UAS

GAME DESIGN AND DEVELOPMENT

SPEED LANCE

Disusun Oleh :

MOKHAMAD WIJAYA (17090092)

REZA ADITYA SAM MARHABAN (17090147)

DIV TEKNIK INFORMATIKA

POLITEKNIK HARAPAN BERSAMA TEGAL

2020

DAFTAR ISI

Halaman

A. TENTANG GAME

Speed Lance adalah game balapan sederhana dimana kendaraan (mobil *hover*) melintasi lintasan balap selama 3 *lap* untuk mencapai finish.

B. KONSEP DAN PENGEMBANGAN GAME

1. Konsep

   Konsep dari game ini sangat sederhana, yaitu pemain mulai mengendarai kendaraan dan melintasi lintasan balap selama 3 lap. Dalam melintasi lintasan balap, pemain dituntut untuk secepat mungkin melesat tanpa membuat gerakan yang sia-sia baik dari berbelok maupun saat lurus agar mendapatkan lama waktu paling cepat.

2. Pengembangan Game.

   Game ini terdapat komponen-komponen penting yang menyusunnya, yaitu sebagai berikut.

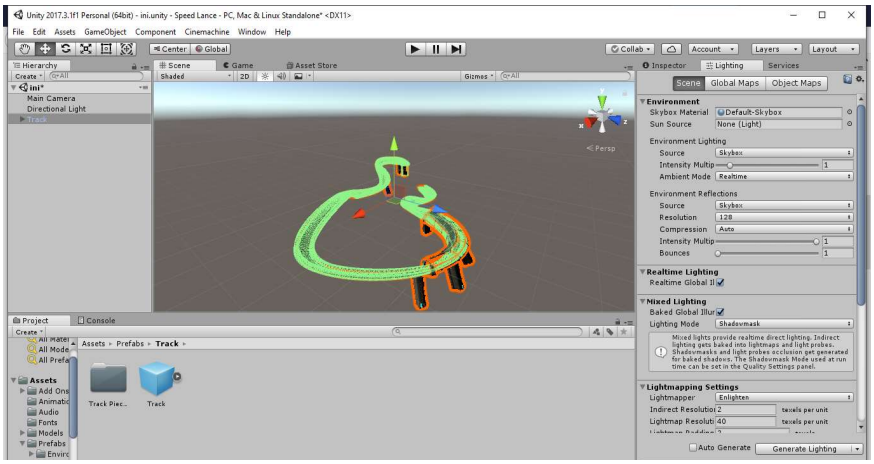
   a. Kendaraan: kendaraan yang tersedia hanya 1 jenis saja.
   b. *Lap*: Berapa kali kendaraan melewati seluruh lintasan, di mana tiap 1 putaran akan dicatat angkanya.
   c. Lintasan: tempat kendaraan melintas untuk mencapai garis *finish*.

C. CARA KERJA PERMAINAN

Cara kerja game ini mirip game balapan lain. Pemain dapat mengandalkan [W] untuk melaju, [SHIFT] untuk rem, [D] untuk belok ke kanan, dan [A] untuk belok ke kiri. Pemain melintasi lintasan balap untuk sampai ke garis finish dengan melalui 3 *lap*. Tiap durasi satu *lap* akan tercatat di sebelah kiri atas dan jumlah lap tercatat di bagian kanan belakang kendaraan. Kecepatan tercatat di bagian kiri belakang kendaraan.

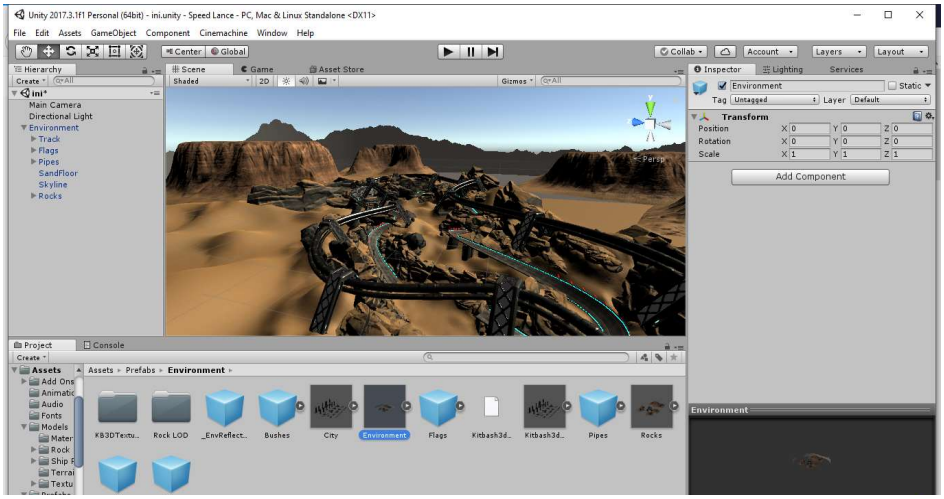
D. PEMBUATAN

1. Pertama kita buat new project dengan setting 3D. Lalu *import asset* yang sudah disediakan ke dalam *project*. Kemudian ke folder “Track” dan seret (drag) Track.prefab ke *tab hierarchy*.



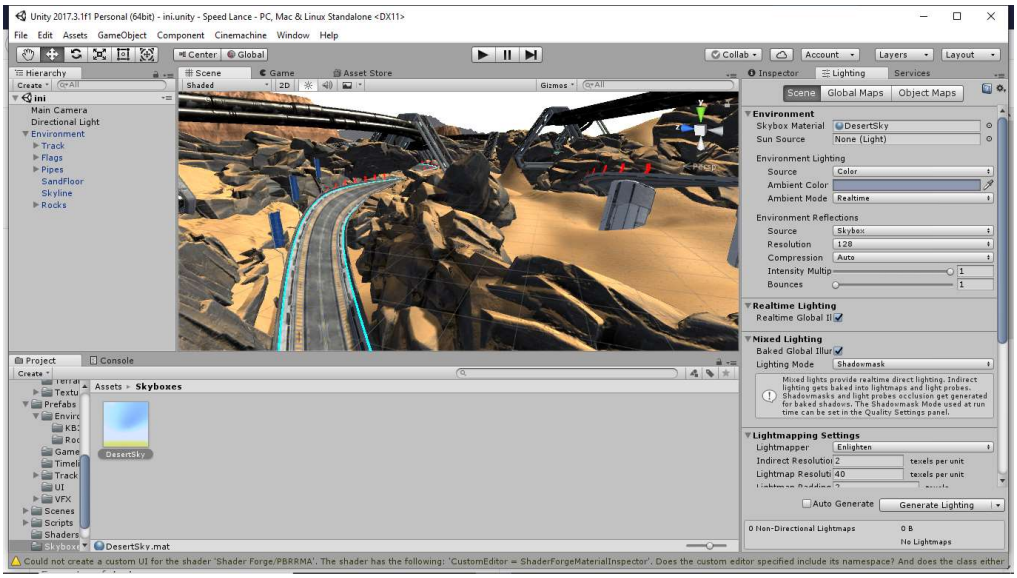
Gambar 1 *Import* Track.prefab

2. Lalu buatlah environment disekitar lintasan supaya terlihat lebih hidup dengan cara import prefab-nya (Pipes, Flags, SandFloor, dan Skylane). Lalu buat empty object dan set position untuk semua sumbu ke koordinat 0. Kemudian pilih prefab yang sudah di import lalu dimasukkan ke dalam empty object dan rename menjadi environment. Kemudian seret objek environment ke dalam folder Enviroment.



Gambar 2 Pembuatan Environment

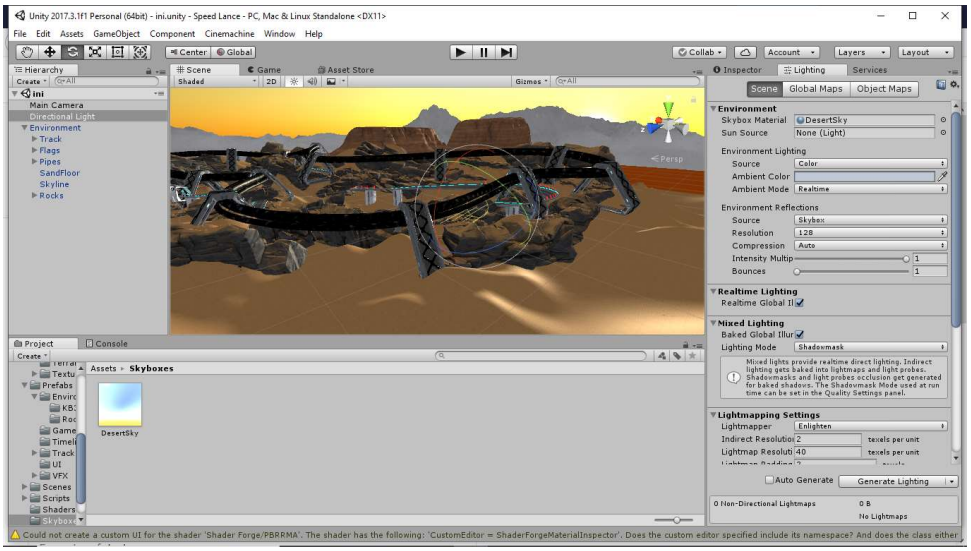
3. Terlihat bahwa lingkungannya kekurangan cahaya, dan untuk memperbaiki hal tersebut, maka seret DesertSky.mat yang ada di dalam folder Skyboxes ke arah langit, lalu ke *tab Lighting* dan ganti Source Environment Lighting dari Skybox menjadi color, dan atur Ambient Color



sehingga pemandangannya lebih cerah dan tidak terlalu gelap.

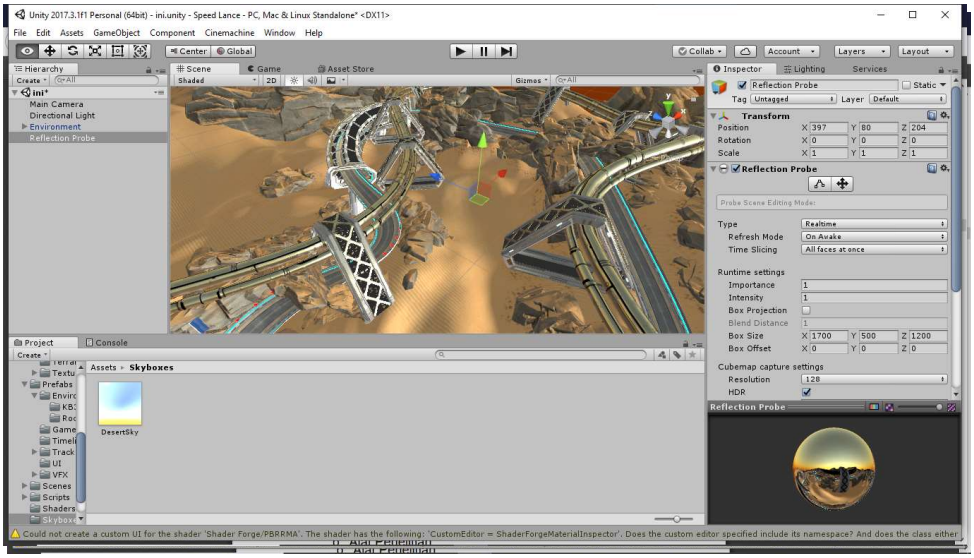
Gambar 3 Penyesuaian *Lighting*

Lalu sesuaikan Directional Light untuk mengatur posisi matahari, apakah ingin pencahayaan siang, sore atau malam.



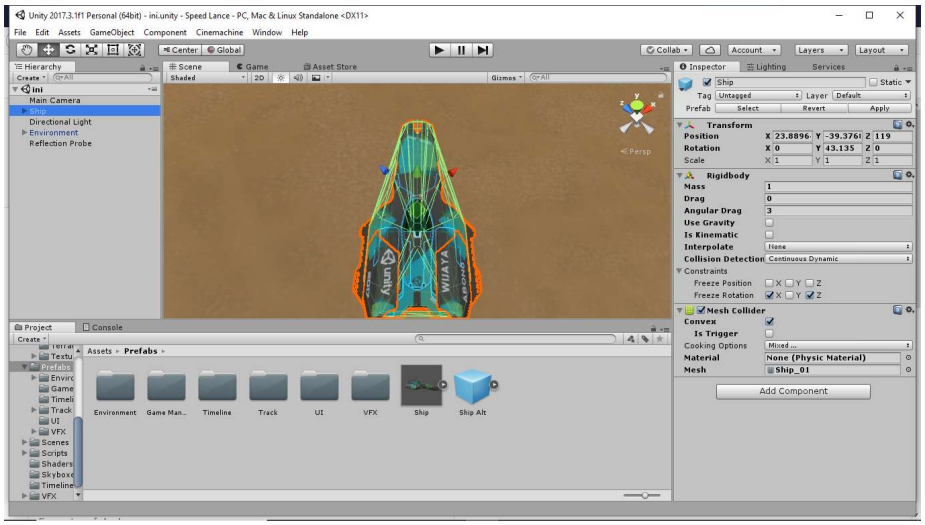
Gambar 4 Penyesuaian Directional Light (posisi matahari)

Lalu tambahkan Reflection Probe dan ganti type-nya dari Baked menjadi Realtime dan atur size agar mencakup semua lintasan balap.



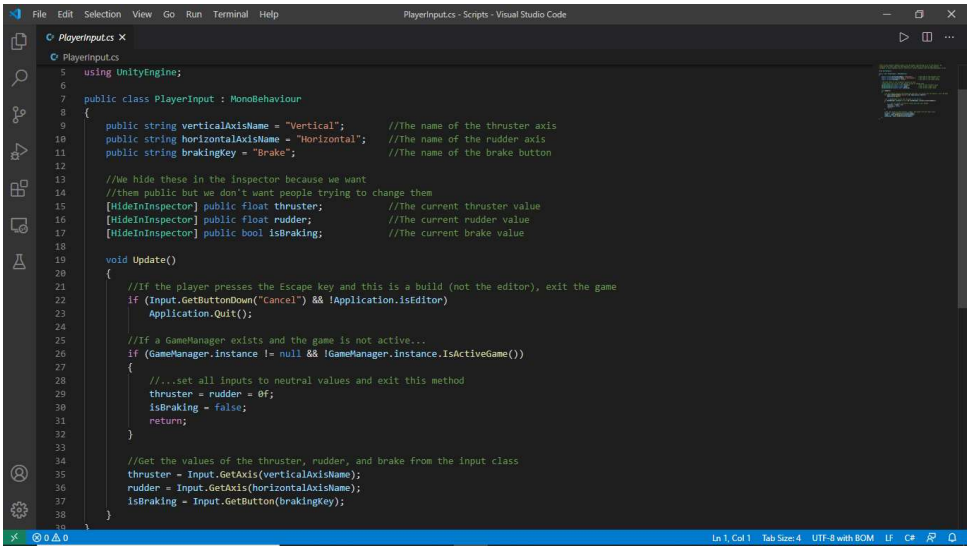
Gambar 5 Menambahkan Reflection Probe

4. Pilih Ship.prefab pada folder Prefabs dan seret ke dalam scene pada Unity, lalu tambahkan component Rigidbody dan Mesh Collider dengan konfigurasi seperti gambar di bawah ini.



Gambar 6 Rigidbody Ship

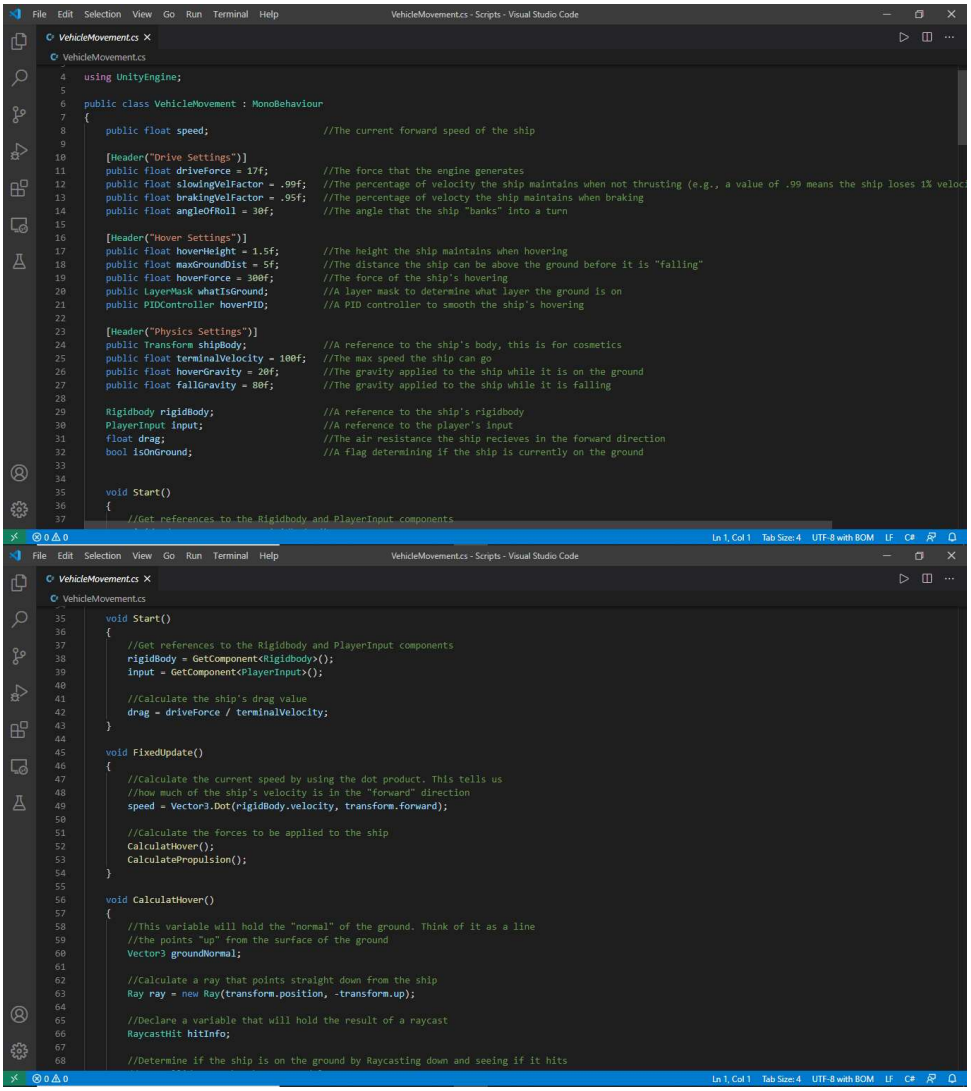
Lalu buatlah script untuk pergerakan kendaraan seperti sumber kode yang ada di bawah ini.



```
using UnityEngine;

public class PlayerInput : MonoBehaviour
{
    public string verticalAxisName = "Vertical";        //The name of the thruster axis
    public string horizontalAxisName = "Horizontal";    //The name of the rudder axis
    public string brakingKey = "Brake";                 //The name of the brake button

    //We hide these in the inspector because we want
    //them public but we don't want people trying to change them
    [HideInInspector] public float thruster;            //The current thruster value
    [HideInInspector] public float rudder;              //The current rudder value
    [HideInInspector] public bool isBraking;            //The current brake value

    void Update()
    {
        //If the player presses the Escape key and this is a build (not the editor), exit the game
        if (Input.GetButtonDown("Cancel") && !Application.isEditor)
            Application.Quit();

        //If a GameManager exists and the game is not active...
        if (GameManager.instance != null && !GameManager.instance.IsActiveGame())
        {
            //...set all inputs to neutral values and exit this method
            thruster = rudder = 0f;
            isBraking = false;
            return;
        }

        //Get the values of the thruster, rudder, and brake from the input class
        thruster = Input.GetAxis(verticalAxisName);
        rudder = Input.GetAxis(horizontalAxisName);
        isBraking = Input.GetButton(brakingKey);
    }
}
```

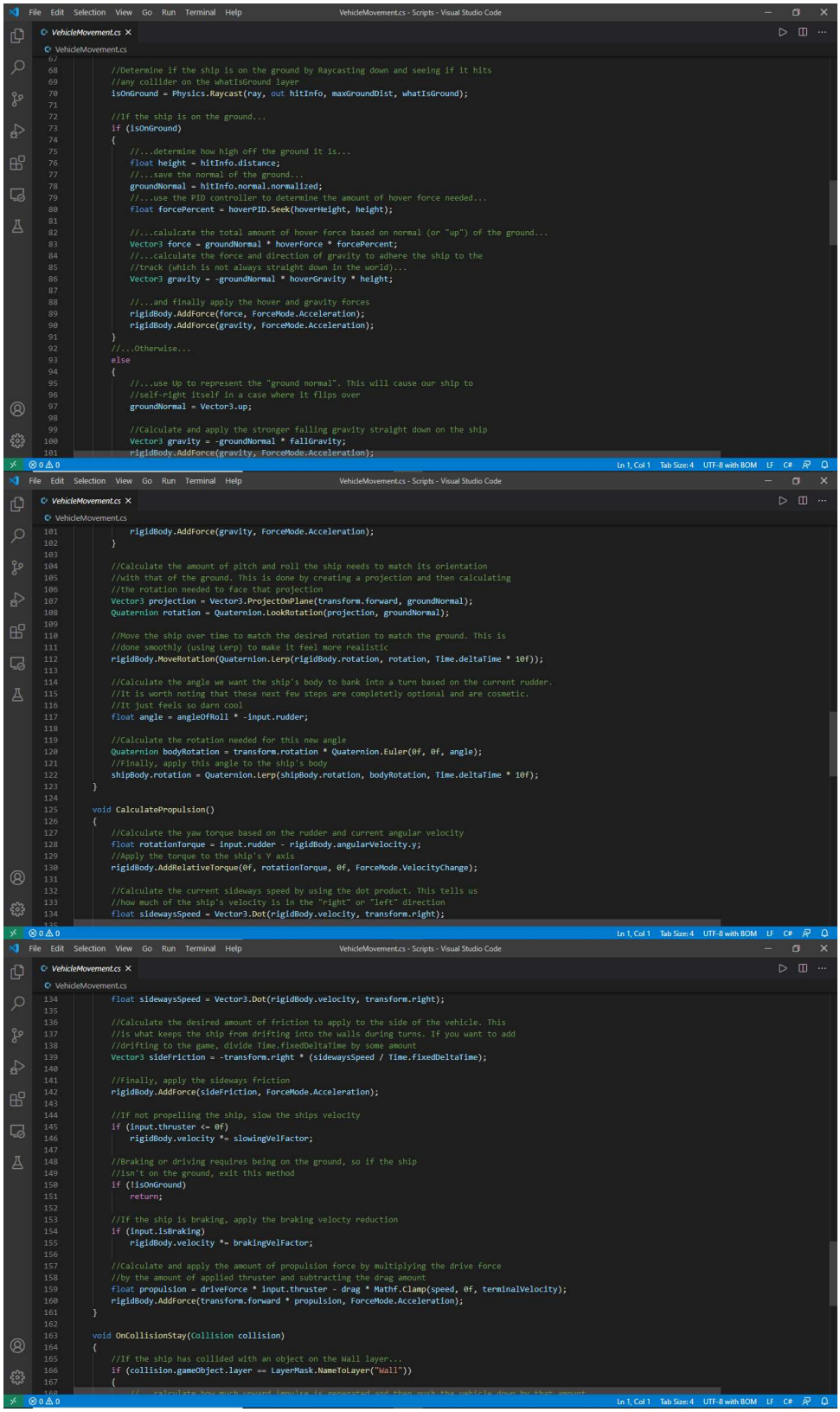
Gambar 7 kode PlayerInputs.cs

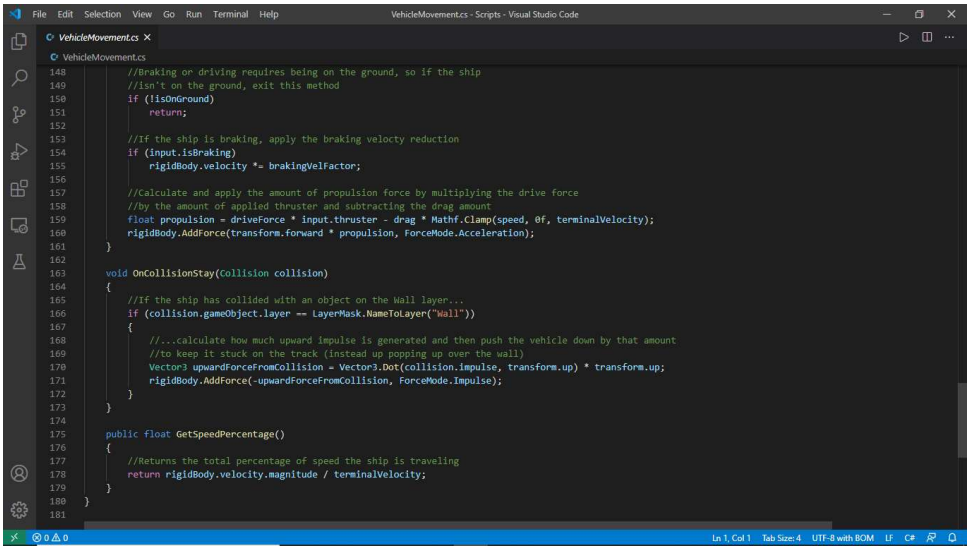
Kemudian buatlah script untuk menangani berbagai fenomena fisik yang terjadi pada kendaraan dalam balapan seperti kode di bawah ini.



```
using UnityEngine;

public class VehicleMovement : MonoBehaviour
{
    public float speed;                     //The current forward speed of the ship

    [Header("Drive Settings")]
    public float driveForce = 17f;          //The force that the engine generates
    public float slowingVelFactor = .99f;   //The percentage of velocity the ship maintains when not thrusting (e.g., a value of .99 means the ship loses 1% veloc
    public float brakingVelFactor = .95f;   //The percentage of velocty the ship maintains when braking
    public float angleOfRoll = 30f;         //The angle that the ship "banks" into a turn

    [Header("Hover Settings")]
    public float hoverHeight = 1.5f;        //The height the ship maintains when hovering
    public float maxGroundDist = 5f;        //The distance the ship can be above the ground before it is "falling"
    public float hoverForce = 300f;         //The force of the ship's hovering
    public LayerMask whatIsGround;          //A layer mask to determine what layer the ground is on
    public PIDController hoverPID;          //A PID controller to smooth the ship's hovering

    [Header("Physics Settings")]
    public Transform shipBody;              //A reference to the ship's body, this is for cosmetics
    public float terminalVelocity = 100f;   //The max speed the ship can go
    public float hoverGravity = 20f;        //The gravity applied to the ship while it is on the ground
    public float fallGravity = 80f;         //The gravity applied to the ship while it is falling

    Rigidbody rigidBody;                    //A reference to the ship's rigidbody
    PlayerInput input;                      //A reference to the player's input
    float drag;                             //The air resistance the ship recieves in the forward direction
    bool isOnGround;                        //A flag determining if the ship is currently on the ground


    void Start()
    {
        //Get references to the Rigidbody and PlayerInput components
```

```
    void Start()
    {
        //Get references to the Rigidbody and PlayerInput components
        rigidBody = GetComponent<Rigidbody>();
        input = GetComponent<PlayerInput>();

        //Calculate the ship's drag value
        drag = driveForce / terminalVelocity;
    }

    void FixedUpdate()
    {
        //Calculate the current speed by using the dot product. This tells us
        //how much of the ship's velocity is in the "forward" direction
        speed = Vector3.Dot(rigidBody.velocity, transform.forward);

        //Calculate the forces to be applied to the ship
        CalculatHover();
        CalculatePropulsion();
    }

    void CalculatHover()
    {
        //This variable will hold the "normal" of the ground. Think of it as a line
        //the points "up" from the surface of the ground
        Vector3 groundNormal;

        //Calculate a ray that points straight down from the ship
        Ray ray = new Ray(transform.position, -transform.up);

        //Declare a variable that will hold the result of a raycast
        RaycastHit hitInfo;

        //Determine if the ship is on the ground by Raycasting down and seeing if it hits
```

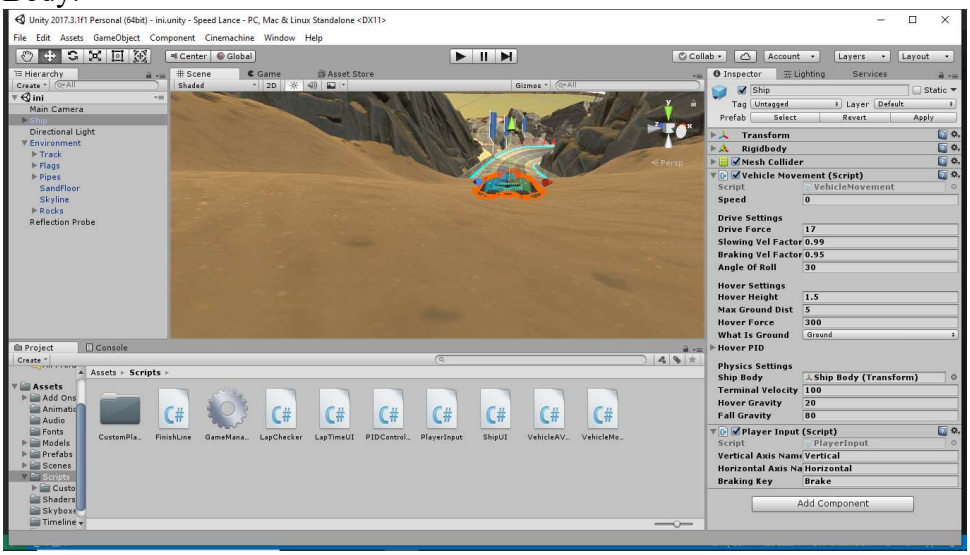

File Edit Selection View Go Run Terminal Help
VehicleMovement.cs - Scripts - Visual Studio Code
VehicleMovement.cs
//Determine if the ship is on the ground by Raycasting down and seeing if it hits
//any collider on the whatIsGround layer
isOnGround = Physics.Raycast(ray, out hitInfo, maxGroundDist, whatIsGround);
//If the ship is on the ground...
if (isOnGround)
{
//...determine how high off the ground it is...
float height = hitInfo.distance;
//...save the normal of the ground...
groundNormal = hitInfo.normal.normalized;
//...use the PID controller to determine the amount of hover force needed...
float forcePercent = hoverPID.Seek(hoverHeight, height);
//...calulcate the total amount of hover force based on normal (or "up") of the ground...
Vector3 force = groundNormal * hoverForce * forcePercent;
//...calculate the force and direction of gravity to adhere the ship to the
//track (which is not always straight down in the world)...
Vector3 gravity = -groundNormal * hoverGravity * height;
//...and finally apply the hover and gravity forces
rigidBody.AddForce(force, ForceMode.Acceleration);
rigidBody.AddForce(gravity, ForceMode.Acceleration);
}
//...Otherwise...
else
{
//...use Up to represent the "ground normal". This will cause our ship to
//self-right itself in a case where it flips over
groundNormal = Vector3.up;
//Calculate and apply the stronger falling gravity straight down on the ship
Vector3 gravity = -groundNormal * fallGravity;
rigidBody.AddForce(gravity, ForceMode.Acceleration);
Ln 1, Col 1 Tab Size: 4 UTF-8 with BOM LF C#
File Edit Selection View Go Run Terminal Help
VehicleMovement.cs - Scripts - Visual Studio Code
VehicleMovement.cs
rigidBody.AddForce(gravity, ForceMode.Acceleration);
}
//Calculate the amount of pitch and roll the ship needs to match its orientation
//with that of the ground. This is done by creating a projection and then calculating
//the rotation needed to face that projection
Vector3 projection = Vector3.ProjectOnPlane(transform.forward, groundNormal);
Quaternion rotation = Quaternion.LookRotation(projection, groundNormal);
//Move the ship over time to match the desired rotation to match the ground. This is
//done smoothly (using Lerp) to make it feel more realistic
rigidBody.MoveRotation(Quaternion.Lerp(rigidBody.rotation, rotation, Time.deltaTime * 10f));
//Calculate the angle we want the ship's body to bank into a turn based on the current rudder.
//It is worth noting that these next few steps are completetly optional and are cosmetic.
//It just feels so darn cool
float angle = angleOfRoll * -input.rudder;
//Calculate the rotation needed for this new angle
Quaternion bodyRotation = transform.rotation * Quaternion.Euler(0f, 0f, angle);
//Finally, apply this angle to the ship's body
shipBody.rotation = Quaternion.Lerp(shipBody.rotation, bodyRotation, Time.deltaTime * 10f);
}
void CalculatePropulsion()
{
//Calculate the yaw torque based on the rudder and current angular velocity
float rotationTorque = input.rudder - rigidBody.angularVelocity.y;
//Apply the torque to the ship's Y axis
rigidBody.AddRelativeTorque(0f, rotationTorque, 0f, ForceMode.VelocityChange);
//Calculate the current sideways speed by using the dot product. This tells us
//how much of the ship's velocity is in the "right" or "left" direction
float sidewaysSpeed = Vector3.Dot(rigidBody.velocity, transform.right);
Ln 1, Col 1 Tab Size: 4 UTF-8 with BOM LF C#
File Edit Selection View Go Run Terminal Help
VehicleMovement.cs - Scripts - Visual Studio Code
VehicleMovement.cs
float sidewaysSpeed = Vector3.Dot(rigidBody.velocity, transform.right);
//Calculate the desired amount of friction to apply to the side of the vehicle. This
//is what keeps the ship from drifting into the walls during turns. If you want to add
//drifting to the game, divide Time.fixedDeltaTime by some amount
Vector3 sideFriction = -transform.right * (sidewaysSpeed / Time.fixedDeltaTime);
//Finally, apply the sideways friction
rigidBody.AddForce(sideFriction, ForceMode.Acceleration);
//If not propelling the ship, slow the ships velocity
if (input.thruster <= 0f)
rigidBody.velocity *= slowingVelFactor;
//Braking or driving requires being on the ground, so if the ship
//isn't on the ground, exit this method
if (!isOnGround)
return;
//If the ship is braking, apply the braking velocty reduction
if (input.isBraking)
rigidBody.velocity *= brakingVelFactor;
//Calculate and apply the amount of propulsion force by multiplying the drive force
//by the amount of applied thruster and subtracting the drag amount
float propulsion = driveForce * input.thruster - drag * Mathf.Clamp(speed, 0f, terminalVelocity);
rigidBody.AddForce(transform.forward * propulsion, ForceMode.Acceleration);
}
void OnCollisionStay(Collision collision)
{
//If the ship has collided with an object on the Wall layer...
if (collision.gameObject.layer == LayerMask.NameToLayer("Wall"))
{
Ln 1, Col 1 Tab Size: 4 UTF-8 with BOM LF C#



```
        //Braking or driving requires being on the ground, so if the ship
        //isn't on the ground, exit this method
        if (!isOnGround)
            return;

        //If the ship is braking, apply the braking velocty reduction
        if (input.isBraking)
            rigidBody.velocity *= brakingVelFactor;

        //Calculate and apply the amount of propulsion force by multiplying the drive force
        //by the amount of applied thruster and subtracting the drag amount
        float propulsion = driveForce * input.thruster - drag * Mathf.Clamp(speed, 0f, terminalVelocity);
        rigidBody.AddForce(transform.forward * propulsion, ForceMode.Acceleration);
    }

    void OnCollisionStay(Collision collision)
    {
        //If the ship has collided with an object on the Wall layer...
        if (collision.gameObject.layer == LayerMask.NameToLayer("Wall"))
        {
            //...calculate how much upward impulse is generated and then push the vehicle down by that amount
            //to keep it stuck on the track (instead up popping up over the wall)
            Vector3 upwardForceFromCollision = Vector3.Dot(collision.impulse, transform.up) * transform.up;
            rigidBody.AddForce(-upwardForceFromCollision, ForceMode.Impulse);
        }
    }

    public float GetSpeedPercentage()
    {
        //Returns the total percentage of speed the ship is traveling
        return rigidBody.velocity.magnitude / terminalVelocity;
    }
}
```

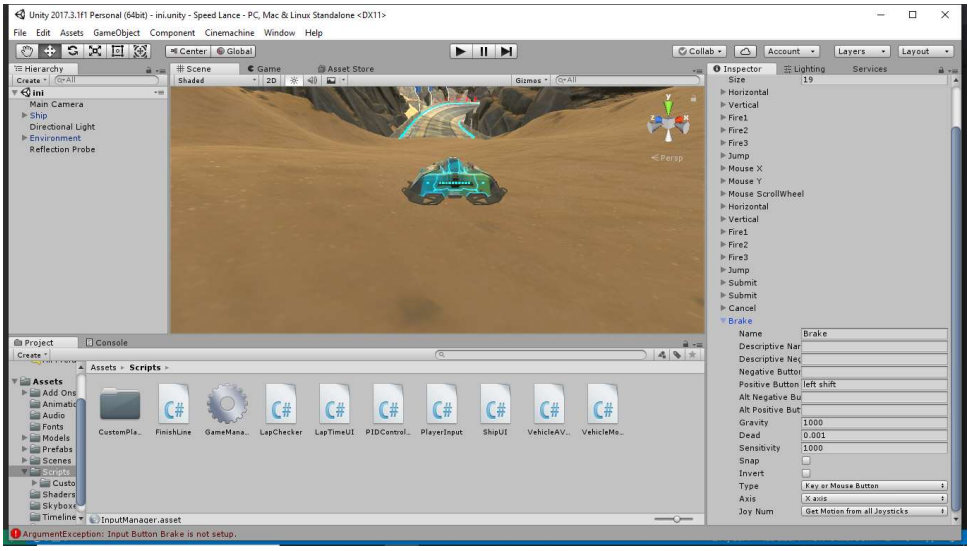
Gambar 8 sumber kode VehicleMovement

Kemudian seret kedua script tersebut ke dalam objek ship dan buat layer baru dengan nama Ground, lalu pada VehicleMovement Inspector, “What is Groud” diubah menjadi layer Groud dan “Ship Body” menjadi Ship Body.



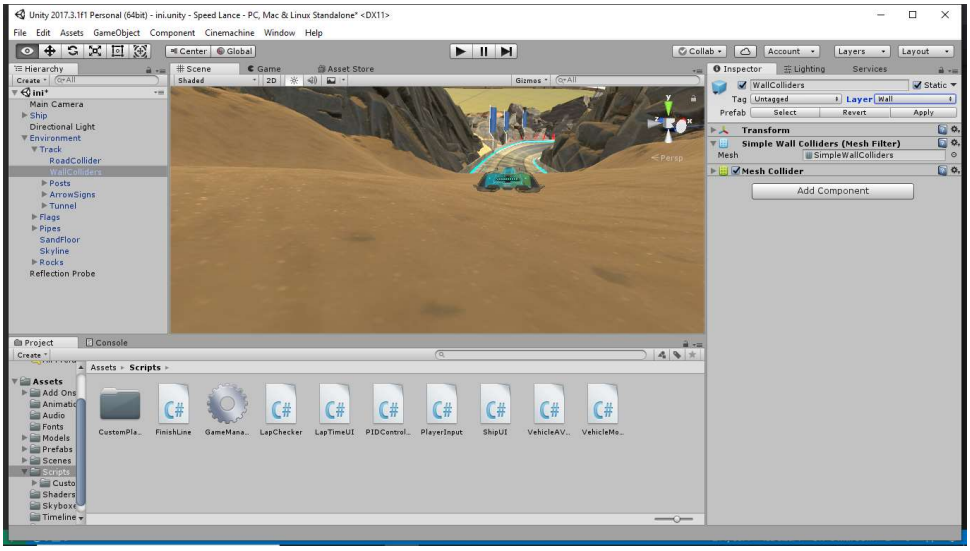
Gambar 9 Tambah Script PlayerInput.cs dan VehicleMovement.cs

Lalu klik menu Edit > Project Settings > Input, Lalu tambahkan jumlah Axes sesuai kebutuhan karena akan ditambahkan Brake yang menggunakan [SHIFT] kiri.



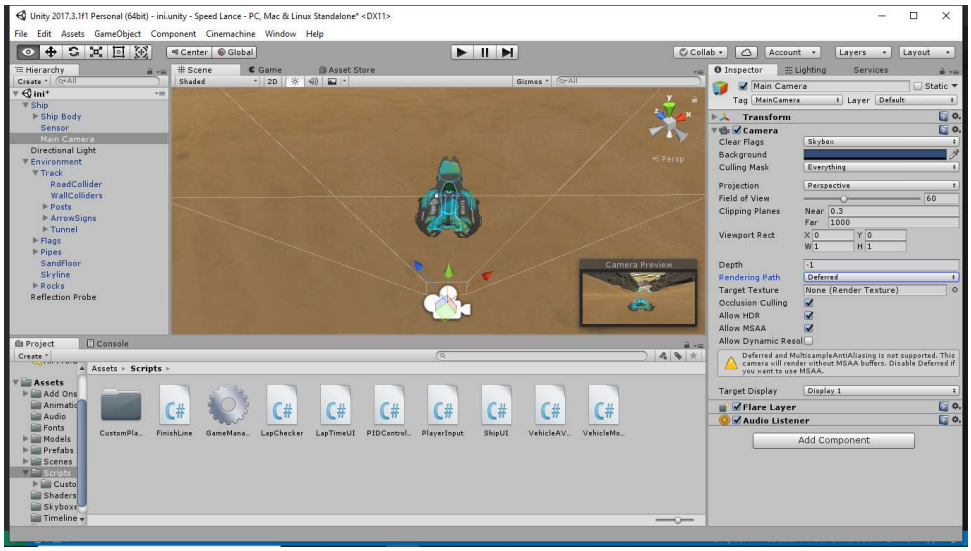
Gambar 10 Menambahkan Input Brake

Kemudian tambahkan lagi Layer selain Ground, yaitu Wall dan Post Processing. Lalu atur agar SandFloor dan RoadCollider berada pada layer Ground, dan objek WallColliders berada di layer Wall.



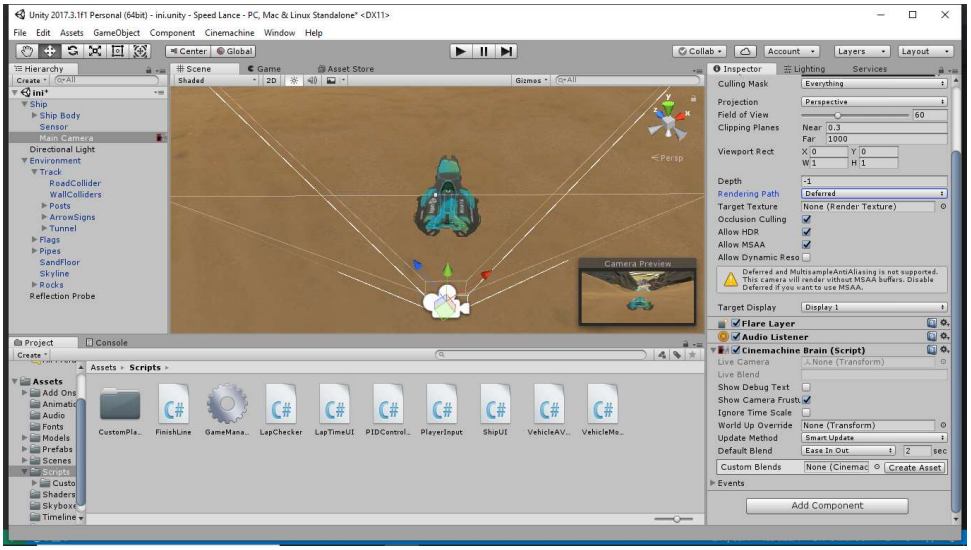
Gambar 11 Atur Layer

5. Untuk membuat kamera mengikuti kendaraan, maka pilih objek Main Camera, lalu pilih menu GameObject > Align With View. Lalu seret objek main camera ke dalam objek Ship. Lalu atur Rendering Path ke Deffered



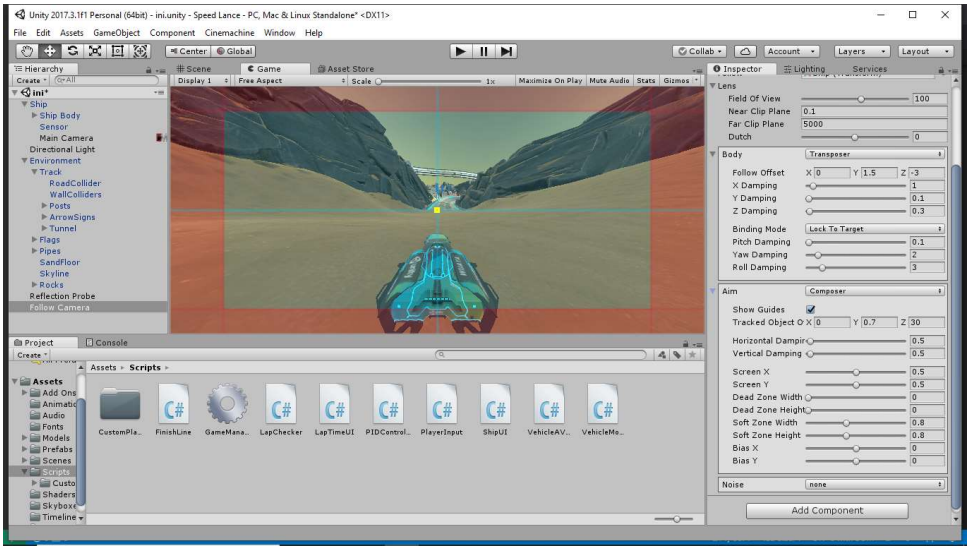
Gambar 12 Atur Main Camera

Kemudian tambahkan component pada objek Main Camera berupa cinemachine dan pilih CinemacihineBrain



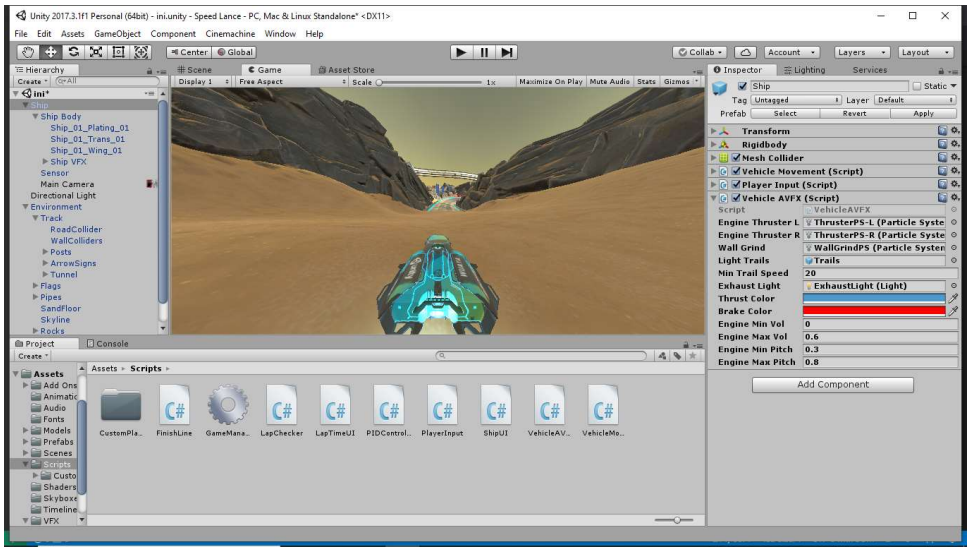
Gambar 13 Menambahkan CinemachineBrain

Lalu pilih menu Cinemachine > Create Virtual Camera. Setelah terbuatnya objek tersebut, ubah nama menjadi Follow Camera. Lalu atur Cinemachine pada kolom Loot At dan Follow menjadi Ship. Lalu atur component Cinemachine seperti gambar di bawah ini.



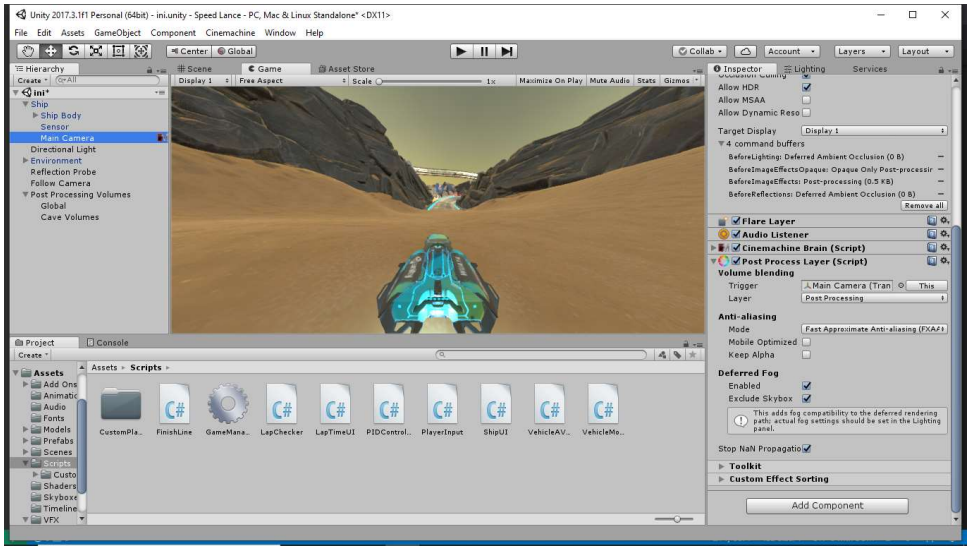
Gambar 14 Mengatur Objek CinemachineBrain

6. Untuk menambahkan VFX, masuk ke direktori Prefabs > VFX lalu seret Ship VFX.prefab ke dalam objek Ship Body. Lalu tambahkan VehicleAVFX.cs dalam direktori Scripts kepada objek Ship.



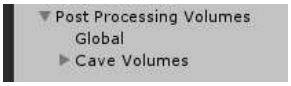
Gambar 15 Menambah VFX pada objek Ship

7. Untuk menambahkan Post Processing, pilih objek Main Camera, lalu tambahkan component Post-process Layer dan atur Layer component ke Post Processing. Uncheck Allow MSAA pada component Camera, dan atur Anti-aliasing Mode menjadi Fast Approximate Anti-aliasing (FXAA).

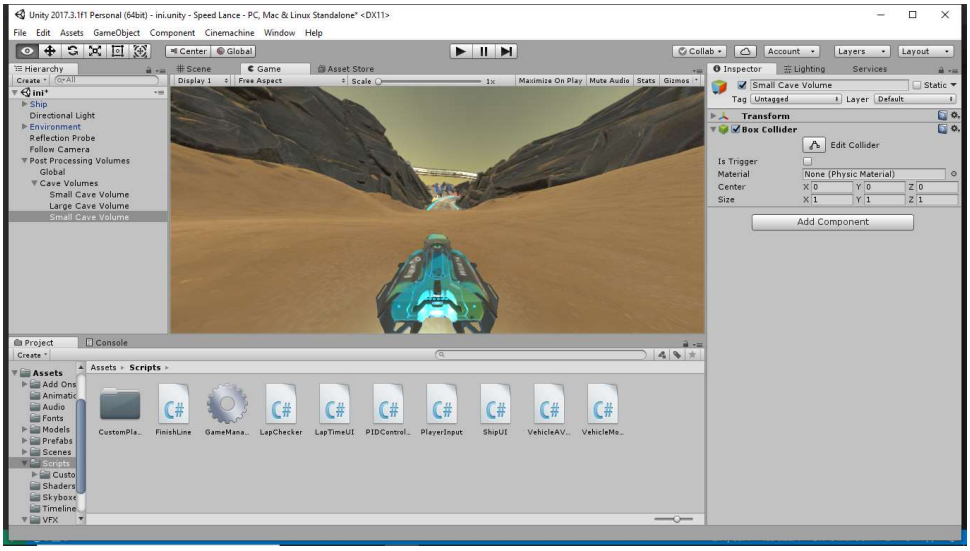


Gambar 16 Menambahkan Post-process Layer

Lalu tambahkan empty object dengan susunan seperti gambar di samping, lalu buat lagi empty

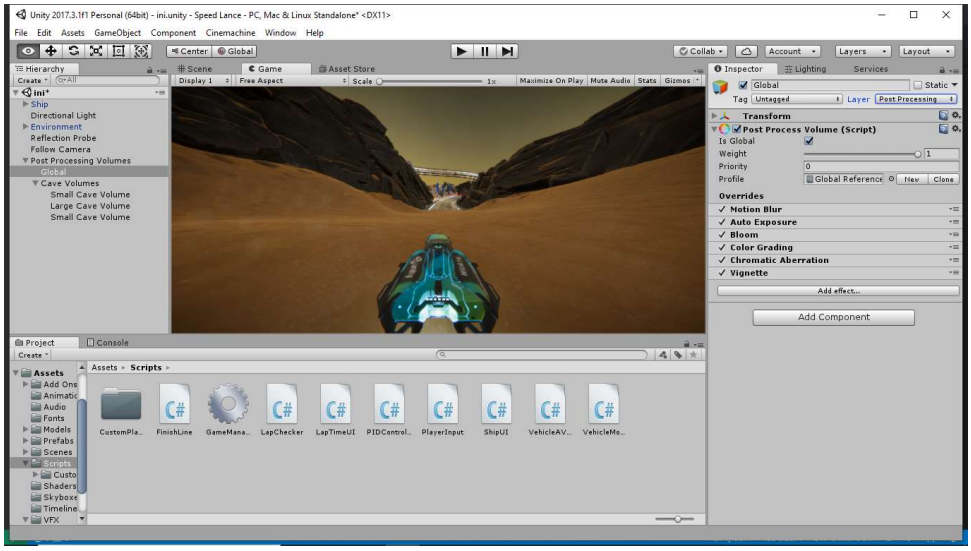


object seperti gambar di bawah, namun isi per objek dengan Box Collider



Gambar 17 Menambah Empty Object dan Box Collider

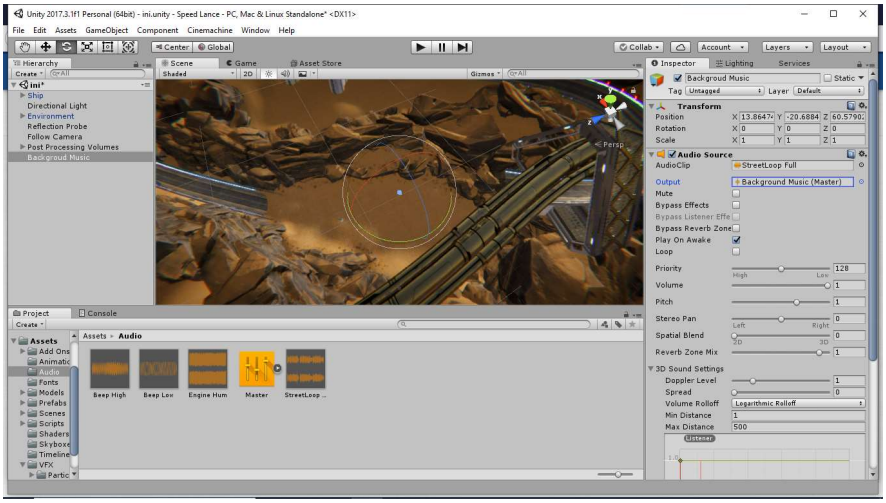
Kemudian, pilih objek Global, ubah Layer ke Post Processing dan tambahkan component Post-process Volume, centang Is Global dan atur Profile ke Global Reference.



Gambar 18 Mengatur Post Processing

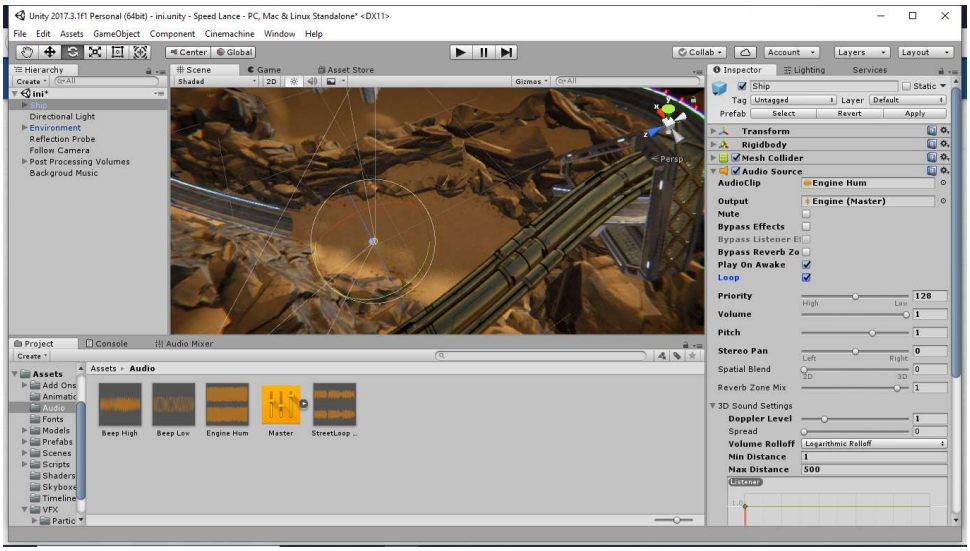
Kemudian taruh masing-masing objek yang ada di dalam objek Cave Volumes ke terowongan gua yang ada di lintasan, lalu pilih ketiganya dan tambahkan component Post-process Volume dan atur Profile menjadi Cave Volume, centang Is Trigger pada component Box Collider, atur Blend Distance dengan nilai 4 dan Priority dengan nilai 1 pada Post-process Volume, dan atur Layer agar berada pada Post Processing.

8. Untuk menambahkan audio, maka buatlah objek Audio Source dan ubah nama menjadi Backgroud Music, lalu pada AudioClip, pilih StreetLoop Full, dan untuk Output pilih Background Music, centang Loop.



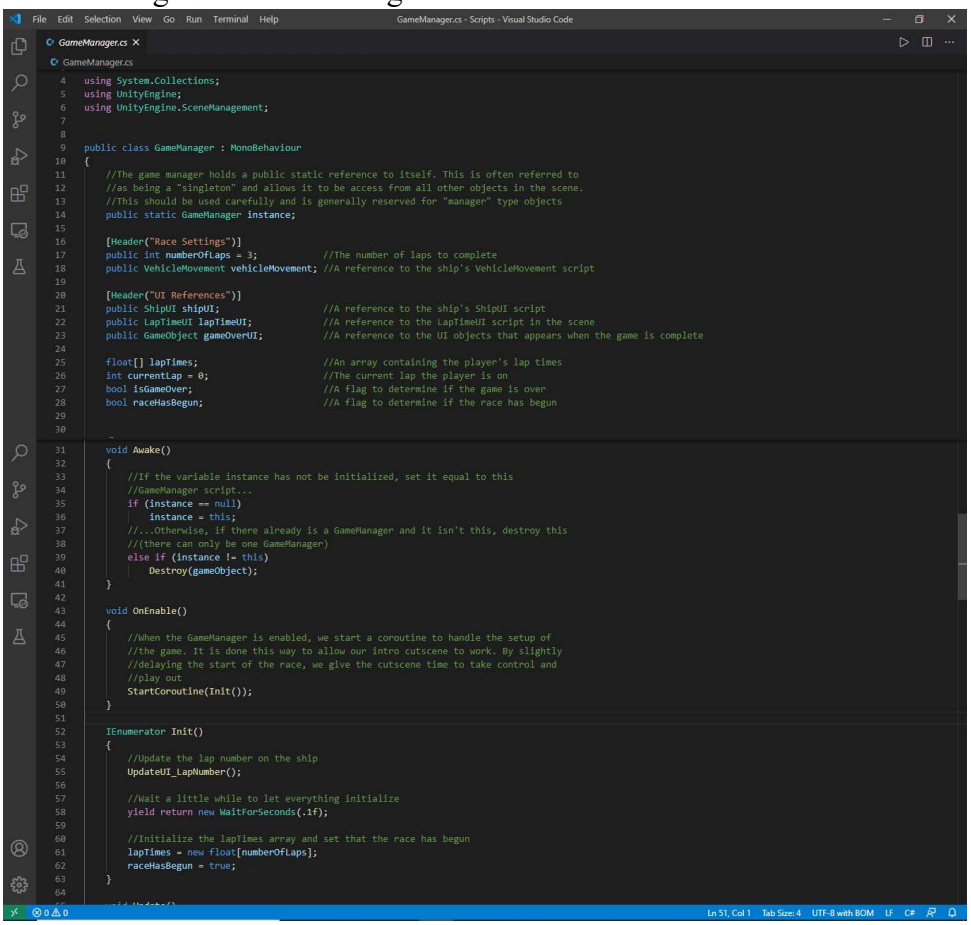
Gambar 19 Background Music

Lalu pada objek Ship, tambahkan component Audio Source dan atur AudioClip menjadi Engine Hum dan Output menjadi Engine serta centang Loop.

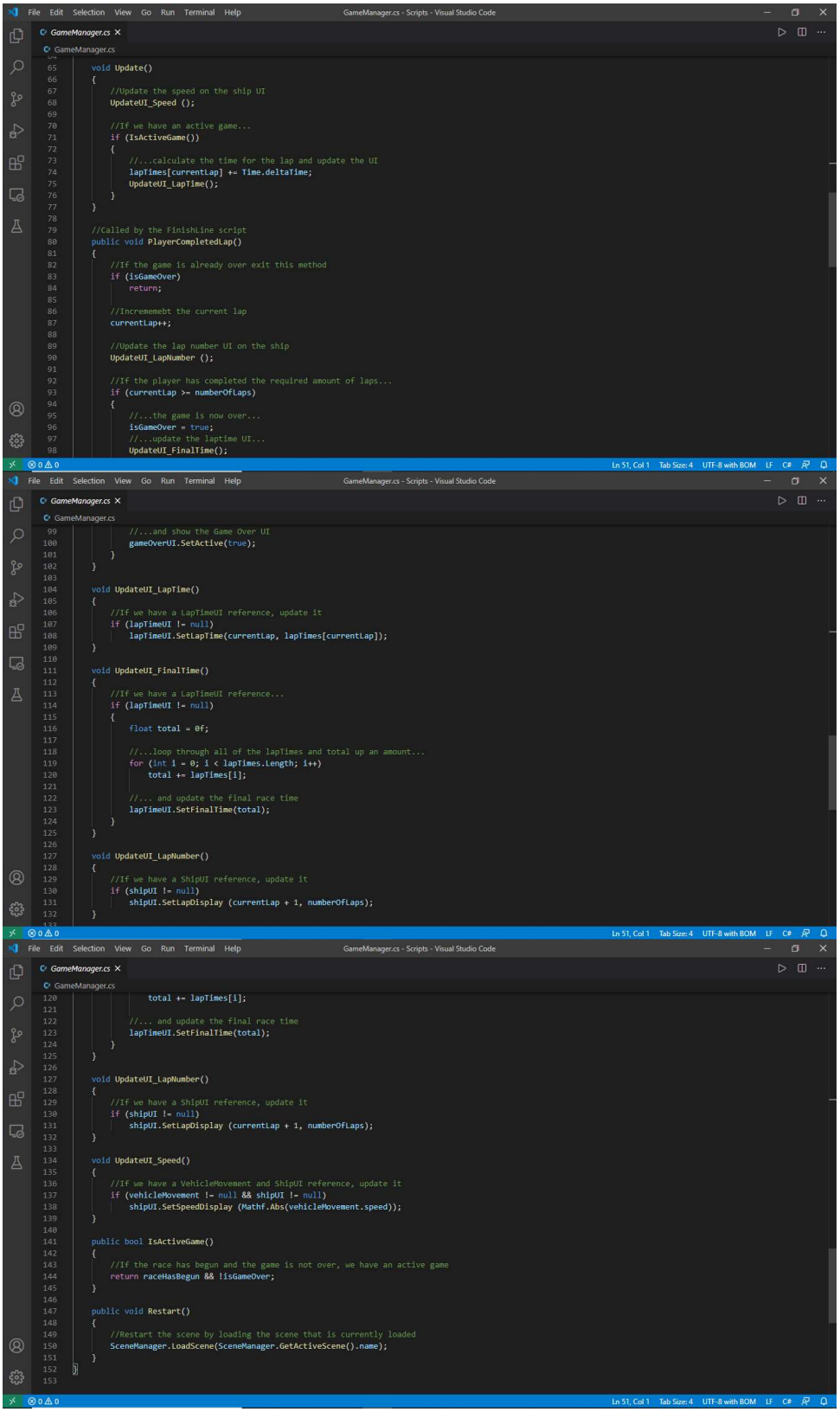


Gambar 20 Menambahkan Suara Mesin Kendaraan

9. Untuk menambahkan GameManager, buatlah script dengan nama GameManager dan isikan dengan kode berikut.



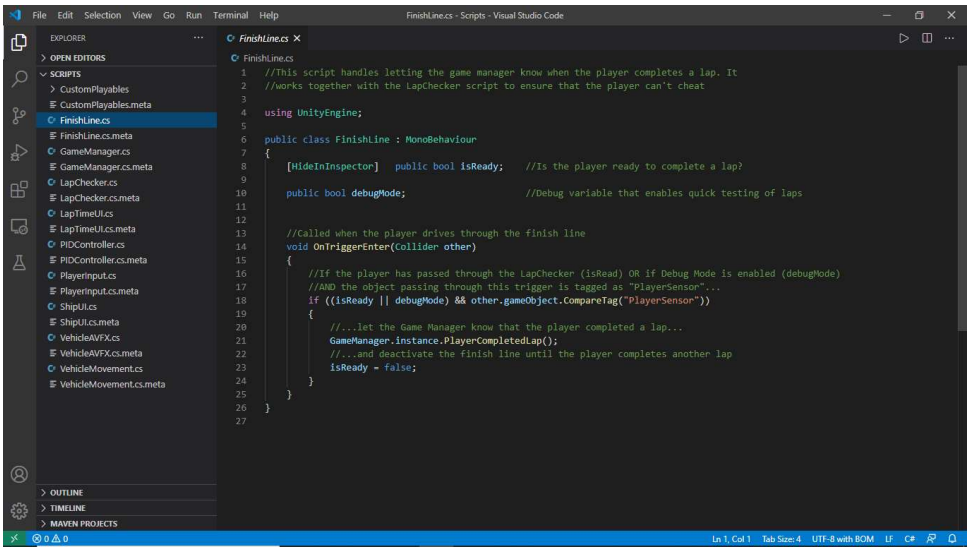
```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;


public class GameManager : MonoBehaviour
{
    //The game manager holds a public static reference to itself. This is often referred to
    //as being a "singleton" and allows it to be access from all other objects in the scene.
    //This should be used carefully and is generally reserved for "manager" type objects
    public static GameManager instance;

    [Header("Race Settings")]
    public int numberOfLaps = 3;            //The number of laps to complete
    public VehicleMovement vehicleMovement; //A reference to the ship's VehicleMovement script

    [Header("UI References")]
    public ShipUI shipUI;                   //A reference to the ship's ShipUI script
    public LapTimeUI lapTimeUI;             //A reference to the LapTimeUI script in the scene
    public GameObject gameOverUI;           //A reference to the UI objects that appears when the game is complete

    float[] lapTimes;                       //An array containing the player's lap times
    int currentLap = 0;                     //The current lap the player is on
    bool isGameOver;                        //A flag to determine if the game is over
    bool raceHasBegun;                      //A flag to determine if the race has begun


    void Awake()
    {
        //If the variable instance has not be initialized, set it equal to this
        //GameManager script...
        if (instance == null)
            instance = this;
        //...Otherwise, if there already is a GameManager and it isn't this, destroy this
        //(there can only be one GameManager)
        else if (instance != this)
            Destroy(gameObject);
    }

    void OnEnable()
    {
        //When the GameManager is enabled, we start a coroutine to handle the setup of
        //the game. It is done this way to allow our intro cutscene to work. By slightly
        //delaying the start of the race, we give the cutscene time to take control and
        //play out
        StartCoroutine(Init());
    }

    IEnumerator Init()
    {
        //Update the lap number on the ship
        UpdateUI_LapNumber();

        //Wait a little while to let everything initialize
        yield return new WaitForSeconds(.1f);

        //Initialize the lapTimes array and set that the race has begun
        lapTimes = new float[numberOfLaps];
        raceHasBegun = true;
    }
```



```
void Update()
{
    //Update the speed on the ship UI
    UpdateUI_Speed ();

    //If we have an active game...
    if (IsActiveGame())
    {
        //...calculate the time for the lap and update the UI
        lapTimes[currentLap] += Time.deltaTime;
        UpdateUI_LapTime();
    }
}

//Called by the FinishLine script
public void PlayerCompletedLap()
{
    //If the game is already over exit this method
    if (isGameOver)
        return;

    //Incrememebt the current lap
    currentLap++;

    //Update the lap number UI on the ship
    UpdateUI_LapNumber ();

    //If the player has completed the required amount of laps...
    if (currentLap >= numberOfLaps)
    {
        //...the game is now over...
        isGameOver = true;
        //...update the laptime UI...
        UpdateUI_FinalTime();
        //...and show the Game Over UI
        gameOverUI.SetActive(true);
    }
}

void UpdateUI_LapTime()
{
    //If we have a LapTimeUI reference, update it
    if (lapTimeUI != null)
        lapTimeUI.SetLapTime(currentLap, lapTimes[currentLap]);
}

void UpdateUI_FinalTime()
{
    //If we have a LapTimeUI reference...
    if (lapTimeUI != null)
    {
        float total = 0f;

        //...loop through all of the lapTimes and total up an amount...
        for (int i = 0; i < lapTimes.Length; i++)
            total += lapTimes[i];

        //... and update the final race time
        lapTimeUI.SetFinalTime(total);
    }
}

void UpdateUI_LapNumber()
{
    //If we have a ShipUI reference, update it
    if (shipUI != null)
        shipUI.SetLapDisplay (currentLap + 1, numberOfLaps);
}

void UpdateUI_Speed()
{
    //If we have a VehicleMovement and ShipUI reference, update it
    if (vehicleMovement != null && shipUI != null)
        shipUI.SetSpeedDisplay (Mathf.Abs(vehicleMovement.speed));
}

public bool IsActiveGame()
{
    //If the race has begun and the game is not over, we have an active game
    return raceHasBegun && !isGameOver;
}

public void Restart()
{
    //Restart the scene by loading the scene that is currently loaded
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
}
```

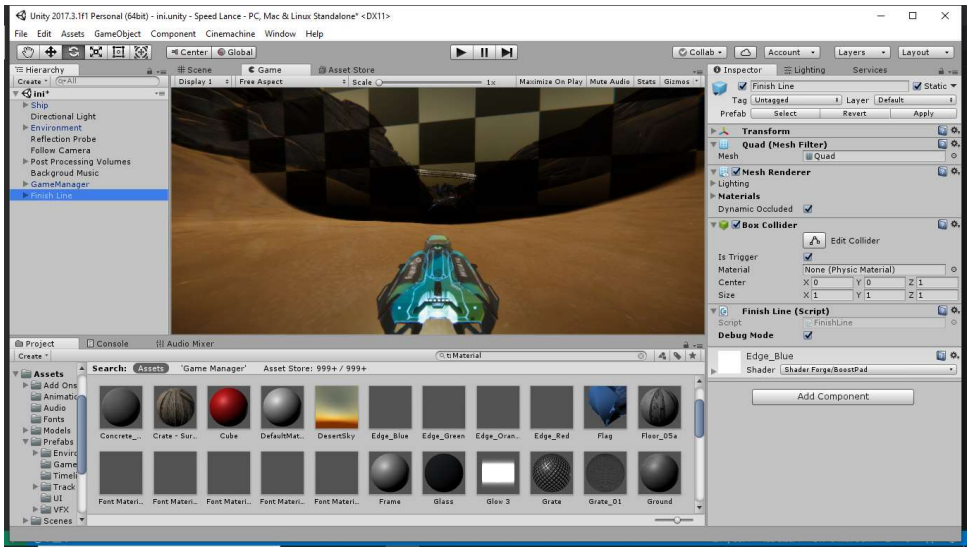
Gambar 21 Script GameManager

Kemudian buat GameManager.prefab ke dalam Scene lalu seret script yang sudah dibuat ke dalam component dari prefab tersebut. Atur Vehicle menjadi Ship. Seret Finish Line.prefab ke dalam Scene dan cari material bernama Edge_Blue dan seret ke objek Finish Line serta centang Debug Mode pada Finish Line Script. Berikut adalah kode Finish Line.



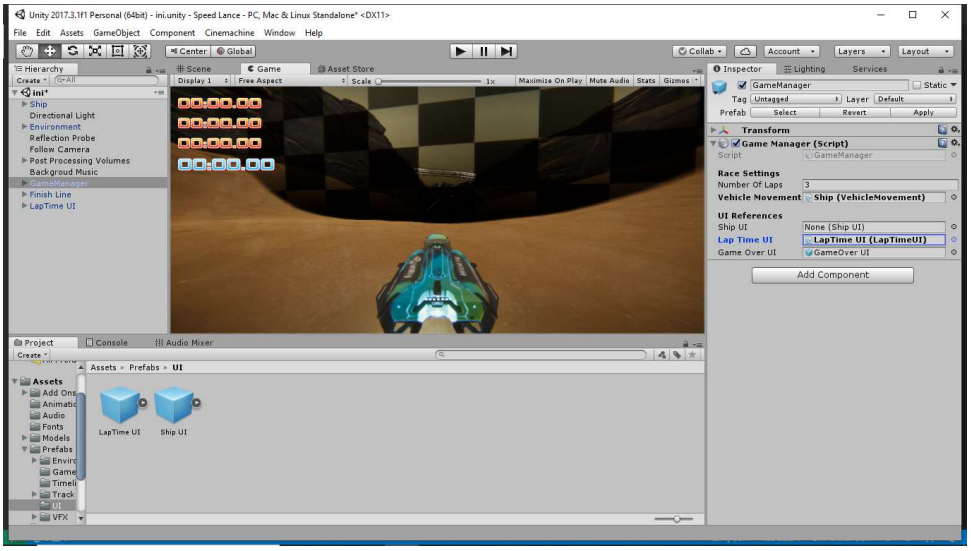
```
//This script handles letting the game manager know when the player completes a lap. It
//works together with the LapChecker script to ensure that the player can't cheat

using UnityEngine;

public class FinishLine : MonoBehaviour
{
    [HideInInspector]   public bool isReady;    //Is the player ready to complete a lap?

    public bool debugMode;                      //Debug variable that enables quick testing of laps


    //Called when the player drives through the finish line
    void OnTriggerEnter(Collider other)
    {
        //If the player has passed through the LapChecker (isRead) OR if Debug Mode is enabled (debugMode)
        //AND the object passing through this trigger is tagged as "PlayerSensor"...
        if ((isReady || debugMode) && other.gameObject.CompareTag("PlayerSensor"))
        {
            //...let the Game Manager know that the player completed a lap...
            GameManager.instance.PlayerCompletedLap();
            //...and deactivate the finish line until the player completes another lap
            isReady = false;
        }
    }
}
```

Gambar 22 Kode Finish Line



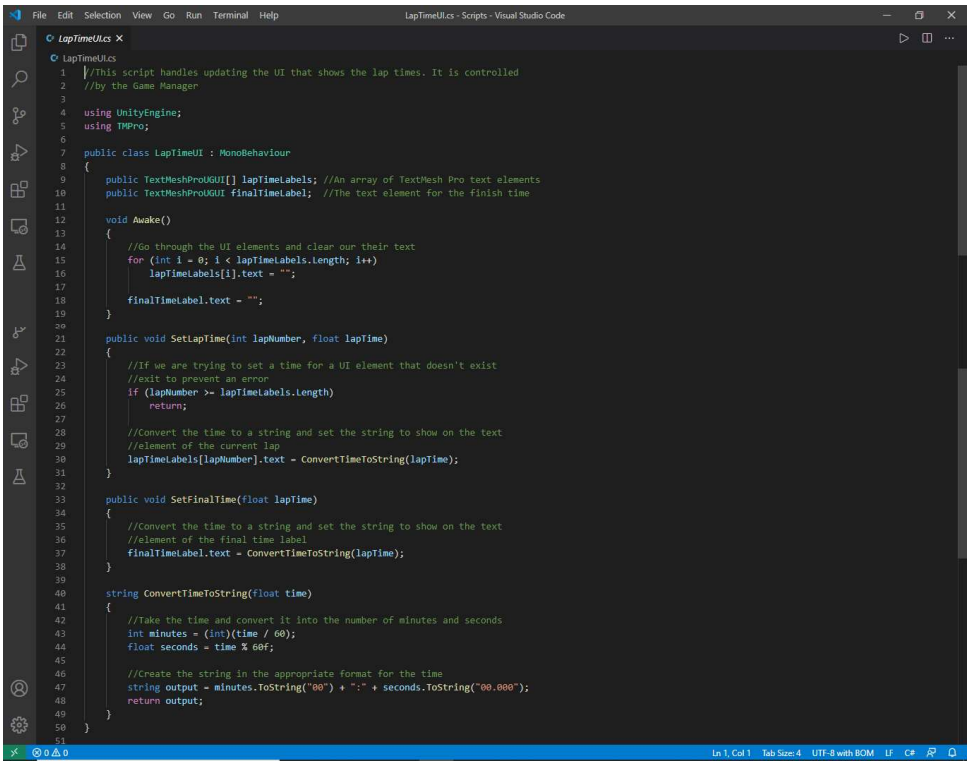
Gambar 23 Mengatur Konfigurasi Finish Line

10. Untuk menampilkan jumlah *lap*, maka tambahkan objek LapTime UI.prefab ke dalam Scene, lalu di objek GameManager, atur Lap Time UI menjadi LapTime UI dari prefab tadi.



Gambar 24 Konfigurasi Lap Time UI

Berikut adalah kode untuk LapTime UI.prefab yang bernama Lap Time UI.cs

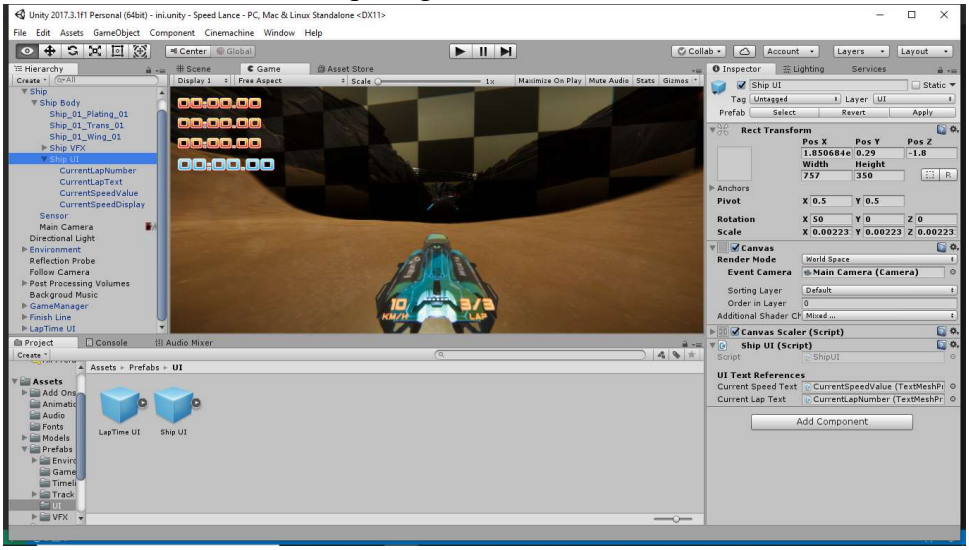


```
//This script handles updating the UI that shows the lap times. It is controlled
//by the Game Manager

using UnityEngine;
using TMPro;

public class LapTimeUI : MonoBehaviour
{
    public TextMeshProUGUI[] lapTimeLabels; //An array of TextMesh Pro text elements
    public TextMeshProUGUI finalTimeLabel;  //The text element for the finish time

    void Awake()
    {
        //Go through the UI elements and clear our their text
        for (int i = 0; i < lapTimeLabels.Length; i++)
            lapTimeLabels[i].text = "";

        finalTimeLabel.text = "";
    }

    public void SetLapTime(int lapNumber, float lapTime)
    {
        //If we are trying to set a time for a UI element that doesn't exist
        //exit to prevent an error
        if (lapNumber >= lapTimeLabels.Length)
            return;

        //Convert the time to a string and set the string to show on the text
        //element of the current lap
        lapTimeLabels[lapNumber].text = ConvertTimeToString(lapTime);
    }

    public void SetFinalTime(float lapTime)
    {
        //Convert the time to a string and set the string to show on the text
        //element of the final time label
        finalTimeLabel.text = ConvertTimeToString(lapTime);
    }

    string ConvertTimeToString(float time)
    {
        //Take the time and convert it into the number of minutes and seconds
        int minutes = (int)(time / 60);
        float seconds = time % 60f;

        //Create the string in the appropriate format for the time
        string output = minutes.ToString("00") + ":" + seconds.ToString("00.000");
        return output;
    }
}
```

Gambar 25 Kode Lap Time UI.cs

11. Untuk menambahkan informasi yang tertempel pada kendaraan, maka seret Ship UI ke dalam Scene, atur Render Mode menjadi World Space,

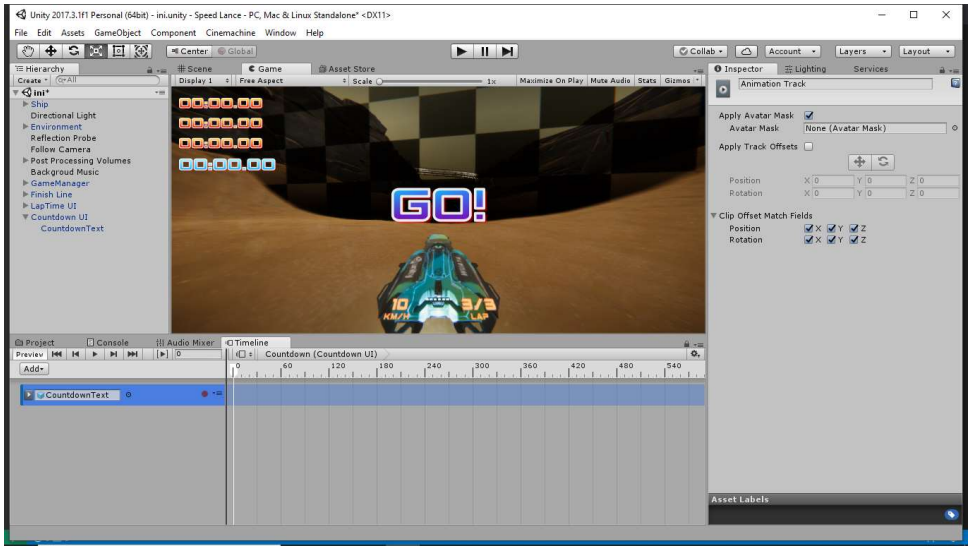
Event Camera menjadi Main Camera, seret Ship UI ke dalam Ship Body, lalu atur Rect Transform seperti gambar di bawah ini.



Gambar 26 Atur Ship UI

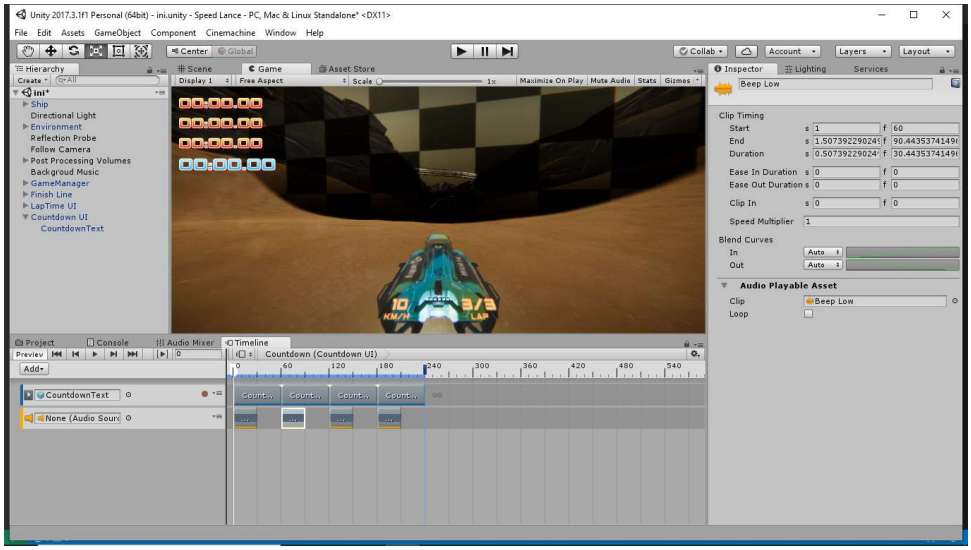
Lalu pilih objek GameManager dan atur Ship UI menjadi Ship UI.

12. Selanjutnya, akan ditambahkan Countdown sebelum balapan, dengan cara seret Countdown UI.prefab ke dalam Scene, lalu buatlah objek Timeline dalam folder Timeline, beri nama Countdown dan seret ke dalam objek Countdown UI. Lalu pilih menu Window > Timeline Editor, lalu seret CountdownText ke dalamnya



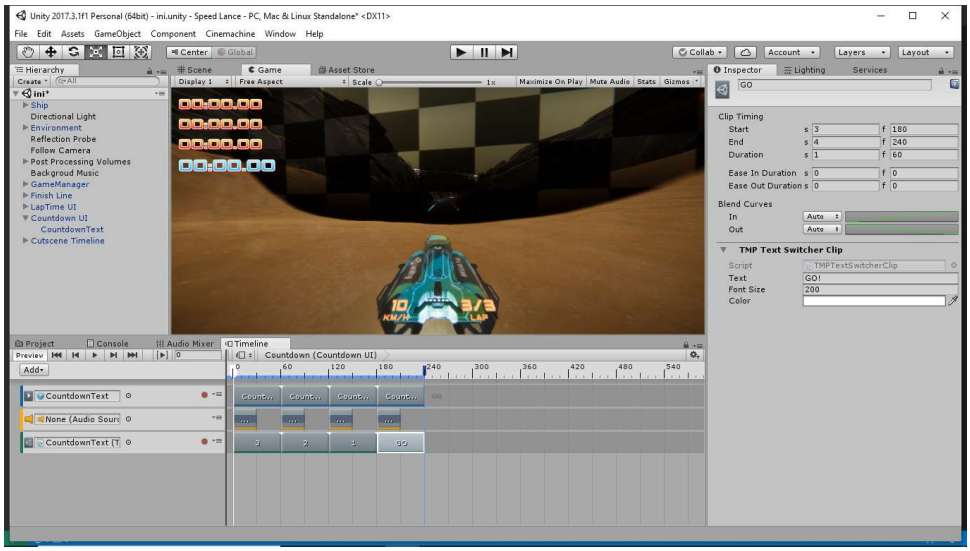
Gambar 27 Timeline Editor

Klik kanan pada CountdownText dan pilih Add From Animation Clip > CountdownTextAnimation, lalu duplicate menjadi 4. Klik kanan di bawah CountdownText dalam Timeline Editor, pilih Audio Track, klik kanan pada timeline dari Audio Track > Add From Audio Clip > Beep Low, lalu duplicate menjadi 3 dan klik kanan lagi > Add From Audio Clip > Beep High, lalu atur waktunya menyesuaikan dengan animasi.



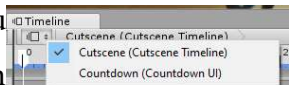
Gambar 28 Mengatur Animasi dan Audio Countdown

Kemudian seret lagi objek CountdownText ke dalam Timeline Editor > TMP Text Switcher Track, lalu klik kanan pada timeline-nya > Add TMP Text Switcher Clip Clip, lalu sesuaikan durasi menjadi 1 detik, ubah namanya menjadi “3” (tanpa tanda petik dua), isi Text dengan “3”, Font Size 150, lalu duplicate sebanyak 3 kali, kemudian sesuaikan tiap duplikasi tersebut agar menjadi hitungan mundur dari 3 sampai 1 dan “GO” dan sesuaikan juga ukuran font.

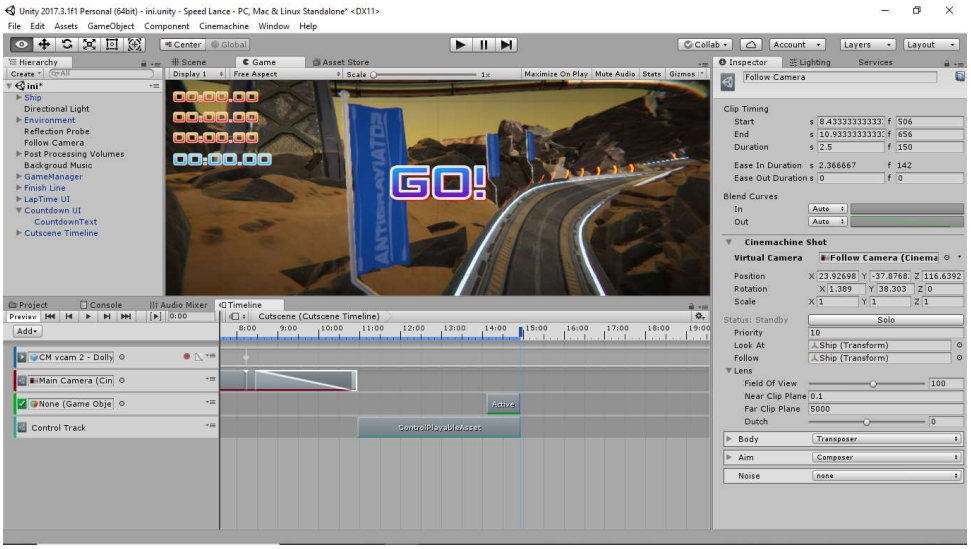


Gambar 29 Mengatur Animasi Countdown

13. Seret Cutscene Timeline.prefab ke dalam Scene, lalu

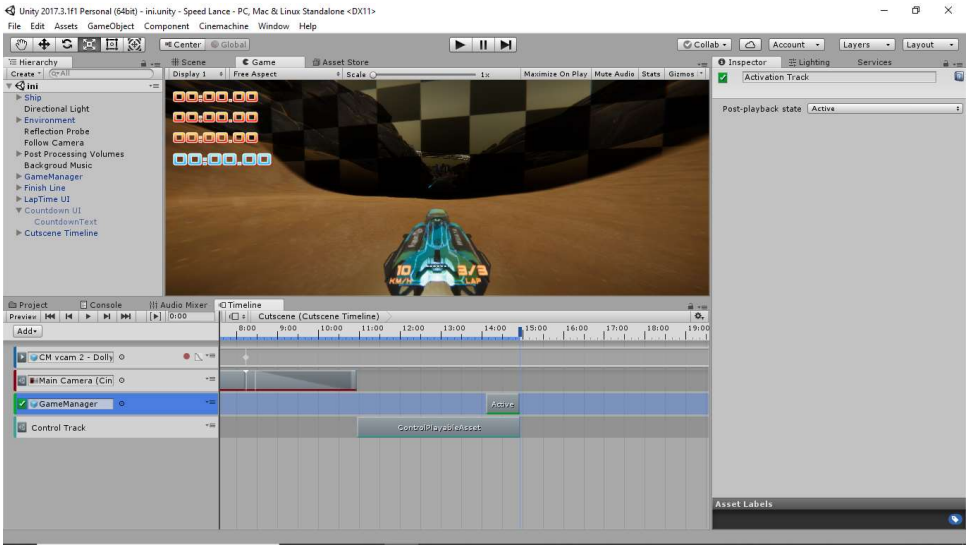


ke Timeline Editor dan pilih Cutscene Timeline, dan klik bundaran kecil pada Cinemachine dan pilih Main Camera, lalu pada timeline tersebut di bagian akhir dari animasinya, klik kanan pada ruang kosong > Add Cinemachine Shot Clip, lalu atur Virtual Camera dengan Follow Camera, lalu geser perlahan dari Clip tersebut ke arah kiri menumpuk clip sebelahnya sesuai selera



Gambar 30 Mengatur Cutscene Timeline

14. Klik Control Track, atur Source Game Object ke Countdown UI, lalu klik Active yang ada di dalam Timeline Editor, lalu klik bundaran kecil dan pilih Game Manager.



Gambar 31 Atur Cutscene Sebelum Melaju