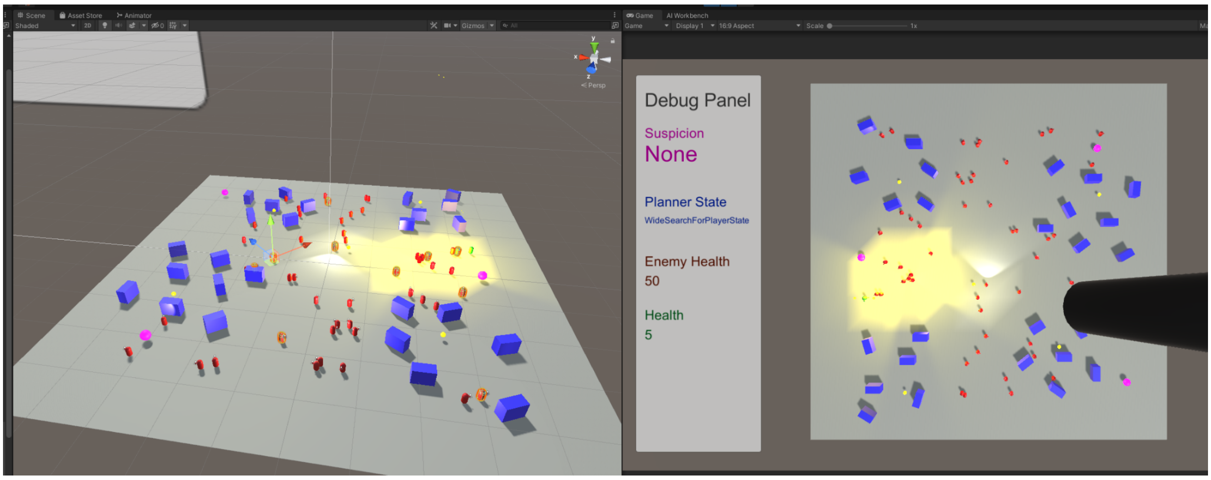
Performance Analysis and Post Mortem

GAM457.2

ROO KOOISTRA - 1011439

The purpose of this report is to analyse the effectiveness of the AI algorithms implemented in the GAM457.1 assignment. The project was to build an AI behaviour that demonstrated working knowledge of their systems. I chose to use a planner for this behaviour. This behaviour was one of a search and destroy where the agent needed to maintain its energy level, search for the player and report the player's location to the overhead guns that would then fire upon the player. The agent had stored knowledge on the players last known whereabouts from both the vision and hearing abilities and I will discuss them further in this paper.

It is usual to give a detailed performance analysis regarding performance and identify bottlenecks. In this case, there weren't any performance issues to report. Even when there were fifty AI's running at the same time, the performance was almost identical to when there were one.



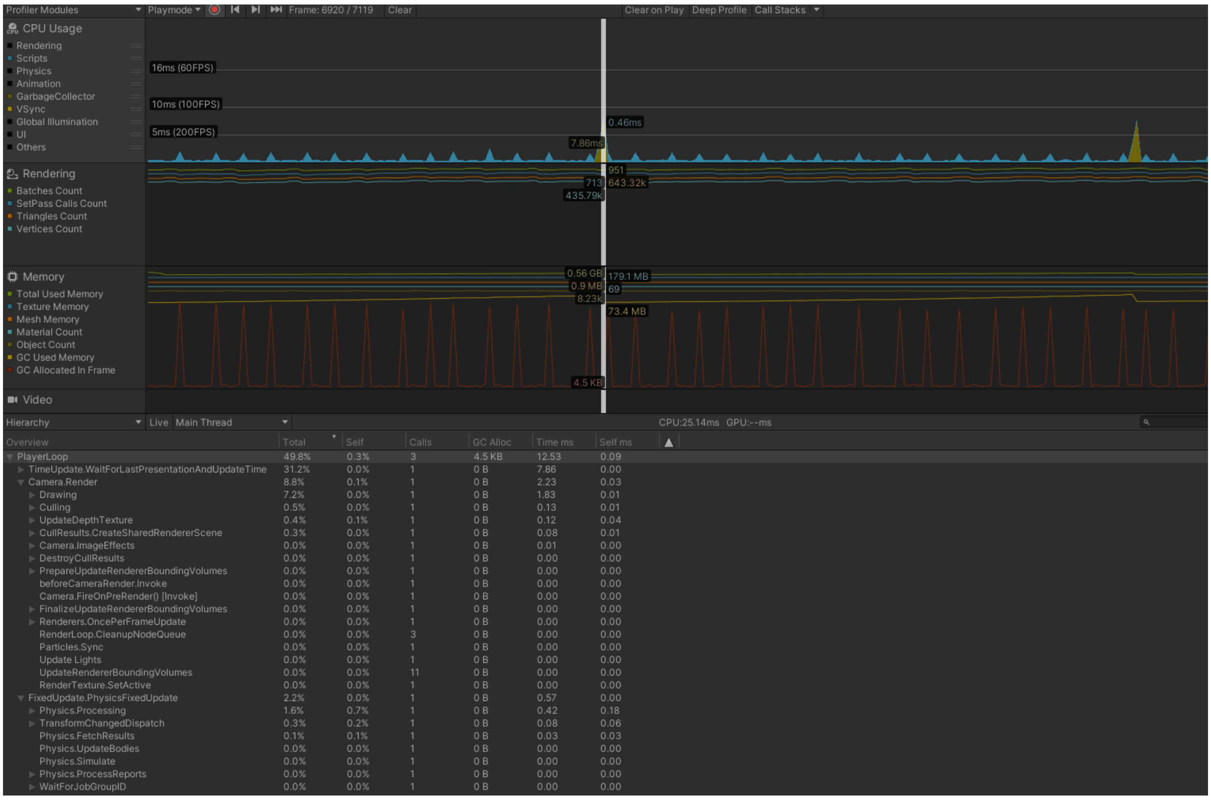
Debug Panel

Suspicion
None

Planner State
WideSearchForPlayerState

Enemy Health
50

Health
5



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
PlayerLoop	49.8%	0.3%	3	4.5 KB	12.53	0.09
TimeUpdate.WaitForLastPresentationAndUpdateTime	31.2%	0.0%	1	0 B	7.86	0.00
Camera.Render	8.8%	0.1%	1	0 B	2.23	0.03
Drawing	7.2%	0.0%	1	0 B	1.83	0.01
Culling	0.5%	0.0%	1	0 B	0.13	0.01
UpdateDepthTexture	0.4%	0.1%	1	0 B	0.12	0.04
CullResults.CreateSharedRendererScene	0.3%	0.0%	1	0 B	0.08	0.01
Camera.ImageEffects	0.0%	0.0%	1	0 B	0.01	0.00
DestroyCullResults	0.0%	0.0%	1	0 B	0.00	0.00
PrepareUpdateRendererBoundingVolumes	0.0%	0.0%	1	0 B	0.00	0.00
beforeCameraRender.Invoke	0.0%	0.0%	1	0 B	0.00	0.00
Camera.FireOnPreRender() [Invoke]	0.0%	0.0%	1	0 B	0.00	0.00
FinalizeUpdateRendererBoundingVolumes	0.0%	0.0%	1	0 B	0.00	0.00
Renderers.OncePerFrameUpdate	0.0%	0.0%	1	0 B	0.00	0.00
RenderLoop.CleanupNodeQueue	0.0%	0.0%	3	0 B	0.00	0.00
Particles.Sync	0.0%	0.0%	1	0 B	0.00	0.00
Update Lights	0.0%	0.0%	1	0 B	0.00	0.00
UpdateRendererBoundingVolumes	0.0%	0.0%	11	0 B	0.00	0.00
RenderTexture.SetActive	0.0%	0.0%	1	0 B	0.00	0.00
FixedUpdate.PhysicsFixedUpdate	2.2%	0.0%	1	0 B	0.57	0.00
Physics.Processing	1.6%	0.7%	1	0 B	0.42	0.18
TransformChangedDispatch	0.3%	0.2%	1	0 B	0.08	0.06
Physics.FetchResults	0.1%	0.1%	1	0 B	0.03	0.03
Physics.UpdateBodies	0.0%	0.0%	1	0 B	0.00	0.00
Physics.Simulate	0.0%	0.0%	1	0 B	0.00	0.00
Physics.ProcessReports	0.0%	0.0%	1	0 B	0.00	0.00
WaitForJobGroupID	0.0%	0.0%	1	0 B	0.00	0.00

With no noticeable performance loss between one and fifty AI agents, and the program running well within acceptable parameters, I would say there is no reason to investigate this further.

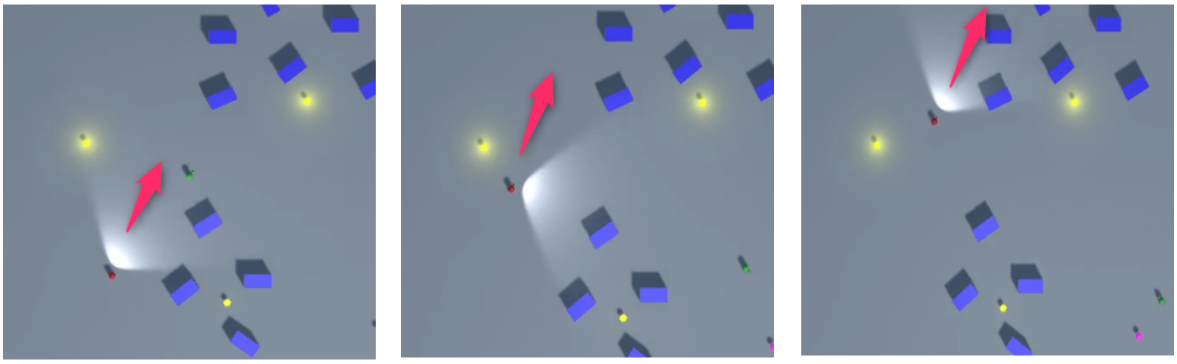
WHAT WORKED.

VISION SYSTEM.

I was very happy with the vision system as it went further than just a raycast with the player being seen or not. The idea was to deliver a list of everything the eyes could see to a logic script that then processed this visual data. A player could be immediately seen by the agent but not yet recognised. Recognition was based on a suspicion score which was accrued from the amount of consecutive time a player was seen as well as what distance they were from the agent. The score also had a cooldown so the player could almost be recognised but then return to no interest after a period of time. For the distance check, I used the formula

(viewMaxDistance - playerdistance) / (viewMaxDistance - viewMaxDistance * 0.5f) * distanceSuspicionMultiplyer

The maximum recognition power would be the same from the position of the agent until 50% of the max viewing distance. From there it would roll off to eventually equal 0 by the time it reached the maximum distance value set in the inspector. The agent would also point its searchlight at the player once the suspicion score reached a certain value. This meant that even though the agent could be heading in a direction away from the player, the eyes were still sending visual data to be processed and the suspicion score could potentially rise. The end result looked as though the agent took interest in the player before giving chase or losing interest and moving on.



Furthermore, this would handle any number of players as once a player was spotted they were put into a list that was constantly checked for suspicion score, last known position, vector3 heading, speed etc.

HEARING

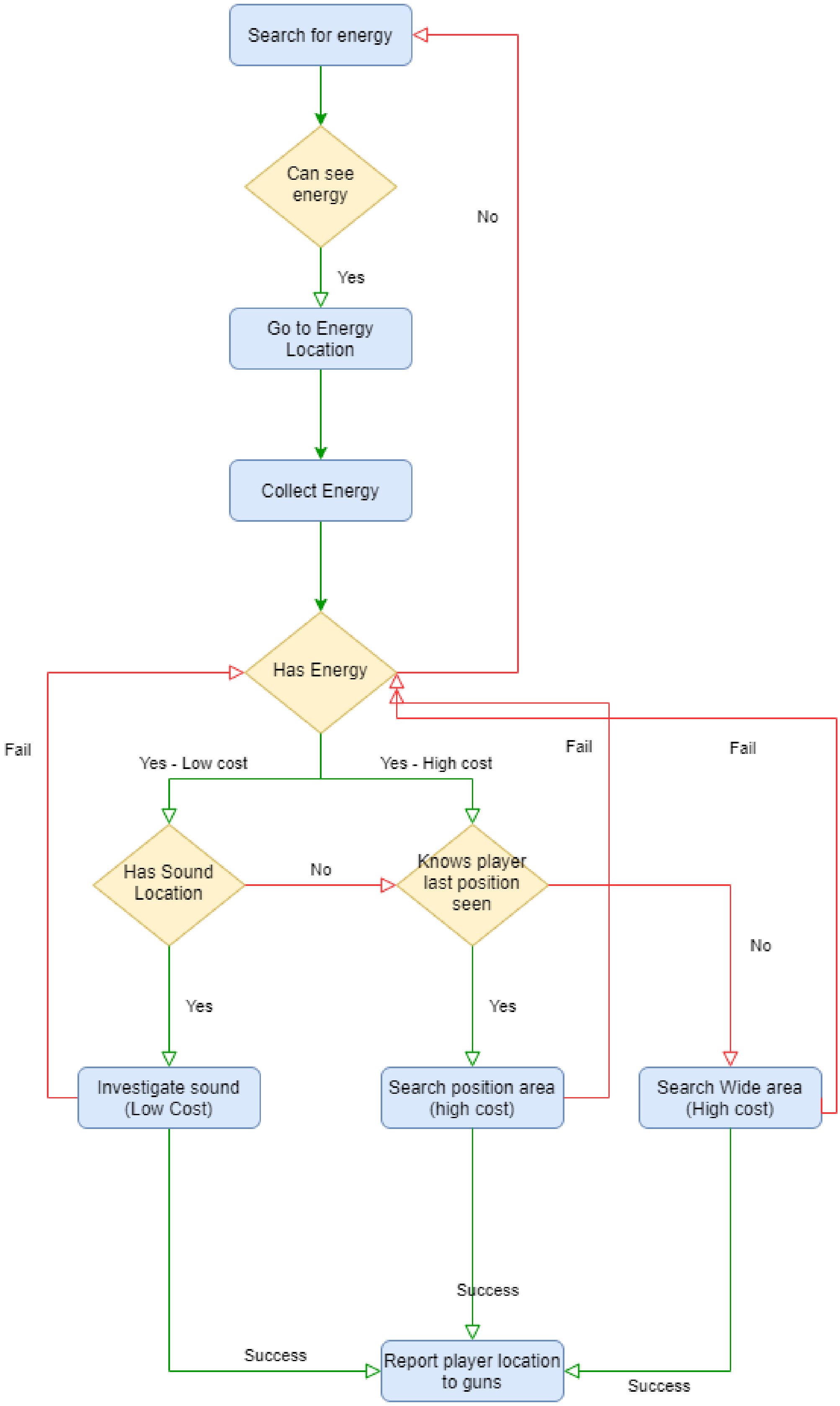
The agent’s hearing was a Vector3.Distance() check from where the sound event occurred. If the player was travelling less than a chosen speed, they would not be emitting any sound. Once travelling faster, a sound event was called and any agent that was in earshot would respond by travelling to that location. This was a simple implementation however felt smart enough when it occurred. If the player stopped making noise and sneaked away from the area, the agent would then search closely in the immediate vicinity making it feel like an intelligent reaction.

SEARCHING CLOSE

Once a known last location was acquired, the agent would generate a list of all known game objects that could be used as cover and systematically search each position. This did mean that a player could sneak away and leave the agent searching the area which felt like an accomplishment if they were able to do so without creating noise.

REACTION TO SOUND POSITION OVER LAST KNOWN POSITION IN GOAP

Using the costing on some of the pathways, the agent would abandon searches in favour of sound if the hearing system detected movement. Once the position was investigated, the agent would revert back to the previous method. Originally the agent would never react to sound until I managed to grasp the costing concept and how that related to pathfinding in the GOAP system.

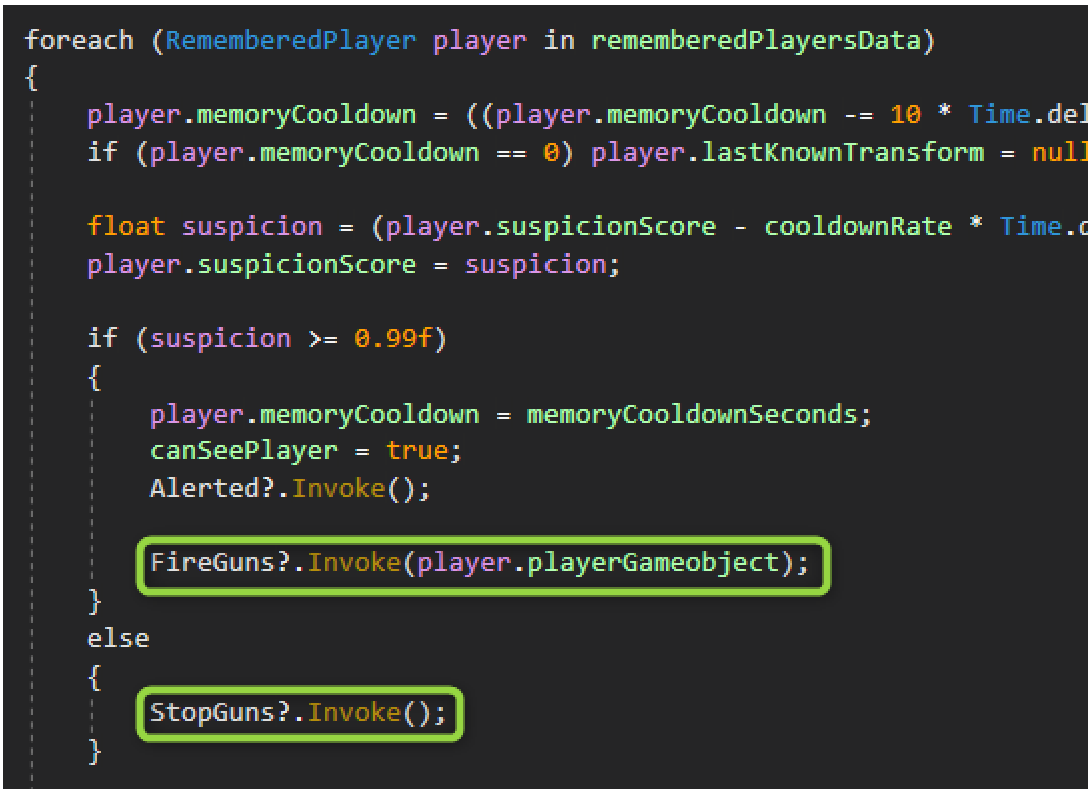


Flow chart of GOAP pathways

.

WHAT DID NOT WORK

Once the agent reached the state of reporting the player's position to the guns there was a problem with the way this was handled so that if there were multiple agents, the gun would not fire correctly.



```
foreach (RememberedPlayer player in rememberedPlayersData)
{
    player.memoryCooldown = ((player.memoryCooldown -= 10 * Time.del
    if (player.memoryCooldown == 0) player.lastKnownTransform = null

    float suspicion = (player.suspicionScore - cooldownRate * Time.d
    player.suspicionScore = suspicion;

    if (suspicion >= 0.99f)
    {
        player.memoryCooldown = memoryCooldownSeconds;
        canSeePlayer = true;
        Alerted?.Invoke();

        FireGuns?.Invoke(player.playerGameobject);
    }
    else
    {
        StopGuns?.Invoke();
    }
```

While coding for a single agent, there was an if-else statement, where the event was triggered to fire at the player or to stop the firing. This became problematic when multiple agents were used as almost all of them were creating the event to stop the guns when only a few were calling for them to fire. This would have been better without the StopGuns event, and a different method to handle the multiple agents in the gun script.

CODE ARCHITECTURE

Throughout GAM456 and 457, I have been slowly learning better coding practices however, I am still finding the architecture quite challenging. There are methods that seem to be in one script that requires to be used and accessed from others. A smarter approach would need to be researched in order to minimise this problem. At present, it is of minor consequence although has the potential to be problematic as the programming becomes more complex. Using namespaces and having a method library may be one solution to this problem as currently, I see this as the biggest barrier moving forward in future projects.