Racing Games – Scripts

# StateDrivenBrain

using UnityEngine;  
using System.Collections;  
  
public class StateDrivenBrain : BasicAIController  
{  
    *// Declares the constant state names*  
    public enum CarStates { Idle, Launch, Drive, Finish };  
    *// Sets up state machine variable*  
    public FSM<CarStates> carStateMachine;  
    *// Puts a 0.4s interval on the state machine*  
    protected float thinkInterval = 0.4f;  
    *// Initialises carStateActive to true*  
    public bool carStateActive = true;  
    *//Creates a variable that can be used to refer to other scripts*  
    public Engine engine;  
    public Node node;  
    public Node2 node2;  
    *// Initialises End to true*  
    public bool End = false;  
    public float countdown = 5.0f;  
    public GameObject RedCD1;  
    public GameObject RedCD2;  
    public GameObject RedCD3;  
    public GameObject GreenCD1;  
    public GameObject GreenCD2;  
    public GameObject GreenCD3;  
    public GameObject Text1;  
    public GameObject Text2;  
  
    protected void Awake()  
    {  
        *//Finds these scripts to allow use of their properties*  
        engine = GetComponent<Engine> ();  
        node = GetComponent<Node> ();  
        node2 = GetComponent<Node2> ();  
  
        *// Sets up a new FSM*  
        carStateMachine = new FSM<CarStates>();  
        *// This adds in all the new states into the state machine using variables declared*  
        carStateMachine.AddState(new Idle<CarStates>(CarStates.Idle, this, 3.0f));  
        carStateMachine.AddState(new Launch<CarStates>(CarStates.Launch, this, 0f));  
        carStateMachine.AddState(new Drive<CarStates>(CarStates.Drive, this, 0f));  
        carStateMachine.AddState(new Finish<CarStates>(CarStates.Finish, this, 0f));  
        *// Sets Idle as first state to be in once plays been activated*  
        carStateMachine.SetInitialState (CarStates.Idle);  
  
        *// These are all the transitions of the states I will require within this program*  
        carStateMachine.AddTransition (CarStates.Idle, CarStates.Launch);  
        carStateMachine.AddTransition (CarStates.Launch, CarStates.Drive);  
        carStateMachine.AddTransition (CarStates.Drive, CarStates.Finish);  
        carStateMachine.AddTransition (CarStates.Finish, CarStates.Idle);  
    }  
  
    *// These Guards determine the conditions states need to meet to be initialised*  
    public bool GuardIdleToLaunch(State<CarStates> currentState)  
    {  
        return (End == false);  
    }  
    public bool GuardLaunchToDrive(State<CarStates> currentState)  
    {  
        return (countdown <= 0.0f);  
    }  
    public bool GuardDriveToFinish(State<CarStates> currentState)  
    {  
        return (node.currentNode == (node.routeNodes.Length - 1) || node2.currentNode == (node2.routeNodes.Length - 1));  
    }  
    public bool GuardFinishToIdle(State<CarStates> currentState)  
    {  
        return (End == true);  
    }  
  
    public void Start()  
    {  
        *// Calls to Think function*  
        StartCoroutine(Think());  
    }  
  
    public void Update()  
    {  
        base.Update ();  
        *// Refers to all act callbacks within other scripts*  
        if (carStateActive)   
        {  
            carStateMachine.CurrentState.Act ();  
        }  
    }  
  
    protected IEnumerator Think()  
    {  
        yield return new WaitForSeconds(thinkInterval);  
        *// Checks state machine*  
        carStateMachine.Check();  
        *// Calls to Think function*  
        StartCoroutine(Think());  
    }  
  
    public void Race()  
    {  
        *//Activates engine script*  
        engine.enabled = true;  
    }  
}

# Idle

using UnityEngine;  
using System.Collections;  
  
*// The idle state is for when the Car has no functions to carry out*  
  
*// The Idle class inherits from the AIState script*  
public class Idle<T> : AIState<T>   
{  
    *// Sets up FSM*  
    public Idle(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
        *//Picks a random integer value out of 0 and 1 (2 isn't counted as a value)*  
        int i = Random.Range (0, 2);  
  
        *//Activates node script if random value is 0*  
        if (i == 0)   
        {  
            Debug.Log ("AI has chosen Route 1");  
            brain.node.enabled = true;  
            brain.Text1.SetActive (true);  
        }  
  
        *//Activates node2 script if random value is 1 or higher*  
        if (i >= 1)   
        {  
            Debug.Log ("AI has chosen Route 2");  
            brain.node2.enabled = true;  
            brain.Text2.SetActive (true);  
        }  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
        brain.Text1.SetActive (false);  
        brain.Text2.SetActive (false);  
    }  
  
    public override void Act()  
    {  
        *// No instructions are needed as the car is idle*  
    }  
}

# Launch

using UnityEngine;  
using System.Collections;  
  
  
public class Launch<T> : AIState<T>   
{  
    *// Sets up FSM*  
    public Launch(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
        brain.RedCD1.SetActive (true);  
        brain.RedCD2.SetActive (true);  
        brain.RedCD3.SetActive (true);  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
        brain.GreenCD1.SetActive (false);  
        brain.GreenCD2.SetActive (false);  
        brain.GreenCD3.SetActive (false);  
    }  
  
    public override void Act()  
    {  
        brain.countdown -= Time.deltaTime;  
  
        if (brain.countdown <= 4.0f)   
        {  
            brain.RedCD1.SetActive (false);  
        }  
        if (brain.countdown <= 3.0f)   
        {  
            brain.RedCD2.SetActive (false);  
        }  
        if (brain.countdown <= 2.0f)   
        {  
            brain.RedCD3.SetActive (false);  
        }  
        if (brain.countdown <= 1.0f)   
        {  
            brain.GreenCD1.SetActive (true);  
            brain.GreenCD2.SetActive (true);  
            brain.GreenCD3.SetActive (true);  
        }  
    }  
}

# Drive

using UnityEngine;  
using System.Collections;  
  
  
public class Drive<T> : AIState<T>   
{  
  
    *// Sets up FSM*  
    public Drive(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
        *//Calls to function from within StateDrivenBrain*  
        brain.Race ();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
  
    }  
}

# Finish

using UnityEngine;  
using System.Collections;  
  
  
public class Finish<T> : AIState<T>   
{  
  
    *// Sets up FSM*  
    public Finish(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
        *//Converts 'End' boolean to true from within the StateDrivenBrain*  
        brain.End = true;  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
  
    }  
}

# Wheel

using UnityEngine;  
using System.Collections;  
using UnityEngine.UI;  
  
public class Wheel : MonoBehaviour   
{  
    public Transform wheelModel;  
    private WheelCollider wheelCollider;  
  
    void Awake()   
    {  
        *//Locates the wheel collider script*  
        wheelCollider = GetComponent<Collider>() as WheelCollider;  
    }  
  
    *//Alligns the wheels position from the wheel collider*  
    private void FixedUpdate()   
    {  
        Vector3 position;  
        Quaternion rotation;  
        *//Get the collider's position and rotation*  
        wheelCollider.GetWorldPose(out position, out rotation);  
        wheelModel.position = new Vector3(position.x, position.y, position.z);  
        wheelModel.transform.rotation = rotation;  
        WheelHit hit;  
        *//Applies friction to wheels*  
        if (wheelCollider.GetGroundHit(out hit)) {  
            PhysicMaterial material = hit.collider.material;  
            WheelFrictionCurve forwardCurve = wheelCollider.forwardFriction;  
            forwardCurve.stiffness = material.dynamicFriction;  
            wheelCollider.forwardFriction = forwardCurve;  
        }  
    }  
}

# PID

using UnityEngine;  
using System.Collections;  
  
public class PID   
{  
    *//This script works out the error values needed to work with physics on the car*  
  
    public float pFactor, iFactor, dFactor;  
    private float kp,ki,kd;  
    private float lastError;  
  
    public PID(float pFactor, float iFactor, float dFactor)   
    {  
        this.pFactor = pFactor;  
        this.iFactor = iFactor;  
        this.dFactor = dFactor;  
        lastError = 0f;  
        ki = 0f;  
    }  
  
    public float Update(float R, float Y, float timeFrame)   
    {  
        float e = R - Y;  
        kp = e \* pFactor;  
        ki = ki + e \* Time.deltaTime \* iFactor;  
        kd = ((e - lastError) / Time.deltaTime) \* dFactor;  
        lastError = e;  
        return kp + ki + kd;  
    }  
  
}

# Engine

using UnityEngine;  
using System.Collections;  
  
public class Engine : MonoBehaviour   
{  
    *//Sets wheel collider array and gameobjects*  
    public WheelCollider[] wheels;  
    public WheelCollider fl;  
    public WheelCollider fr;  
    public WheelCollider rl;  
    public WheelCollider rr;  
    *//Sets value for torque*  
    private float torque = -400f;   
    *//Shows in inspector whats the next target node for the car*  
    public Transform targetNodes;  
    [HideInInspector]  
    *//Car physics values, can be changed to adjust car performance*  
    public float turnAngle;  
    private float currentAngle = 0;  
    private float turningSpeed = 25.0f;   
    private float currentMaxTurnAngle = 35.0f;   
    private bool stop = false;  
    public float brakeForce = 20000f;  
    public PID pid;  
    public float TargetVelocity = 15.0f;   
    private Rigidbody rb;  
  
    void Awake()   
    {  
  
        turnAngle = 30f;  
        *//Place the wheel colliders in an array for easier processing*  
        wheels = new WheelCollider[4];  
        wheels[0] = fr;  
        wheels[1] = fl;  
        wheels[2] = rr;  
        wheels[3] = rl;  
        *//These values can be changed to tune the car*  
        pid = new PID (50.0f, 15.0f, 15.0f);   
        rb = GetComponent<Rigidbody>();  
    }  
  
    void Update()   
    {  
        float Velocity = 0.0f;  
        *//This calculation determines the car speed*  
        Velocity = (rb.velocity.magnitude) \* 1.4f;   
        *//Uses PID to set up torque on the vehicle*  
        torque = -(pid.Update(TargetVelocity, Velocity, Time.deltaTime));  
  
        *//This if statement will move the car and steer is in refernece to a target (node)*  
        if (!stop)   
        {  
            Vector3 moveDirection = (targetNodes.position - transform.position).normalized;  
            *//Underneath a ray is drawn demonstrating a vision model (senses)*  
            Debug.DrawRay(transform.position, moveDirection \* 150, Color.cyan);  
            Vector3 localTarget = transform.InverseTransformPoint(targetNodes.position);  
            localTarget = localTarget \* -1;  
            float targetAngle = Mathf.Atan2(localTarget.x, localTarget.z) \* Mathf.Rad2Deg;  
            if (currentAngle < targetAngle)   
            {  
                currentAngle = currentAngle + (Time.deltaTime \* turningSpeed);  
                if (currentAngle > targetAngle)   
                {  
                    currentAngle = targetAngle;  
                }  
            }  
            else if (currentAngle > targetAngle)   
            {  
                currentAngle = currentAngle - (Time.deltaTime \* turningSpeed);  
                if (currentAngle < targetAngle)   
                {  
                    currentAngle = targetAngle;  
                }  
            }  
            turnAngle = Mathf.Clamp(currentAngle, (-1) \* currentMaxTurnAngle, currentMaxTurnAngle);  
        }  
    }  
  
    void FixedUpdate()   
    {  
        *//This if statement stops the car using torque and break force*  
        if (stop) {  
            for (int n = 0; n < 4; n++)   
            {  
                *// Apply brakes*  
                wheels[n].brakeTorque = brakeForce;  
                wheels[n].motorTorque = 0;  
            }  
        }  
        else   
        {  
            for (int n = 0; n < 4; n++)   
            {  
                *// Apply torque*  
                wheels[n].motorTorque = torque;  
            }  
            fl.steerAngle = turnAngle;  
            fr.steerAngle = turnAngle;  
        }  
    }  
  
    public void Stop()   
    {  
        stop = true;  
    }  
}

# Node

using UnityEngine;  
using System.Collections;  
using System.Collections.Generic;  
  
public class Node : MonoBehaviour   
{  
      
    public enum DriveStates { Drive, Stop, OutOfControl };  
    public DriveStates driveState;  
    public Transform[] routeNodes;  
    public int currentNode = 0;  
    private Engine engine;  
    private List<Collider> visitedNodes;  
  
    protected void Awake()   
    {  
        *//Inherits the Engine script to allow driving force towards nodes*  
        engine = GetComponent<Engine>();  
        driveState = DriveStates.Drive;  
        engine.targetNodes = routeNodes[currentNode];  
        visitedNodes = new List<Collider>();  
    }  
  
    protected void Update()   
    {  
        if (driveState == DriveStates.Stop)   
        {  
            *//Stops the driving force*  
            engine.Stop();  
        }  
    }  
  
    *//This function determines when a node has been met and what its next target (node) is*  
    void OnTriggerEnter(Collider collider)   
    {  
        if (collider.tag == "node")   
        {  
            if (!visitedNodes.Exists(delegate(Collider col) { return col == collider; }))   
            {  
                visitedNodes.Add(collider);  
  
                if (currentNode == (routeNodes.Length - 1))   
                {  
                    driveState = DriveStates.Stop;  
                }  
                else   
                {  
                    engine.targetNodes = routeNodes[++currentNode];  
                }  
            }  
        }  
    }  
}

# Node2

using UnityEngine;  
using System.Collections;  
using System.Collections.Generic;  
  
public class Node2 : MonoBehaviour   
{  
  
    public enum DriveStates { Drive, Stop, OutOfControl };  
    public DriveStates driveState;  
    public Transform[] routeNodes;  
    public int currentNode = 0;  
    private Engine engine;  
    private List<Collider> visitedNodes;  
  
    protected void Awake()   
    {  
        *//Inherits the Engine script to allow driving force towards nodes*  
        engine = GetComponent<Engine>();  
        driveState = DriveStates.Drive;  
        engine.targetNodes = routeNodes[currentNode];  
        visitedNodes = new List<Collider>();  
    }  
  
    protected void Update()   
    {  
        if (driveState == DriveStates.Stop)   
        {  
            *//Stops the driving force*  
            engine.Stop();  
        }  
    }  
  
    *//This function determines when a node has been met and what its next target (node) is*  
    void OnTriggerEnter(Collider collider)   
    {  
        if (collider.tag == "node")   
        {  
            if (!visitedNodes.Exists(delegate(Collider col) { return col == collider; }))   
            {  
                visitedNodes.Add(collider);  
  
                if (currentNode == (routeNodes.Length - 1))   
                {  
                    driveState = DriveStates.Stop;  
                }  
                else   
                {  
                    engine.targetNodes = routeNodes[++currentNode];  
                }  
            }  
        }  
    }  
}