# Unit 3 – Lesson 9. Intelligent Seeking and Pursuing

**Aim:**

* How do we design a game with the enemy game object automatically seeking / pursuing a player game object?

**Objectives:** After the lesson, students should be able to:

* Obtain better understanding of Unity C# class hierarchies
* Design scripts to make a game object automatically pursuing / seeking a player controlled game object or any other game object

**CLASS PROCEDURE:**

***Do Now:*** Create a new Unity 3D project. Import some character package and some vehicle package. Create a new scene and call it “Seeking and Pursuing”. Drop a person character (i.e., Ethan) to your scene, and then add two vehicles. Say we want the user to control the movements of the person character, and as the person character moving around, we want the vehicles automatically “seeking” and “pursuing” the person character. How can we accomplish the task?

***Discussion / Presentation:***

1. How do we allow the user to control the movements of the person character? (Reminder: Last school year in the coin collection project, we did this task together!)
2. How do we track and save the movements of the person character?

* One of the Unity 5 AI tools we can use is the steering and steering functions
* We can use steering to store data for target movements (vectors)

1. How do we make the vehicle (agent / non – player controlled game object) automatically seeking and following the person character?

* In this unit, we will use a couple of classes again and again. In order not to reinvent the wheels every single time, we create an Agent class that can take quite a few parameters (some for other purposes in other class activities), and keep reusing it.
* We can add two classes (scripts) to one game object in Unity, and please remember – the sequence of the scripts on the Inspector menu REALLY MATTERS! It determines which script will run first!

***Pair – sharing Activity:***

1. Work with your partner, write a script and attach to the person character to allow the player controls the game object. If you already forget how to write the script, use the solution below as a reference to refresh your memory.
2. Study the Steering class, AgentBehaviour class, Agent class and the Seek and Pursue classes provided in the Solution below, discuss with your partner on how the programs work. Attach the Agent and then the Pursue scripts to the vehicle game object in your scene, save and play the game. In the game, when the player moves the person character around, the vehicle shall seek and pursue the person character.
3. Discuss with your partner and use the Unity online documentations to find the answers for the following questions:
   1. How does the game detect and keep tracking the movements of the player controlled game character?
   2. How does the game make the agent (vehicle) seeking and following the player controlled game character?
   3. What is the difference between Update() and LateUpdate()? In the Agent class, why do we use both methods?
   4. What is the difference between Start() and Awake()? Which method is executed first by Unity?
   5. What is pipeline? What is the purpose of using a pipeline?
   6. What is linear? What is angular?
   7. How do we interpret the following segment of code in the Agent class?
4. foreach (int gid in gIdList)
5. {
6. steering = new Steering();
7. foreach (Steering singleSteering in groups[gid])
8. {
9. steering.linear += singleSteering.linear;
10. steering.angular += singleSteering.angular;
11. }
12. if (steering.linear.magnitude > priorityThreshold ||
13. Mathf.Abs(steering.angular) > priorityThreshold)
14. {
15. return steering;
16. }
17. }
    1. In the Pursue class, why do we need the onDestroy() method?

***Solution / Summary:***

Steering class – serves as a custom data type for storing the movement and rotation of the agent:

using UnityEngine;

using System.Collections;

public class Steering {

public float angular;

public Vector3 linear;

public Steering()

{

angular = 0.0f;

linear = new Vector3();

}

}

AgentBehaviour class – a template class for most of behaviors that we can use and reuse in the unit.

using UnityEngine;

using System.Collections;

public class AgentBehavior : MonoBehaviour {

public GameObject target;

protected Agent agent;

public virtual void Awake()

{

agent = gameObject.GetComponent<Agent>();

}

// Update is called once per frame

void Update () {

agent.SetSteering(GetSteering());

}

public virtual Steering GetSteering()

{

return new Steering();

}

}

Agent class – this is where we create intelligent movements so that the agent game object can seek the player controlled game object:

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

public class Agent : MonoBehaviour

{

public bool blendWeight = false;

public bool blendPriority = false;

public float priorityThreshold = 0.2f;

public bool blendPipeline = false;

public float maxSpeed;

public float maxAccel;

public float maxRotation;

public float maxAngularAccel;

public float orientation;

public float rotation;

public Vector3 velocity;

protected Steering steering;

private Dictionary<int, List<Steering>> groups;

void Start ()

{

velocity = Vector3.zero;

steering = new Steering();

groups = new Dictionary<int, List<Steering>>();

}

public virtual void Update ()

{

Vector3 displacement = velocity \* Time.deltaTime;

orientation += rotation \* Time.deltaTime;

if (orientation < 0.0f)

orientation += 360.0f;

else if (orientation > 360.0f)

orientation -= 360.0f;

transform.Translate(displacement, Space.World);

transform.rotation = new Quaternion();

transform.Rotate(Vector3.up, orientation);

}

public virtual void LateUpdate ()

{

if (blendPriority)

{

steering = GetPrioritySteering();

groups.Clear();

}

velocity += steering.linear \* Time.deltaTime;

rotation += steering.angular \* Time.deltaTime;

if (velocity.magnitude > maxSpeed)

{

velocity.Normalize();

velocity = velocity \* maxSpeed;

}

if (rotation > maxRotation)

{

rotation = maxRotation;

}

if (steering.angular == 0.0f)

{

rotation = 0.0f;

}

if (steering.linear.sqrMagnitude == 0.0f)

{

velocity = Vector3.zero;

}

steering = new Steering();

}

public void SetSteering (Steering steering)

{

this.steering = steering;

}

public void SetSteering (Steering steering, float weight)

{

this.steering.linear += (weight \* steering.linear);

this.steering.angular += (weight \* steering.angular);

}

public void SetSteering (Steering steering, int priority)

{

if (!groups.ContainsKey(priority))

{

groups.Add(priority, new List<Steering>());

}

groups[priority].Add(steering);

}

public void SetSteering (Steering steering, bool pipeline)

{

if (!pipeline)

{

this.steering = steering;

return;

}

}

private Steering GetPrioritySteering ()

{

Steering steering = new Steering();

float sqrThreshold = priorityThreshold \* priorityThreshold;

List<int> gIdList = new List<int>(groups.Keys);

gIdList.Sort();

foreach (int gid in gIdList)

{

steering = new Steering();

foreach (Steering singleSteering in groups[gid])

{

steering.linear += singleSteering.linear;

steering.angular += singleSteering.angular;

}

if (steering.linear.magnitude > priorityThreshold ||

Mathf.Abs(steering.angular) > priorityThreshold)

{

return steering;

}

}

return steering;

}

}

Seek class:

using UnityEngine;

using System.Collections;

public class Seek : AgentBehavior {

public override Steering GetSteering()

{

Steering steering = new Steering();

steering.linear = target.transform.position - transform.position;

steering.linear.Normalize();

steering.linear = steering.linear \* agent.maxAccel;

return steering;

}

}

Pursue class:

using UnityEngine;

using System.Collections;

public class Pursue : Seek

{

public float maxPrediction;

private GameObject targetAux;

private Agent targetAgent;

public override void Awake()

{

base.Awake();

targetAgent = target.GetComponent<Agent>();

targetAux = target;

target = new GameObject();

}

public override Steering GetSteering()

{

Vector3 direction = targetAux.transform.position - transform.position;

float distance = direction.magnitude;

float speed = agent.velocity.magnitude;

float prediction;

if (speed <= distance / maxPrediction)

prediction = maxPrediction;

else

prediction = distance / speed;

target.transform.position = targetAux.transform.position;

target.transform.position += targetAgent.velocity \* prediction;

return base.GetSteering();

}

void OnDestroy ()

{

Destroy(targetAux);

}

}

AgentPlayer class – takes user inputs and move the player controlled game object:

using UnityEngine;

using System.Collections;

public class AgentPlayer : Agent

{

public override void Update()

{

velocity.x = Input.GetAxis("Horizontal");

velocity.z = Input.GetAxis("Vertical");

velocity \*= maxSpeed;

Vector3 translation = velocity \* Time.deltaTime;

transform.Translate(translation, Space.World);

transform.LookAt(transform.position + velocity);

orientation = transform.rotation.eulerAngles.y;

}

}

***HW:*** Continue to work with your partner on the Maze game.