Artificial Intelligence

Table of contents

INTRODUCTION

This is my first unity project in which I made a survival game where the player is constantly attacked by endless waves of enemies trying to surround and eventually kill the player. In order to do this I wrote the necessary AI scripts to handle my enemies' behavior. The AI concept I will focus on during this process is called 'Situational Awareness'. All scripts are written in C# and compiled by the Unity game engine.

WHAT IS SITUATIONAL AWARENESS?

Situational awareness is the ability to analyze one's surroundings and acting accordingly. This can mean either being able to guarantee one's own survival or being able to complete a mission or achieve a goal using teamwork and/or the environment. This requires reasoning and is therefore strictly limited to living organisms capable of thinking rationally.

SITUATIONAL AWARENESS IN GAME AI

Computers on the other hand cannot think in the same way humans or animals do and are thus incapable of having situational awareness in a self-thought way. This means AI isn't truly intelligent and is merely built up by a series of logical tests. When a bot faces a problem in the same circumstances 10 times it will always solve the problem the same way and the result will never change. Knowing this we can however attempt to recreate realism by using probability and random values in our behavior scripts which allows us to achieve different results in identical situations. This 'randomness' of events can make things a lot more interesting and forces the player to analyze the situation instead of learning the bot's pattern and causes the game to feel more challenging instead of 'grindy'. For example; random spawning makes sure the player never faces the exact same scenario more than once.

POSSIBLE AI TOPICS

- Collecting objects & pick-ups
- Lign of sight (hide, take cover)
- Movement prediction
- Avoid danger (e.g. player has power-up → flee)
- …

Spawn Pickup

I'm using a pickup manager script to handle the spawning of my pickups. Their position and rotation is randomly generated within the scene bounds and then snapped to the navigation mesh. My pickup manager has a property called 'Pickup Type List' which is a List I use to store all the pickup prefabs. This way I can choose a pickup to spawn at random and alter the likeliness of a certain pickup spawning by adding multiple copies of it to the list. The following function handles the spawning of my pickups.

```
void SpawnPickup()
{
    // Get random x- & y-position inside scene bounds
    Vector3 randomPos = Vector3.zero;
    randomPos.x = Random.Range(_sceneBounds.min.x, _sceneBounds.max.x);
    randomPos.y = (_sceneBounds.max.y - _sceneBounds.min.y) / 2 + _sceneBounds.min.y;   // Get average scene height position
    randomPos.z = Random.Range(_sceneBounds.min.z, _sceneBounds.max.z);

    // Sample point to snap it onto the nearest point of the navMesh
    NavMeshHit hitInfo;
    NavMesh.SamplePosition(randomPos, out hitInfo, 100, -1);
    Vector3 sampledPos = hitInfo.position;

    // Get random rotation
    Quaternion randomQuat = new Quaternion(0.0f, Random.Range(360.0f, 0.0f), 0.0f, 0.0f);

    // Get distance between the player and the sampled position
    float distance = Vector3.Distance(GameObject.FindGameObjectWithTag("Player").transform.position, sampledPos);

    if (distance > NoSpawnRadius)
    {
        // Get random pickup
        int randomIndex = Random.Range(0, PickupTypeList.Count);

        // Spawn the pickup on the navmash given a random position and rotation, outside of the player's no-spawn zone
        PickupList.Add((GameObject)Instantiate(PickupTypeList[randomIndex], sampledPos + new Vector3(0, HeightOffset, 0), randomQuat));
    }
    else
    {
        // If the generated position lies within the no-spawn zone, restart the method to generate another position and rotation
        SpawnPickup();
    }
}
```

However an enemy can only pickup certain pickups, one of which being armor. All enemies will be using flocking in order to chase the player, relying on them to accidently walk over an armor pickup wouldn't be very intelligent. To make sure my enemies are able to detect nearby armor pickups I attach a sphere collider to every armor pickup. This way I can more efficiently check whether an enemy is in range without having to individually calculate the distance from every enemy to every pickup and comparing it to a predefined value.

PICKUP DETECTION

I use the OnTriggerEnter function of Unity to detect enemies entering the armor's pickup range. First of all I give every pickup object a box collider and the pickups obtainable by enemies and extra sphere collider. This way I can use the first collider to check range and the second collider to check whether the pickup has been reached.

Every enemy has a Boolean property called IsCollecting. If this value is true the enemy will ignore the steering towards the player and instead change its destination to the location of the pickup. However, every pickup that can be collected by an enemy also has a CollectionStatus script attached which determines if the pickup is already being collected or still available for collection. When an enemy dies while collecting the pickup the collection status' IsBeingCollected boolean is set to false and the pickup can be collected by another enemy.

In the OnTriggerEnter function I check which collider is hit. When the sphere collider is hit the IsCollecting Boolean is set to true, the pickup GameObject is passed to the agent movement script's PickupObject value and the pathing is altered accordingly. The pickup's collection status is also updated to include the GameObject of the collector and the IsBeingCollected boolean is set to true. Upon reaching the box collider of the pickup is destroyed, IsCollecting boolean is set to false, the PickupObject variable of the AgentMovement script is set to null and the enemy continues to pursue the player.

```
0 references
void OnTriggerEnter(Collider other)
{
    // If the other collider is a BoxCollider, the object is a pickup
    if (other.GetType() == typeof(BoxCollider) && other.tag == "Armor")
    {
        // Apply pickup effects
        _enemyArmor.AddArmor(ArmorValue);

        // Remove from list
        for (int i = 0; i < _pickupManager.PickupList.Count; i++)
        {
            if (_pickupManager.PickupList[i] == other.gameObject) _pickupManager.PickupList.RemoveAt(i);
        }

        // Destroy gameobject
        Destroy(other.gameObject);

        // Set agent variables
        gameObject.GetComponent<AgentMovement>().IsCollecting = false;
        gameObject.GetComponent<AgentMovement>().PickupObject = null;
    }

    // If the other collider is a SphereCollider, move towards the GameObject to collect the pickup
    else if(other.GetType() == typeof(SphereCollider) && other.gameObject.tag == "Armor")
    {
        if (other.gameObject.GetComponent<CollectionStatus>().IsBeingCollected == false)
        {
            // If the distance between the enemy and the pickup is smaller than the distance between the enemy and his current destination
            float distanceToPickup = Vector3.Distance(gameObject.transform.position, other.GetComponent<Transform>().position);
            float distanceToDestination = Vector3.Distance(gameObject.transform.position, gameObject.GetComponent<AgentMovement>().Destination);

            if (distanceToPickup < distanceToDestination)
            {
                // Change destination to the pickup's position by setting isCollecting and passing the target object to the AgentMovement script
                gameObject.GetComponent<AgentMovement>().IsCollecting = true;
                gameObject.GetComponent<AgentMovement>().PickupObject = other.gameObject;

                // Set collection status
                other.gameObject.GetComponent<CollectionStatus>().IsBeingCollected = true;
                other.gameObject.GetComponent<CollectionStatus>().Collector = gameObject;
            }
        }

        // Otherwise, ignore the pickup
    }
}
```

COLLECTING THE PICKUP

First we perform a raycast to check for lign of sight. If a hit is detected an obstacle is blocking the way. We then tell the agent to use the obstacle avoidance of the Unity navmesh and no manual steering (flocking) is used. If no hits are detected we use manual steering to make the enemies flock and attack the layer as a group.

Then we perform a series of logical tests and set the destination accordingly

- Is the enemy stunned (performing an attack)?
- Is he dead?
- Is he currently collecting a pickup?

```
// SET DESTINATION
//****************
Vector3 avatarPos = _avatar.GetComponent<Transform>().position;
Vector3 agentPos = _agent.GetComponent<Transform>().position;

// If the stunned or dead -> Destination = agent position
if (IsStunned || _enemyHealth._isDead)
{
    Destination = agentPos;
}

// If the agent is collecting a pickup and is not stunned or dead -> Destination = pickup
else if (IsCollecting)
{
    if (PickupObject == null)
    {
        IsCollecting = false;
        Destination = avatarPos;
    }
    else
    {
        Destination = PickupObject.transform.position;
    }
}

// If the agent is not collecting a pickup, stunned or dead -> Destination = player
else
{
    Destination = avatarPos;

    // Use flocking when active
    if (UseFlocking) FlockToTarget(_avatar);
}

_agent.destination = Destination;
```

Conclusion

This is my first time making a game in Unity and therefore I'm happy with the results I managed to achieve. I was quite intimidated at first but there are a lot of resources and tutorials that allowed me to quickly learn how to work with the game engine.

I am happy my pickup collection AI and the way it currently works but there are a couple of things I might want to do differently. One of them is the fact that an enemy only checks if the pickup is available upon entering the sphere collider. This means that when 2 enemies enter the collider at the same time but the first one dies, the second enemy simply ignores the pickup thinking it is already being collected. This can be resolved by using the OnCollisionStay function instead of OnCollisionEnter but seemed way less performant.

I will definitely continue working on this project in the near future since I really enjoyed working on it and a lot of mechanics can be added such as special attacks different enemies with unique properties.

Special thanks

I would like to thank AxeyWorks and BitGem for letting me use their low poly environment and character assets for free.
Unity's tutorials and API also proved to be extremely useful in the learning process.

References

https://unity3d.com/learn/tutorials/projects/survival-shooter-tutorial (Basics)

https://www.youtube.com/watch?v=wdOk5QXYC6Y (Animation tutorial)

https://www.assetstore.unity3d.com/en/ (Assets)

https://shop.bitgem3d.com/ (Characters and animations)