ROCKFALL

Kristian Paul Bigornia
Calvin Klein Catalon
Dhann Angelo Guerrero

PROJECT START

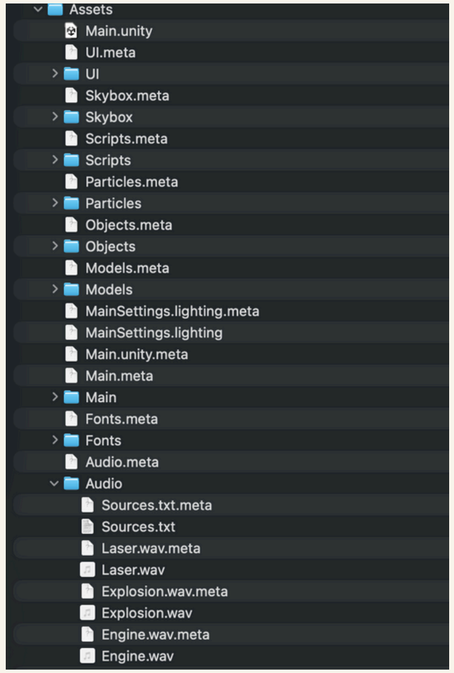
Rockfall is a 3d Shooter Tutorial game for Beginners

First Step is to Install Unity And install the latest editor available

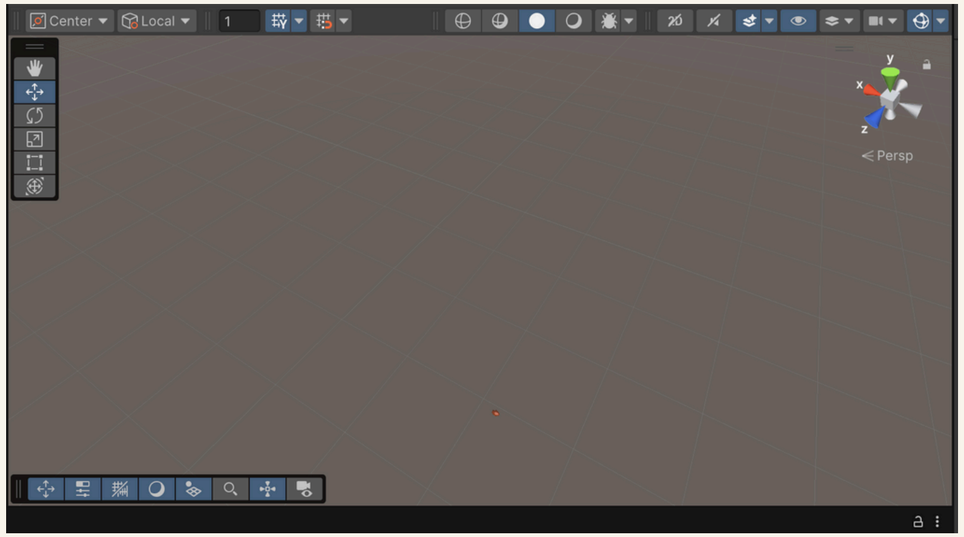
Create A project with 3d universal selected

PROJECT START

Import The Assets in to the editor

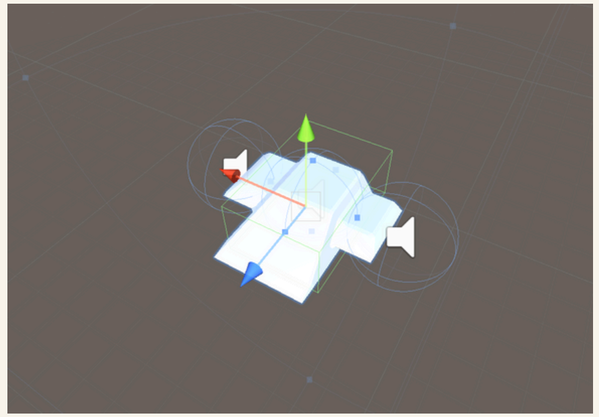


Create the Scene for the game

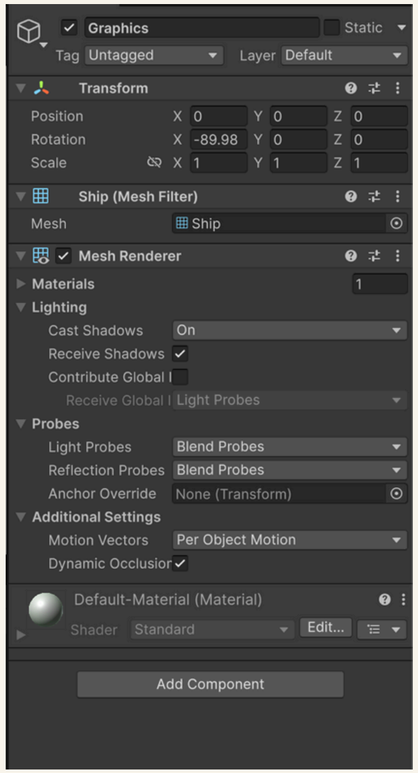


PROJECT START

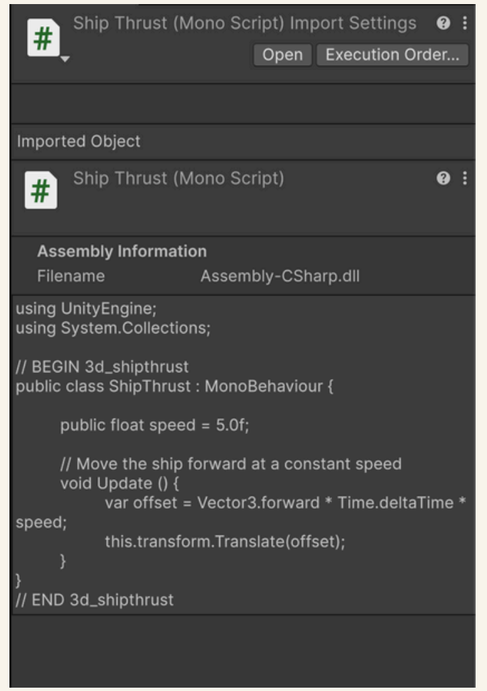
Create the Ship object and drag the ship model to the object



Create another Object name graphics and put these properties



Make ShiptrustScript to make the ship move



PROJECT START

Make the Smooth Follow Script and put the ship object

Imported Object

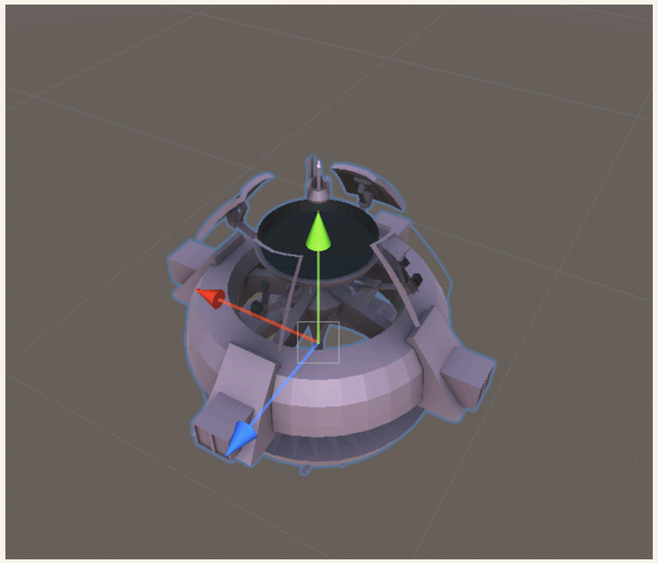
Smooth Follow (Mono Script)

Assembly Information

Filename Assembly-CSharp.dll

```
using UnityEngine;

// BEGIN 3d_smoothfollow
public class SmoothFollow : MonoBehaviour
{

	// The target we are following
	public Transform target;

	// The height we want the camera to be above the
target
	public float height = 5.0f;

	// The distance to the target, not counting height
	public float distance = 10.0f;

	// How much we slow down changes in rotation
and height
	public float rotationDamping;
	public float heightDamping;

	// Update is called once per frame
	void LateUpdate()
	{
		// Bail out if we don't have a target
		if (!target)
			return;

		// Calculate the current rotation angles
		var wantedRotationAngle =
target.eulerAngles.y;
		var wantedHeight = target.position.y + height;
		var wantedHeight = target.position.y + height;

		// Note where we're currently positioned and
looking
		var currentRotationAngle =
transform.eulerAngles.y;
		var currentHeight = transform.position.y;

		// Damp the rotation around the y-axis
		currentRotationAngle =
Mathf.LerpAngle(currentRotationAngle,
			wantedRotationAngle, rotationDamping
* Time.deltaTime);

		// Damp the height
		currentHeight = Mathf.Lerp(currentHeight,
			wantedHeight, heightDamping *
Time.deltaTime);

		// Convert the angle into a rotation
		var currentRotation = Quaternion.Euler(0,
currentRotationAngle, 0);

		// Set the position of the camera on the x-z
plane to:
		// "distance" meters behind the target
		transform.position = target.position;
		transform.position -= currentRotation *
Vector3.forward * distance;

		// Set the position of the camera using our
new height
		transform.position = new
Vector3(transform.position.x,
			currentHeight, transform.position.z);

		// Finally, look at where the target is looking
		transform.rotation =
Quaternion.Lerp(transform.rotation,
			target.rotation, rotationDamping *
Time.deltaTime);

	}
}
// END 3d_smoothfollow
```

make a game object named space station and drag the model as a child object

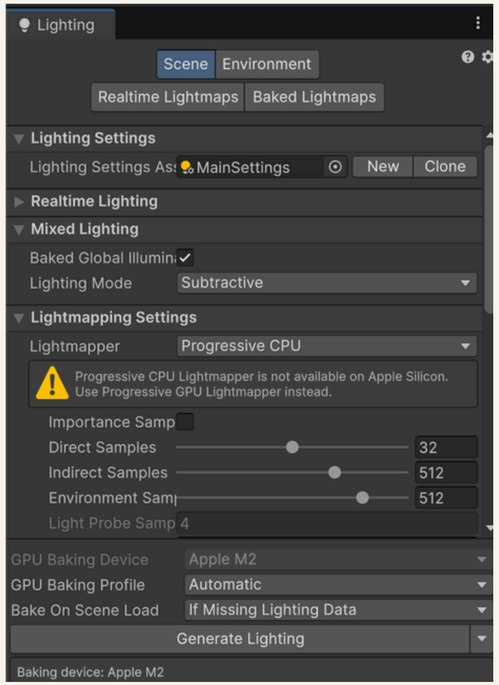


PROJECT START

Lets now create the Space

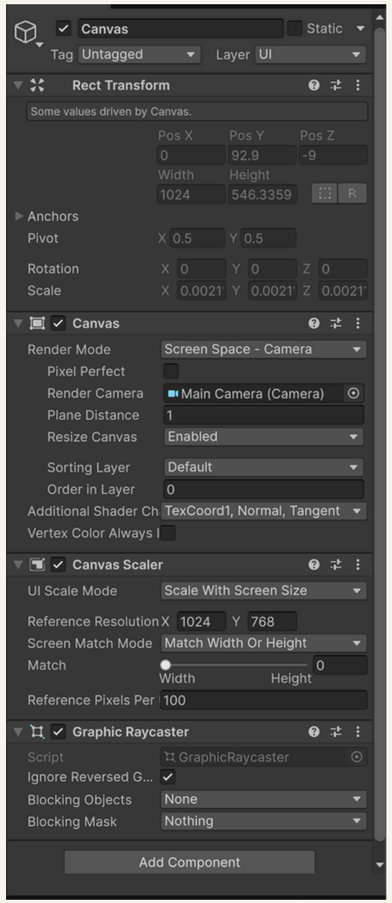
make a material called material and put skybox and put these properties

put the skybox on the lighting settings



PROJECT START

Lets now create the canvas and put these properties



After creating the canvas lets condigure the input controls

add a joystick to the scene

PROJECT START

After creating the joystick add the virtual joystick script

Imported Object

Virtual Joystick (Mono Script)

Assembly Information

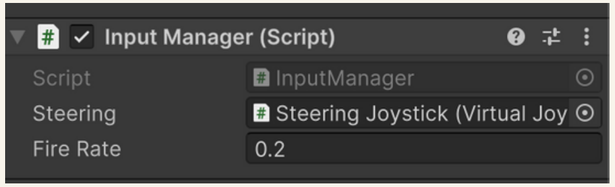
Filename Assembly-CSharp.dll

```
using UnityEngine;
using System.Collections;

// BEGIN 3d_virtualjoystick
// Get access to the Event interfaces
using UnityEngine.EventSystems;

// Get access to UI elements
using UnityEngine.UI;

public class VirtualJoystick : MonoBehaviour,
IBeginDragHandler,
        IDragHandler, IEndDragHandler {

        // The sprite that gets dragged around
        public RectTransform thumb;

        // The locations of the thumb and joystick when no
dragging is happening
        private Vector2 originalPosition;
        private Vector2 originalThumbPosition;

        // The distance that the thumb has been dragged
away from its
        // original position
        public Vector2 delta;

        void Start () {
                // When the joystick starts up, record the
original positions
                originalPosition =
this.GetComponent<RectTransform>().localPosition;
                originalThumbPosition = thumb.localPosition;

                // Disable the thumb so that it's not visible
                thumb.gameObject.SetActive(false);

                // Reset the delta to zero
                delta = Vector2.zero;
        }

        // Called when dragging starts
        public void OnBeginDrag (PointerEventData
eventData) {

                // Make the thumb visible
                thumb.gameObject.SetActive(true);

                // Figure out where in world-space the drag
started
                Vector3 worldPoint = new Vector3();

RectTransformUtility.ScreenPointToWorldPointInRectangl
e(
                        this.transform as RectTransform,
                eventData.position,
                eventData.enterEventCamera,
                out worldPoint);


                // Place the joystick at that point

this.GetComponent<RectTransform>().position =
worldPoint;

                // Ensure that the thumb is in its original
location,
                // relative to the joystick
                thumb.localPosition = originalThumbPosition;
        }

        // Called when the drag moves
public void OnDrag (PointerEventData eventData) {

    // Work out where the drag is in world space now
    Vector3 worldPoint = new Vector3();

RectTransformUtility.ScreenPointToWorldPointInRectangl
e(
```

Then create the input manager script

```
using UnityEngine;
using System.Collections;


// BEGIN 3d_inputmanager
public class InputManager : Singleton<InputManager> {

        // The joystick used to steer the ship.
        public VirtualJoystick steering;

        // BEGIN 3d_inputmanager_weapons
        // The delay between firing shots, in seconds.
        public float fireRate = 0.2f;

        // The current ShipWeapons script to fire from.
        private ShipWeapons currentWeapons;

        // If true, we are currently firing weapons.
        private bool isFiring = false;

        // Called by ShipWeapons to update the
currentWeapons
        // variable.
        public void SetWeapons(ShipWeapons weapons) {
                this.currentWeapons = weapons;
        }

        // Likewise; called to reset the currentWeapons
variable.
        public void RemoveWeapons(ShipWeapons
weapons) {

                // If the currentWeapons object is 'weapons',
                // set it to null.
                if (this.currentWeapons == weapons) {
                        this.currentWeapons = null;
                }
        }

        // Called when the user starts touching the Fire
button.
        public void StartFiring() {

                // Kick off the routine that starts firing shots.
                StartCoroutine(FireWeapons());
        }


        IEnumerator FireWeapons() {

                // Mark ourself as firing shots
                isFiring = true;

                // Loop for as long as isFiring is true
                while (isFiring) {

                        // If we have a weapons script, tell it to
fire
                        // a shot!
                        if (this.currentWeapons != null) {
                                currentWeapons.Fire();
                        }

                        // Wait for fireRate seconds before
firing the
                        // next shot
                        yield return new
WaitForSeconds(fireRate);

                }

        }

        // Called when the user stops touching the Fire
button
        public void StopFiring() {

                // Setting this to false will end the loop in
FireWeapons
                isFiring = false;
        }


        // END 3d_inputmanager_weapons
}
// END 3d_inputmanager
```

make an object and put the virtual joystick in the input manager



PROJECT START

Lets now configure the flight control by making the ship steering script

Ship Steering (Mono Script)

Assembly Information

Filename Assembly-CSharp.dll

```
using UnityEngine;
using System.Collections;

// BEGIN 3d_shipsteering
public class ShipSteering : MonoBehaviour {

	// The rate at which the ship turns
	public float turnRate = 10.0f;

	// The strength with which the ship levels out
	public float levelDamping = 1.0f;

	void Update () {

		// Create a new rotation by multiplying the joystick's direction
		// by turnRate, and clamping that to 90% of half a circle.

		// First, get the user's input.
		var steeringInput = InputManager.instance.steering.delta;

		// Now, create a rotation amount, as a vector.
		var rotation = new Vector2();

		rotation.y = steeringInput.x;
		rotation.x = steeringInput.y;

		// Multiply by turnRate to get the amount we want to steer by.
		rotation *= turnRate;

		// Turn this into radians by multiplying by 90% of a half-circle
		rotation.x = Mathf.Clamp(rotation.x, -Mathf.PI * 0.9f, Mathf.PI * 0.9f);

		// And turn those radians into a rotation quaternion!
		var newOrientation = Quaternion.Euler(rotation);

		// Combine this turn with our current orientation
		transform.rotation *= newOrientation;

		// Next, try to minimise roll!

		// Start by working out what our orientation
		// would be if we weren't rolled around the Z
		// axis at all
		var levelAngles = transform.eulerAngles;
		levelAngles.z = 0.0f;
		var levelOrientation = Quaternion.Euler(levelAngles);

		// Combine our current orientation with a small amount of this
		// "zero-roll" orientation; when this happens over multiple
		// frames, the object will slowly level out to zero roll
		transform.rotation = Quaternion.Slerp(transform.rotation,
			levelOrientation, levelDamping * Time.deltaTime);

	}
}
// END 3d_shipsteering
```

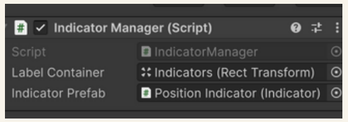
After that lets create indicators for the objects in the game, first create the Indicators scipt

```
using UnityEngine;
using System.Collections;

// BEGIN 3d_indicator
// Get access to the UI classes
using UnityEngine.UI;

public class Indicator : MonoBehaviour {

	// The object we're tracking.
	public Transform target;

	// Measure the distance from 'target' to this transform.
	public Transform showDistanceTo;

	// The label that shows the distance we're measuring.
	public Text distanceLabel;

	// How far we should be from the screen edges.
	public int margin = 50;

	// Our image's tint colour.
	public Color color {
		set {
			GetComponent<Image>().color = value;
		}
		get {
			return GetComponent<Image>().color;
		}
	}

  // Set up the indicator
	void Start() {
		// Hide the label; it will be re-enabled in
		// Update if we have a target
		distanceLabel.enabled = false;

		// On start, wait a frame before appearing to prevent visual glitches
		GetComponent<Image>().enabled = false;

	}

	// Update the indicator's position every frame
	void Update()
	{
```

```
distance.ToString() + "m";
		} else {
			// Don't show the label
			distanceLabel.enabled = false;
		}

		GetComponent<Image>().enabled = true;

		// Work out where in screen-space the object is
		var viewportPoint = Camera.main.WorldToViewportPoint(target.position);

		// Is the point behind us?
		if (viewportPoint.z < 0) {
			// Push it to the edges of the screen
			viewportPoint.z = 0;
			viewportPoint = viewportPoint.normalized;
			viewportPoint.x *= -Mathf.Infinity;
		}

		// Work out where in view-space we should be
		var screenPoint = Camera.main.ViewportToScreenPoint(viewportPoint);

		// Clamp to screen edges
		screenPoint.x = Mathf.Clamp(screenPoint.x, margin, Screen.width - margin * 2);
		screenPoint.y = Mathf.Clamp(screenPoint.y, margin, Screen.height - margin * 2);

		// Work out where in the canvas-space the view-space coordinate is
		var localPosition = new Vector2();
		RectTransformUtility.ScreenPointToLocalPointInRectangle(
			transform.parent.GetComponent<RectTransform>(),
			screenPoint,
			Camera.main,
			out localPosition);

		// Update our position
		var rectTransform = GetComponent<RectTransform>();
		rectTransform.localPosition = localPosition;
```

then lets make an indicator manager script,create an object and put the script then put these properties.

```
using UnityEngine;
using System.Collections;

// BEGIN 3d_indicatormanager
using UnityEngine.UI;

public class IndicatorManager : Singleton<IndicatorManager> {

	// The object that all indicators will be children of
	public RectTransform labelContainer;

	// The prefab we'll instantiate for each indicator
	public Indicator indicatorPrefab;

	// This method will be called by other objects
	public Indicator AddIndicator(GameObject target, Color color,
		Sprite sprite = null) {

		// Create the label object
		var newIndicator = Instantiate(indicatorPrefab);

		// Make it track the target
		newIndicator.target = target.transform;

		// Update its color
		newIndicator.color = color;

		// If we received a sprite, set the indicator's sprite to that
		if (sprite != null) {
			newIndicator.GetComponent<Image>().sprite = sprite;
		}

		// Add it to the container.
		newIndicator.transform.SetParent(labelContainer, false);

		return newIndicator;
	}
}
// END 3d_indicatormanager
```



PROJECT START

Lets now create the Weapons, first make the shot script

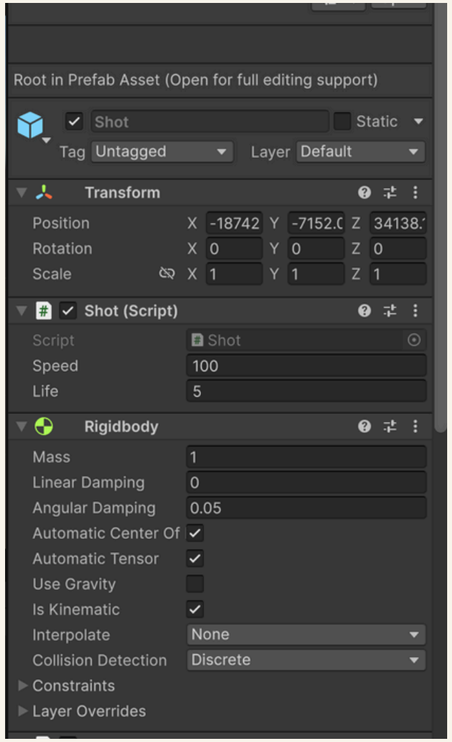
Shot (Mono Script)

Assembly Information

Filename Assembly-CSharp.dll

```
using UnityEngine;
using System.Collections;

// BEGIN 3d_shot
// Moves forward at a certain speed, and dies after a
certain time.
public class Shot : MonoBehaviour {

	// The speed at which the shot will move forward
	public float speed = 100.0f;

	// Remove this object after this many seconds
	public float life = 5.0f;

	void Start() {
		// Destroy after 'life' seconds
		Destroy(gameObject, life);
	}

	void Update () {
		// Move forward at constant speed
		transform.Translate(Vector3.forward * speed *
Time.deltaTime);
	}
}
// END 3d_shot
```

and make an object called shot and putthe shot script in it and put these scripts

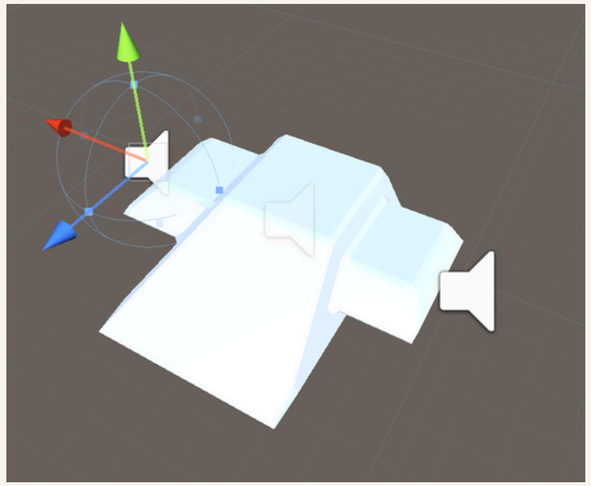


after that lets make the ship weapons script

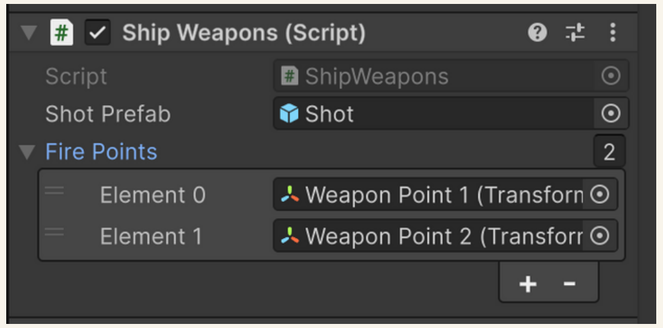
```
using UnityEngine;
using System.Collections;

// BEGIN 3d_shipweapons
public class ShipWeapons : MonoBehaviour {

	// The prefab to use for each shot
	public GameObject shotPrefab;

	// BEGIN 3d_shipweapons_inputmanager
	public void Awake() {
		// When this object starts up, tell the input
manager
		// to use me as the current weapon object
		InputManager.instance.SetWeapons(this);
	}

	// Called when the object is removed
	public void OnDestroy() {
		// Don't do this if we're not playing
		if (Application.isPlaying == true) {

InputManager.instance.RemoveWeapons(this);
		}
	}
	// END 3d_shipweapons_inputmanager

	// The list of places where a shot can emerge from
	public Transform[] firePoints;

	// The index into firePoints that the next shot will
fire from
	private int firePointIndex;

	// Called by InputManager.
	public void Fire() {

		// If we have no points to fire from, return
		if (firePoints.Length == 0)
			return;

		// Work out which point to fire from
		var firePointToUse =
firePoints[firePointIndex];

		// Create the new shot, at the fire point's
position
		// and with its rotation
		Instantiate(shotPrefab,
			firePointToUse.position,
			firePointToUse.rotation);

		// BEGIN 3d_shipweapons_audio
		// If the fire point has an audio source
component,
		// play its sound effect
		var audio =
firePointToUse.GetComponent<AudioSource>();
		if (audio) {
			audio.Play();
		}
		// END 3d_shipweapons_audio

		// Move to the next fire point
		firePointIndex++;

		// If we've moved past the last fire point in
the list,
		// move back to the start of the queue
		if (firePointIndex >= firePoints.Length)
			firePointIndex = 0;

	}
}
// END 3d_shipweapons
```

PROJECT START

put the firepoint on both side of the wings on the ship



put the script ont he ship and put the two firepoint



on the scene create the fire button that will be used by the input manager

PROJECT START

make the target reticle, first create shiptarget script

and put it into the reticle sprite

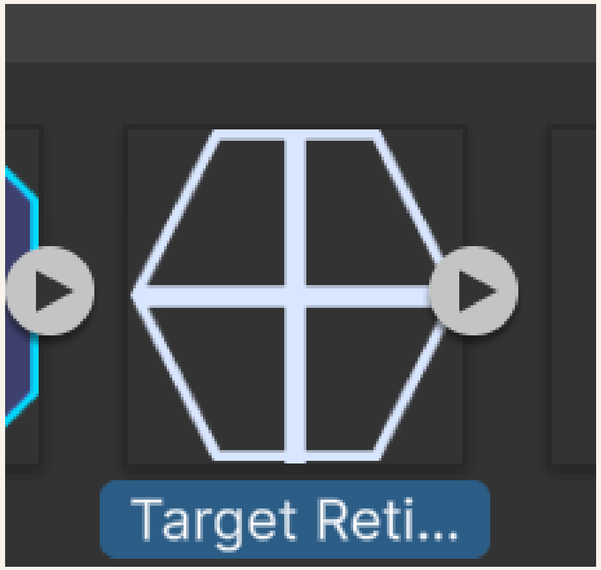
Imported Object

Ship Target (Mono Script)

Assembly Information

Filename Assembly-CSharp.dll

```
using UnityEngine;
using System.Collections;

// BEGIN 3d_shiptarget
public class ShipTarget : MonoBehaviour {

    // The sprite to use for the target reticle.
    public Sprite targetImage;

    void Start () {

        // Register a new indicator that tracks this
object, using a
        // yellow color and the custom sprite.

IndicatorManager.instance.AddIndicator(gameObject,
                Color.yellow, targetImage);
    }


// END 3d_shiptarget
```

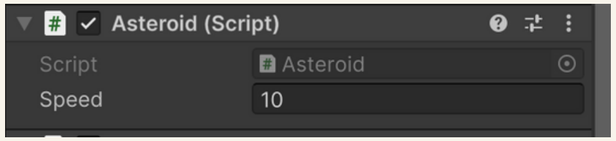


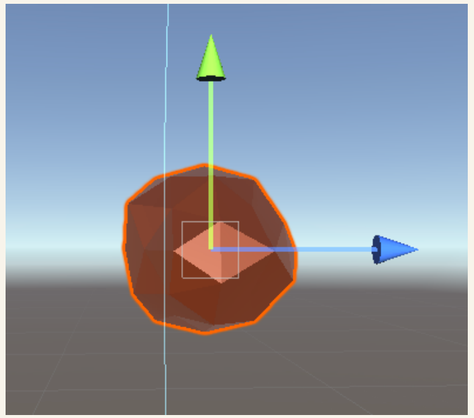
PROJECT START

lets now create the asteroid, first create the asteroid script

```
using UnityEngine;
using System.Collections;

// BEGIN 3d_asteroid
public class Asteroid : MonoBehaviour {

	// The speed at which the asteroid moves.
	public float speed = 10.0f;

	void Start () {
		// Set the velocity of the rigidbody
		GetComponent<Rigidbody>().linearVelocity =
transform.forward * speed;

		// Create a red indicator for this asteroid
		var indicator =
IndicatorManager.instance.AddIndicator(gameObject,
Color.red);

		// BEGIN 3d_asteroid_gamemanager
		// Track the distance from this object to the
current space station
		// that's managed by the GameManager
	indicator.showDistanceTo =
GameManager.instance.currentSpaceStation.transform;
		// END 3d_asteroid_gamemanager
	}

}
// END 3d_asteroid
```

then make a model and put the script in it



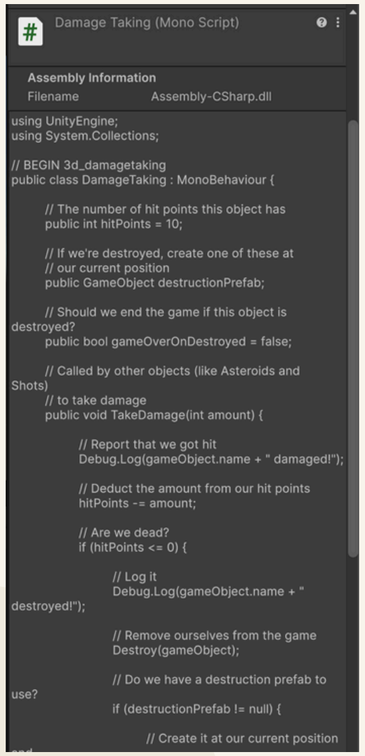


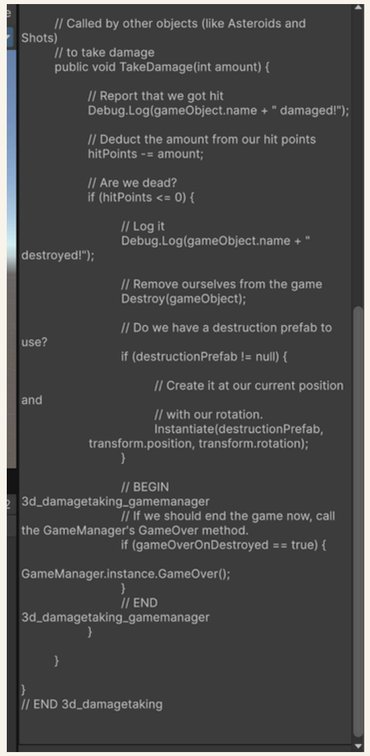
after that make the asteroid spawner script

```
using UnityEngine;
using System.Collections;

// BEGIN 3d_asteroidspawner
public class AsteroidSpawner : MonoBehaviour {

	// The radius of the spawn area
	public float radius = 250.0f;

	// The asteroids to spawn
	public Rigidbody asteroidPrefab;

	// Wait spawnRate ± variance seconds between
each asteroid
	public float spawnRate = 5.0f;
	public float variance = 1.0f;

	// The object to aim the asteriods at
	public Transform target;

	// If false, disable spawning
	public bool spawnAsteroids = false;

	void Start () {
		// Start the coroutine that creates asteroids
immediately
		StartCoroutine(CreateAsteroids());
	}

	IEnumerator CreateAsteroids() {

		// Loop forever
		while (true) {

			// Work out when the next asteroid
should appear
			float nextSpawnTime = spawnRate +
Random.Range(-variance, variance);

			// Wait that much time
			yield return new
WaitForSeconds(nextSpawnTime);

			// Additionally, wait until physics is
about to update
			yield return new WaitForFixedUpdate();

			// Create the asteroid
			CreateNewAsteroid();
		}
	}

	void CreateNewAsteroid() {

		// If we're not currently spawning asteroids,
bail out
		if (spawnAsteroids == false) {
			return;
		}

		// Randomly select a point on the surface of
the sphere
		var asteroidPosition = Random.onUnitSphere
* radius;

		// Scale this by the object's scale
		asteroidPosition.Scale(transform.lossyScale);

		// And offset it by the asteroid spawner's
location
		asteroidPosition += transform.position;

		// Create the new asteroid
		var newAsteroid = Instantiate(asteroidPrefab);

		// Place it at the spot we just calculated
		newAsteroid.transform.position =
asteroidPosition;

		// Aim it at the target
		newAsteroid.transform.LookAt(target);
	}

	// Called by the editor while the spawner object is
selected.
	void OnDrawGizmosSelected() {

		// We want to draw yellow stuff
		Gizmos.color = Color.yellow;

		// Tell the Gizmos drawer to use our current
position and scale
		Gizmos.matrix =
transform.localToWorldMatrix;

		// Draw a sphere representing the spawn area
		Gizmos.DrawWireSphere(Vector3.zero,
radius);
	}

	public void DestroyAllAsteroids() {
		// Remove all asteroids in the game
		foreach (var asteroid in
FindObjectsOfType<Asteroid>()) {
			Destroy (asteroid.gameObject);
		}
	}

}
// END 3d_asteroidspawner
```

PROJECT START

lets now crate the damage dealing , first kets create the script

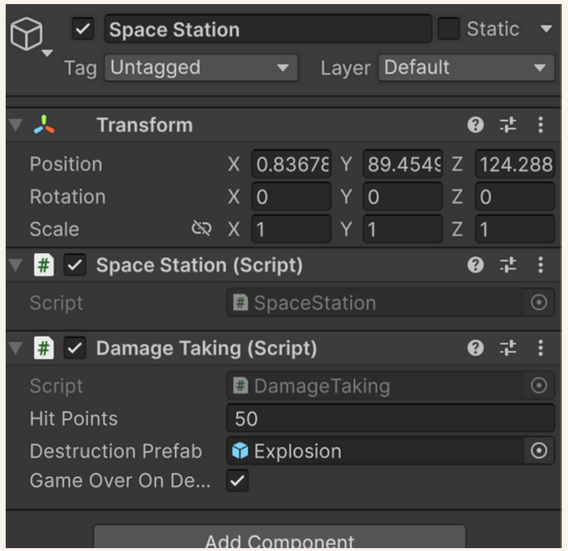




then make a damageoncollide script

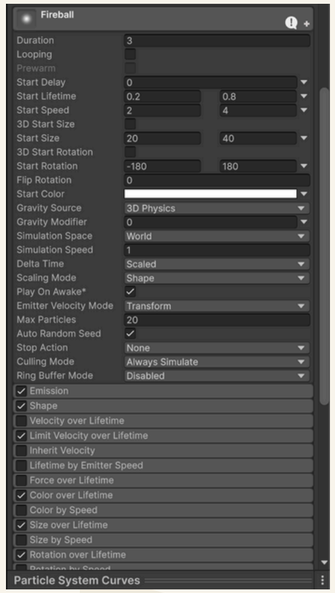
```
public class DamageOnCollide : MonoBehaviour {

  // The amount of damage we'll deal to anything we hit.
  public int damage = 1;

  // The amount of damage we'll deal to ourselves when we
  // hit something.
  public int damageToSelf = 5;

  void HitObject(GameObject theObject) {
    // Do damage to the thing we hit, if possible
    var theirDamage =
      theObject.GetComponentInParent<DamageTaking>();
    if (theirDamage) {
      theirDamage.TakeDamage(damage);
    }

    // Do damage to ourself, if possible
    var ourDamage =
      this.GetComponentInParent<DamageTaking>();
    if (ourDamage) {
      ourDamage.TakeDamage(damageToSelf);
    }
  }

  // Did an object enter this trigger area?
  void OnTriggerEnter(Collider collider) {
    HitObject(collider.gameObject);
  }

  // Did an object collide with us?
  void OnCollisionEnter(Collision collision) {
    HitObject(collision.gameObject);
  }
}
```

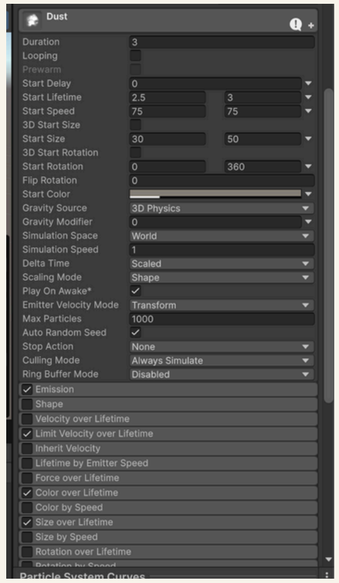
make a object called space station and put these two



PROJECT START

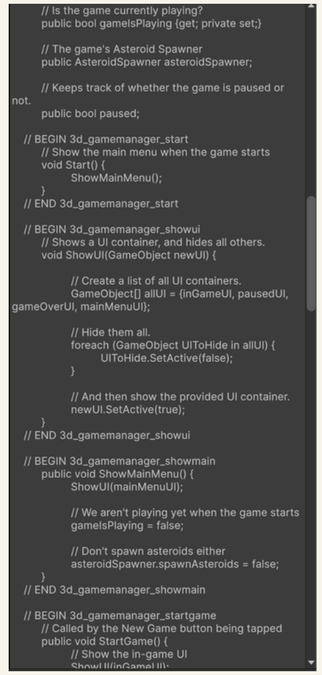
lets now add explosions, make these 2 materials and put it as child of the explosion material





after that we can move to the menus

after that lets crate the death using the game manager , first create a game manager script



PROJECT START

continuation of game manager script

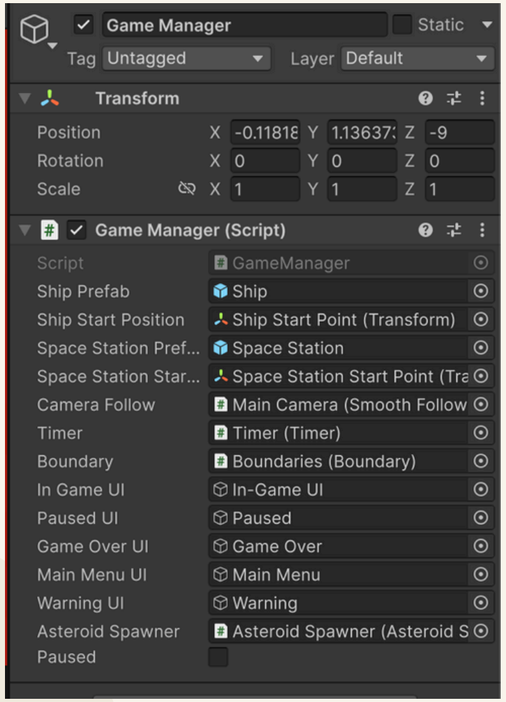
```
            // We're now playing
            gameIsPlaying = true;

            // If we happen to have a ship, destroy it
            if (currentShip != null) {
                Destroy(currentShip);
            }

            // Likewise for the station
            if (currentSpaceStation != null) {
                Destroy(currentSpaceStation);
            }

            // Create a new ship, and place it at the start
position
            currentShip = Instantiate(shipPrefab);
            currentShip.transform.position =
shipStartPosition.position;
            currentShip.transform.rotation =
shipStartPosition.rotation;

            // And likewise for the station
            currentSpaceStation =
Instantiate(spaceStationPrefab);

            currentSpaceStation.transform.position =
        spaceStationStartPosition.position;

            currentSpaceStation.transform.rotation =
        spaceStationStartPosition.rotation;

            // Make the follow script track the new ship
            cameraFollow.target = currentShip.transform;

            // Start spawning asteroids
            asteroidSpawner.spawnAsteroids = true;

            // And aim the the spawner at the new space
station
            asteroidSpawner.target =
currentSpaceStation.transform;

            // BEGIN 3d_gamemanager_timer
            timer.StartClock();
            // END 3d_gamemanager_timer
        }
    // END 3d_gamemanager_startgame

    // BEGIN 3d_gamemanager_gameover
        // Called by objects that end the game when
they're destroyed
        public void GameOver() {
```

```
they're destroyed
        public void GameOver() {
            // Show the game over UI
            ShowUI(gameOverUI);

            // We're no longer playing
            gameIsPlaying = false;

            // Destroy the ship and the station
            if (currentShip != null)
                Destroy (currentShip);

            if (currentSpaceStation != null)
                Destroy (currentSpaceStation);

            // BEGIN 3d_gamemanager_boundary
            // Stop showing the warning UI, if it was
visible
            warningUI.SetActive(false);
            // END 3d_gamemanager_boundary

            // Stop spawning asteroids
            asteroidSpawner.spawnAsteroids = false;

            // And remove all lingering asteroids from the
game
            asteroidSpawner.DestroyAllAsteroids();
        }
    // END 3d_gamemanager_gameover

    // BEGIN 3d_gamemanager_setpaused
        // Called when the Pause or Resume buttons are
tapped
        public void SetPaused(bool paused) {

            // Switch between the in-game and paused UI
            inGameUI.SetActive(!paused);
            pausedUI.SetActive(paused);

            // If we're paused..
            if (paused) {
                // Stop time
                Time.timeScale = 0.0f;
            } else {
                // Resume time
                Time.timeScale = 1.0f;
            }
        }
    // END 3d_gamemanager_setpaused
```

```
    // BEGIN 3d_gamemanager_boundary
        public void Update() {

            // If we don't have a ship, bail out
            if (currentShip == null)
                return;

            // If the ship is outside the Boundary's
Destroy Radius,
            // game over. If it's within the Destroy Radius,
but outside
            // the Warning radius, show the Warning UI. If
it's within both,
            // don't show the Warning UI.

            float distance =
                (currentShip.transform.position
                    -
boundary.transform.position).magnitude;

            if (distance > boundary.destroyRadius) {
                // The ship has gone beyond the
destroy radius, so it's game over
                GameOver();
            } else if (distance > boundary.warningRadius)
{
                // The ship has gone beyond the
warning radius, so show the
                // warning UI
                warningUI.SetActive(true);
            } else {
                // It's within the warning threshold, so
don't show the warning UI
                warningUI.SetActive(false);
            }


        }
        // END 3d_gamemanager_boundary

}
// END 3d_gamemanager
```

PROJECT START

After that we can now configure the game manager object and put the game manager script



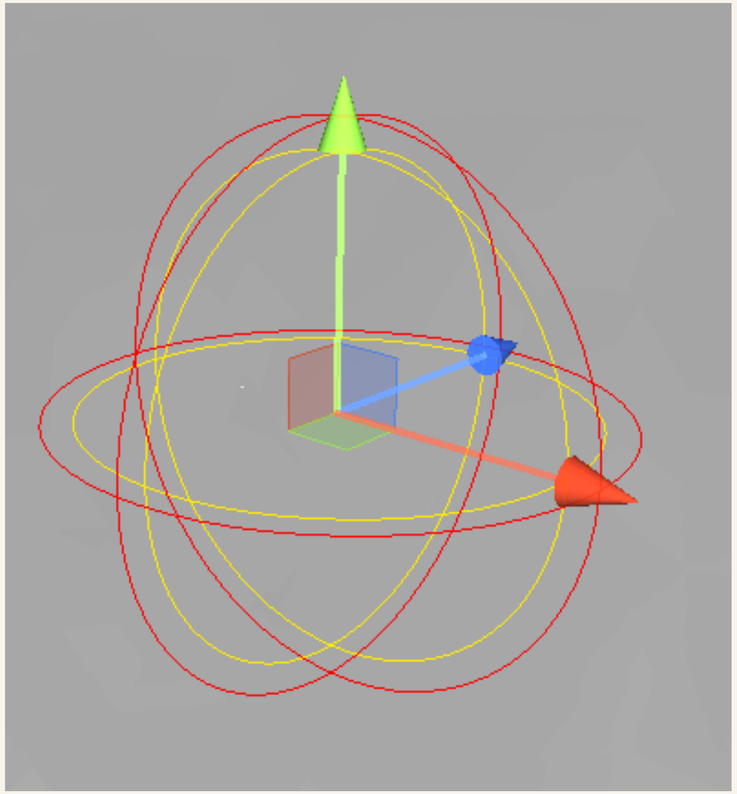
Lets now create the Boundary pf the game , first. lets code the boundary.cs

Boundary (Mono Script)

Assembly Information

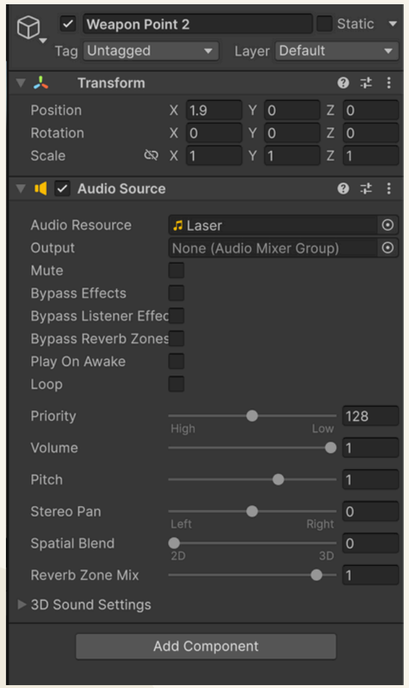
Filename Assembly-CSharp.dll

```
using UnityEngine;
using System.Collections;

// BEGIN 3d_boundary
public class Boundary : MonoBehaviour {

	// Show the warning UI when the player is this far
from the center
	public float warningRadius = 400.0f;

	// End the game when the player is this far from the
center
	public float destroyRadius = 450.0f;

	public void OnDrawGizmosSelected() {
		// Show a yellow sphere with the warning radius
		Gizmos.color = Color.yellow;
		Gizmos.DrawWireSphere(transform.position,
warningRadius);

		// And show a red sphere with the destroy
radius
		Gizmos.color = Color.red;
		Gizmos.DrawWireSphere(transform.position,
destroyRadius);
	}
}
// END 3d_boundary
```



PROJECT START

After that lets just put the audioes to each part we want like this.



And then The rockfall game is done base on the book. With this tutorial we can now make the rockfall game from the ground up, and understanding how we can make a simple 3d game.