2020

Tallinna ülikool

Team:
Anastasiia Tcepeleva
Syeda Ghazal

ARTIFICIAL INTELLIGENCE FOR GAMES

AI Unity Assignment

Artificial intelligence

In games, artificial intelligence (AI) is used to create responsive, adaptive or intelligent actions mainly in non-player characters (NPCs) close to human-like intelligence.

Unity AI Project:

Functions: There are two cubes one is in the Idle position and the other one is **patrolling** around the map. When you try to reach them they start **chasing** you. There are also spheres flocking and flying around.

There are two scripts responsible for cube's AI: Patrol.cs and EnemyController.cs.

There are 6 scripts responsible for flocking AI behaviour: FlockBehaviour.cs, AlignmentBehaviour.cs, AvoidanceBehaviour.cs, SteeredCohesionBehaviour.cs, StayInRadiusBehaviour.cs, CompositeBehaviour.cs,.

Patrol.cs Code

```
public class Patrol : MonoBehaviour
{
    public float lookRadius = 10f;

    Transform target; //Reference to the player
    NavMeshAgent agent; //Reference to the NavMeshAgent
    CharacterCombat combat;

    public float speed;
    private float waitTime;

    public Transform[] moveSpots;
    private int randomSpot;

    void Start()
    {
        target = PlayerManager.instance.player.transform;
        agent = GetComponent<NavMeshAgent>();
        combat = GetComponent<CharacterCombat>();
        waitTime = UnityEngine.Random.Range(0, 4);
}

    void Awake()
    {
        agent = GetComponent<NavMeshAgent>();
        agent.speed = speed;
        randomSpot = UnityEngine.Random.Range(0, moveSpots.Length);
    }

    void Update()
    {
        float distance = Vector3.Distance(target.position, transform.position);
```

```
        if (distance <= lookRadius)
        {
            // Move towards the target
            agent.SetDestination(target.position);

            //If within attacking distance
            if (distance <= agent.stoppingDistance)
            {
                CharacterStats targetStats = target.GetComponent<CharacterStats>();
                if (targetStats != null)
                {
                    // Attack the target
                    combat.Attack(targetStats);
                }
                // Face the target
                FaceTarget();
            }
        }
        else
        {
            Wander();
        }
    }
```

This code makes enemy move towards the player if the player is within the looking radius. If the distance between him and player is less or equal than an acceptable, he stops. If he reaches the player (the distance between him and the player is less than stopping distance), it starts to attack. If enemy does not see the player, it patrols.

```
void Wander()
{
    //moves to the random spot
    if (Vector3.Distance(transform.position, moveSpots[randomSpot].position) >= 2f)
    {
        agent.SetDestination(moveSpots[randomSpot].position);
    }
    //if reaches the spot, then waits
    if (Vector3.Distance(transform.position, moveSpots[randomSpot].position) < 2f)
    {
        //and finds another spot
        if (waitTime <= 0)
        {
            int currentSpot = randomSpot;
            randomSpot = UnityEngine.Random.Range(0, moveSpots.Length);
            if (moveSpots.Length > 1)
            {
                while (currentSpot == randomSpot)
                {
                    randomSpot = UnityEngine.Random.Range(0, moveSpots.Length);
                }
            }
            waitTime = UnityEngine.Random.Range(0, 4);
        }
        else
        {
            waitTime -= Time.deltaTime;
        }
    }
}
```

This code is about patrolling. The enemy is moving between spots placed on the ground. All spots are inside moveSpots array. If enemy reaches the point, he starts to wait for a random time between 0 and 4 seconds, and after that finds another random spot. If this spot is the same as the current spot, he finds another one (if there is more than 1 spot in array).

The speed of the enemy and the looking radius are editable inside Unity Inspector.

```
    void FaceTarget()
    {
        Vector3 direction = (target.position - transform.position).normalized;
        Quaternion lookRotation = Quaternion.LookRotation(new Vector3(direction.x, 0, direction.z));
        transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation, Time.deltaTime * 5f);
    }

    void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, lookRadius);
    }
}
```

FaceTarget() method is to make enemy look at the player while chasing. OnDrawGizmosSelected() makes a sphere gizmo around the enemy to help user edit the looking radius.

EnemyController.cs Code

```
public class EnemyController : MonoBehaviour
{
    public float lookRadius = 10f;

    Transform target; //Reference to the player
    NavMeshAgent agent; //Reference to the NavMeshAgent
    CharacterCombat combat;

    public float speed;

    // Start is called before the first frame update
    void Start()
    {
        target = PlayerManager.instance.player.transform;
        agent = GetComponent<NavMeshAgent>();
        combat = GetComponent<CharacterCombat>();
        agent.speed = speed;

    }

    // Update is called once per frame
    void Update()
    {
        float distance = Vector3.Distance(target.position, transform.position);

        if(distance <= lookRadius)
        {
            // Move towards the target
            agent.SetDestination(target.position);

            //If within attacking distance
            if (distance <= agent.stoppingDistance)
            {
                CharacterStats targetStats = target.GetComponent<CharacterStats>();
                if (targetStats != null)
                {
                    // Attack the target
                    combat.Attack(targetStats);
                }
                // Face the target
                FaceTarget();
            }
        }
    }
```

```
    void FaceTarget()
    {
        Vector3 direction = (target.position - transform.position).normalized;
        Quaternion lookRotation = Quaternion.LookRotation(new Vector3(direction.x, 0, direction.z));
        transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation, Time.deltaTime * 5f);
    }

    void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, lookRadius);
    }
}
```

This code is for enemy in the Idle position. Exactly like the patrolling enemy, this NPC starts chasing the player when he is withing the looking radius and attacks if reaches him. The speed of the enemy and the looking radius are also editable inside Unity Inspector.

Flocking AI

Basic models of flocking behaviour are controlled by three simple rules:

1. Avoiding - avoid crowding neighbours (short range repulsion)
2. Alignment - steer towards average heading of neighbours
3. Cohesion - steer towards average position of neighbours (long range attraction)

FlockBehaviour.cs Code

```
Ссылок: 12
public abstract class FlockBehaviour : ScriptableObject
{      2
    Ссылок: 9
    public abstract Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock);
}
```

This is an abstract class with an abstract method CalculateMove. We override it in each Behaviour script.

AlignmentBehaviour.cs Code

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

[CreateAssetMenu(menuName = "Flock/Behaviour/Alignment")]
Ссылок: 0
public class AlignmentBehaviour : FilteredFlockBehaviour
{
    Ссылок: 9
    public override Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock)
    {
        //if no neighbours, maintain current alignment
        if (context.Count == 0)
            return agent.transform.up;

        //add all points together and average
        Vector3 alignmentMove = Vector3.zero;
        List<Transform> filteredContext = (filter == null) ? context : filter.Filter(agent, context);
        foreach (Transform item in filteredContext)
        {
            alignmentMove += (Vector3)item.transform.up;
        }
        alignmentMove /= context.Count;

        return alignmentMove;
    }
}
```

In this code we made a behavior for alignment. If an object does not have neighbors, it just moves. If the object has any neighbors, we add all points together and calculate the average of them. In this CalculateMove method we return the Vector3 variable for alignment moving. Transform.up moves the object while also considering its rotation.

AvoidanceBehaviour.cs Code

```
[CreateAssetMenu(menuName = "Flock/Behaviour/Avoidance")]
Ссылок: 0
public class AvoidanceBehaviour : FilteredFlockBehaviour
{
    Ссылок: 9
    public override Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock)
    {
        //if no neighbours return no adjustments
        if (context.Count == 0)
            return Vector3.zero;

        //add all points together and average
        Vector3 avoidanceMove = Vector3.zero;
        int nAvoid = 0;
        List<Transform> filteredContext = (filter == null) ? context : filter.Filter(agent, context);
        foreach (Transform item in filteredContext)
        {
            if(Vector3.SqrMagnitude(item.position - agent.transform.position) < flock.SquareAvoidanceRadius)
            {
                nAvoid++;
                avoidanceMove += (Vector3)(agent.transform.position - item.position);
            }
        }
        if (nAvoid > 0)
        {
            avoidanceMove /= nAvoid;
        }
        return avoidanceMove;
    }
}
```

In this code we calculate the movement for Avoidance rule. If an object does not have neighbors, it doesn't need to move, so we return zero Vector3. If the distance between the agent and each of his neighbors is lower than the avoidance radius, we add a movement in a Vector 3 variable. If the number of these movements is greater than 0, we calculate the average.

SteeredCohesionBehaviour.cs Code

```
[CreateAssetMenu(menuName = "Flock/Behaviour/SteeredCohesion")]
Ссылок: 0
public class SteeredCohesionBehaviour : FilteredFlockBehaviour
{

    Vector3 currentVelocity;
    public float agentSmoothTime = .5f;
      2
    Ссылок: 9
    public override Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock)
    {
        //if no neighbours return no adjustments
        if (context.Count == 0)
            return Vector3.zero;

        //add all points together and average
        Vector3 cohesionMove = Vector3.zero;
        List<Transform> filteredContext = (filter == null) ? context : filter.Filter(agent, context);
        foreach (Transform item in filteredContext)
        {
            cohesionMove += (Vector3)item.position;
        }
        cohesionMove /= context.Count;

        //create offset from agent position
        cohesionMove -= (Vector3)agent.transform.position;
        cohesionMove = Vector3.SmoothDamp(agent.transform.up, cohesionMove, ref currentVelocity, agentSmoothTime);
        return cohesionMove;
    }
}
```

In this code we calculate the movement for Cohesion rule. Again, if an object does not have any neighbors, it does not need to move. If it has neighbors, we add a position of each neighbor in a cohesionMove variable and then calculate the average. After that, we create an offset from the position of the object and gradually change a vector over time.

StayInRadiusBehaviour.cs Code

```
[CreateAssetMenu(menuName = "Flock/Behaviour/Stay In Radius")]
Ссылок: 0
public class StayInRadiusBehaviour : FlockBehaviour
{
    public Vector3 center;
    public float radius = 15f;

    Ссылок: 9
    public override Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock)
    {
        Vector3 centerOffset = center - (Vector3)agent.transform.position;
        float t = centerOffset.magnitude / radius;
        if (t < 0.9f)
        {
            return Vector3.zero;
        }

        return centerOffset * t * t;
    }
}
```

This code is to avoid flock's flying away. Here we calculate the center of the flock and for each agent check if we are inside the radius.

CompositeBehaviour.cs Code

```
public class CompositeBehaviour : FlockBehaviour
{
    public FlockBehaviour[] behaviours;
    public float[] weights;

    Ссылок: 9
    public override Vector3 CalculateMove(FlockAgent agent, List<Transform> context, Flock flock)
    {
        //handle data mismatch
        if (weights.Length != behaviours.Length)
        {
            Debug.LogError("Data mismatch in " + name, this);
            return Vector3.zero;
        }

        //set up move
        Vector3 move = Vector3.zero;

        //iterate through behaviours
        for (int i = 0; i < behaviours.Length; i++)
        {
            Vector3 partialMove = behaviours[i].CalculateMove(agent, context, flock) * weights[i];

            if (partialMove != Vector3.zero)
            {
                if (partialMove.sqrMagnitude > weights[i] * weights[i])
                {
                    partialMove.Normalize();
                    partialMove *= weights[i];
                }

                move += partialMove;
            }
        }

        return move;
    }
}
```

This code is about combining all the behaviors. First, we must check if the number of weights matches the number of behaviors. If not, we return Vector3.zero to not move anything and show the error. If it matches, we calculate the total movement depending on behavior's weights.

It is possible in Unity Inspector to add all Behaviors or to delete some of them and to set Weights for each.