ARTIFICIAL INTELLIGENCE: SITUATIONAL AWARENESS

So first of all before we start, we have to ask ourselves: What’s situational awareness exactly. A quick google search reveals that:

**WHAT IS SITUATIONAL AWARENESS EXACTLY?**

**Situational Awareness:** is the ability to identify, process, and comprehend the critical elements of information about what is happening to the team with regards to the mission. More simply, it's knowing what is going on around you.

---

**LEGEND**

---

**INTRODUCTION**

So first off I made an ItemSpawnScript which will spawn our armor pickups. These will need to be spawned in at random positions around the map. But also preferable in a radius around the player.

So I made an overloaded function to generate a random spawn point which is sampled on the nav mesh. The first function will just generate a random position on the nav mesh. For the second
function you can enter a parameter (GameObject), which will result in the Item – In this case it’s our armor – being spawned randomly inside a radius around the given parameter.

Here are the two functions calls:

Random locations on the nav mesh:

```csharp
//Generate SpawnPos inside bounding box on navmesh
Vector3 GenerateSpawnPos() {
    Vector3 spawnPos;
    //Generate random pos
    spawnPos.x = Random.Range(-_size.x, _size.x);
    spawnPos.y = 1;
    spawnPos.z = Random.Range(-_size.z, _size.z);
    int layerMask = (1 << NavMesh.GetAreaFromName("Walkable"));
    NavMeshHit hit;
    //Get closest pos to random pos on navmesh
    NavMesh.SamplePosition(spawnPos, out hit, 30, layerMask);
    Vector3 generatedPos = hit.position;
    generatedPos.y += 1;
    //Return newly generated pos
    return generatedPos;
}
```

Random locations in a radius around a given parameter (GameObject):

```csharp
//Generates SpawnPos in a radius around target pos
Vector3 GenerateSpawnPos(Vector3 targetPos, float radius) {
    //Generate random values
    float xValue = Random.Range(-radius, radius);
    float zValue = Random.Range(-radius, radius);
    Vector3 generatedPos = new Vector3( targetPos.transform.position.x +xValue, 1,
                                         targetPos.transform.position.z + zValue);

    int layerMask = (1 << NavMesh.GetAreaFromName("Walkable"));
    NavMeshHit hit;
    //Get closest pos to generated random pos on navmesh
    NavMesh.SamplePosition(generatedPos, out hit, 30, layerMask);
    Vector3 generatedPos = hit.position;
    //Offset the y value otherwise the item will be stuck in the ground
    generatedPos.y += 1;
    //Return newly generated pos
    return generatedPos;
}
```
AI: ASSESSING INFORMATION

So first of all we need to assess all the information in the immediate surroundings of our agent. In our case the agent is an enemy zombie. We need to go over all the possible decisions our agent can make, and all the possible outcomes of those decisions.

The goal of the agent is to try and damage the player. But it is easier for the player to kill the zombie because the player moves much faster in his vehicle than the zombie moves by foot. Basically the player’s strength lies in his speed and manoeuvrability. Whereas the zombie’s strength lies in numbers, the difficulty level of the game should lie in the huge number of zombies that are thrown at the player.

So when would it be more efficient for the zombie to go off route and pickup armor first. Regroup with the hoard and then attack the player with a higher chance of success rate.

For our agent to go and pickup an armor piece and still have an advantage we need to know:

- Distance to the armor pickup
- Distance to Target (player)
- Distance to the Hoard (Average position of all flocked zombies)
- A list with all possible armor pickups
- A list with all other zombies (agents)
  - So we can determine if there isn’t an agent which is closer to the wanted armor pickup
- Armor desirability when armor is closer than the target
- Armor desirability when armor is further away than the target

AI: IMPLEMENTATION METHOD

First of all we loop through our list of possible armor pickups. The list is always changing in size, because of armor pickups that spawn in and player picking them up. So we need to check if the item in our list of pickups isn’t null. After that we check if our current agent isn’t already on its way to an armor pickup.

Now we calculate the distance from our agent to the current armor pickup (inside the foreach loop). If this distance is smaller than to the agent’s destination then we generate a random float between 0.0 and 1.0. If this newly generated number is lower the 0.8 then we continue the calculations. If this is not the case we stop here. For our last step we check if the armor pickup our agent now wants already as an agent on its way to that armor piece. But if that isn’t so then we can set the newly picked armor pickup as the agent’s new destination.
I implemented on last piece of code to make it a little more random. So if the agent is closer to the destination than to the armor pickup there is still a 10% chance that the agent will go for the armor than directly for the destination (Player).

For debug purposes I made an agent that is on its way to an armor pickup blue and those on their way to the player red. This way it's much easier to debug.

```csharp
void DesireItem()
{
    //If item is snatched = reset
    if (_gettingItem && _itemTarget == null)
    {
        _gettingItem = false;
    }

    //Check is our agent already has armor active and enabled
    if (!GetComponent<ArmorBehaviour>().enabled)
    {
        //variables
        var ItemList = GameObject.Find("GameManager").GetComponent<ItemManager>().GetArmorList.FindAll(x => x != this);
        var zombielist = GameObject.Find("GameManager").GetComponent<EnemyManager>().GetEnemyList.FindAll(x => x != this);

        float insideChance = 0.80f;
        float outsideChance = 0.10f;
        float distanceToTarget = Vector3.Distance(_agent.transform.position, _agent.destination);
        float distanceToItem;
        Vector3 averageZombiePos = Vector3.zero;
        foreach (var zombie in zombielist)
        {
            averageZombiePos += zombie.transform.position;
        }
        averageZombiePos /= zombielist.Count;
        float distanceToHoard = Vector3.Distance(_agent.transform.position, averageZombiePos);

        foreach (var item in ItemList)
        {
            //If item doesn't exist anymore the continue
            if (item == null)
            {
                continue;
            }

            //If Agent already has a desired item
            if (_gettingItem)
            {
                continue;
            }

            //Calculate distance to Item
            distanceToItem = Vector3.Distance(_agent.transform.position, item.transform.position);

            //If Armor is Closer than Target ----> 80% Desirability
            //If Armor is Further than Target ----> 20% Desirability
            if ((distanceToTarget > distanceToItem && Random.value <= insideChance) || Random.value <= outsideChance)
            {
                //Check if is already wanted
                if (!item.GetComponent<ArmorPickupBehaviour>().IsWanted)
                {
```
AI: FLOCKING VS. NAV MESH

So the AI concept I’m using for the zombie hoard is flocking. To flock means to travel together usually in a big group. So when we make our agents flock together when traveling the player. We basically give the zombies strength in numbers. Which should make the game more challenging for the player.

So for flocking we use the unity nav mesh to get us the where we need to be. The only thing we do differently is we manually override the steer vector of our agent. We make our agent steer toward each other and toward the target at the same time. So we form a hoard on the way to the player.

But in order to make a decent zombie hoard we need to implement a couple of vectors containing parameters. which are the following:

- Alignment vector (make them align with each other)
- Separation vector (we want them to group, but not too much.)
- Cohesion vector (make them ‘stick’ together like a hoard)
- Steer vector (a vector steering them to the destination)

But when we are over writing the whole movement of the agent with our flocking algorithm our zombie’s ignore pathfinding when our target is not in its line of sight.

How I fix this is with implementing a ray trace from each agent to the player. When our ray doesn’t connect with the player it means there is an obstacle blocking our path. In this case we switch over to the normal unity nav mesh path finding.

Once our agent’s ray trace does connect with the player – Which means are on a clear path towards the player without any obstacles – we switch back over to our flocking algorithm and override all movement of our agent.
AI: SLIPSTREAM

When a zombie has armour equipped he has a knockback effect on the player. Which can be really annoying for the player because he can’t just plow through the zombie hoard. He will be knock back by zombies with armour and lose all his speed. After which the player is stuck in the middle of the zombie hoard.

But the only way this is going to benefit the zombie’s is when the zombies with amor walk in front of the hoard. If they would be in the back of the hoard then the player could just run over all the zombies and be stopped by the last tough ones. Who the player can deal with separately. But if the armored zombie walk in front of the hoard it would make it much harder for the player to just run them all over cause of the knockback effect the armored zombies have.

Implemented this by giving the armored zombies a little speed up. Not too much of a speed boost, I made them just a little bit faster, you can barely notice it. But this way if the hoard is on its way to the player and there are armored zombies in the hoard they slowly but steady overtake the other zombies. Thus making them be the front line of the hoard.

The zombies that are running away from the hoard to get a amor piece also gain a small amount of slipstream to make up for the time they lose not in pursuit of the player.

Sources:  http://www.aerohabitat.eu/uploads/media/Automation_and_Situation_Awareness_-_Endsley.pdf
http://www.gdcvault.com/play/1015443/Situational-Awareness-Terrain-Reasoning-for