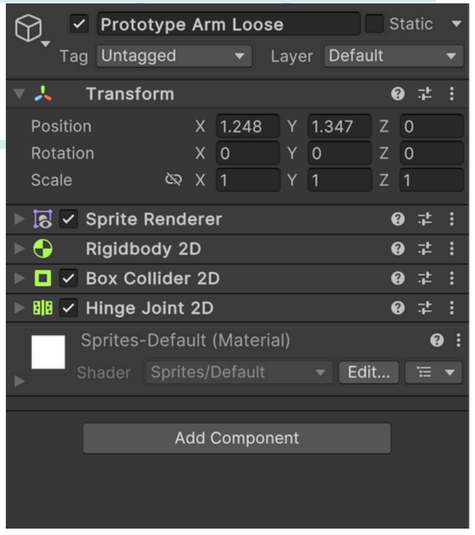
Prototype Leg Rope



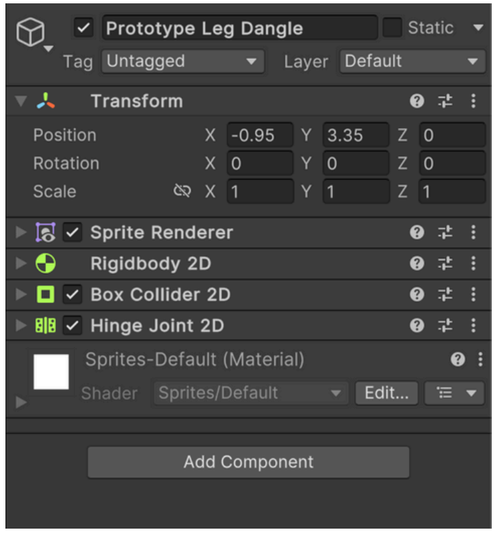
STEP 7

We clicked each part of the body and added components

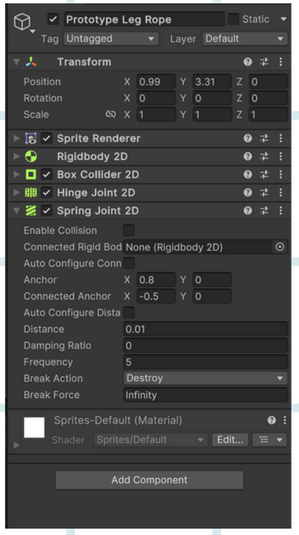
Prototype Arm Loose



Prototype Leg Dangle



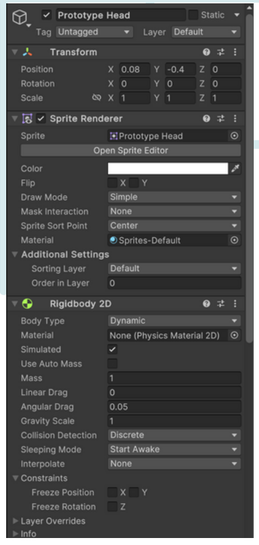
Prototype Leg Rope

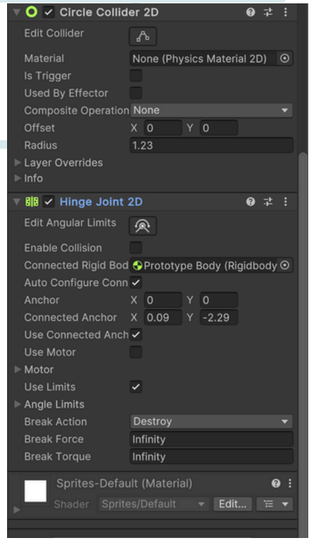


STEP 7.5

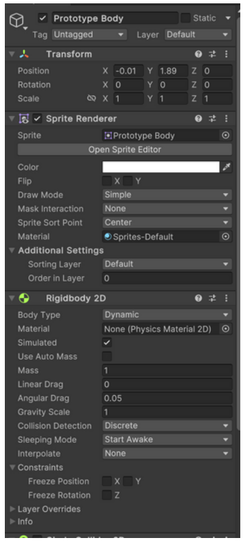
And Each Component with these settings

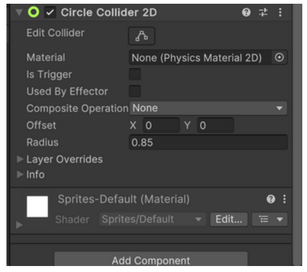
Prototype Head



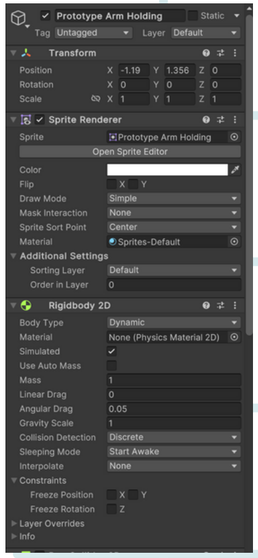


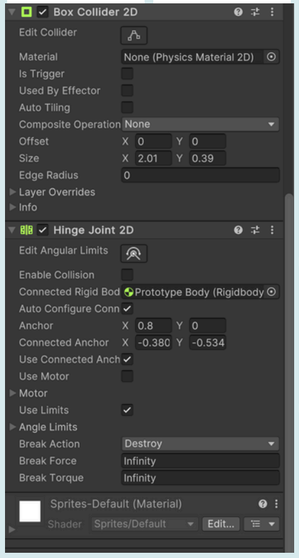
Prototype Body





Prototype Arm Holding

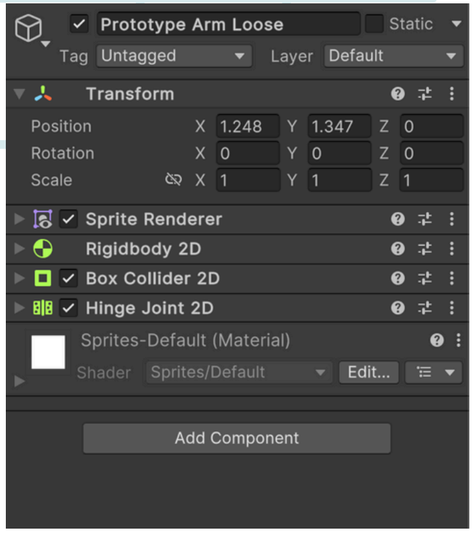




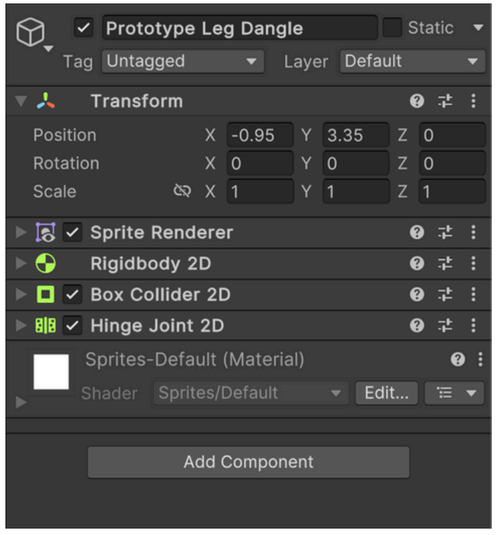
STEP 7.5

We clicked each part of the body and added components

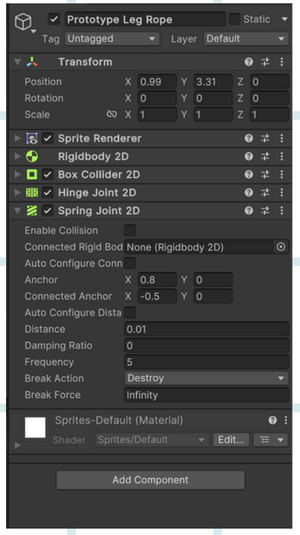
Prototype Arm Loose



Prototype Leg Dangle



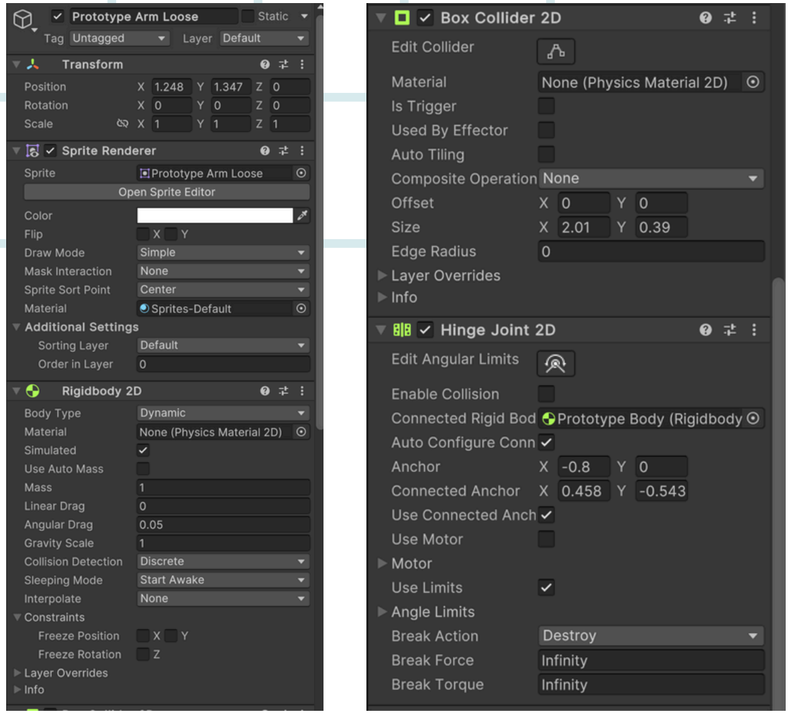
Prototype Leg Rope



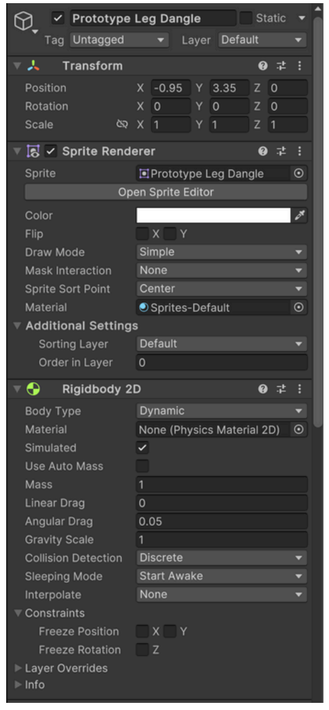
STEP 7.5

We clicked each part of the body and added components

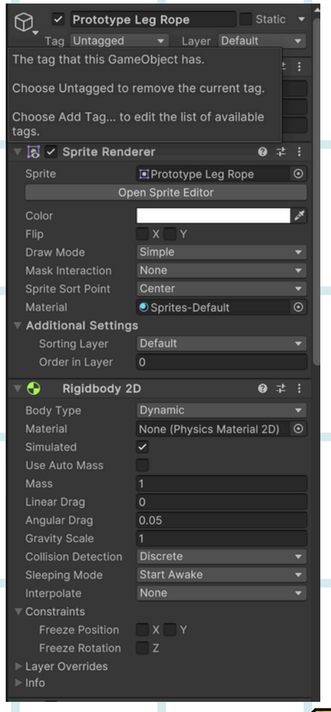
Prototype Arm Loose

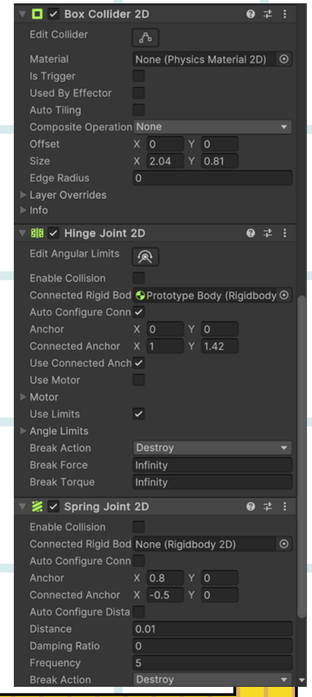


Prototype Leg Dangle



Prototype Leg Rope

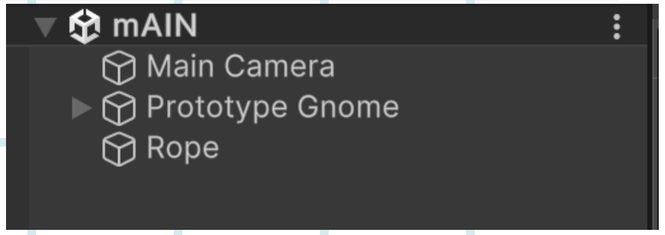




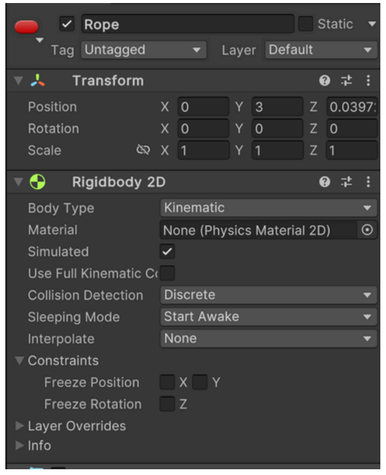
STEP 8

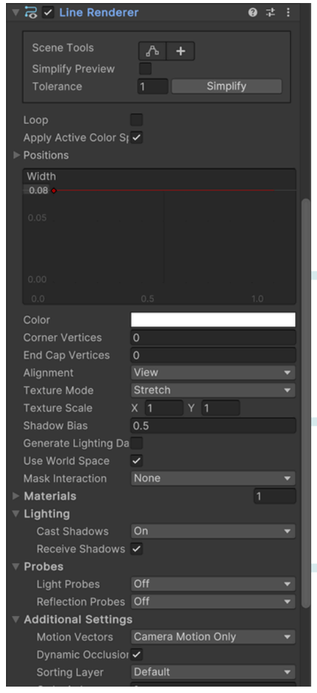
Configure the rope physics

Add Rope Object

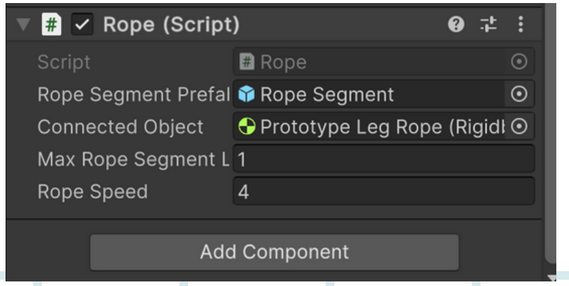


Add these settings





Add this C# code to make it work

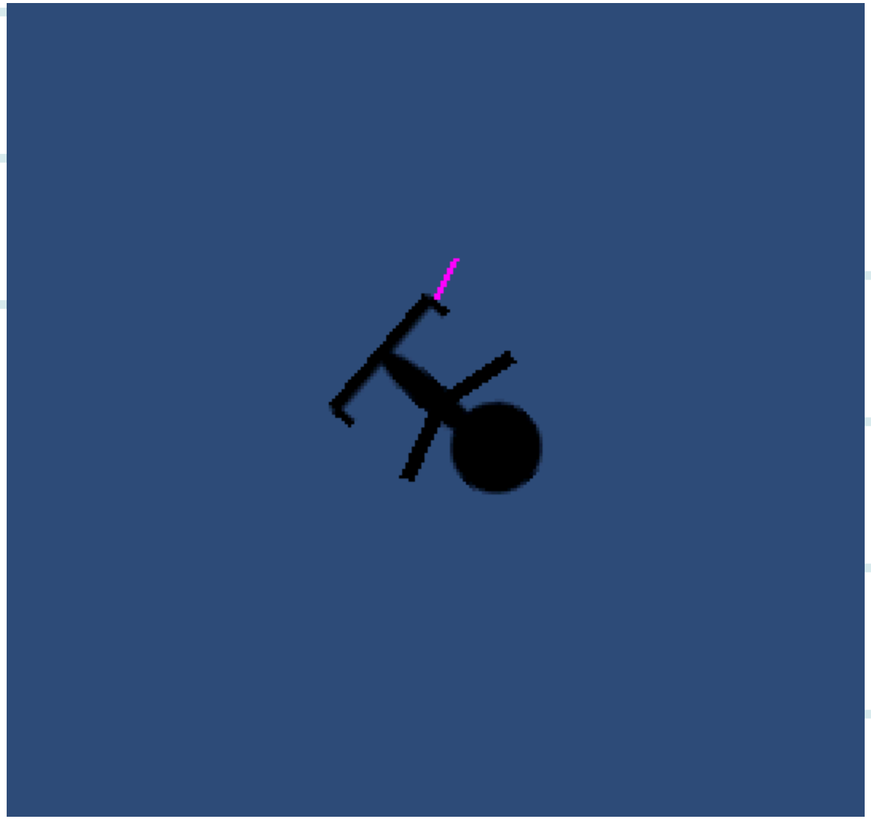


With these code

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
// The connected rope.
public class Rope : MonoBehaviour
{
    // The Rope Segment prefab to use.
    public GameObject ropeSegmentPrefab;
    // Contains a list of Rope Segment objects.
    List<GameObject> ropeSegments = new
List<GameObject>();
    // Are we currently extending or retracting the
rope?
    public bool isIncreasing { get; set; }
    public bool isDecreasing { get; set; }
    // The rigidbody object that the end of the rope
    // should be attached to.
    public Rigidbody2D connectedObject;
    // The maximum length a rope segment should be
(if we
    // need to extend by more than this, create a new
rope
    // segment).
    public float maxRopeSegmentLength = 1.0f;
    // How quickly we should pay out new rope.
    public float ropeSpeed = 4.0f;
    // The LineRenderer that renders the actual rope.
    LineRenderer lineRenderer;
    void Start()
    {
Asset Labels
```

STEP 9

Connect evrything and run the game

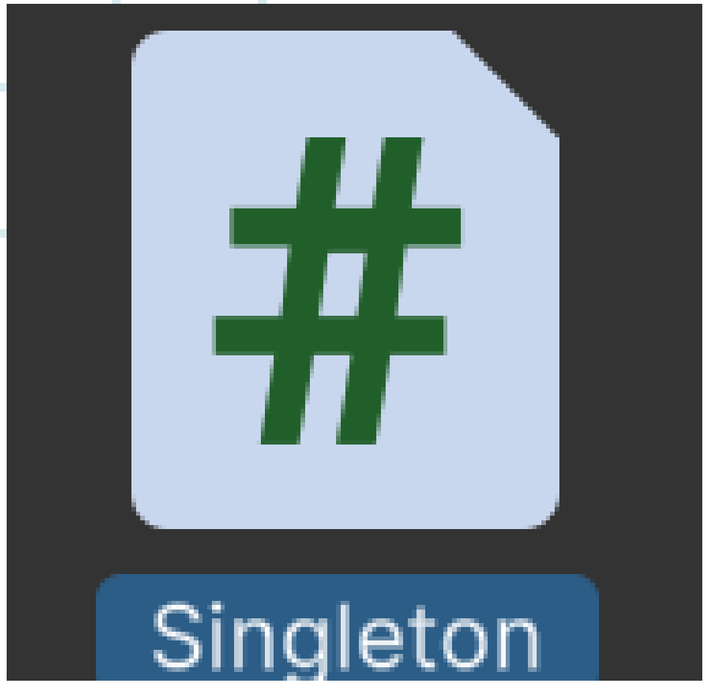


PROTOTYPE

These Steps are to create a prototype of the game Gnomes Well, Where we learned that this simple components and sprites are what games are and the endless possibilities we can do to improve the game.

CREATING BUTTONS

Create another cs file names singleton



and add this code

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;
    }
  }
}
```

CREATING BUTTONS

Create an object named inoutmanager and put this code

Input Manager (Mono Script) Import Settin

Open | Execution Order...

Imported Object

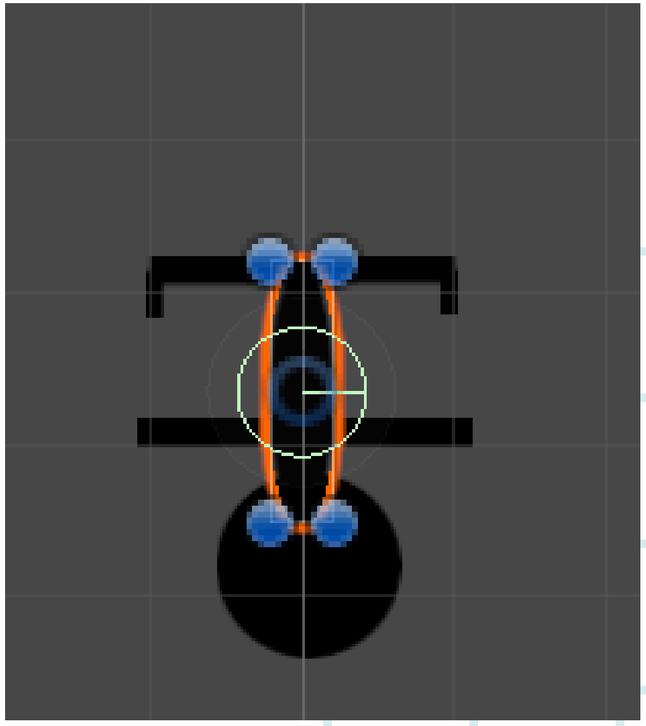
Input Manager (Mono Script)

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;
    }
}
```

Select the body of the gnome and add a component cs file and put this code



Imported Object

Swinging (Mono Script)

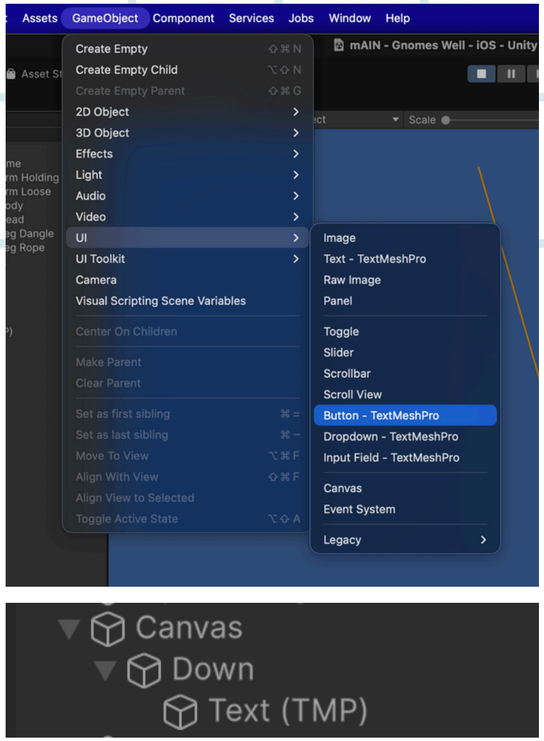
Assembly Information

Filename Assembly-CSharp.dll

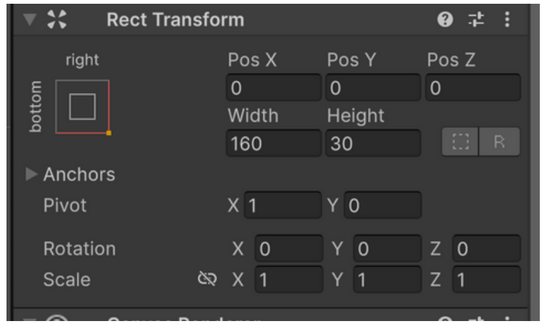
```
using UnityEngine;
using System.Collections;
// Uses the input manager to apply sideways forces to an
// object. Used to make the gnome swing side-to-side.
public class Swinging : MonoBehaviour
{
    // How much should we swing by? Bigger numbers =
more
    // swing
    public float swingSensitivity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

CREATING BUTTONS

Add new gameobject called button and rename it to Down

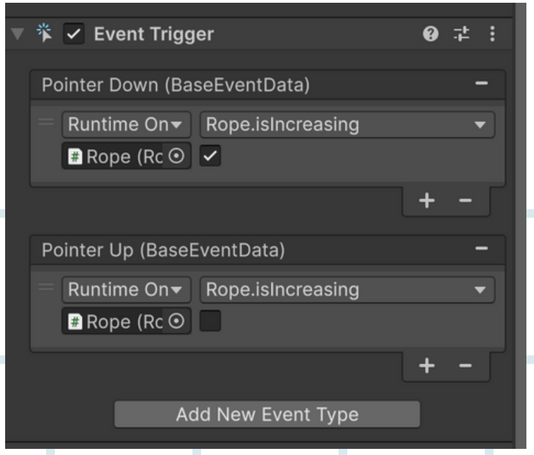


Click down and configure this



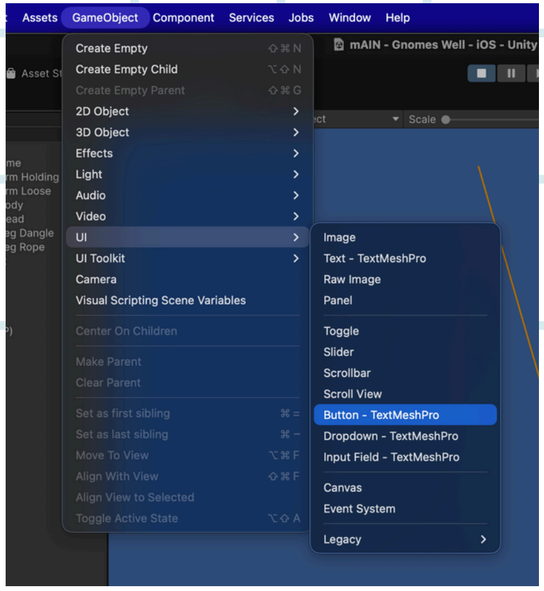
Change the text child of the button to Down

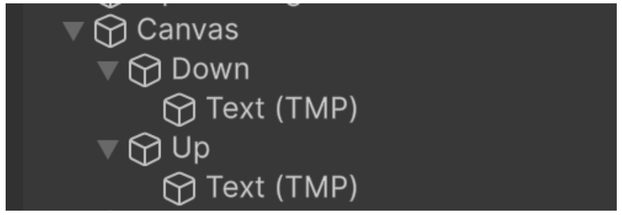
Add Event Trigger component and add this event type



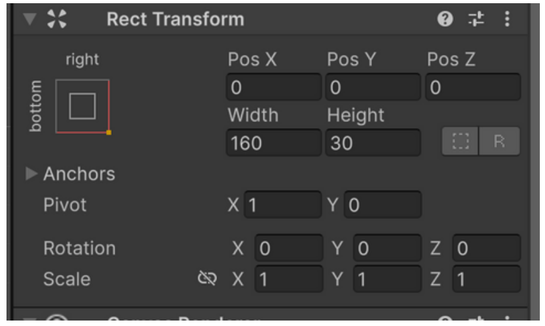
CREATING BUTTONS

Add new gameobject called button and rename it to Down and up





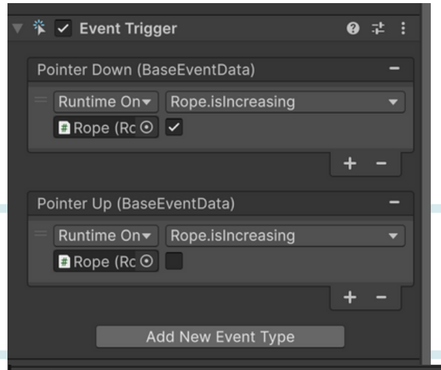
Click down and configure this



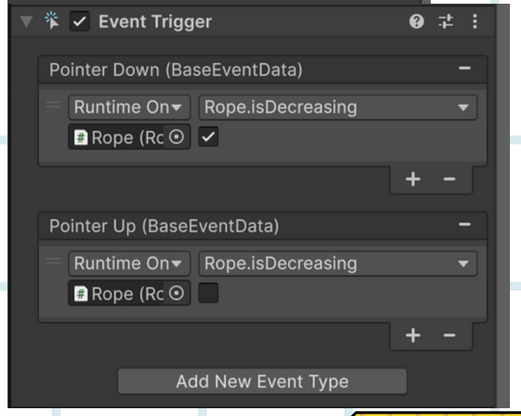
Change the text child of the button to Down and up

Add Event Trigger component and add this event type for both up and down

down



up



MAKING THE CAMERA FOLLOW

ADD this code in the 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);
    }
}
```

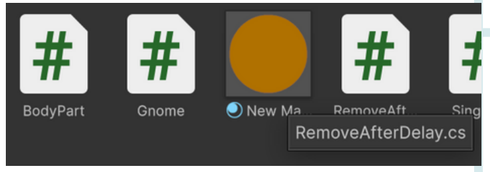
ADD this code in the Gnome.cs

Gnome ▸ ReleaseGhost()

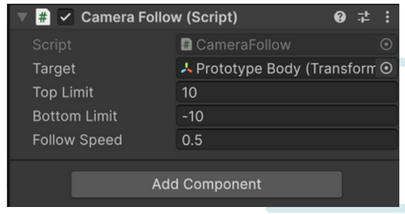
```
using System.Collections;
using UnityEngine;
public class Gnome : MonoBehaviour
{
    // The object that the camera should follow.
    public Transform cameraFollowTarget;
    public RigidBody2D ropeBody;
    public Sprite armHoldingEmpty;
    public Sprite armHoldingTreasure;
    public SpriteRenderer 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, 2) == 0;
                if (shouldBurn)
                {
                    part.ApplyDamageSprite(type);
                }
                break;
            case DamageType.Slicing:
                // Slice damage always applies a damage sprite
                part.ApplyDamageSprite(type);
                break;
        }
        // 1 in 3 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
                    );
                    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
        );
    }
}
```

Create this 3 cs files called BodyPart,Gnome and Removeafter delay



Add Camera Follow script on the main camera object AND DRAg prototype body in the target



Add this 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;
        }
    }
}
```

CONFIGURING THE GAME

Add this to the RemoveAfterDelay.cs

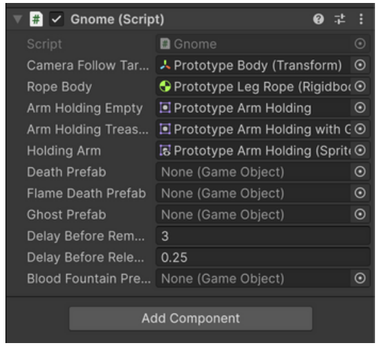
RemoveAfterDelay ▸ Remove()

```
// 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.
    }
}
```

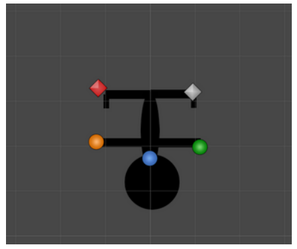
Select all the body parts and put the BodyPart.cs

Select Prototype Gnome and put the Gnome.cs

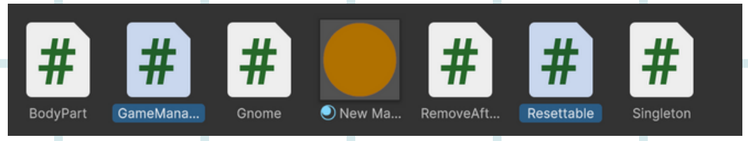
Configure Gnome.cs



Configure the 5 Blood Fountain Objects and put it in their respective places



Create 2 More cs names GameManager and resettable



Put this in the Resettable

```
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();
    }
}
```

CONFIGURING THE GAME

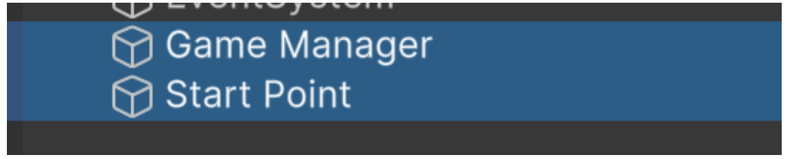
Add this code to the game manager

```
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
        // reset
        var resetObjects = FindObjectsOfType<Resettable>();


        foreach (Resettable r in resetObjects)
        {
            r.Reset();
        }
        // Make a new gnome
        CreateNewGnome();
        // Un-pause the game
        Time.timeScale = 1.0f;
    }
        // 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";
            }
            // Mark ourselves as not currently having a
            // gnome
            currentGnome = null;
        }
    }
    // Kills the gnome.
    void KillGnome(Gnome.DamageType damageType)
    {
        // If we have an audio source, play "gnome died"
        // sound
        var audio = GetComponent<AudioSource>();
        if (audio)
        {
            audio.PlayOneShot(this.gnomeDiedSound);
        }
        // Show the damage effect
        currentGnome.ShowDamageEffect(damageType);
        // If we're not invincible, reset the game and make
        // the gnome not be the current player.

    if (gnomeInvincible == false)
        {
            // Tell the gnome that it died
            currentGnome.DestroyGnome(damageType);
            // Remove the Gnome
            RemoveGnome();
            // Reset the game
            StartCoroutine(ResetAfterDelay());
        }
    }
    // Called when gnome dies.
    IEnumerator ResetAfterDelay()
    {
        // Wait for delayAfterDeath seconds, then call Reset
        yield return new WaitForSeconds(delayAfterDeath);
        Reset();
    }
    // Called when the player touches a trap
    public void TrapTouched()
    {
        KillGnome(Gnome.DamageType.Slicing);
    }
        // Called when the player touches a fire trap
        public void FireTrapTouched()
        {
            KillGnome(Gnome.DamageType.Burning);
        }


    // Called when the gnome picks up the treasure.
    public void TreasureCollected()
    {
        // Tell the currentGnome that it should have the
        // treasure.
        currentGnome.holdingTreasure = true;
    }
    // Called when the player touches the exit.
    public void ExitReached()
    {
        // If we have a player, and that player is holding
        // treasure, game over!
        if (currentGnome != null &&
        currentGnome.holdingTreasure == true)
        {
            // If we have an audio source, play the "game
            // over" sound
            var audio = GetComponent<AudioSource>();
            if (audio)
            {
                audio.PlayOneShot(this.gameOverSound);

            }
            // Pause the game
            Time.timeScale = 0.0f;
            // Turn off the Game Over menu, and turn on the
            // "game over" screen!
            if (gameOverMenu)
            {
                gameOverMenu.gameObject.SetActive(true);
            }
            if (gameplayMenu)
            {
                gameplayMenu.gameObject.SetActive(false);
            }
        }
    }
    // Called when the Menu button is tapped, and when the
    // Resume Game button is tapped.
    public void SetPaused(bool paused)
    {
        // If we're paused, stop time and enable the menu (and
        // disable the game overlay)
        if (paused)
        {
            Time.timeScale = 0.0f;
            mainMenu.gameObject.SetActive(true);
            gameplayMenu.gameObject.SetActive(false);
        }
        else
        {
            // If we're not paused, resume time and disable
            // the menu (and enable the game overlay)
            Time.timeScale = 1.0f;
            mainMenu.gameObject.SetActive(false);
            gameplayMenu.gameObject.SetActive(true);
        }
    }
    // Called when the Restart button is tapped.
    public void RestartGame()
    {
        // Immediately remove the gnome (instead of killing it)
        Destroy(currentGnome.gameObject);
        currentGnome = null;
        // Now reset the game to create a new gnome.
        Reset();
    }
}
```

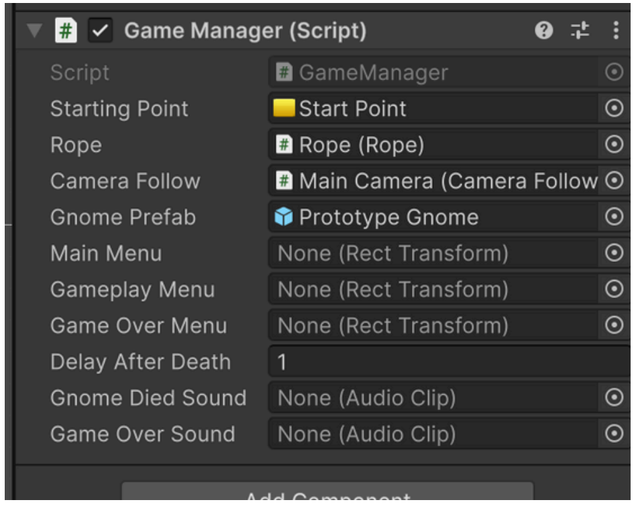
UPDATE THE GNOME

make the Prototype Gnome in a prefab by dragging it into the Gnome asset folder

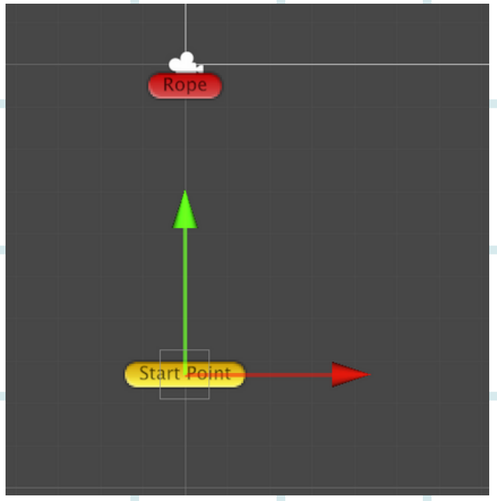
Create 2 game object named Game manager and Start Point



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

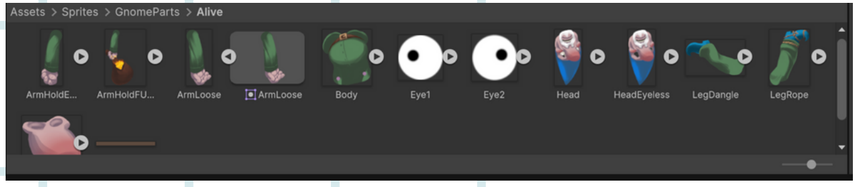


Make Sure to put the Start point object in the gnome itself



UPDATE THE GNOME

To make the gnome more polished lets add the sprites



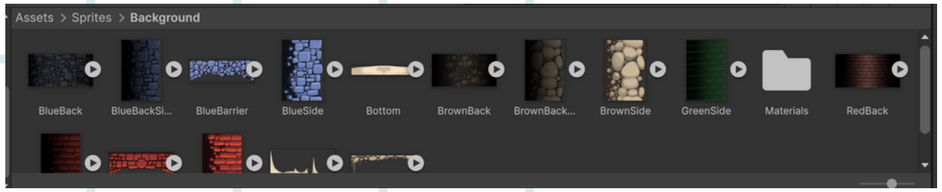
And replace the prefab witht the actual sprites

Add an collider object to each part except the body

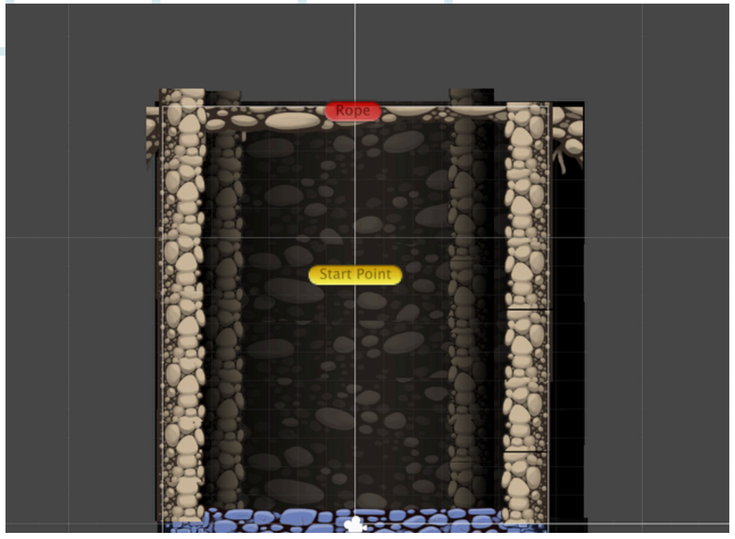
and inside the colider add. polygoncollider and make it like this

MAKE THE LEVEL

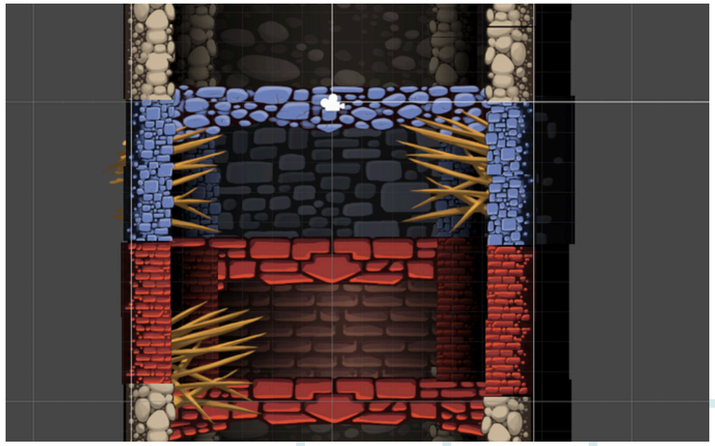
Import the background assets



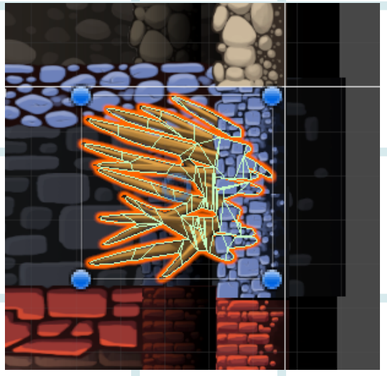
and arrange them like this



put spikes too for the death

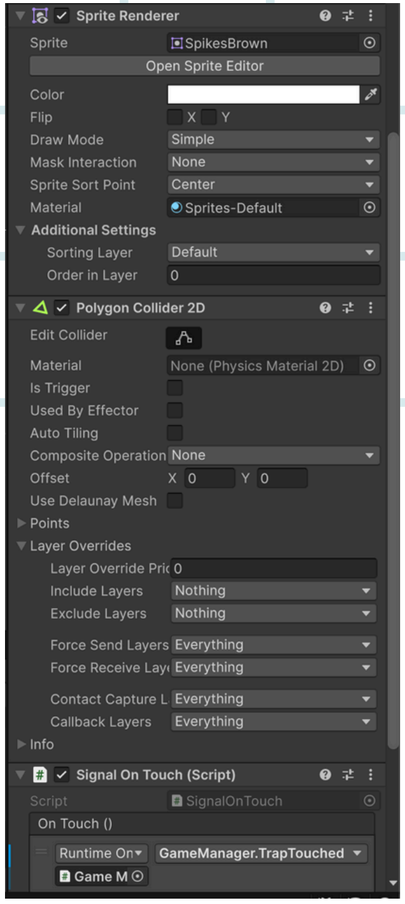


To configure it



MAKE THE LEVEL

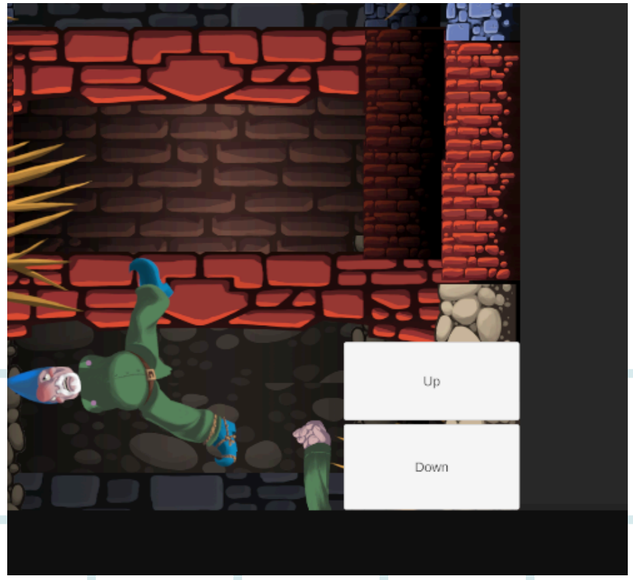
Add this components to the spike



create a new script called SignalOntouch.cs

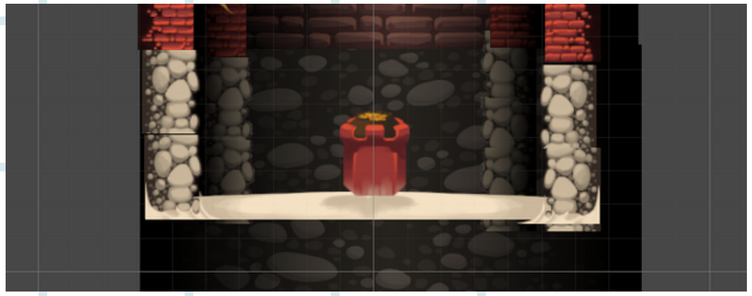
```
using UnityEngine;
using UnityEngine.Events;
// Invokes a UnityEvent when the Player collides with this
// object.
[RequireComponent(typeof(Collider2D))]
public class SignalOnTouch : MonoBehaviour
{

// 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();
        }
    }
}
```

When the gnome touches the spike it will die and fall off

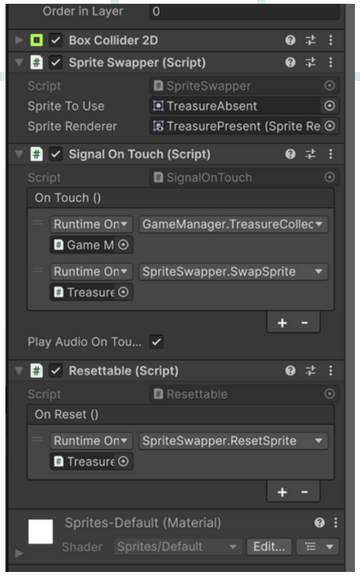


CREATING BUTTONS

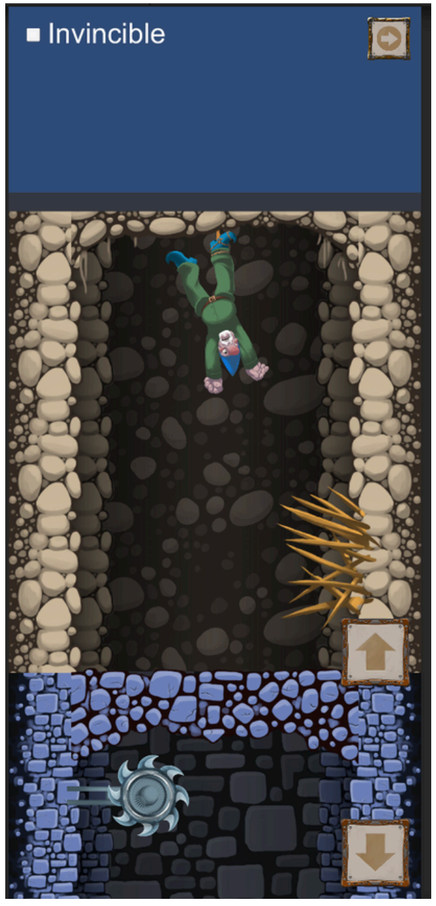
Lets now create the Treasure



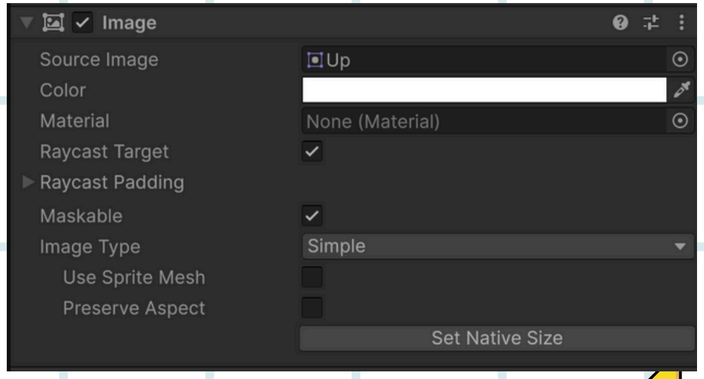
Add these components



Lets now update the uii and add other functions of ui



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

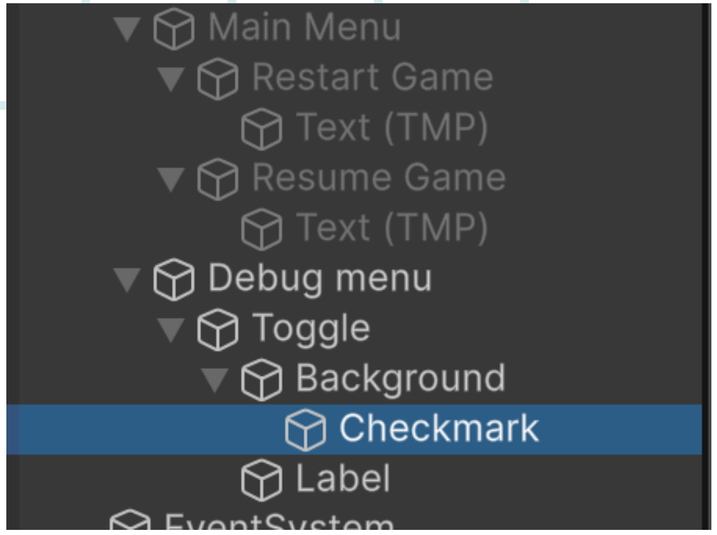


Do the same with the down button

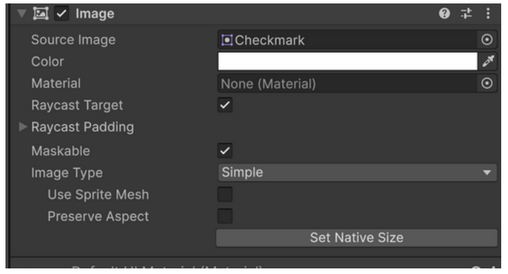
CREATE MECHANICS

Lets now make the Menu and invincible button

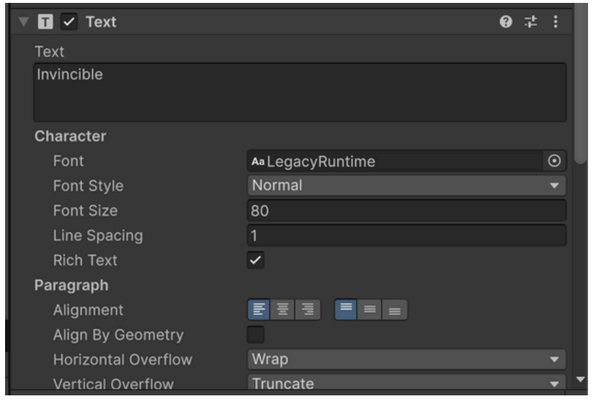
Make this objects with parent and child objects like this



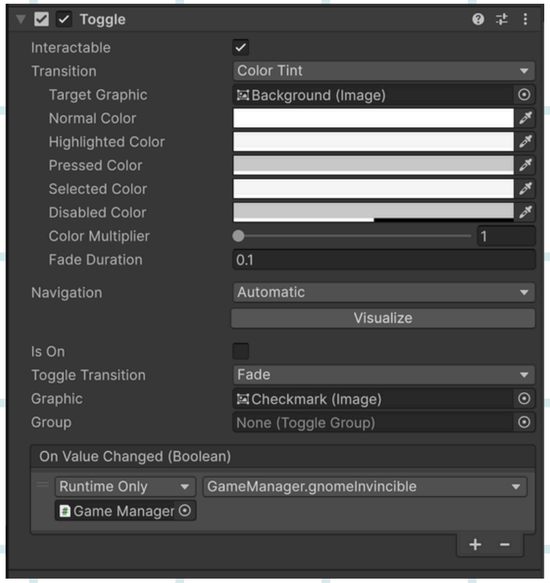
Add these on the checkmark object



And modify level to this

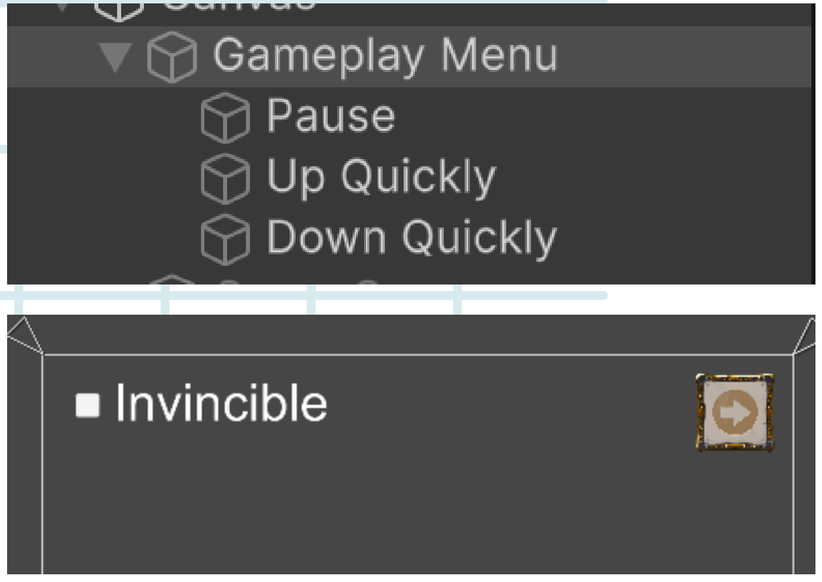


And put these on the toggle object

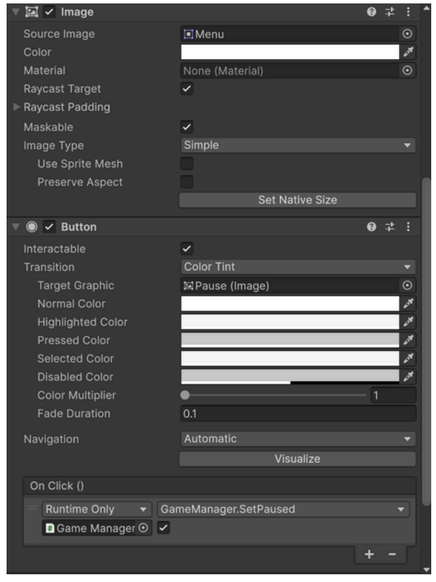


CREATE MECHANICS

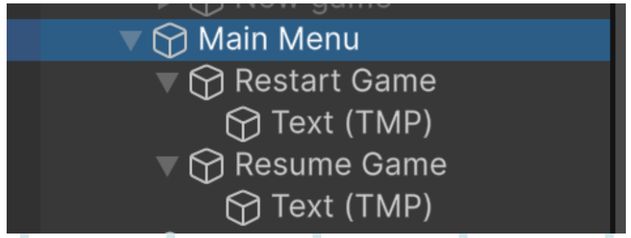
Add pause to the gameplay menu and put it on the top right



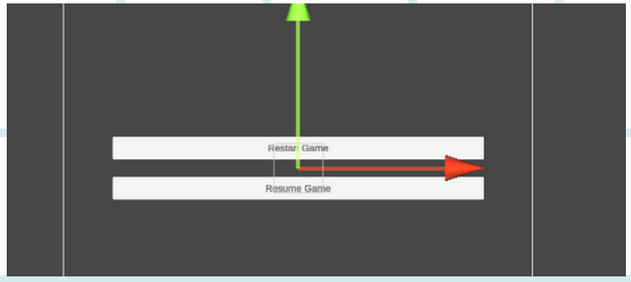
Add these components to the object pause



make another objects called main menu and add these objects under it

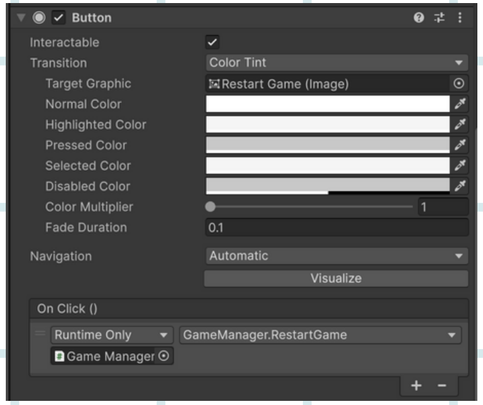


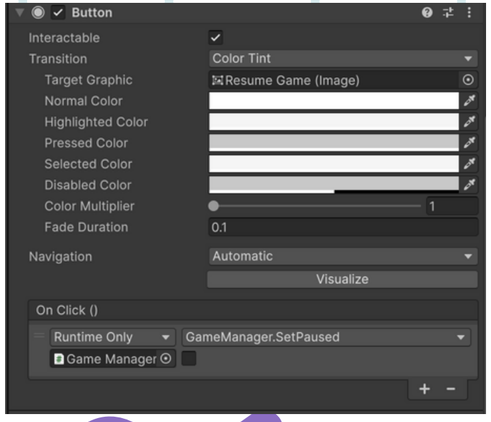
and Make them Look like this



CREATE GAMEOVER

Add these componnets respectively

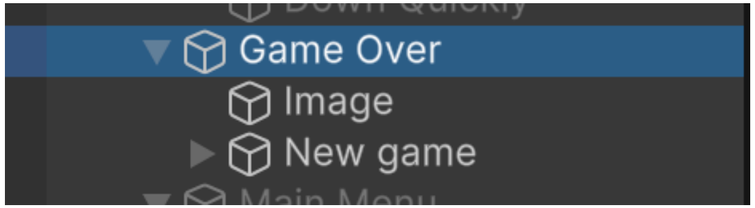




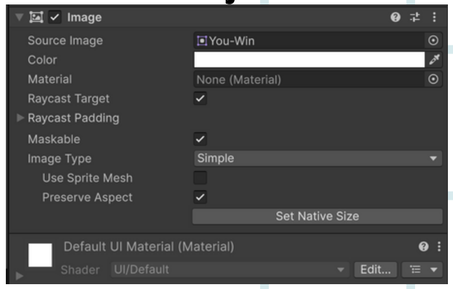
And set the game manager to main menu and gameplay menu

Lets create the game over menu

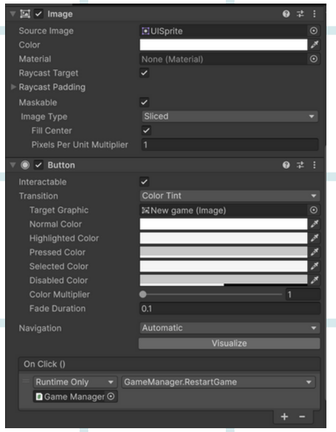
Create these 3 objects



Put these on the image object



Put these on the new game object

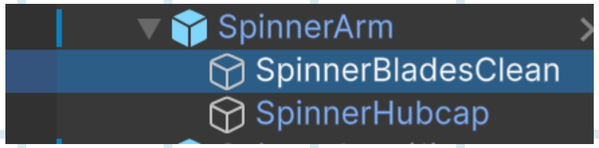


and configure game manager

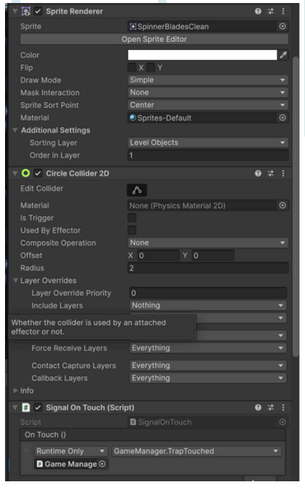
ADD MORE TRAPS

Lets add another trap called spinner

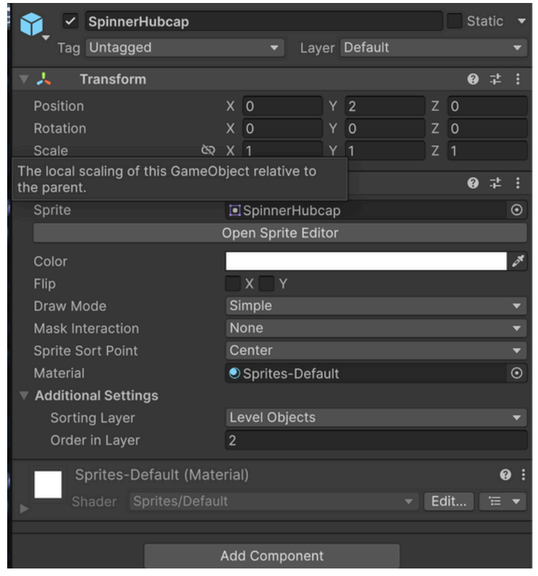
Make these objects



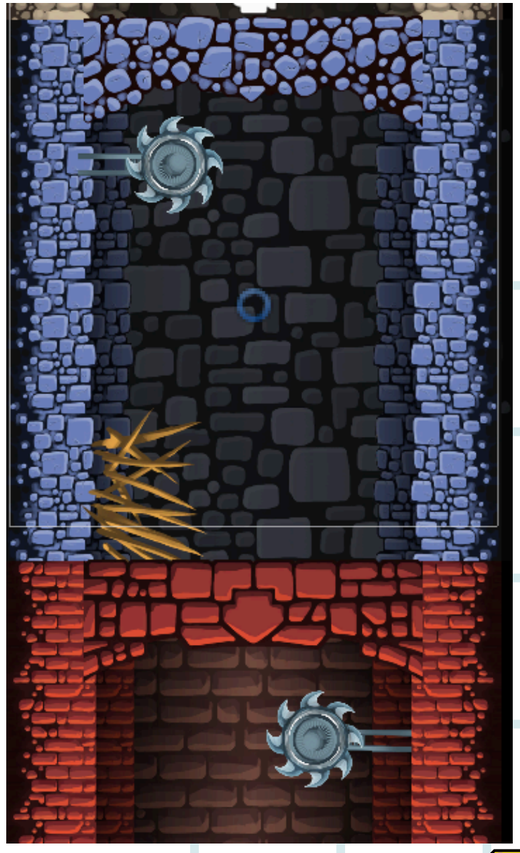
Add this components to spinnerbladesclean



copy this setting to the spinner hub cap

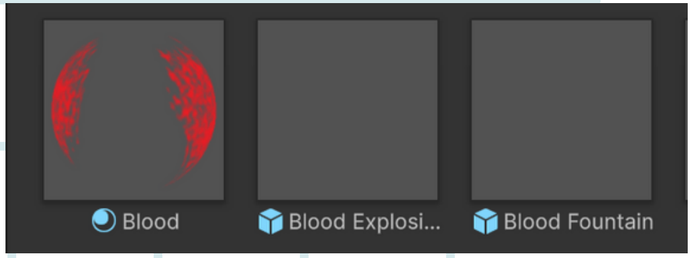


And put it anywhere on the map as you like

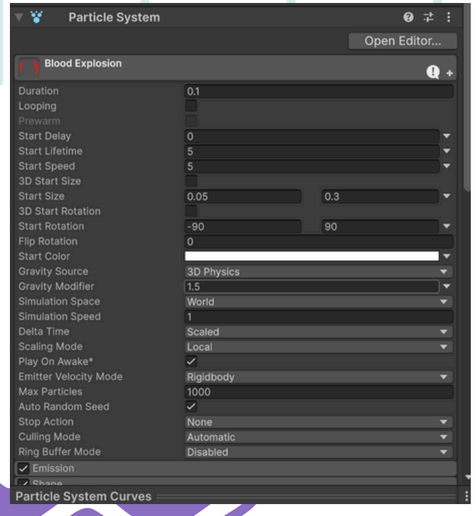


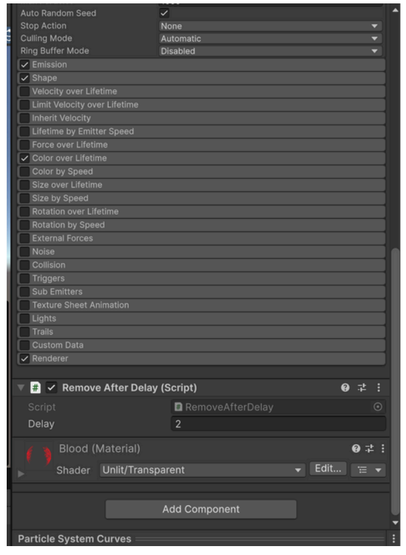
CREATING BUTTONS

Create this prefabs called blood explosion and bloof fountain into a particle system

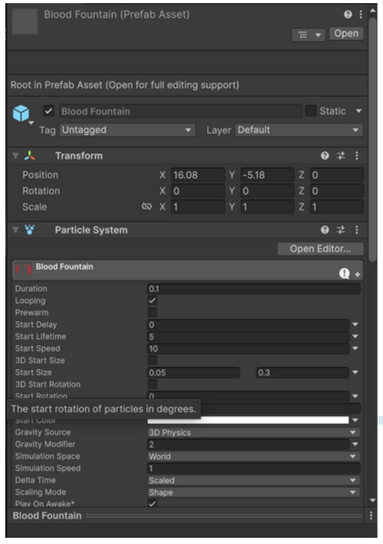


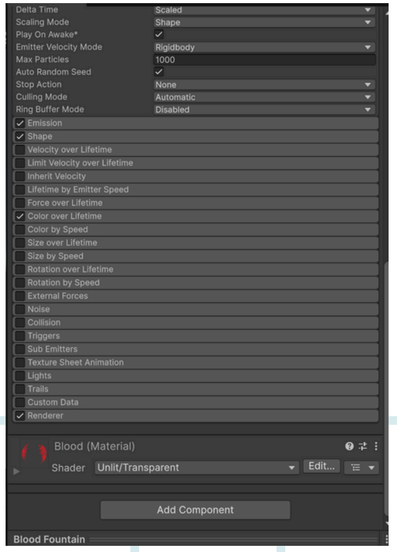
Copy these settings to blood explosion



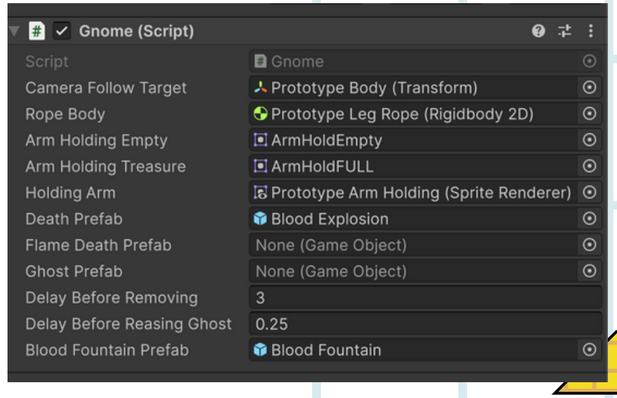


Copy these settings to blood fountain



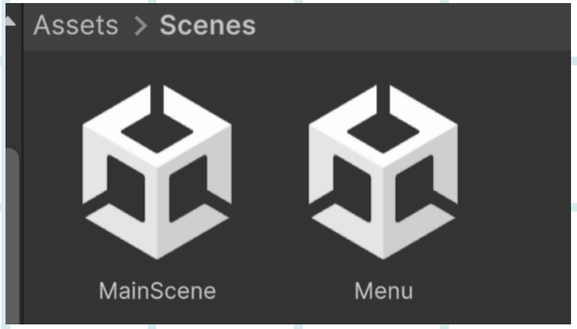


Update the gnome script in th Gnome prefab

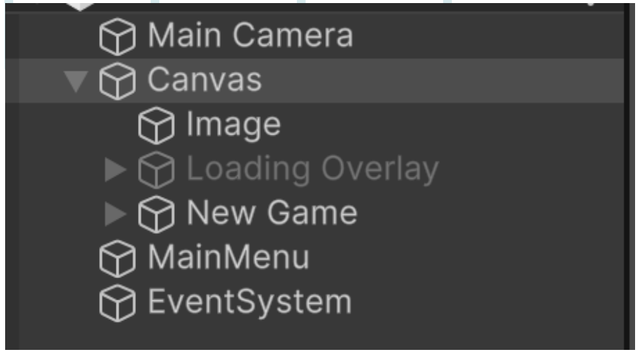


CREATE THE MENU

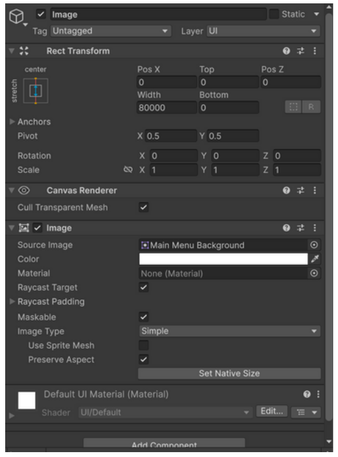
Create a new scene Called Menu



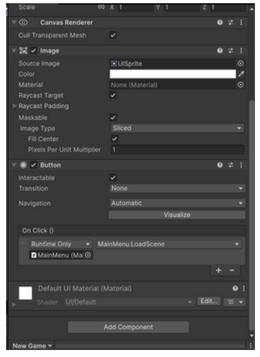
And Make this objects



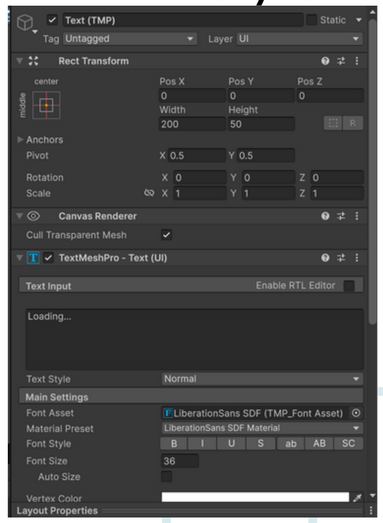
Make these image settings



Like in the pause game add a new game button and add these buttons



make these in the loading overlay text



The Scene should look like this

CREATING THE MENU

Make Main menu code like this and put it on the main menu object

Main Menu (Mono Script) Import Settings

Open | Execution Order...

Loading Overlay: None (Rect Transform)

Default references will only be applied in edit mode.

Imported Object

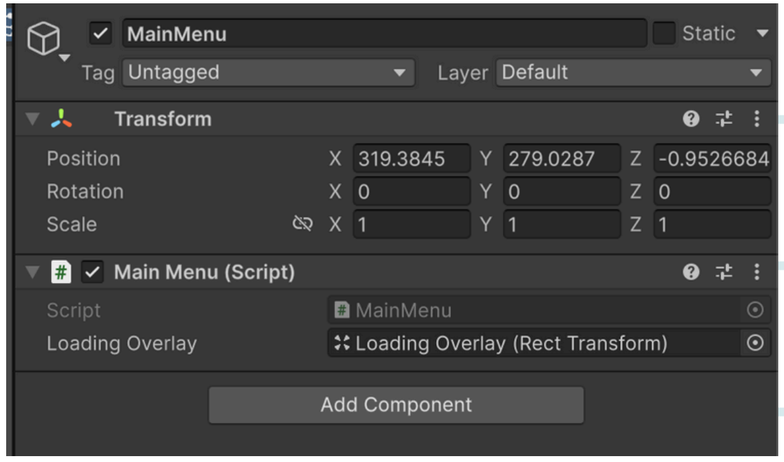
Main Menu (Mono Script)

Assembly Information

Filename: Assembly-CSharp.dll

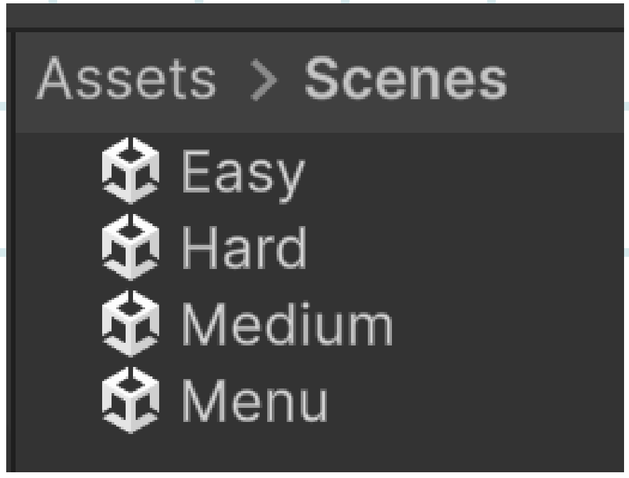
```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    public RectTransform loadingOverlay;

    private AsyncOperation _sceneLoadingOperation;

    private void Start()
    {
        loadingOverlay.gameObject.SetActive(false);

        _sceneLoadingOperation =
SceneManager.LoadSceneAsync(Scenes.MAIN_SCENE);
        _sceneLoadingOperation.allowSceneActivation = false;
    }

    public void LoadScene()
    {
        loadingOverlay.gameObject.SetActive(true);

        _sceneLoadingOperation.allowSceneActivation = true;
    }
}
```

Add mainmenu.cs to the mainmenu object

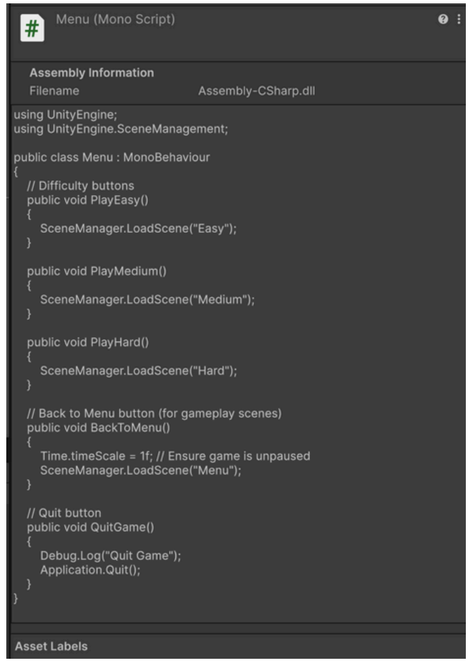


CREATING LEVELS

Duplicate the main scene and create all these scenes



Create Menu.cs for the Menu Scene



Make buttons and arrange Menu Scene Like this

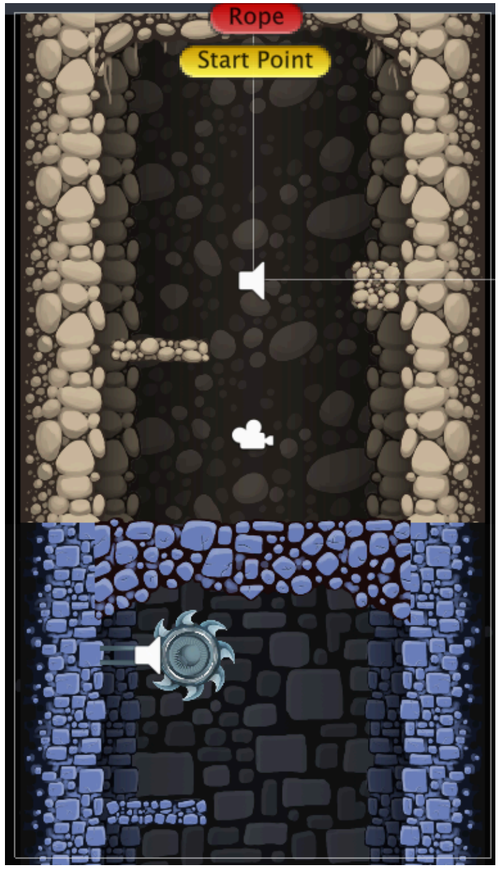
And Arrange the Duplicated Scene

CREATING LEVELS

And Update the Duplicated levels to match the difficulty

Easy

Medium



Medium

FINAL OUTPUT

After All the steps that we've done we should have a decent 2D game output that runs with unity which are these pictures, in the left with the Menu scene and in the right the Game scene.