using System;

using UnityEngine;

namespace UnityStandardAssets.Utility

{

public class WaypointProgressTracker : MonoBehaviour

{

// This script can be used with any object that is supposed to follow a

// route marked out by waypoints.

// This script manages the amount to look ahead along the route,

// and keeps track of progress and laps.

[SerializeField] private WaypointCircuit circuit; // A reference to the waypoint-based route we should follow

[SerializeField] private float lookAheadForTargetOffset = 5;

// The offset ahead along the route that the we will aim for

[SerializeField] private float lookAheadForTargetFactor = .1f;

// A multiplier adding distance ahead along the route to aim for, based on current speed

[SerializeField] private float lookAheadForSpeedOffset = 10;

// The offset ahead only the route for speed adjustments (applied as the rotation of the waypoint target transform)

[SerializeField] private float lookAheadForSpeedFactor = .2f;

// A multiplier adding distance ahead along the route for speed adjustments

[SerializeField] private ProgressStyle progressStyle = ProgressStyle.SmoothAlongRoute;

// whether to update the position smoothly along the route (good for curved paths) or just when we reach each waypoint.

[SerializeField] private float pointToPointThreshold = 4;

// proximity to waypoint which must be reached to switch target to next waypoint : only used in PointToPoint mode.

public enum ProgressStyle

{

SmoothAlongRoute,

PointToPoint,

}

// these are public, readable by other objects - i.e. for an AI to know where to head!

public WaypointCircuit.RoutePoint targetPoint { get; private set; }

public WaypointCircuit.RoutePoint speedPoint { get; private set; }

public WaypointCircuit.RoutePoint progressPoint { get; private set; }

public Transform target;

private float progressDistance; // The progress round the route, used in smooth mode.

private int progressNum; // the current waypoint number, used in point-to-point mode.

private Vector3 lastPosition; // Used to calculate current speed (since we may not have a rigidbody component)

private float speed; // current speed of this object (calculated from delta since last frame)

// setup script properties

private void Start()

{

// we use a transform to represent the point to aim for, and the point which

// is considered for upcoming changes-of-speed. This allows this component

// to communicate this information to the AI without requiring further dependencies.

// You can manually create a transform and assign it to this component \*and\* the AI,

// then this component will update it, and the AI can read it.

if (target == null)

{

target = new GameObject(name + " Waypoint Target").transform;

}

Reset();

}

// reset the object to sensible values

public void Reset()

{

progressDistance = 0;

progressNum = 0;

if (progressStyle == ProgressStyle.PointToPoint)

{

target.position = circuit.Waypoints[progressNum].position;

target.rotation = circuit.Waypoints[progressNum].rotation;

}

}

private void Update()

{

if (progressStyle == ProgressStyle.SmoothAlongRoute)

{

// determine the position we should currently be aiming for

// (this is different to the current progress position, it is a a certain amount ahead along the route)

// we use lerp as a simple way of smoothing out the speed over time.

if (Time.deltaTime > 0)

{

speed = Mathf.Lerp(speed, (lastPosition - transform.position).magnitude/Time.deltaTime,

Time.deltaTime);

}

target.position =

circuit.GetRoutePoint(progressDistance + lookAheadForTargetOffset + lookAheadForTargetFactor\*speed)

.position;

target.rotation =

Quaternion.LookRotation(

circuit.GetRoutePoint(progressDistance + lookAheadForSpeedOffset + lookAheadForSpeedFactor\*speed)

.direction);

// get our current progress along the route

progressPoint = circuit.GetRoutePoint(progressDistance);

Vector3 progressDelta = progressPoint.position - transform.position;

if (Vector3.Dot(progressDelta, progressPoint.direction) < 0)

{

progressDistance += progressDelta.magnitude\*0.5f;

}

lastPosition = transform.position;

}

else

{

// point to point mode. Just increase the waypoint if we're close enough:

Vector3 targetDelta = target.position - transform.position;

if (targetDelta.magnitude < pointToPointThreshold)

{

progressNum = (progressNum + 1)%circuit.Waypoints.Length;

}

target.position = circuit.Waypoints[progressNum].position;

target.rotation = circuit.Waypoints[progressNum].rotation;

// get our current progress along the route

progressPoint = circuit.GetRoutePoint(progressDistance);

Vector3 progressDelta = progressPoint.position - transform.position;

if (Vector3.Dot(progressDelta, progressPoint.direction) < 0)

{

progressDistance += progressDelta.magnitude;

}

lastPosition = transform.position;

}

}

private void OnDrawGizmos()

{

if (Application.isPlaying)

{

Gizmos.color = Color.green;

Gizmos.DrawLine(transform.position, target.position);

Gizmos.DrawWireSphere(circuit.GetRoutePosition(progressDistance), 1);

Gizmos.color = Color.yellow;

Gizmos.DrawLine(target.position, target.position + target.forward);

}

}

}

}