Lab 5 - Implementing Sensors

COMP396-Game Programming 2

Purpose: Implementing Sensors such as **Sight** and **Touch**.

Contents

1. Intro

We will:

- Set Up the Demo Scene
- Create the Player and Aspect classes
- Create the AI Character's classes
  - Sense.cs
  - Sight.cs
  - Touch.cs
- Test the Game

1.1. Setting Up the Demo Scene

- Create a word document named **Lab5_Snapshots_{YourInitials}** to hold your snapshots.
- Open the project **COMP396_001_F24_{YourInitials}** from last labs.
- Create a folder under Scenes folder named **Lab5_{YourInitials}**.
- Create a new scene named **Lab5_Sensors_{YourInitials}** and save it in the above folder.
- $Take\ Snapshot$
- Set up the scene with the following game objects:

Game Object	Type	Parent	P(x,y,z)	R(x,y,z)	S(x,y,z)	Color
Floor	Plane	Root	(0,0,0)	(0,0,0)	(10,1,10)	Beige
Obstacles	Empty	Root	(0,0,0)	(0,0,0)	(1,1,1)	-
Wall 1-6	Cube	Obstacles	Random	Random	Random	White
Player	Cube	Root	Random	Random	Random	Blue
NPC	Cube	Root	Random	Random	Random	Red
Target	Sphere	Root	Random	Random	Random	Green

- *Take Snapshot*

1.2. The Player and Aspect classes

1.2.1 Target.cs

- Create **Target.cs** (set it via point-clicks as NPC *destination*)

```
using UnityEngine;
public class Target : MonoBehaviour {
  [SerializeField]
  private float hOffset = 0.2f;
  void Update () {
    int button = 0;
    //Get the point of the hit position when the mouse is being clicked
    if(Input.GetMouseButtonDown(button)) {
      Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
      RaycastHit hitInfo;
      if (Physics.Raycast(ray.origin, ray.direction, out hitInfo)) {
        Vector3 targetPosition = hitInfo.point;
        transform.position = targetPosition + new Vector3(0.0f, hOffset, 0.0f);
      }
    }
  }
}
```

- Attach it to the **Target** game object.
- *Take Snapshot*

1.2.2 PlayerController.cs

- Tag the Player object as **Player**.
- Add a **Rigidbody** to the Player game object and set it as **non-kinematic**.
- Create **PlayerController.cs**

```
using UnityEngine;
public class PlayerController : MonoBehaviour {
  public Transform targetTransform;
  [SerializeField]
  private float movementSpeed = 10.0f;
  [SerializeField]
  private float rotSpeed = 2.0f;
  [SerializeField]
  private float targetReactionRadius = 5.0f;
  void Update () {
    //Stop once you reached near the target position
    if (Vector3.Distance(transform.position, targetTransform.position) < targetReactionRadius)
```

```
   return;
  //Calculate direction vector from current position to target position
  Vector3 tarPos = targetTransform.position;
  tarPos.y = transform.position.y;
  Vector3 dirRot = tarPos - transform.position;
  //Build a Quaternion for this new rotation vector using LookRotation method
  Quaternion tarRot = Quaternion.LookRotation(dirRot);
  //Move and rotate with interpolation
  transform.rotation= Quaternion.Slerp(transform.rotation, tarRot, rotSpeed * Time.deltaTime);
  transform.Translate(new Vector3(0, 0, movementSpeed * Time.deltaTime));
 }
}
```

- Attach it to the **Player** game object.
- Drop the Target game object on the **targetTransform** of the PlayerController slot.
- *Take Snapshot*

1.2.3 Aspect.cs

- Create Aspect.cs

```
using UnityEngine;
public class Aspect : MonoBehaviour {
  public enum Affiliation {
    Player,
    Enemy
  }
  public Affiliation affiliation;
}
```

- Notice that an enum and a corresponding variable are defined, named respectively **Affiliation** and **affiliation**
- Drop **Aspect.cs** on NPC GameObject
- Select **Enemy** for affiliation.
- *Take Snapshot*

1.3. The AI Character classes

1.3.1 Wander.cs

- Create **Wander.cs** to control NPC's movement.

```
using UnityEngine;
using System.Collections;
public class Wander : MonoBehaviour {
 private Vector3 tarPos;
 [SerializeField] private float movementSpeed = 5.0f;
 [SerializeField] private float rotSpeed = 2.0f;
 [SerializeField] private float minX = -45.0f;
 [SerializeField] private float maxX = 45.0f;
 [SerializeField] private float minZ = -45.0f;
 [SerializeField] private float maxZ = 45.0f;
 [SerializeField] private float targetReactionRadius = 5.0f;
 [SerializeField] private float targetVerticalOffset = 0.5f;
 void Start () {
  //Get Wander Position
  GetNextPosition();
 }
 void Update () {
  // Check if we're near the destination position
  if (Vector3.Distance(tarPos, transform.position) ≤ targetReactionRadius)
   GetNextPosition();
  // generate new random position
  // Set up quaternion for rotation toward destination
  Quaternion tarRot = Quaternion.LookRotation(tarPos - transform.position);
  // Update rotation and translation
```

```
    transform.rotation = Quaternion.Slerp(transform.rotation, tarRot, rotSpeed * Time.deltaTime);
    transform.Translate(new Vector3(0, 0, movementSpeed * Time.deltaTime));
  }
  void GetNextPosition() {
    tarPos = new Vector3(Random.Range(minX, maxX), targetVerticalOffset, Random.Range(minZ, maxZ));
  }
}
```

- Attach it to the NPC.
- Notice that the **Wander.cs** script generates a new *random position* in a specified range whenever an NPC character reaches its current *destination point*. Then, the *Update* method **rotates** the NPCs and moves them toward their new destination.
- *Take Snapshot*

1.3.2 Sense.cs (base class for Sight and Touch)

- Create **Sense.cs** base class with two virtual methods:

```
using UnityEngine;
public class Sense : MonoBehaviour {
  public bool bDebug = true;
  public Aspect.Affiliation targetAffiliation = Aspect.Affiliation.Enemy;
  public float detectionRate = 1.0f;
  protected float elapsedTime = 0.0f;
  protected virtual void Initialize() { }
  protected virtual void UpdateSense() { }
  void Start () {
    Initialize();
  }
  void Update () {
    UpdateSense();
  }
}
```

- *Take Snapshot*

1.3.3 Sight.cs

- Create Sight.cs derived from Sense.

```
using UnityEngine;
public class Sight: Sense {
  public int FieldOfView = 45;
  public int ViewDistance = 100;
  private Transform playerTrans;
  private Vector3 rayDirection;
  protected override void Initialize() {
    //Find player position
    playerTrans = GameObject.FindGameObjectWithTag("Player").transform;
  }
  protected override void UpdateSense() {
    elapsedTime += Time.deltaTime;
    // Detect perspective sense if within the detection rate
    if (elapsedTime ≥ detectionRate) {
      DetectAspect();
      elapsedTime = 0.0f;
    }
  }
  //Detect perspective field of view for the AI Character
  void DetectAspect() {
    //Direction from current position to player position
    rayDirection = (playerTrans.position - transform.position).normalized;
    //Check the angle between the AI character's forward vector and the direction vector between
    //player and AI to detect if the Player is in the field of view.
    if ((Vector3.Angle(rayDirection, transform.forward)) < FieldOfView) {
      RaycastHit hit;
      if (Physics.Raycast(transform.position, rayDirection, out hit, ViewDistance)) {
        Aspect aspect = hit.collider.GetComponent<Aspect>();
        if (aspect != null) {
          //Check the aspect
          if (aspect.affiliation == targetAffiliation) {
            print("Enemy Detected");
          }
        }
      }
    }
  }
  void OnDrawGizmos() {
    if (!Application.isEditor|| playerTrans == null)
      return;
    Debug.DrawLine(transform.position, playerTrans.position, Color.red);
    Vector3 frontRayPoint = transform.position + (transform.forward * ViewDistance);
    //Approximate perspective visualization
    Vector3 leftRayPoint = Quaternion.Euler(0,FieldOfView * 0.5f ,0) * frontRayPoint;
    Vector3 rightRayPoint = Quaternion.Euler(0,- FieldOfView*0.5f, 0) * frontRayPoint;
    Debug.DrawLine(transform.position, frontRayPoint, Color.green);
    Debug.DrawLine(transform.position, leftRayPoint, Color.green);
    Debug.DrawLine(transform.position, rightRayPoint, Color.green);
  }
}
```

- Attach it to the NPC game object.
- *Take Snapshot*

1.3.4 Touch.cs

- Create Touch.cs derived from Sense (using **OnTriggerEnter**)

```
using UnityEngine;
public class Touch : Sense {
  void OnTriggerEnter(Collider other) {
    Aspect aspect = other.GetComponent<Aspect>();
    if (aspect != null) {
      //Check the aspect
      if (aspect.affiliation == targetAffiliation) {
        print("Enemy Touch Detected");
      }
    }
  }
}
```

- Attach it to the NPC game object.
- Add a **BoxCollider** to the NPC (if it doesn't have one already)
- Check **Is Trigger** property of the BoxCollider.
- *Take Snapshot*

1.4. Testing the Game

- Press *Play*
- Move player by point-and-clicking on the ground.
- Bring the player within FOV of AI.
  - Notice the message *“Player Detected”*
- Bring the player close to the NPC.
  - Notice the message *“Player Touch Detected”*
- *Take Snapshot*

1.5. Video

- Make a short **video** (~1-2 min) with playtesting **Sight** and **Touch** senses as above. Name it **Lab5_Video_{YourInitials}**.

1.6. Deliverables

- Make a unitypackage out of **Lab5_{YourInitials}** unity folder.
- Zip together:
  - The Snapshots document.
  - The unity package created above
  - the video created above.
- Name the Zip file **Lab5_Sensors_{YourInitials}.zip** (nor .rar files please!!!).
- Submit in eCentennial

1.7. Summary

Here we dealt with the following:

- Implemented **PlayerController** to control the player via mouse point-and-click.
- Implemented the **Wander** for the NPCs
- Implemented the **Sight** and **Touch** senses for the NPCs, based on the **Sense** base class.
- Playtested the system.

Next week we will implement **flocking** behaviour.