**PROJECT PROTOTYPE**

**3102GFS – Advanced Game Development**

**Assignment 1 – Research Blog**

**3rd October 2017**

**Dylan Lee – s5045023**

The goal of Project Prototype is to explore an area or areas of advanced game development. There are no restrictions on to what software, program, or language the code can be written in. The prototype will be presented on the 14th of August, it must demonstrate a clear concept and approach as well as present potential for full development. This blog will outline the research, methods and processes taken during the development of the prototype elaborating on the challenges presented and how they were overcome.

**Session 1 – Multiple Targets**

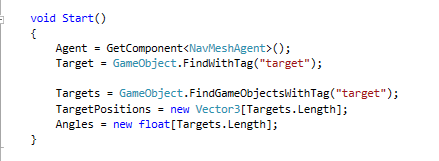
Something that hadn’t been previously tested is what the AI would do if there was more than one target within its sight. This was tested and revealed large problems with the AI, it paths very incorrectly and becomes inaccurate. Looking into the script for its targeting, the AI was using the position of a singular public game object as its new position when it had spotted a target. This needed to be changed to set its position to the target its looking toward at any given time.

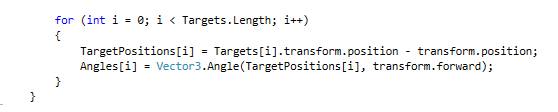
This proved to be very difficult, the first approach to solving this problem was to set the destination to the hit point of the raycast. While this would work for one target, this didn’t fix the problem for multiple targets. This revealed that similar to setting the AI’s new position, the Vector that was being used for the AI’s sight was using the same singular target. This meant that no matter how many targets in the scene, the AI was only going to be able to look at one.

To combat this issue, the original vector and float used to create the AI’s vision were edited to try and include multiple target positions. To do this, arrays were created to try and incorporate them into the vector and float. After many failed attempts at different combinations of array, vector and float code, time was taken to deconstruct the problem.

There needed to be a way of tracking how many targets were in the scene, where the targets were at any time and the angle from the AI to that target. The best way to keep track of multiple items is through arrays. The goal from this stage was to create arrays containing this information that could then be utilized later to check which target the AI can see and should target. As there were three different things that needed to be tracked, three arrays were created. The TargetPositions array and the Angles array were filled with a for loop using the Targets array.



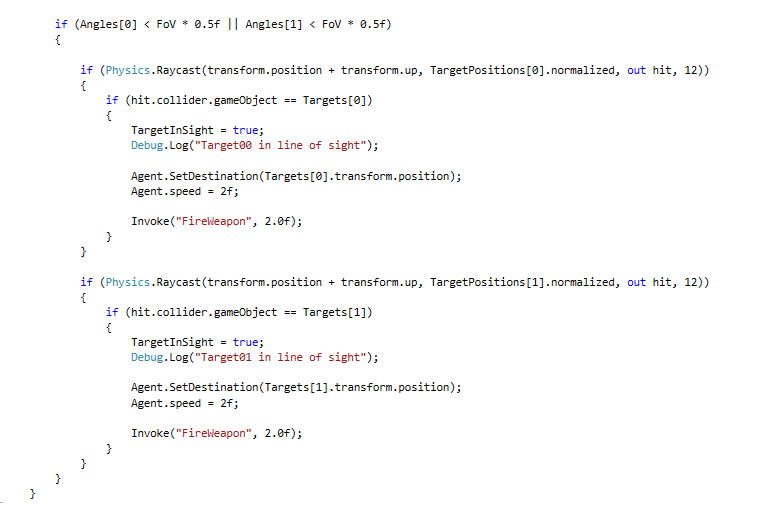




Now that the script could track all of the information needed to create a FoV (Field of View), the next challenge was checking every angle and efficiently creating raycasts for every target instead of just the one. First the if statement that checked the angle of the FoV needed to be corrected for multiple targets. To do this an ‘or’ was added to the if statement that checked both angles currently in the array.



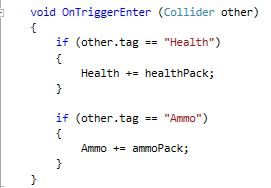
From this stage, the idea was to create a raycast that would point to each object, after attempting this method many times, it was discovered that a single raycast couldn’t point to multiple objects unless the function RaycastAll was utilized. Research done into the use of RaycastAll didn’t prove helpful and wasn’t going to work for what this project needed. The goal was to create one neat block of code that created all of the rays, told the AI if it could see or was in range of them and fire its weapon. It was eventually learned that this was either not possible or incredibly difficult. To achieve the desired multiple targeting system, under the if statement checking the angle, there would be a raycast for each target.



While this script worked as intended, it was large and bulky. A goal for future development is to condense this down into as little code as possible. Feedback from tutor Gordon Moyes and peers during the tutorial lesson following the date of *Session 1 – Multiple targeting* will be helpful in achieving this.

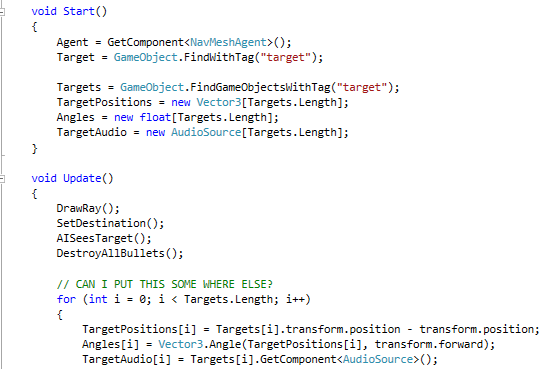
**Session 2 – Ammo & Health packs**

The previous implementation of the health and ammo packs didn’t give the AI any ammunition or health, while a minor update, this was essential to the AI functioning properly. Two new floats were created for health and ammunition, healthPack and ammoPack. The ammunition and health objects were converted to triggers and given separate tags, when the AI enters a collider, it checks for the tag of the collider then adds health or ammo accordingly.

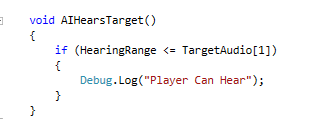


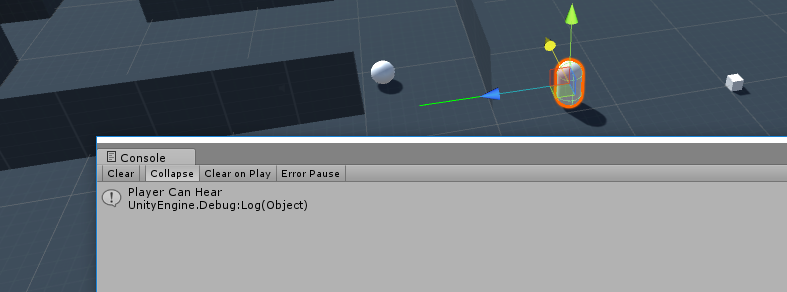
The next goal for the AI was to implement a hearing system, research was done into *Unity’s* audio component to learn what variables we could use. It was learned audio in *Unity* has a maxDistance variable, the distance in which the player or object can hear the audio source from. Attempts at using this variable for the range in which the AI can hear did not yield successful. At this stage, the idea was to use a spherical trigger, and if the player was inside, the player can hear. While this would work, the problem with this is that the bullets the AI fires destroy objects they collide with, since the trigger would be on the same object, there was no known way around this.

Further research was done and a new method was thought of, create an array with all of the targets audio sources in them and check the AI’s distance from the audio source. A new array for audio sources was created and using the for loop already implemented into the project, the audio scene of each target was collected and placed into the array.

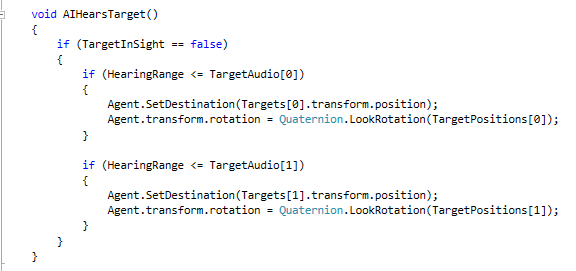


Once this array was implemented, it was then thought that if the audio sources are at the centre of the targets, why does the AI need to track the audio sources, why not just create another set of vectors. The array used to create the vectors for the field of view was copied and renamed for the AI’s hearing. A float was set up to be the range at which the AI could hear and a function was written that checks that distance against the new vector, if it’s the same or smaller, the AI is close enough to hear.





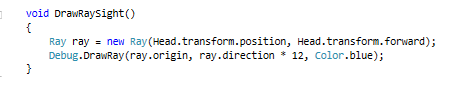
The goal from here was to get the AI to not only move to the location from which it heard the target, but to begin looking in that direction, prepared to fire. Within this new function, the AIs’ destination was set which revealed a problem. If the AI could both see and hear the target, the AIs’ pathing and rotation completely breaks. To fix this, a Boolean already implemented into the script was used to check if the AI could see the target, if not, then continue. The next goal was to get the AI to look at the direction of the target if it could hear it, even if it was behind an obstacle. Research was done into the best way to face the AI in one direction but move another.

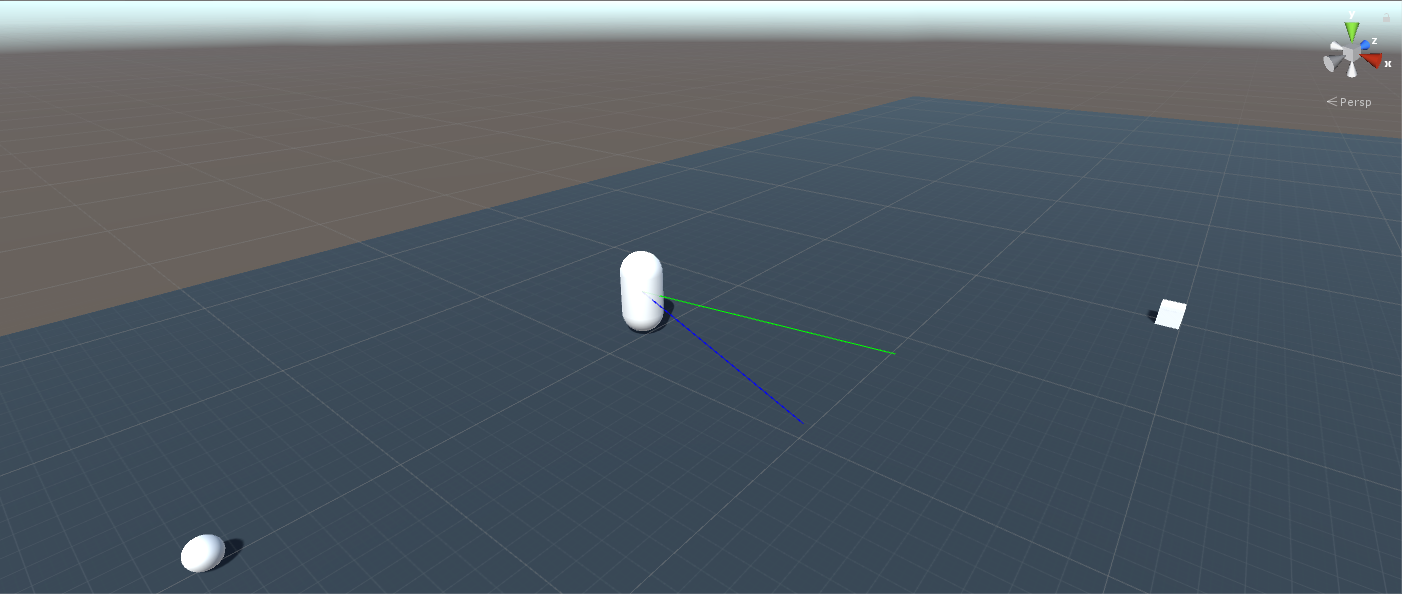


The script above is the final version of code for this period of work, while successful at turning the AI toward the target while it moved toward its position, there was a problem. The AI cannot look one way and move another, its movement and look rotation combat each other causing the AIs’ face direction to flick back and forth as it continued to move. At this point in time, the planned fix for this is to create a separate head and body for the AI, so it can look around smoothly while maintaining its pathing.   
  
**Session 3 – Look Direction & Head**

The goal for this session was to implement an actual look direction for the AI to go along with its vision, the point of this was to have the AI look in one direction and move in another if it could hear something. To do this, a new object was created and parented to the AI’s body, the plan was to rotate this object in the direction the AI was facing. This proved to be a very difficult task, time taken on this was approximately 5 hours.

Firstly, a raycast was written to project forwards from the head object so the rotation could be followed in the editor to help better understand what the code was doing.





The first attempt at rotating the head object was done by trying to set the rotation location to the agents’ destination using Agent.destination. This didn’t work, the rotation of the head object would change but it was always off a large distance, the cause for this was never understood. To combat this problem, multiple different methods were experimented with over the course of the session that were unsuccessful. Some of these methods included trying to create new Vectors or Transforms for the head to rotate to and an attempt to write a function that returned the agents destination as a transform and using that for the rotation.

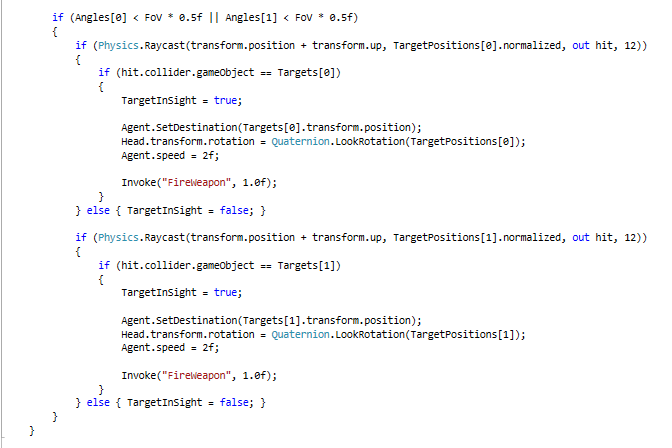
The next option was to rotate the head every time the AI’s destination was set, this resulted in multiple lines of the same code, unique to each possible destination.



This implementation of head rotation worked to some degree, it worked for the first target but wouldn’t continue to update beyond that point. The cause for this was the clash between hearing and seeing a target in the script. To solve the problem, Booleans were used to tell the AI more accurately whether it could see or just hear a target and update the rotation of the head object accordingly. Before reaching this solution, multiple attempts at fixing this took place. Some of these involved adjusting the if statements for the firing and hearing, and rotating the head with in its own function, all of which were unsuccessful.

At one point during this session, the head rotation broke all together, the cause for this was unknown. Reloading an old version of the script was required to fix the problem.



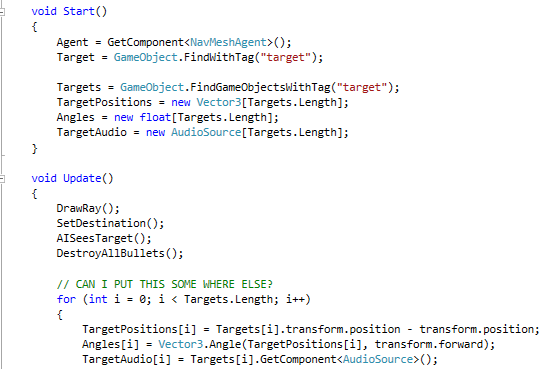


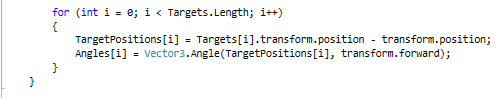
The rotation configuration was added for when the AI was heading towards a health or ammunition pack. There is currently a small problem with this code however, the AI’s head will rotate for the health pack but not for the ammunition. The cause of this is currently unknown and will require attention during another session.



There were some other minor changes that added to the quality of the AI’s performance. Previously if the AI ran out of ammo but could see a target, it would stay locked onto the target. An if statement was added to the AI that checked the ammo count, if the AI didn’t have any ammo, return was called to prevent the AI from moving to the target. The same thing happened with the AI’s hearing, the same solution was applied in the hearing area of the script.

It was discovered that there was an array the AI was using that was the exact same as another array, the second array was deleted and the script that utilized the old array was adjusted to use the values from the first.



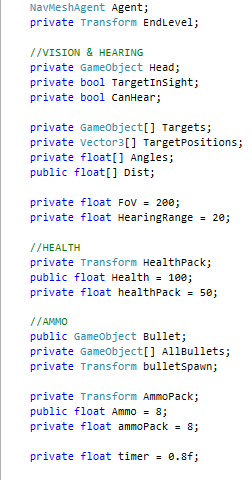


The planned development for this script will be to look over everything written and identifying where code can be more efficient. A full obstacle course will be designed and implemented into the scene with enemy targets that shoot and move around to see how the AI behaves in a full training course simulation.

**Session 4 – Tidy up, Hearing & Head Rotation Optimization**

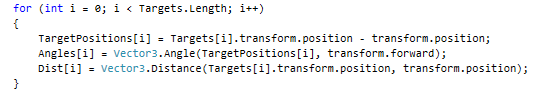
During this session, initial work began on cleaning up code, converting all public variables that should be private was done as well as categorizing them for easy navigation. This was also done for the start function.



In addition to this all the functions were moved around following a similar method, being placed in the order they were called during each update otherwise being placed somewhere relative to its purpose.

Further work on this didn’t continue as to whether or not there was an industry standard method or if it was personal preference within reason was unsure. A small tidy was done for now and questions regarding this would be asked during the next tutorial session.

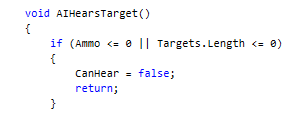
A large optimization in terms of quality was made to the AI’s hearing capabilities, previously the AI was using an array of vector angle, which would change dramatically if the AI rotated, causing problems. A new float array was created that used vector distance rather than angle, providing a much steadier value to check against for hearing. This prevented the AI from turning back and forth awkwardly as it made its way through the space.



The rotation for the AI’s head to indicate where he was looking rather than facing was updated and optimized greatly, instead of calling a head rotation every time the AI’s target changed. A method for rotating the head to the AI’s destination was discovered and implemented.

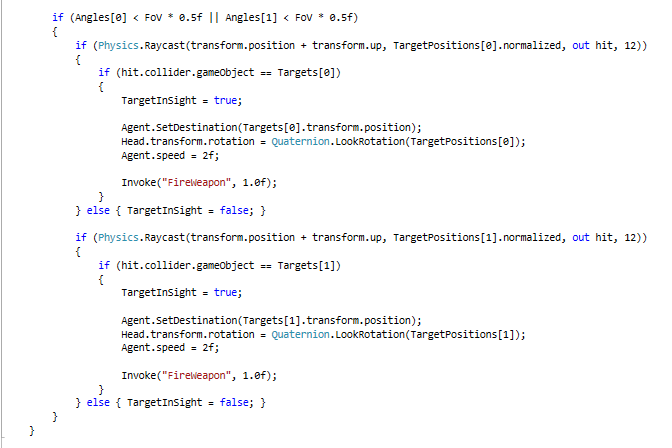
Head.transform.LookAt(Agent.destination);

This line was called during the update function, this worked smoothly up until all the targets in the scene were destroyed. If there weren’t any targets in the scene, the AI’s head rotation set up stopped working all together. This was due to the script still trying to access members of an array that weren’t there. To counter this problem, a condition was added before any of the targeting and hearing code that checked the number of targets in the array, if there were none, return.

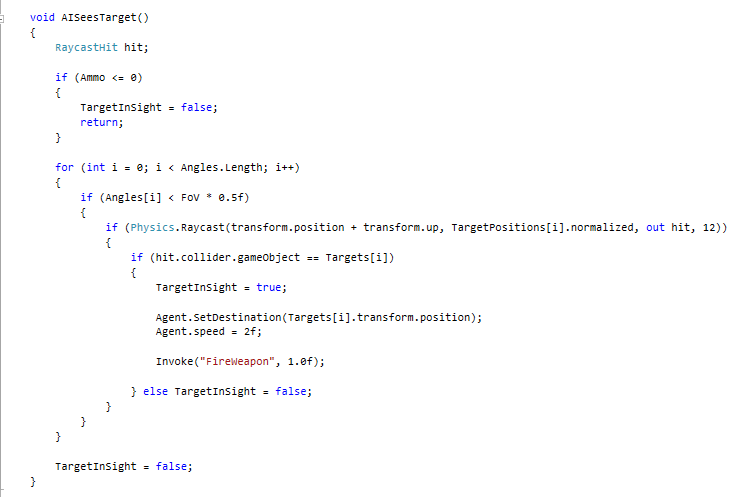


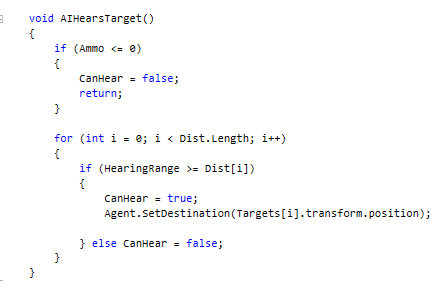
**Session 5 – Hearing and Sight Optimization**

A big problem for the code in its current state was that for each target in the scene, a new condition needed to be written for each target in regard to if the AI could hear and see that target, this took up a lot of space and resulted in the same code been written over and over with only the targets array index changing.

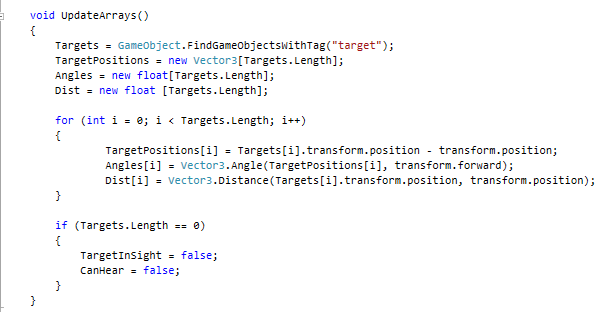


This was updated and re written in the form of a loop that only needed to be written once and would work for any number of targets, each specific target didn’t need to be specified, just the array itself. This same method was applied to both checking if the AI could see the target, and if the AI could hear a target.



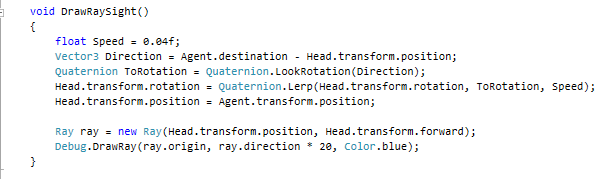


In addition to this, all of the arrays involved with the hearing and sigh script were moved into their own function and called during the update function instead of the start function. The reason for this is so that when a target is destroyed, the array members are removed and not left null or at the values they had when the target died. This was causing problems with the hearing and sight conditions. For example, if the script was checking the distance to a target, and the target died but the distance array wasn’t updated, that distance value stayed at whatever value it was when the target died, meaning a condition using that value would stay true.



There was still a problem every time the last target in the scene died and the Booleans for hearing and sight wouldn’t change back to false, so a condition was added to the array function as a backup. If all the targets were dead, turn both to false.

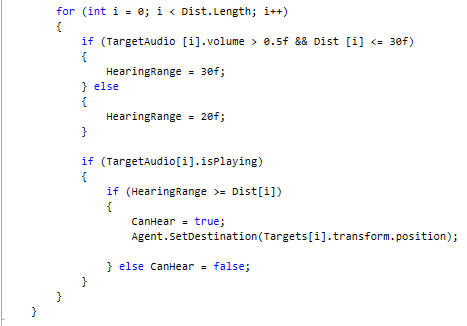
The head rotation code was updated to be a little more lifelike or realistic if the AI was a human character or something similar. It was edited to rotate over to any new position rather than just change instantly and was moved into the same function that drew the array for hearing.



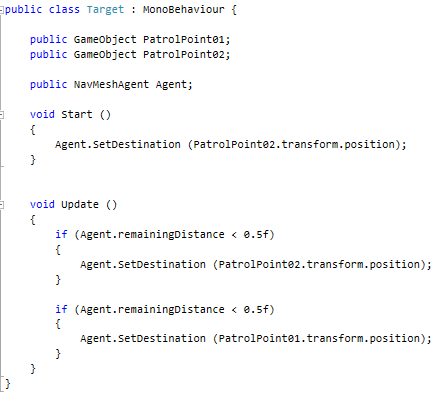
The Head object was unparented from the body in the scene and scripted to follow the body itself to avoid over rotation when the head and body both rotated at the same time.

**Session 6 – Hearing Audio Sources & Target Movement**

To make the hearing more realistic and work on actual in game sound, Audio Sources were added to the enemy targets, given an audio clip and placed into an array. An if condition was added to the hearing function to check if the audio source was playing. In addition to this, a for loop was written that checks the volume of all the audio sources in the scene, if the volume is loud enough, the target can be heard from a further distance.

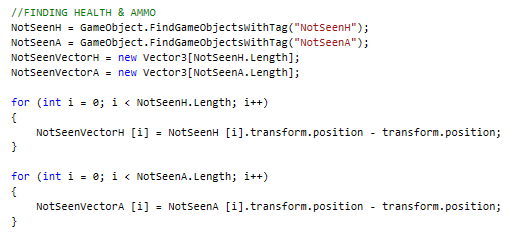


To better test the AI, a target was set up to pace back and forth between two patrol points. The target was given the nav mesh agent component so that it could move around obstacles while still only having the two patrol points. Using nav mesh agent, the target is given a destination, each frame a condition checks the remaining distance between it and the patrol point. If that distance is small enough, set the destination to the opposite patrol point.

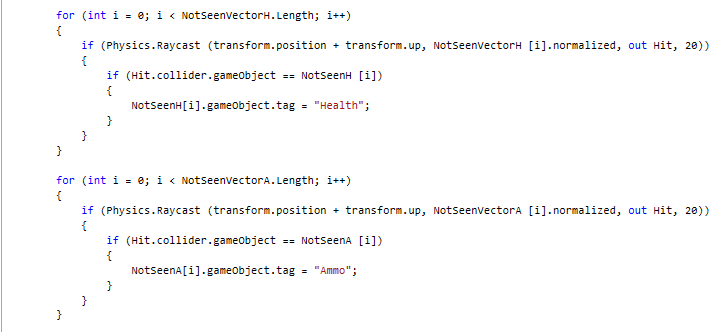


**Session 7 – Remembering Health & Ammo location**

The goal for this session was to get the AI to see and remember the locations of health and ammo packs rather than the AI knowing that from the start. Firstly, health and ammo packs that hadn’t been seen by the AI needed to be distinguished from the ones it had. This was done using tags, a not seen tag was created for both the health and ammunition. An object array was created for each and filled via the tags, then using a for loop and the object arrays, vector arrays were also created for each.

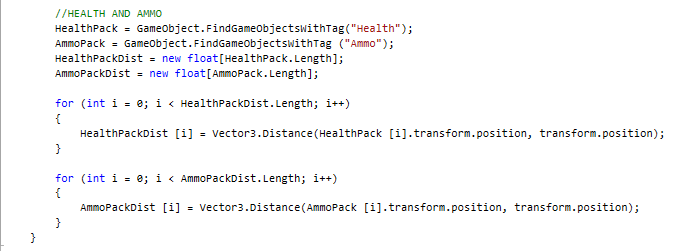


Similar to the AIs vision of targets, these vectors were used with raycasts to check if the AI can see any packs. A raycast is sent out from the AI to the corresponding vector, if it hits an object with the not seen tag then the AI can see that pack. Once the pack has been seen, the tag of that pack is changed to Health or Ammo accordingly.

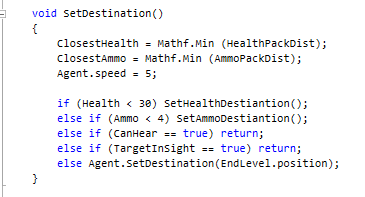


A lot of time here was spent trying to combine the two vector arrays into one so only one for loop was needed, all research lead to the only solutions being in Java Script.

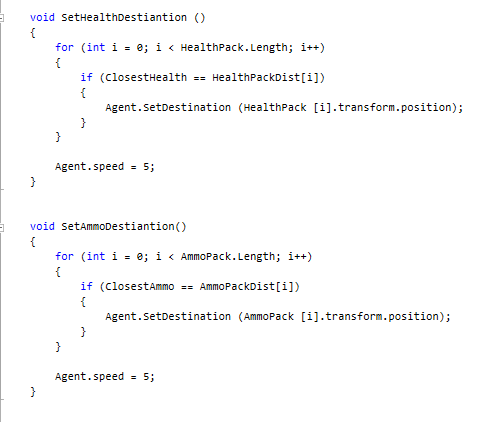
Once the tag of a health or ammo pack has been changed, it is moved into one of two arrays via the new tag. Utilizing these new arrays, floats representing the distance between the AI and the pack are created.



A new float for the closest health and ammo is created using the smallest of the distance floats.

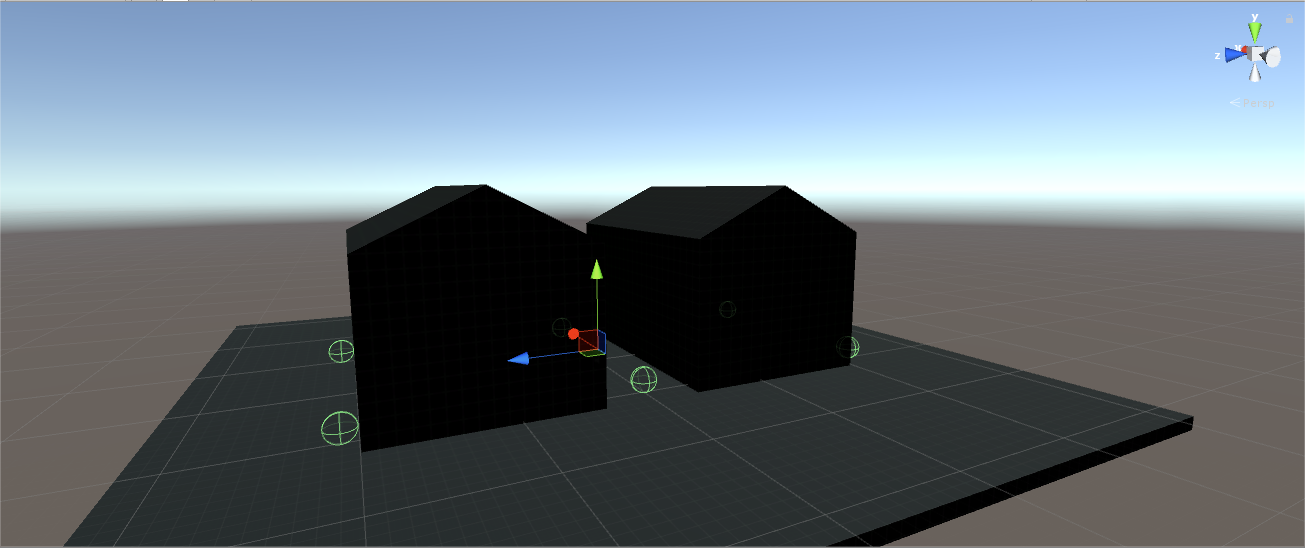


The smallest float is then checked against all the other distances in the distance array. It checks for which member of the array the closest pack float is equal to and then sends the AI to that one.



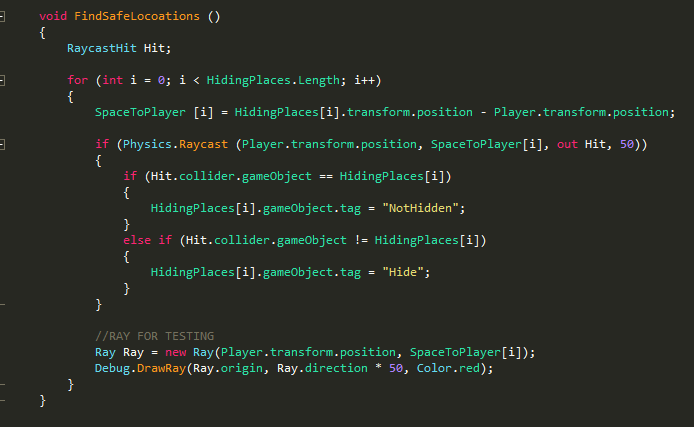
**Session 8 – New Hiding AI**

The goal was to create an AI that once it has been spotted by a player, it runs away and finds a location in which the player can no longer see it. A scene was created with 2 structures in the centre, 6 different empty game objects or Transforms were created and placed around and in between these structures as hiding places.



An FPSContoller from the assets store was imported to be the player and a sphere was created to be the AI. The AI would navigate the scene via the Nav Mesh baked into the scene and the Nav Mesh Agent component.

The first thing the AI needed to know was the location of all the hiding places, an array of game object was created containing all the hiding places via a tag “Hide”. Then the AI needed to know which places were hidden from the player. To do this, a Vector3 array (SpaceToPlayer) was created for all the hiding places in relation to the player. This was used to create a Raycast from the player to each hiding location. If that raycast was hitting any of the hiding places, the tag of that hiding place was changed to “NotHidden”, else if the ray was not hitting the hiding places, the tag remained as “Hide”.



Next, the AI needed to know if the player could see it or not, and if it could be seen, move to a location that the player couldn’t see it at. To do this, a Vector was created for the AI and player and used in a Raycast similar to the raycasts for the player to hiding places. A condition was first checked to see if the Raycast hit any objects tagged with “Obstruction”, this tag was given to the structures placed in the centre of the space. If the Raycast did collide with them, the code would stop running using return as there is something blocking line of sight from the player to the AI.

If the raycast wasn’t hitting an obstruction the AI’s destination was set to the furthest hidden location from the player. To do this, a function was used to create an array of distance Vector 3’s, SpaceDistances, from the safe locations to the player. The safe locations were an array filled with objects tagged with “Hide” which is updated every frame. A float (FurthestSpace) was created which is always equal to the largest value in the SpaceDistances array.

Back in the Raycast, a for loop was used to iterate through all the safe spaces, whichever distance from a safe space to the player was equal to the FurthestSpace float, set the agents or AI’s destination to that space.



Variables, Start & Update



Moving into further development, there a few optimizations for this AI that could be made. In all situations the AI will always run around the outside of the 2 structures, even if the player is standing on the corner. This means that the AI sits in view of the player, when it could turn through the middle to leave line of sight and take that path to the furthest safe space.

**Session 9 – Hiding AI improvements**

During this session there were many attempts to try and get the AI find the closest hiding location and go there first but then continue to move to the furthest. This proved very difficult and all attempts at this were un successful.

The closest the AI got to the desired behaviour was a very small adjustment, reducing the AI’s speed to 0 if the player couldn’t see it. Causing the AI to run away until it was unseen.



Depending which angle the player comes at the AI from, the AI works perfectly and will run in the opposite direction and hide from the player. Other times and less often, it may run past the players position which is sub optimal. Given more time on the Hiding AI, further improvements could be made to minor aspects of its behaviour.