REPUBLIK INDONESIA
KEMENTERIAN HUKUM DAN HAK ASASI MANUSIA

SURAT PENCATATAN
CIPTAAN

Dalam rangka pelindungan ciptaan di bidang ilmu pengetahuan, seni dan sastra berdasarkan Undang-Undang Nomor 28 Tahun 2014 tentang Hak Cipta, dengan ini menerangkan:

Nomor dan tanggal permohonan	:	EC00202246644, 21 Juli 2022
Pencipta		
Nama	:	**Dr. Erwani Merry Sartika, S.T., M.T.,Novie Theresia Br. Pasaribu, S.T., M.T. dkk**
Alamat	:	Jl. Pulolaut No. 6, Bandung , JAWA BARAT, 40114
Kewarganegaraan	:	Indonesia
Pemegang Hak Cipta		
Nama	:	**Universitas Kristen Maranatha**
Alamat	:	Jl. Suria Sumantri No.65, Bandung, JAWA BARAT, 40164
Kewarganegaraan	:	Indonesia
Jenis Ciptaan	:	**Program Komputer**
Judul Ciptaan	:	**Program Komputer Game Driving Berbasis Virtual Reality**
Tanggal dan tempat diumumkan untuk pertama kali di wilayah Indonesia atau di luar wilayah Indonesia	:	4 Juli 2022, di Bandung
Jangka waktu pelindungan	:	Berlaku selama 50 (lima puluh) tahun sejak Ciptaan tersebut pertama kali dilakukan Pengumuman.
Nomor pencatatan	:	000362373

adalah benar berdasarkan keterangan yang diberikan oleh Pemohon.
Surat Pencatatan Hak Cipta atau produk Hak terkait ini sesuai dengan Pasal 72 Undang-Undang Nomor 28 Tahun 2014 tentang Hak Cipta.

a.n Menteri Hukum dan Hak Asasi Manusia
Direktur Jenderal Kekayaan Intelektual
u.b.
Direktur Hak Cipta dan Desain Industri

Anggoro Dasananto
NIP.196412081991031002

Disclaimer:
Dalam hal pemohon memberikan keterangan tidak sesuai dengan surat pernyataan, Menteri berwenang untuk mencabut surat pencatatan permohonan.

LAMPIRAN PENCIPTA

No	Nama	Alamat
1	Dr. Erwani Merry Sartika, S.T., M.T.	Jl. Pulolaut No. 6
2	Novie Theresia Br. Pasaribu, S.T., M.T.	Kompleks Puri Budi Asri E11 Cihanjuang, Parongpong
3	Winda Halim, S.T., M.T.	Jl. Rama No. 17
4	Vieri Candhya Wigayha	Perumahan Resinda Blok C7 No. 15

2022

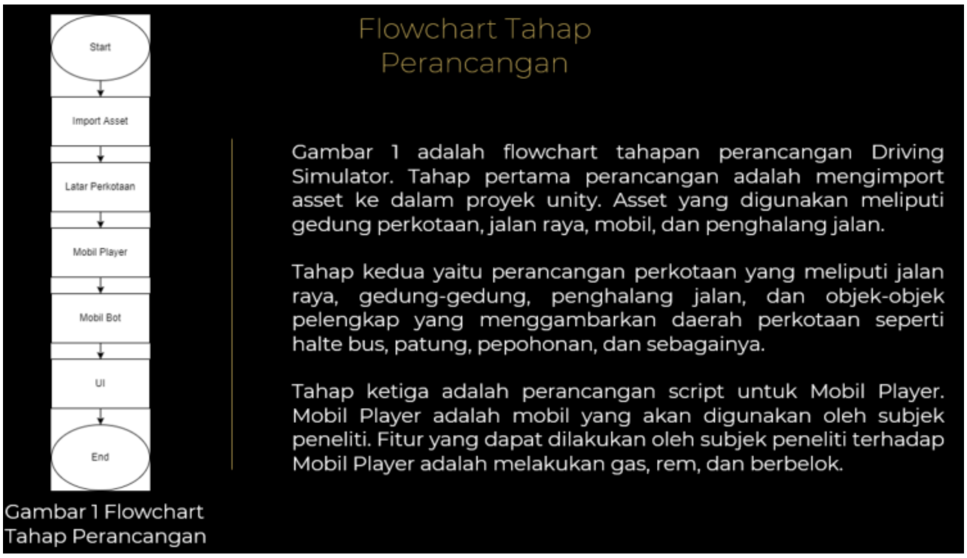
Program Komputer Game Driving berbasis Virtual Reality

Erwani Merry Sartika Novie Theresia Br. Pasaribu Winda Halim Vieri Candhya Wigayha

URAIAN CIPTAAN

Program Komputer “Game Driving berbasis Virtual Reality” merupakan hasil karya yang berisi program komputer untuk mengetahui respon pengendara saat mengalami distraksi berbasis teknologi Virtual Reality. Background dari simulator berupa jalan pada perkotaan (persimpangan, terdapat trotoar, penghalang jalan, terdapat kendaraan lain yang lalu-lalang). Gangguan hanya terdiri dari gangguan luar kendaraan, gangguan berupa penghalang jalan (jumlah dan lokasi), kendaraan yang saling menyalip.

A. Gambaran Perancangan Proyek



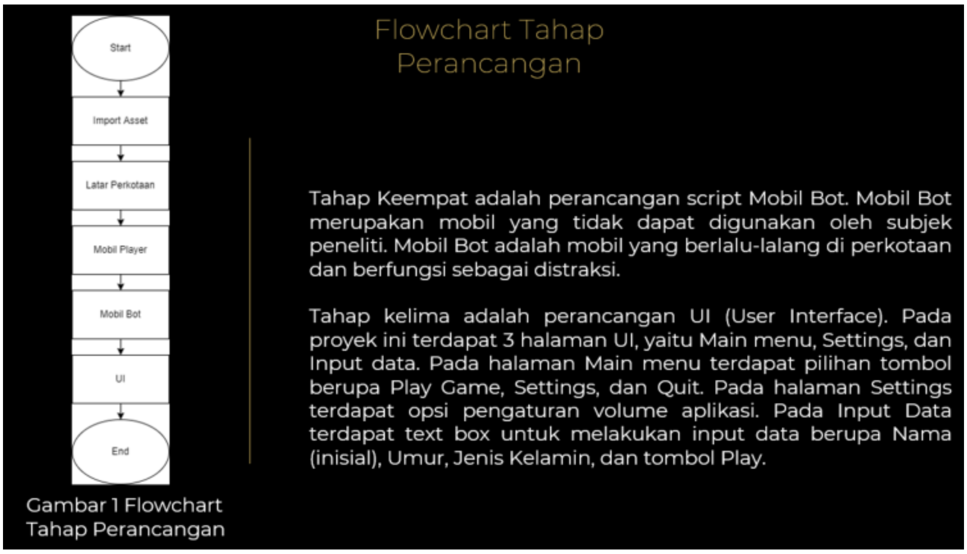
Flowchart Tahap Perancangan

Gambar 1 adalah flowchart tahapan perancangan Driving Simulator. Tahap pertama perancangan adalah mengimport asset ke dalam proyek unity. Asset yang digunakan meliputi gedung perkotaan, jalan raya, mobil, dan penghalang jalan.

Tahap kedua yaitu perancangan perkotaan yang meliputi jalan raya, gedung-gedung, penghalang jalan, dan objek-objek pelengkap yang menggambarkan daerah perkotaan seperti halte bus, patung, pepohonan, dan sebagainya.

Tahap ketiga adalah perancangan script untuk Mobil Player. Mobil Player adalah mobil yang akan digunakan oleh subjek peneliti. Fitur yang dapat dilakukan oleh subjek peneliti terhadap Mobil Player adalah melakukan gas, rem, dan berbelok.

Gambar 1 Flowchart Tahap Perancangan



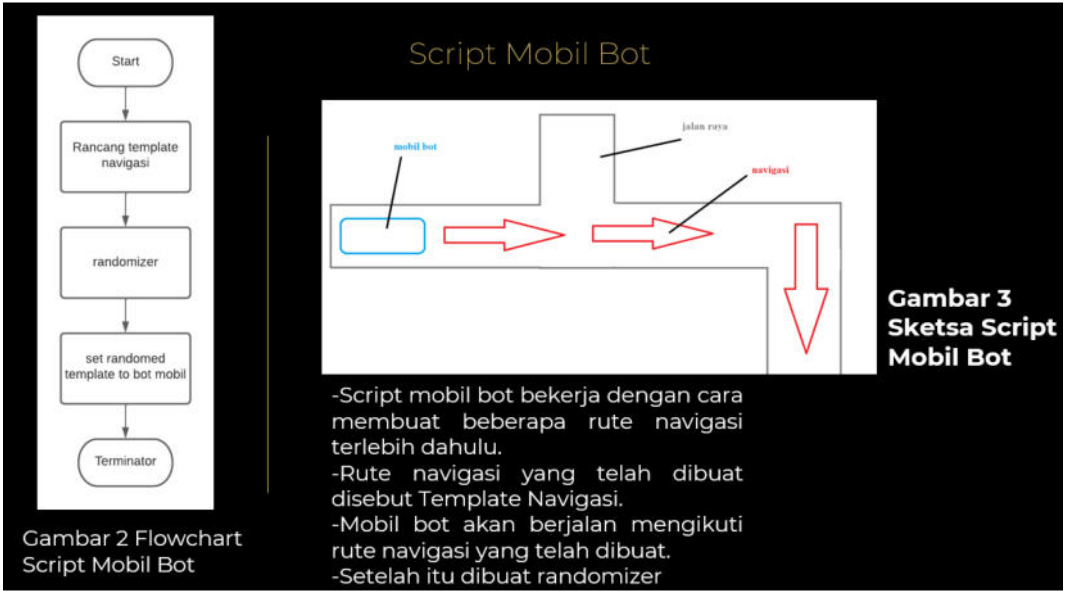
Flowchart Tahap Perancangan

Tahap Keempat adalah perancangan script Mobil Bot. Mobil Bot merupakan mobil yang tidak dapat digunakan oleh subjek peneliti. Mobil Bot adalah mobil yang berlalu-lalang di perkotaan dan berfungsi sebagai distraksi.

Tahap kelima adalah perancangan UI (User Interface). Pada proyek ini terdapat 3 halaman UI, yaitu Main menu, Settings, dan Input data. Pada halaman Main menu terdapat pilihan tombol berupa Play Game, Settings, dan Quit. Pada halaman Settings terdapat opsi pengaturan volume aplikasi. Pada Input Data terdapat text box untuk melakukan input data berupa Nama (inisial), Umur, Jenis Kelamin, dan tombol Play.

Gambar 1 Flowchart Tahap Perancangan

B. Gambaran Script

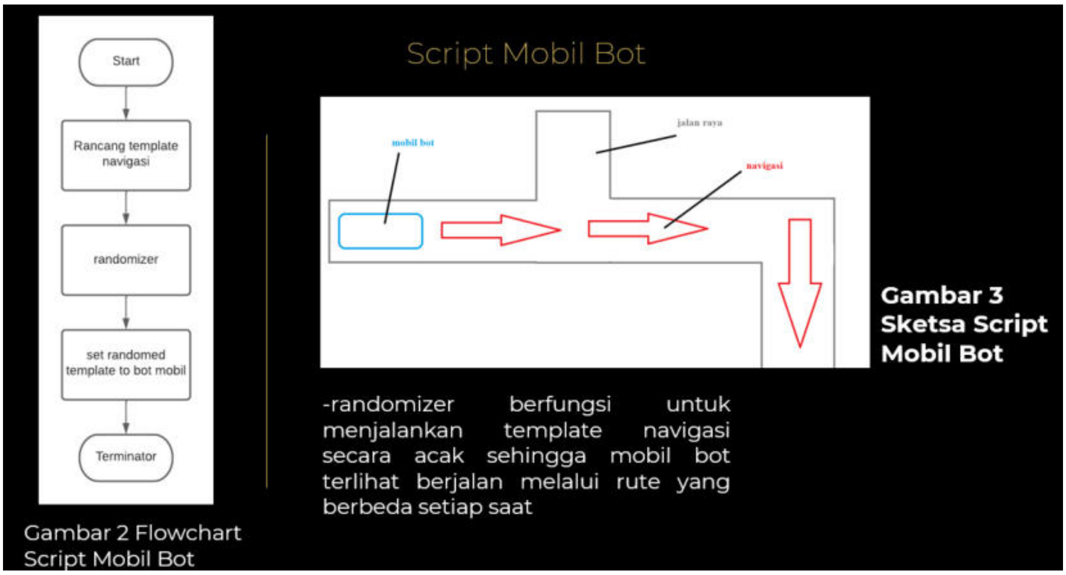


Script Mobil Bot

Gambar 2 Flowchart Script Mobil Bot

Gambar 3 Sketsa Script Mobil Bot

-Script mobil bot bekerja dengan cara membuat beberapa rute navigasi terlebih dahulu.
-Rute navigasi yang telah dibuat disebut Template Navigasi.
-Mobil bot akan berjalan mengikuti rute navigasi yang telah dibuat.
-Setelah itu dibuat randomizer

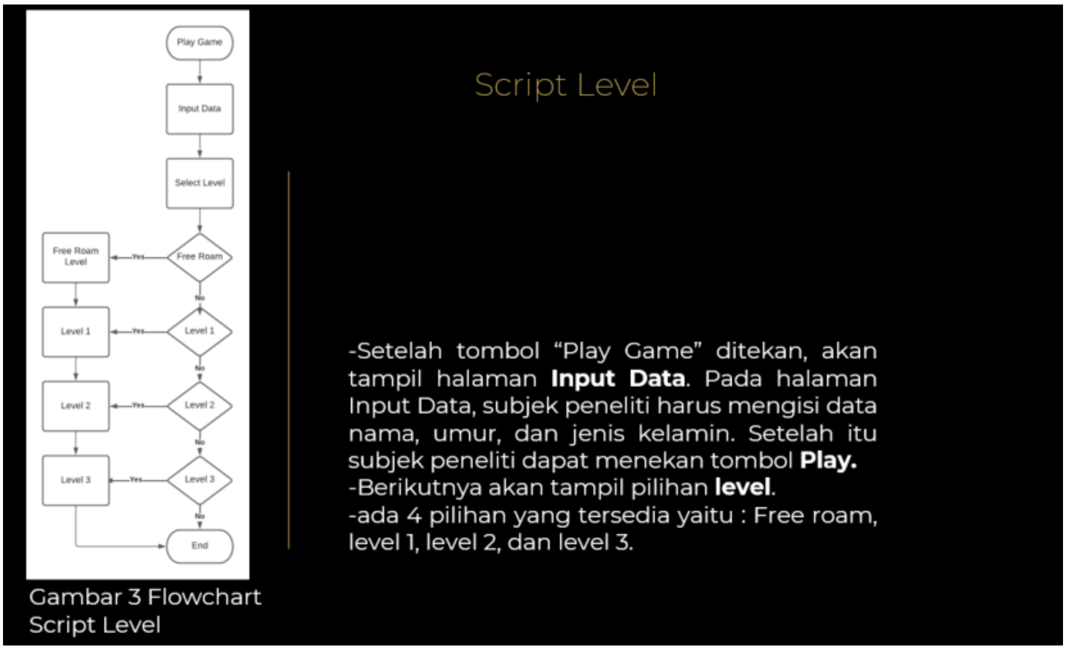


Script Mobil Bot

Gambar 2 Flowchart Script Mobil Bot

Gambar 3 Sketsa Script Mobil Bot

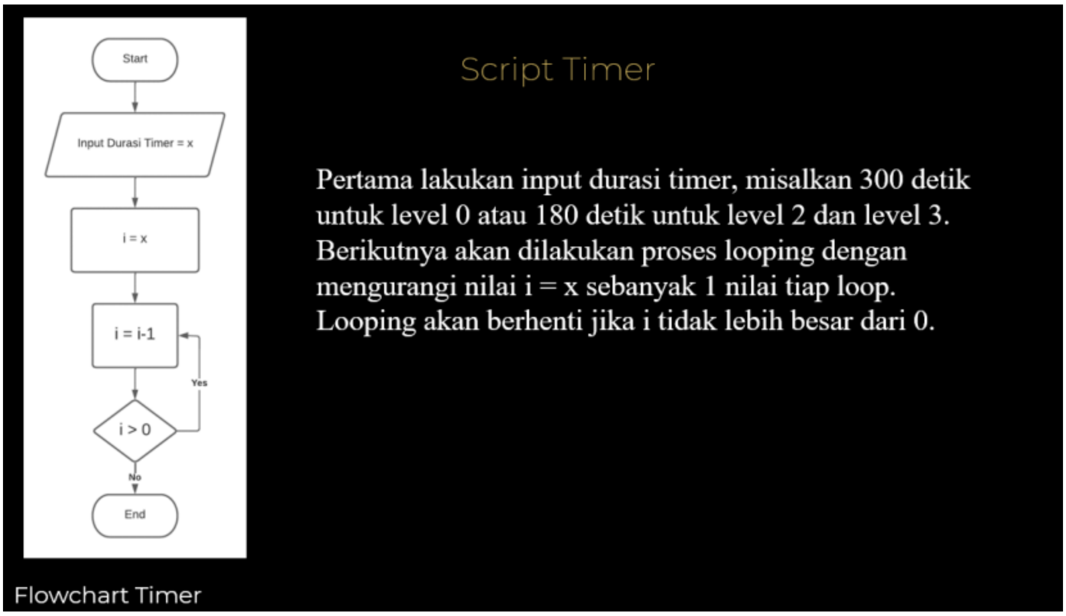
-randomizer berfungsi untuk menjalankan template navigasi secara acak sehingga mobil bot terlihat berjalan melalui rute yang berbeda setiap saat



Script Level

-Setelah tombol “Play Game” ditekan, akan tampil halaman **Input Data**. Pada halaman Input Data, subjek peneliti harus mengisi data nama, umur, dan jenis kelamin. Setelah itu subjek peneliti dapat menekan tombol **Play.**
-Berikutnya akan tampil pilihan **level**.
-ada 4 pilihan yang tersedia yaitu : Free roam, level 1, level 2, dan level 3.

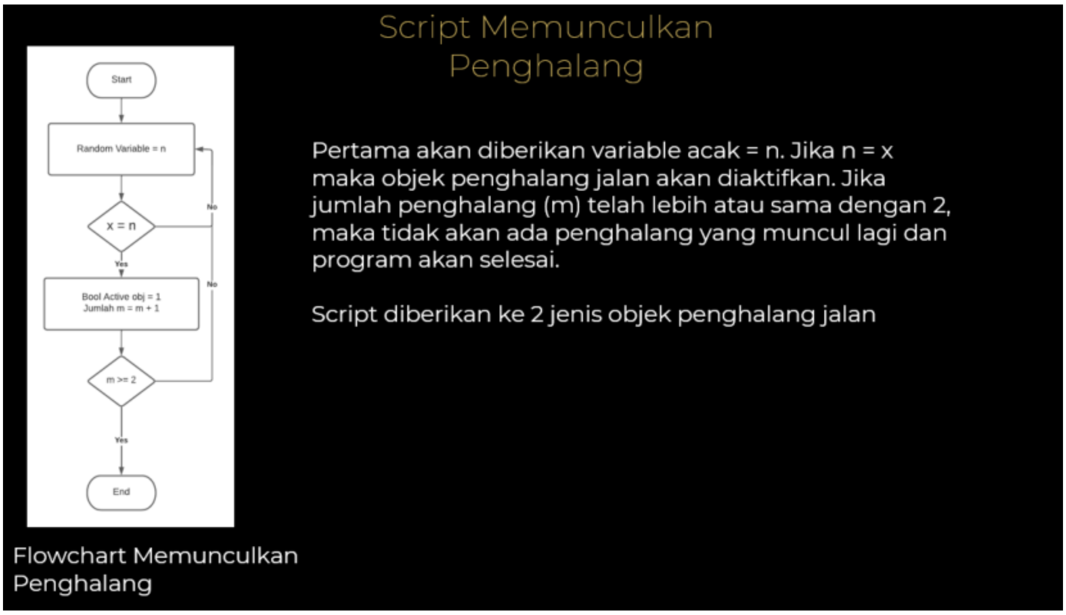
Gambar 3 Flowchart Script Level



Script Timer

Pertama lakukan input durasi timer, misalkan 300 detik untuk level 0 atau 180 detik untuk level 2 dan level 3. Berikutnya akan dilakukan proses looping dengan mengurangi nilai i = x sebanyak 1 nilai tiap loop. Looping akan berhenti jika i tidak lebih besar dari 0.

Flowchart Timer



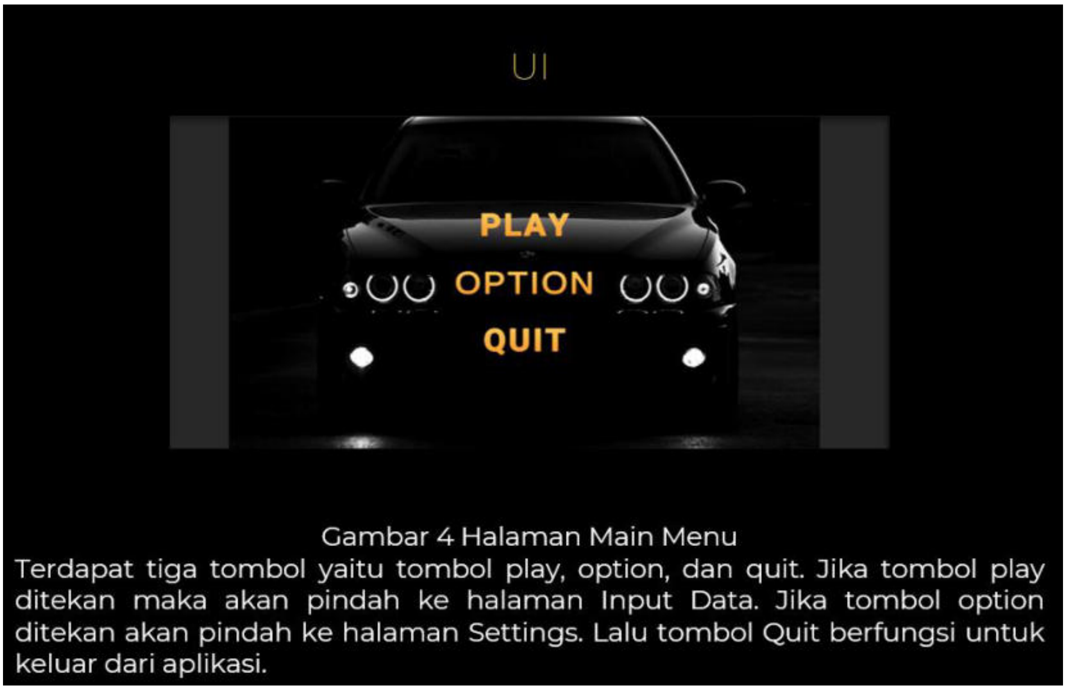
Script Memunculkan Penghalang

Pertama akan diberikan variable acak = n. Jika n = x maka objek penghalang jalan akan diaktifkan. Jika jumlah penghalang (m) telah lebih atau sama dengan 2, maka tidak akan ada penghalang yang muncul lagi dan program akan selesai.

Script diberikan ke 2 jenis objek penghalang jalan

Flowchart Memunculkan Penghalang

C. User Interface

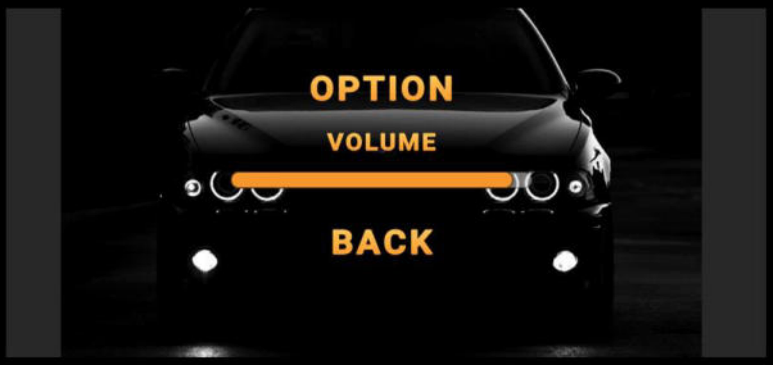


UI

Gambar 4 Halaman Main Menu

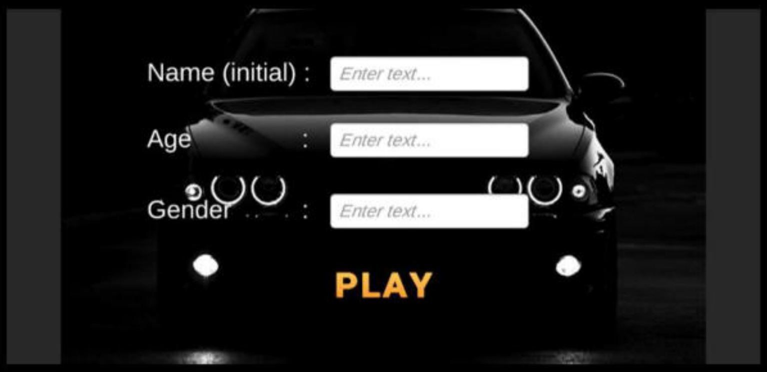
Terdapat tiga tombol yaitu tombol play, option, dan quit. Jika tombol play ditekan maka akan pindah ke halaman Input Data. Jika tombol option ditekan akan pindah ke halaman Settings. Lalu tombol Quit berfungsi untuk keluar dari aplikasi.

UI



Gambar 5 Halaman Settings
Pada halaman Settings terdapat opsi pengaturan volume. Volume dapat diatur menggunakan slider seperti pada di gambar

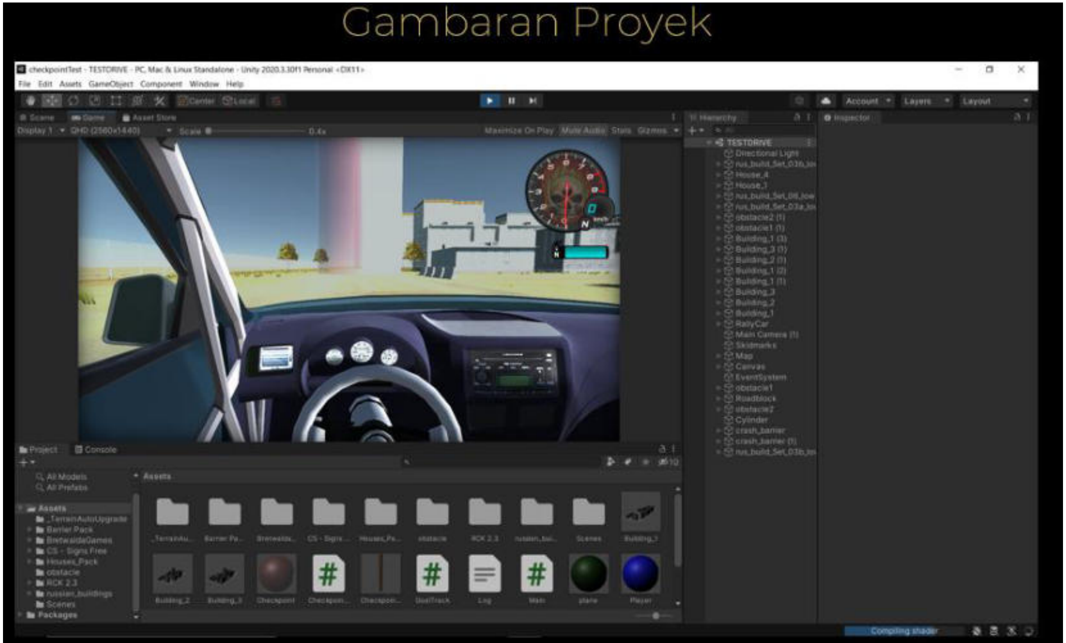
UI



Gambar 6 Input Data
Pada halaman Input Data terdapat tiga text box yang disertai label untuk mengisi data berupa Nama (inisial), Umur, dan Jenis Kelamin. Setelah itu dapat ditekan tombol Play untuk memulai simulasi.

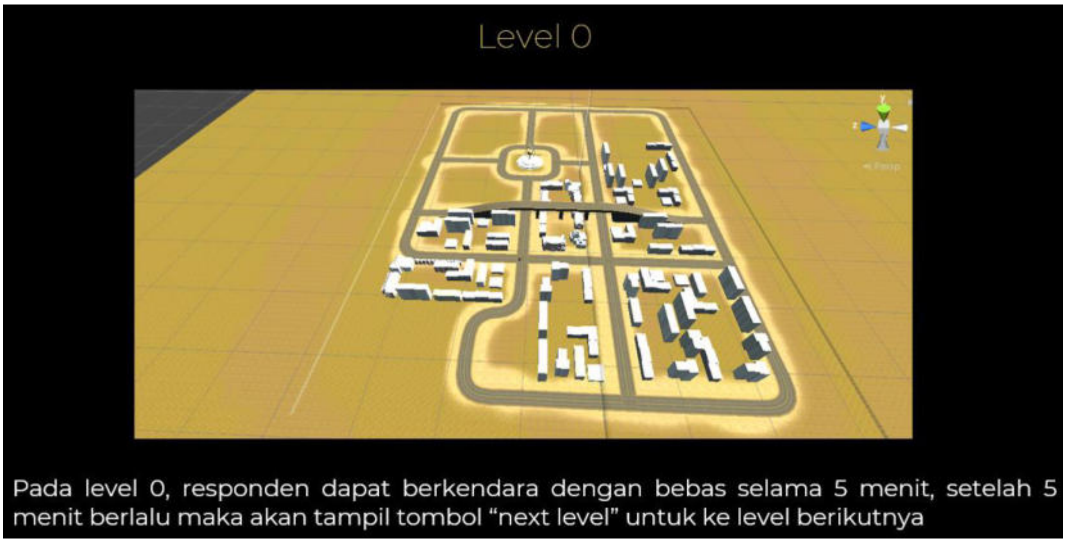
D. Gambaran Proyek

Gambaran Proyek



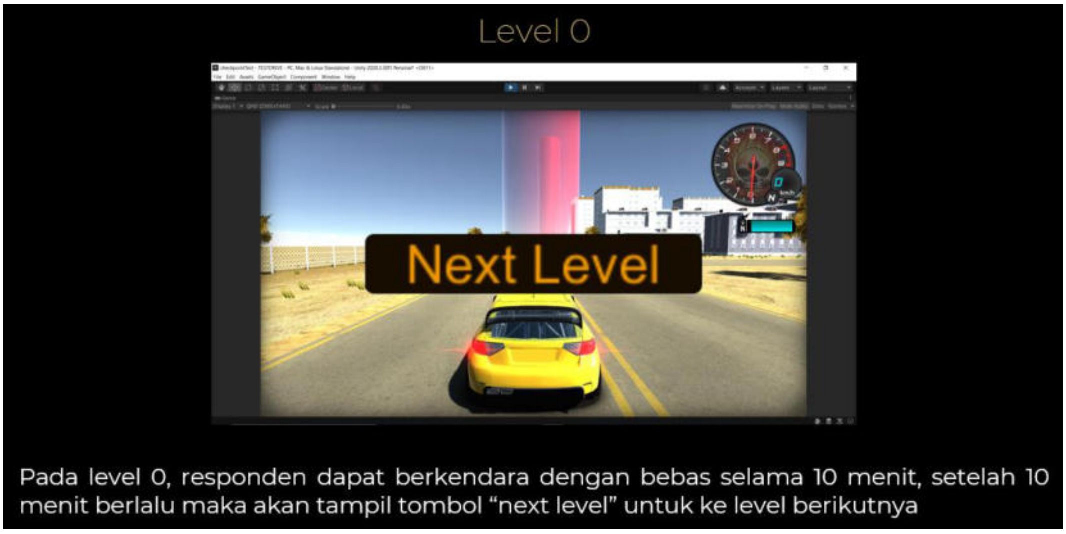
Tampilan Unity Editor
(Game mode)

E. Realisasi



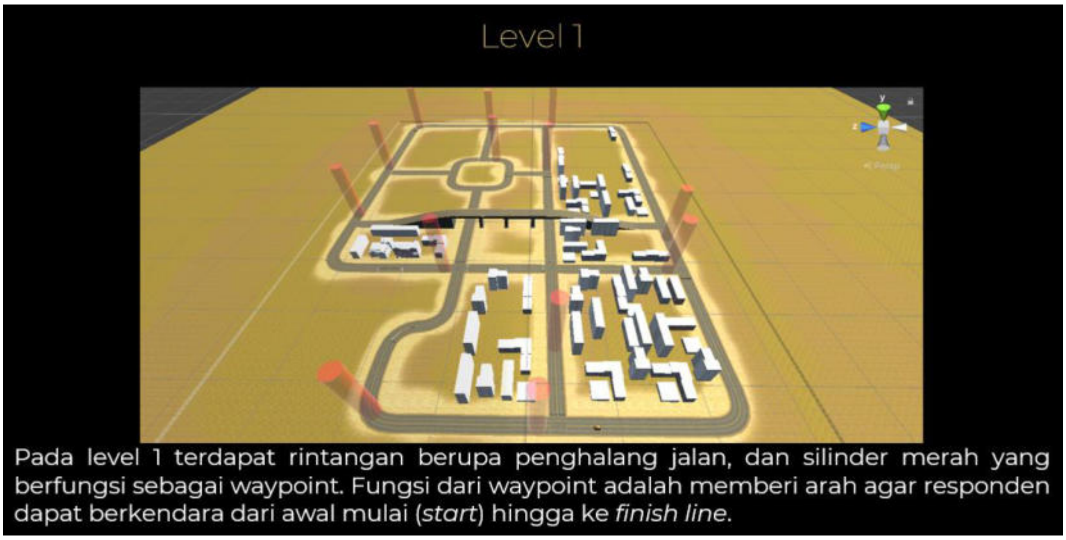
Level 0

Pada level 0, responden dapat berkendara dengan bebas selama 5 menit, setelah 5 menit berlalu maka akan tampil tombol “next level” untuk ke level berikutnya



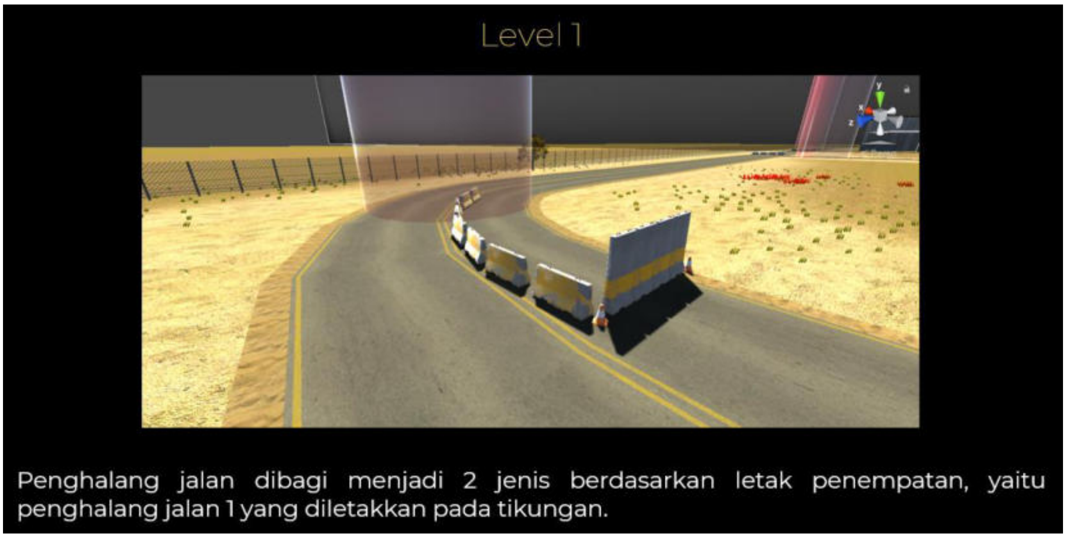
Level 0

Pada level 0, responden dapat berkendara dengan bebas selama 10 menit, setelah 10 menit berlalu maka akan tampil tombol “next level” untuk ke level berikutnya



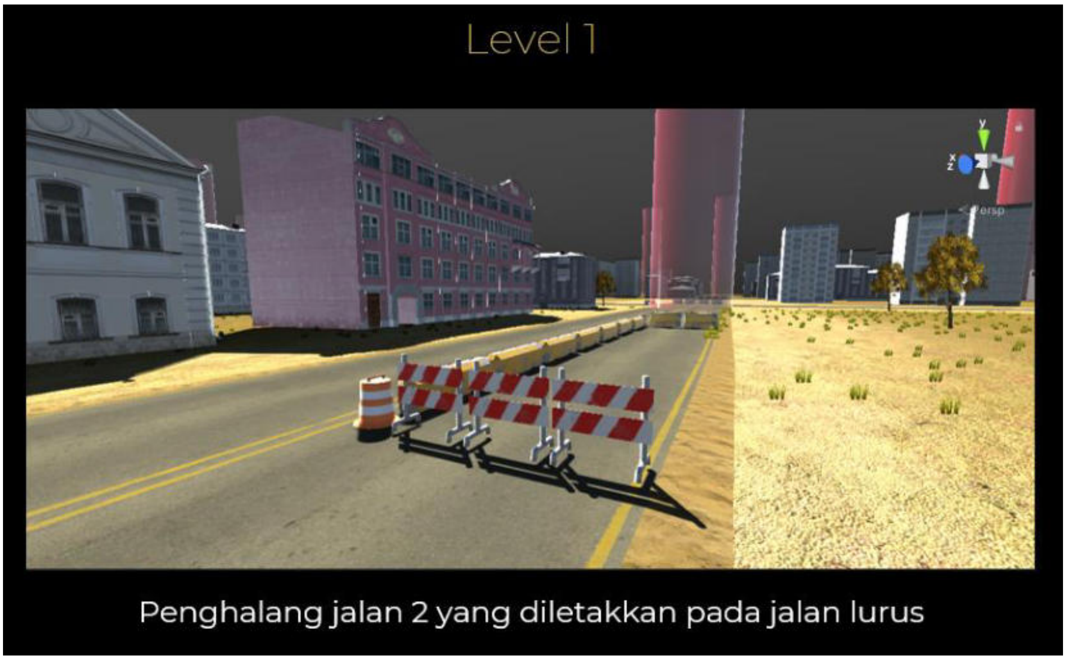
Level 1

Pada level 1 terdapat rintangan berupa penghalang jalan, dan silinder merah yang berfungsi sebagai waypoint. Fungsi dari waypoint adalah memberi arah agar responden dapat berkendara dari awal mulai (*start*) hingga ke *finish line*.



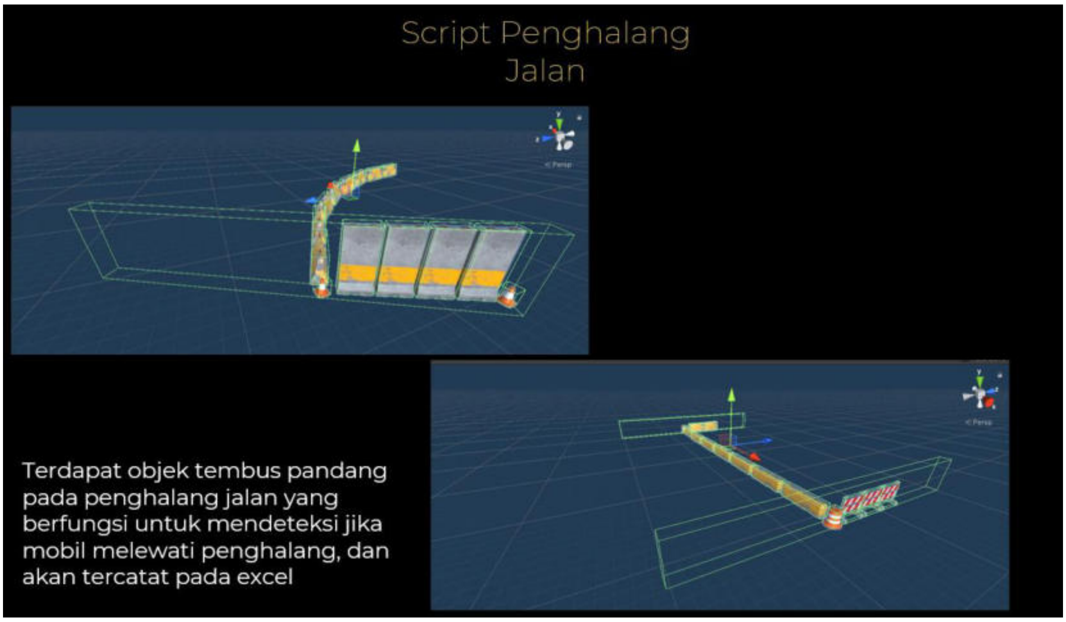
Level 1

Penghalang jalan dibagi menjadi 2 jenis berdasarkan letak penempatan, yaitu penghalang jalan 1 yang diletakkan pada tikungan.



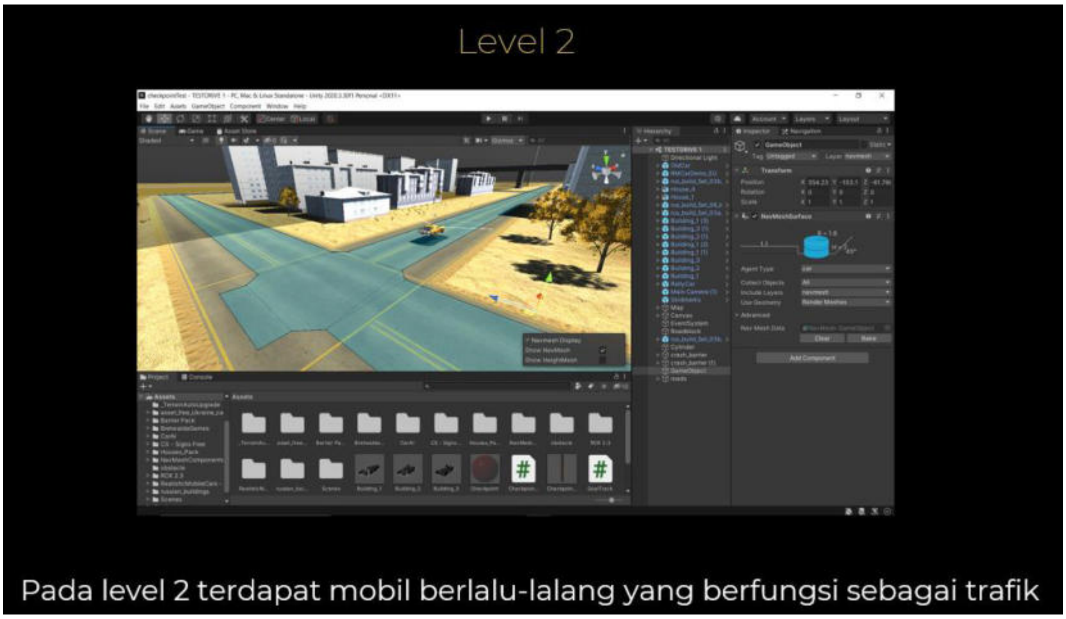
Level 1

Penghalang jalan 2 yang diletakkan pada jalan lurus



Script Penghalang Jalan

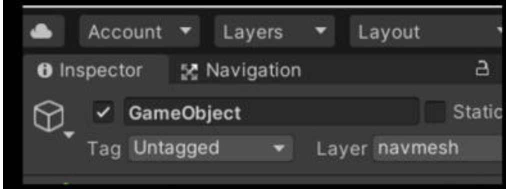
Terdapat objek tembus pandang pada penghalang jalan yang berfungsi untuk mendeteksi jika mobil melewati penghalang, dan akan tercatat pada excel



Level 2

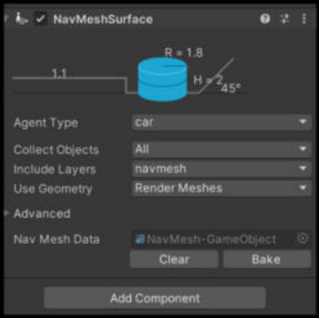
Pada level 2 terdapat mobil berlalu-lalang yang berfungsi sebagai trafik

Script Mobil Bot



☑ GameObject □ Static

Tag Untagged Layer navmesh



☑ NavMeshSurface

Agent Type car

Collect Objects All

Include Layers navmesh

Use Geometry Render Meshes

Advanced

Nav Mesh Data NavMesh-GameObject

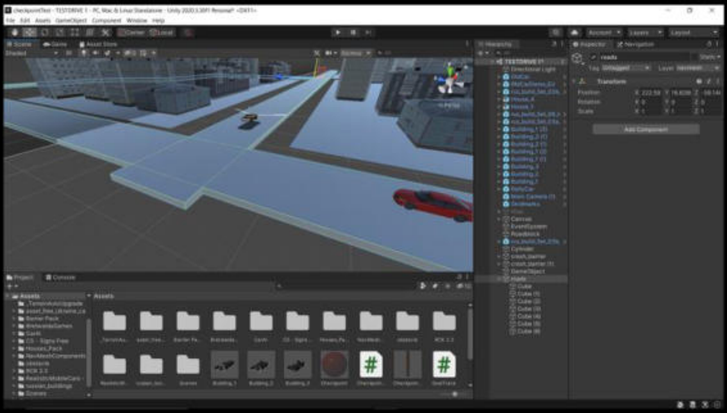
Clear Bake

Add Component

Layer untuk rute navigasi

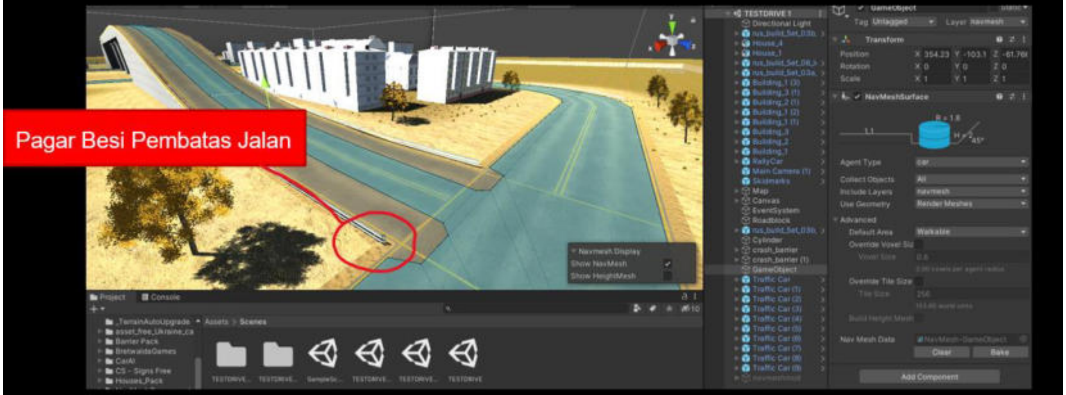
Komponen rute navigasi (NavMesh)

Script Mobil Bot



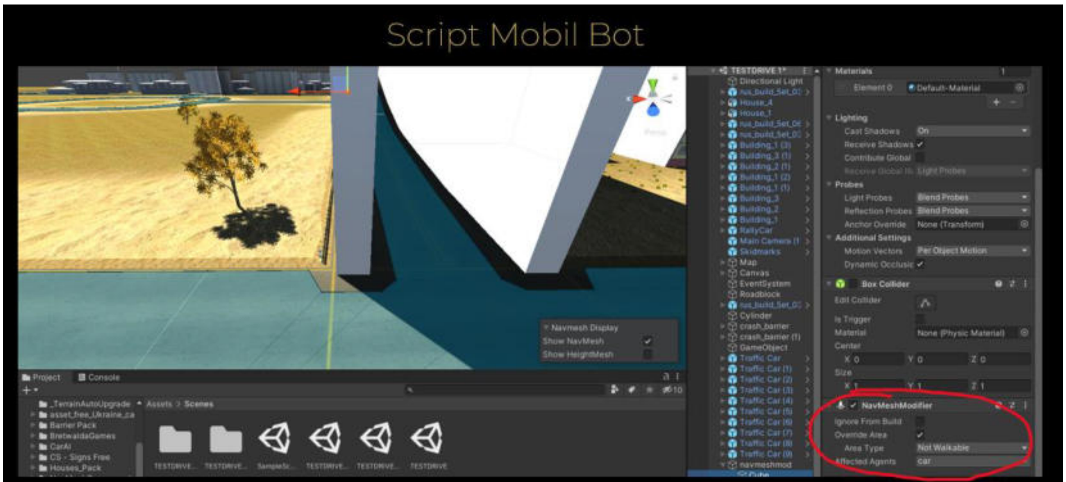
Buat objek kubus sebagai rute navigasi dan ubah layer kubus agar sama dengan komponen NavMesh

Script Mobil Bot



Daerah biru merupakan daerah mobil bot berjalan

Script Mobil Bot

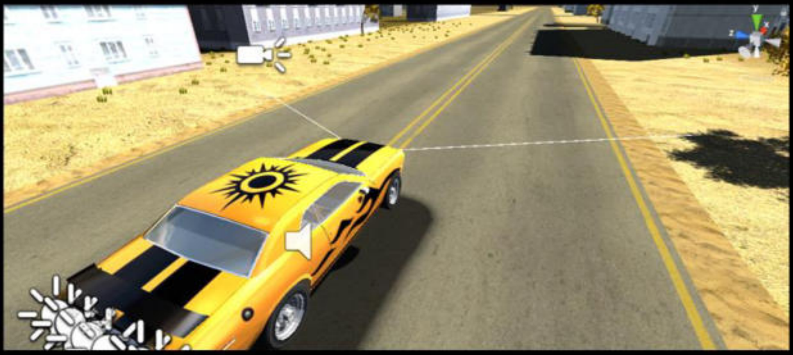


Navmesh Display
Show NavMesh ☑
Show HeightMesh □

NavMeshModifier
Ignore From Build □
Override Area ☑
Area Type Not Walkable
Affected Agents car

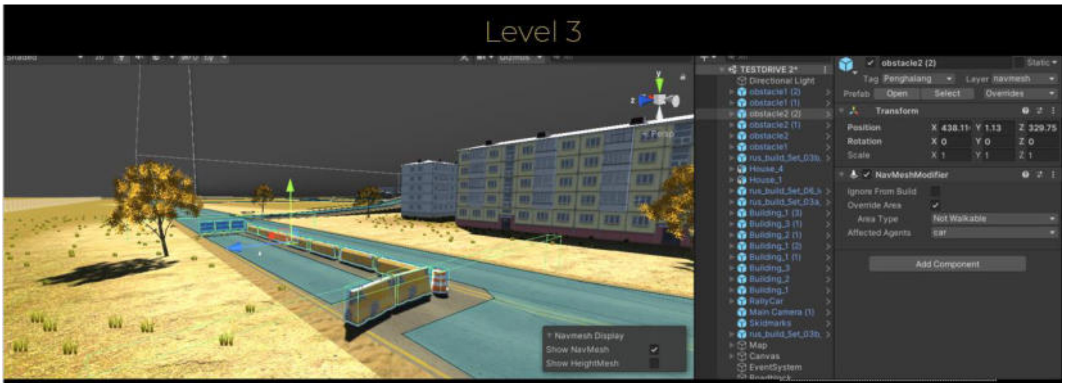
Agar mobil bot tidak tersangkut akibat menabrak pagar besi, dibuat objek invisible sebagai navigasi "not walkable"

Script Mobil Bot



Garis putih didepan mobil bot berfungsi sebagai sudut pandang mobil bot Pada daerah sudut pandang mobil bot, akan dibuat objek sebagai destinasi mobil bot, destinasi hanya dapat dibuat dalam rute NavMesh

Level 3



obstacle2 (2)
Tag Penghalang
Layer navmesh

NavMeshModifier
Ignore From Build □
Override Area ☑
Area Type Not Walkable
Affected Agents car

Navmesh Display
Show NavMesh ☑
Show HeightMesh □

Sama seperti objek kubus pada level 2, gunakan navmesh modifier untuk memotong rute mobil bot agar tidak menabrak penghalang jalan

LISTING PROGRAM

A. PROGRAM TIMER

```
public class Timer : MonoBehaviour
{
    public float timeRemaining = 900;
    public bool timerIsRunning = false;
    public Text timeText;
    private void Start()
    {
        // Starts the timer automatically
        timerIsRunning = true;
    }
    void Update()
    {
        if (timerIsRunning)
        {
            if (timeRemaining > 0)
            {
                timeRemaining -= Time.deltaTime;
                DisplayTime(timeRemaining);
            }
            else
            {
                Debug.Log("Time has run out!");
                timeRemaining = 0;
                timerIsRunning = false;
            }
        }
    }
    void DisplayTime(float timeToDisplay)
    {
        timeToDisplay += 1;
        float minutes = Mathf.FloorToInt(timeToDisplay / 60);
        float seconds = Mathf.FloorToInt(timeToDisplay % 60);
        timeText.text = string.Format("{0:00}:{1:00}", minutes, seconds);
    }
```

B. PROGRAM WAYPOINT

```
public class TrackCheckpoints : MonoBehaviour
{
    private List<CheckpointSingle> checkpointSingleList;
    private int nextCheckpointSingleIndex;

    private void Awake()
    {
        Transform checkpointsTransform = transform.Find("Checkpoints");

        checkpointSingleList = new List<CheckpointSingle>();
        foreach(Transform checkpointSingleTransform in
checkpointsTransform)
        {
            CheckpointSingle checkpointSingle =
checkpointSingleTransform.GetComponent<CheckpointSingle>();
            checkpointSingle.SetTrackCheckpoints(this);

            checkpointSingleList.Add(checkpointSingle);
        }

        nextCheckpointSingleIndex = 0;
    }

    public void PlayerThroughCheckpoint(CheckpointSingle checkpointSingle)
    {
        if (checkpointSingleList.IndexOf(checkpointSingle) ==
nextCheckpointSingleIndex)
        {
            //Correct checkpoint
            Debug.Log("Correct Checkpoint");
            nextCheckpointSingleIndex = (nextCheckpointSingleIndex + 1) %
checkpointSingleList.Count;

        } else
        {
            //Wrong checkpoint
            Debug.Log("Wrong Checkpoint");
            //show chekcpoint current
        }
    }
}
```

C. PROGRAM PENGHALANG

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Penghalang : MonoBehaviour
{
    //private Main main;
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            //main.DeteksiPenghalang(this);
            Debug.Log("Penghalang");
        }
    }

    public void SetMain(Main main)
   {
        //this.main = main;
    }
}
```

D. PROGRAM CAR AI

```
public class CarAI : MonoBehaviour
{
    [Header("Car Wheels (Wheel Collider)")]// Assign wheel Colliders through the
inspector
    public WheelCollider frontLeft;
    public WheelCollider frontRight;
    public WheelCollider backLeft;
    public WheelCollider backRight;
    [Header("Car  Wheels  (Transform)")]//  Assign  wheel  Transform(Mesh  render)
through the inspector
    public Transform wheelFL;
    public Transform wheelFR;
    public Transform wheelBL;
    public Transform wheelBR;
    [Header("Car Front (Transform)")]// Assign a Gameobject representing the front
of the car
    public Transform carFront;
    [Header("General  Parameters")]//  Look  at  the  documentation  for  a  detailed
explanation
    public List<string> NavMeshLayers;
    public int MaxSteeringAngle = 45;
    public int MaxRPM = 150;
    [Header("Debug")]
    public bool ShowGizmos;
    public bool Debugger;
    [Header("Destination Parameters")]// Look at the documentation for a detailed
explanation
    public bool Patrol = true;
    public Transform CustomDestination;
    [HideInInspector] public bool move;// Look at the documentation for a detailed
explanation
    private Vector3 PostionToFollow = Vector3.zero;
    private int currentWayPoint;
    private float AIFOV = 60;
    private bool allowMovement;
    private int NavMeshLayerBite;
    private List<Vector3> waypoints = new List<Vector3>();
    private float LocalMaxSpeed;
    private int Fails;
    private float MovementTorque = 1;

    void Awake()
    {
        currentWayPoint = 0;
        allowMovement = true;
        move = true;
    }

    void Start()
    {
        GetComponent<Rigidbody>().centerOfMass = Vector3.zero;
        CalculateNavMashLayerBite();
    }

    void FixedUpdate()
    {
        UpdateWheels();
        ApplySteering();
        PathProgress();
    }

    private void CalculateNavMashLayerBite()
    {
        if (NavMeshLayers == null || NavMeshLayers[0] == "AllAreas")
            NavMeshLayerBite = NavMesh.AllAreas;
        else if (NavMeshLayers.Count == 1)
            NavMeshLayerBite += 1 << NavMesh.GetAreaFromName(NavMeshLayers[0]);
        else
        {
            foreach (string Layer in NavMeshLayers)
            {
                int I = 1 << NavMesh.GetAreaFromName(Layer);
                NavMeshLayerBite += I;
            }
        }
    }

    private void PathProgress() //Checks if the agent has reached the
currentWayPoint or not. If yes, it will assign the next waypoint as the
currentWayPoint depending on the input
    {
        wayPointManager();
        Movement();
        ListOptimizer();
        void wayPointManager()
        {
            if (currentWayPoint >= waypoints.Count)
                allowMovement = false;
            else
            {
                PostionToFollow = waypoints[currentWayPoint];
                allowMovement = true;
                if (Vector3.Distance(carFront.position, PostionToFollow) < 2)
                    currentWayPoint++;
            }

            if (currentWayPoint >= waypoints.Count - 3)
                CreatePath();
        }

        void CreatePath()
        {
            if (CustomDestination == null)
            {
                if (Patrol == true)
                    RandomPath();
                else
                {
                    debug("No custom destination assigned and Patrol is set to
false", false);
                    allowMovement = false;
                }
            }
            else
               CustomPath(CustomDestination);

        }

        void ListOptimizer()
        {
            if (currentWayPoint > 1 && waypoints.Count > 30)
            {
                waypoints.RemoveAt(0);
                currentWayPoint--;
            }
        }
    }

    public void RandomPath() // Creates a path to a random destination
    {
        NavMeshPath path = new NavMeshPath();
        Vector3 sourcePostion;
        if (waypoints.Count == 0)
        {
            Vector3 randomDirection = Random.insideUnitSphere * 100;
            randomDirection += transform.position;
            sourcePostion = carFront.position;
            Calculate(randomDirection, sourcePostion, carFront.forward,
NavMeshLayerBite);
        }
        else
        {
            sourcePostion = waypoints[waypoints.Count - 1];
            Vector3 randomPostion = Random.insideUnitSphere * 100;
            randomPostion += sourcePostion;
            Vector3 direction = (waypoints[waypoints.Count - 1] -
waypoints[waypoints.Count - 2]).normalized;
            Calculate(randomPostion, sourcePostion, direction,
NavMeshLayerBite);
        }

        void Calculate(Vector3 destination, Vector3 sourcePostion, Vector3
direction, int NavMeshAreaByte)
        {
            if (NavMesh.SamplePosition(destination, out NavMeshHit hit, 150, 1 <<
NavMesh.GetAreaFromName(NavMeshLayers[0])) &&
                NavMesh.CalculatePath(sourcePostion, hit.position,
NavMeshAreaByte, path) && path.corners.Length > 2)
            {
                if (CheckForAngle(path.corners[1], sourcePostion, direction))
                {
                    waypoints.AddRange(path.corners.ToList());
                    debug("Random Path generated successfully", false);
                }
                else
                {
                    if (CheckForAngle(path.corners[2], sourcePostion,
direction))
                    {
                        waypoints.AddRange(path.corners.ToList());
                        debug("Random Path generated successfully", false);
                    }
                    else
                    {
                        debug("Failed to generate a random path. Waypoints are
outside the AIFOV. Generating a new one", false);
                        Fails++;
                    }
                }
            }
            else
            {
                debug("Failed to generate a random path. Invalid Path. Generating
a new one", false);
                Fails++;
            }
        }
    }

    public void CustomPath(Transform destination) //Creates a path to the Custom
destination
    {
        NavMeshPath path = new NavMeshPath();
        Vector3 sourcePostion;

        if (waypoints.Count == 0)
        {
            sourcePostion = carFront.position;
            Calculate(destination.position,   sourcePostion,   carFront.forward,
NavMeshLayerBite);
        }
        else
        {
            sourcePostion = waypoints[waypoints.Count - 1];
            Vector3   direction   =   (waypoints[waypoints.Count   -   1]   -
waypoints[waypoints.Count - 2]).normalized;
            Calculate(destination.position,      sourcePostion,      direction,
NavMeshLayerBite);
        }

        void  Calculate(Vector3  destination,  Vector3  sourcePostion,  Vector3
direction, int NavMeshAreaBite)
        {
            if  (NavMesh.SamplePosition(destination,  out  NavMeshHit  hit,  150,
NavMeshAreaBite) &&
                NavMesh.CalculatePath(sourcePostion,               hit.position,
NavMeshAreaBite, path))
            {
                if         (path.corners.ToList().Count()         >         1&&
CheckForAngle(path.corners[1], sourcePostion, direction))
                {
                    waypoints.AddRange(path.corners.ToList());
                    debug("Custom Path generated successfully", false);
                }
                else
                {
                    if (path.corners.Length > 2 && CheckForAngle(path.corners[2],
sourcePostion, direction))
                    {
                        waypoints.AddRange(path.corners.ToList());
                        debug("Custom Path generated successfully", false);
                    }
                    else
                    {
                        debug("Failed to generate a Custom path. Waypoints are
outside the AIFOV. Generating a new one", false);
                        Fails++;
                    }
                }
            }
            else
            {
                debug("Failed to generate a Custom path. Invalid Path. Generating
a new one", false);
                Fails++;
            }
        }
    }

    private bool CheckForAngle(Vector3 pos, Vector3 source, Vector3 direction)
//calculates the angle between the car and the waypoint
    {
        Vector3 distance = (pos - source).normalized;
        float CosAngle = Vector3.Dot(distance, direction);
        float Angle = Mathf.Acos(CosAngle) * Mathf.Rad2Deg;

        if (Angle < AIFOV)
            return true;
        else
            return false;
    }

    private void ApplyBrakes() // Apply brake torque
    {
        frontLeft.brakeTorque = 5000;
        frontRight.brakeTorque = 5000;
        backLeft.brakeTorque = 5000;
        backRight.brakeTorque = 5000;
    }


    private void UpdateWheels() // Updates the wheel's postion and rotation
    {
        ApplyRotationAndPostion(frontLeft, wheelFL);
        ApplyRotationAndPostion(frontRight, wheelFR);
        ApplyRotationAndPostion(backLeft, wheelBL);
        ApplyRotationAndPostion(backRight, wheelBR);
    }
    private void ApplyRotationAndPostion(WheelCollider targetWheel, Transform
wheel) // Updates the wheel's postion and rotation
    {
        targetWheel.ConfigureVehicleSubsteps(5, 12, 15);

        Vector3 pos;
        Quaternion rot;
        targetWheel.GetWorldPose(out pos, out rot);
        wheel.position = pos;
        wheel.rotation = rot;
    }

    void ApplySteering() // Applies steering to the Current waypoint
    {
        Vector3 relativeVector =
transform.InverseTransformPoint(PostionToFollow);
        float SteeringAngle = (relativeVector.x / relativeVector.magnitude) *
MaxSteeringAngle;
        if (SteeringAngle > 15) LocalMaxSpeed = 100;
        else LocalMaxSpeed = MaxRPM;

        frontLeft.steerAngle = SteeringAngle;
        frontRight.steerAngle = SteeringAngle;
    }

    void Movement() // moves the car forward and backward depending on the input
    {
        if (move == true && allowMovement == true)
            allowMovement = true;
        else
            allowMovement = false;

        if (allowMovement == true)
        {
            frontLeft.brakeTorque = 0;
            frontRight.brakeTorque = 0;
            backLeft.brakeTorque = 0;
            backRight.brakeTorque = 0;

            int SpeedOfWheels = (int)((frontLeft.rpm + frontRight.rpm +
backLeft.rpm + backRight.rpm) / 4);

            if (SpeedOfWheels < LocalMaxSpeed)
            {
                backRight.motorTorque = 400 * MovementTorque;
                backLeft.motorTorque = 400 * MovementTorque;
                frontRight.motorTorque = 400 * MovementTorque;
                frontLeft.motorTorque = 400 * MovementTorque;
            }
            else if (SpeedOfWheels < LocalMaxSpeed + (LocalMaxSpeed * 1 / 4))
            {
                backRight.motorTorque = 0;
                backLeft.motorTorque = 0;
                frontRight.motorTorque = 0;
                frontLeft.motorTorque = 0;
            }
            else
                ApplyBrakes();

        }
        else
            ApplyBrakes();
    }

    void debug(string text, bool IsCritical)
    {
        if (Debugger)
        {
            if (IsCritical)
                Debug.LogError(text);
            else
                Debug.Log(text);
        }
    }

    private void OnDrawGizmos() // shows a Gizmos representing the waypoints and
AI FOV
    {
        if (ShowGizmos == true)
        {
            for (int i = 0; i < waypoints.Count; i++)
            {
                if (i == currentWayPoint)
                    Gizmos.color = Color.blue;
                else
                {
                    if (i > currentWayPoint)
                        Gizmos.color = Color.red;
                    else
                        Gizmos.color = Color.green;
                }
                Gizmos.DrawWireSphere(waypoints[i], 2f);
            }
            CalculateFOV();
        }
        void CalculateFOV()
        {
            Gizmos.color = Color.white;
            float totalFOV = AIFOV * 2;
            float rayRange = 10.0f;
            float halfFOV = totalFOV / 2.0f;
            Quaternion    leftRayRotation    =    Quaternion.AngleAxis(-halfFOV,
Vector3.up);
            Quaternion    rightRayRotation    =    Quaternion.AngleAxis(halfFOV,
Vector3.up);
            Vector3 leftRayDirection = leftRayRotation * transform.forward;
            Vector3 rightRayDirection = rightRayRotation * transform.forward;
            Gizmos.DrawRay(carFront.position, leftRayDirection * rayRange);
            Gizmos.DrawRay(carFront.position, rightRayDirection * rayRange);
        }
    }
}
```

E. PROGRAM AMBIL DATA

```
public class Main : MonoBehaviour {

    public Rigidbody MyCar;
    public float speed = 0.0f;
    public float accel = 0.0f;
    public float lspeed = 0.0f;
   // public bool doTextUpdate;
    public bool stop = true;
    public float delay;
    private float timer;
    IEnumerator changeColorCoroutine;
    private string detector;
    private Penghalang penghalang;
    private int fileCounter = 0;

    // Use this for initialization
    void Start () {
        //CreateText();
        timer = delay;
       }

    void Update()
    {
        //doTextUpdate = !doTextUpdate;
        string path = Application.dataPath + "/Log.csv";
        //Create File if it doesn't exist
        if (!File.Exists(path))
        {
            fileCounter++;
            File.WriteAllText(path, "Car Speed : \n\n");
        }
        //Content of the file


        accel = (speed - lspeed);
        lspeed = speed;
        speed = MyCar.velocity.magnitude * 3.6f;
        timer -= Time.deltaTime;
        if (timer <= 0)
        {
            timer = delay;

            string content = speed + " KM/H " + System.DateTime.Now + " " +
"\n";
            File.AppendAllText(path, content);
```