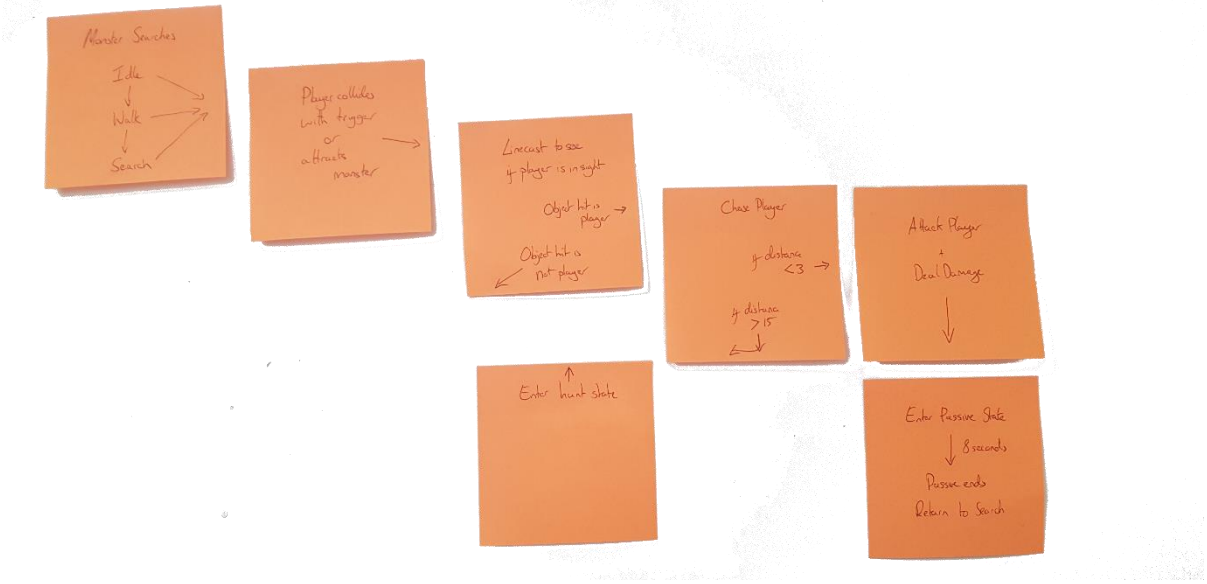
It Will Find You – Monster AI:

Goal: Create a realistic AI for the creature antagonist of It Will Find You, giving it the ability to interact with objects in the world, and make decisions based on player choices and playstyle.

Core Functionality: As a core element of both the narrative and gameplay, it was important to ensure the Monster felt like a living entity to the player, as oppose to the generic basic AI you see in a lot of horror games.

The foundation of this AI is a string-based state system built around the Nav Mesh Agent, that will allow the monster to cycle through different states depending on what scenario it finds itself in. These states can also be referenced and called by external scripts, allowing the Master Event System to influence the monster's behaviour as a secondary AI brain.

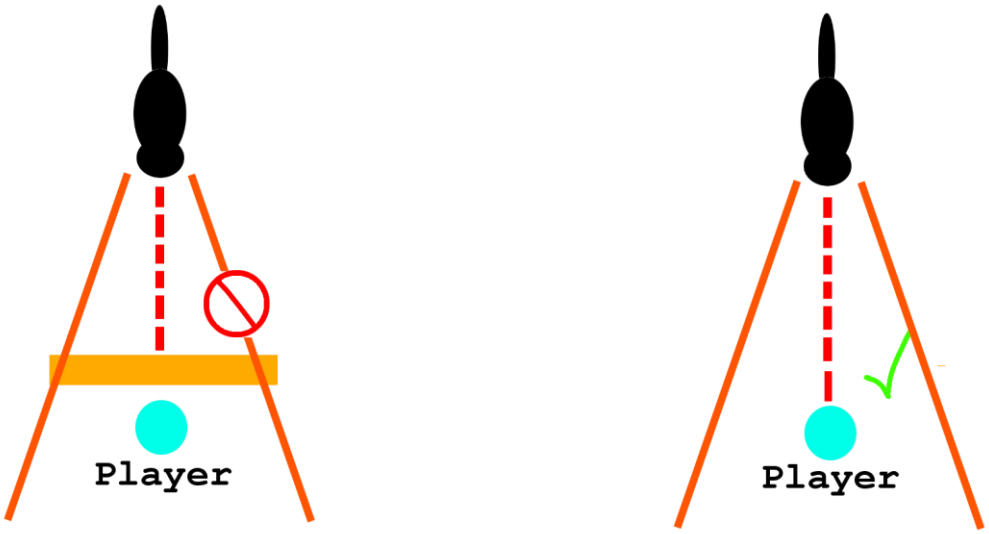


At the core of this system was a Nav Mesh Agent (NMA) attached to the creature, with an adjustable radius and height that could be altered as development continued to ensure the creature had free mobility throughout all environments. The NMA would then have its behaviour determined by the string state machine or external factors like collisions and behave accordingly. This presented a good base to build the AI on, as the inbuilt tools of the NMA and Unity's navigation system meant the creature could navigate the space consistently without too many additional calculations needing to be made to compensate for tight spaces, with the baked mesh often providing all the navigation detail needed.

With this in place I began to create the core cycle the monster would work through during its active phase whilst hunting. The idea was to have an AI loop that it would move through, step by step, whilst exploring its environment that could then be deviated from when exceptions, like contact with the player, occurred. This took the form of an idle phase, a movement phase and a search phase. In the idle phase the creature would pick a random point inside of a unit sphere centred on itself, then set this as the navigation point. The movement phase would then set the NMA speed so that the creature could begin walking towards the selected point, and upon arrival it would search the spot for the player, rotating 360 degrees before returning

to idle, choosing another point and repeating the process if the player was not detected.

This loop was a simple but effective base on which to begin expanding, and with the creature able to move, it was time to give it the ability to chase and catch the player. Using a cone build in Pro Builder12 I created a trigger to act as a base for the monster's vision. This gave the monster a wide and realistic area in front of it which, if entered, would trigger a checksight function that could determine if the creature could see the player. The checks involved were designed to ensure that the creature could see the player directly, and that they were not hidden behind a wall or object, and so upon calling checksight a would be cast from the monster to the player. If the ray intersected any other objects before hitting the player, checksight would come back as false and the monster would be unable to see the player. If the ray hit the player, then the monster would enter its 'hunting' phase. If the player remained within the trigger, checksight would continue being called until either the player or monster left the area.



Entering into the hunting phase the creature sets the NMA destination as the player's position, and increases the NMA speed to allow it to catch up to the player that would now likely be on the run. A Vector3.Distance is then used to detect how far the creature is from the player. If the player manages to put enough distance between the creature and the themselves, the hunt is cancelled, and the creature returns to the idle state with a new alert function.

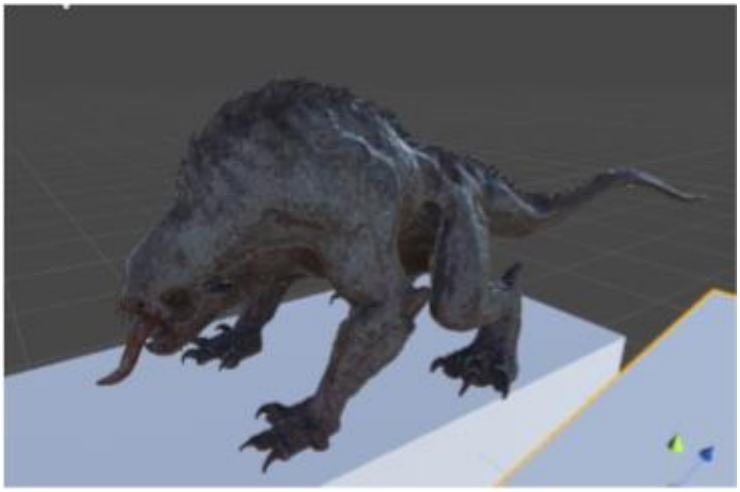
The alert function begins a countdown that determines how long the monster will now search for the player having made contact, with additional clues expanding the search time, and contact refreshing it completely. During this function the monster's speed remains slightly higher than average, and it sets it's NMA destination points based around a random unit sphere centred on the player, so as to mimic the action of tracking.

If the player does not manage to escape, and the distance between both objects is small enough, the creature triggers an attack. To do this he first disables the players character controller so they cannot move, then triggers the camera animator to Slerp to a predesignated kill point attached to the monster. A specific animation is then selected based

on the player's current health, and an animation for both the player camera and the creature would then play, lowering the players health in the process.

Once the animation is complete, the monster directs the player camera to reset, and enters a "Gloating" state where it cannot attack, and essentially rests for 8 seconds before making its next move, allowing the player to escape. Initially this gloating phase did not exist, but escaping the creature became too difficult and he would often take 3 levels of health in short succession causing game over. Introducing the cooldown period also improved realism, as it matched the earlier mentioned behaviour of an apex predator, with the creature landing a blow, then letting its prey run itself out while it stalked them from a distance, making the kill easier.

The AI could now navigate by itself and was capable of catching and killing the player to a limited degree. This became a solid foundation upon which I could develop more advanced behaviour, turning the creature from a simple robot into a more living, breathing predator with its own intelligence which would add to the immersive, and realistic quality of the game.



I discovered during early testing sessions that the monster would use any path it deemed available to get to the player once it had caught sight of them, and this often led to it becoming stuck when it attempted to reach the upper floors of the house by climbing through the window. I also wanted to monster to have a limited duration it could stay within the house to ensure the player would have time to explore after successfully evading the creature, and this meant working out a way to limit what areas the NMA could travel to.

I successfully limited the NMA's movement using Binary values to dictate which Nav Mesh Areas the monster could choose for its waypoint, creating a movement loop that could only progress when the chosen point was not on one of the forbidden meshes. Having never used binary this solution required research and time to apply correctly, but was extremely effective at helping to manage the creatures movements, simply listing the building and roof, which was made accessible through off-mesh links that allowed the NMA to jump as different Nav Mesh Areas that could then be excluded.

The NMA binary enhancement was then supplemented with the addition of a 'hunt timer' that was activated whenever the monster entered the house, with a random range of 2-3 minutes. Whilst the monster is inside the countdown timer activates, with the value increased if the monster encountered the player. If the counter hit 0 then the creature would be forced to leave the house and be unable to use nav waypoints inside the building for a set amount of time, with the exception of specific events, or if it encountered the player.

Interacting with Doors:

As the creature had to be able to enter the house, and is supposed to have a basic level of intelligence, the next step was to give it the ability to open doors. This was done by mimicking the way the player interacted with the doors, creating a Raycast from the creature that could detect if a door was in front of it, and then subsequently open it if was blocking the monster's path.

Initially, I created a system using Inverse Kinematics that would allow the limbs to specifically target the door handle depending on what side of the door it was on, physically interacting and turning it to open the door the creature would then pass through. This was then embedded into a coroutine where the creature would stop, perform the animation, and then continue through the door once the door had opened. The inverse kinematic method was complicated but visually effective as it made the creature interact directly with the world. It was disappointing that following the change in monster model that this was no longer possible.

Whilst this method of opening doors worked fine in the early versions of the game, as doors became more complex with the addition of locks, this system required expansion and diversification. Initially my idea was to apply Nav Mesh Obstacles to each door and have them turn on when the door was locked, thereby excluding the doorway as a path the monster could take. The problem with this was that the creature would then never try to navigate to that area or go anywhere near the door, since it did not recognise it as a viable path, and so the player could not only seal the monster completely out of the house, but also lock it into rooms.

At this point the advantage of having a more bestial creature became apparent, and the idea to give the monster the ability to knock down doors began to develop. To rectify the Nav Mesh Obstacle issue, I removed the ability to trigger it from the locked state, and instead created a door check system within this monsters AI. This meant that the monster would navigate towards doors again, and then perform an action based on the doors state.

The system would then play out in the following manner:

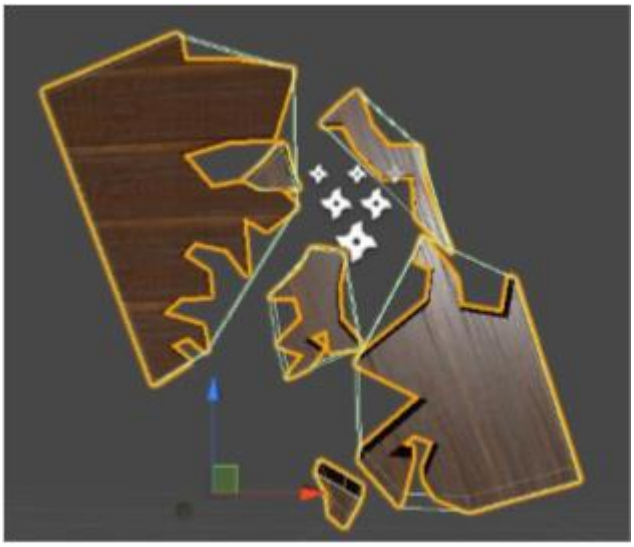
1. The creature arrives at a door
2. If it is unlocked, the creature opens the door, if locked, the creature begins to listen for the player.
3. If the player is within a set distance, and is not holding their breath, then the creature will destroy the door and begin a hunt.
4. If the player is not within a set distance, or is holding their breath, then the door will turn on its Nav Mesh Obstacle for 10 seconds, blocking the path and forcing the monster to find an alternate route.

This system worked well and gave the creature a much more intelligent approach to making its way through the house, while also making the doors a more tactical element of gameplay,

as unnecessarily locking them or failing to hide properly now resulted in the permanent loss of a defence.

This did not entirely fix the problem though, as the creature could still become trapped within a room, and so a dice roll was incorporated into the creature's door interactions. After the previous steps had been taken, instead of the Nav Mesh Obstacle activating, the creature chooses a random number between 0-100, and if the number was higher than an adjustable value it would destroy the door to continue its search.

This system then had a '3 fail chance' implemented to avoid the creature becoming trapped for too long, instantly destroying the door on the third attempt. If an attempt failed, the Nav Mesh Obstacle would then activate, and the creature would walk away to explore another area.



To create the door smashing effect for the creature bursting through, a 3D cast had to be created of each door using a simple cube. Pro builder12 was then used to make polygon shapes representing individual shards of door, and these were then collected into one object, with a particle effect added for fine debris. When the creature destroyed a door, it would instantiate these chunks and particles in the door frame, and then apply an explosive force emanating from the creature to propel the debris away from it, as if the door had just been blown open. This was effective as a base effect but will require further work to make truly realistic and immersive, with more detail applied to the individual shards.

Tracking and Relocation:

With the monster's new improved ability to pursue the player in a more immersive way, I had to ensure that the creature would be an omnipresent threat to the player to maintain tension and encourage progression. My solution to this came in two distinct components, a scent tracking system, and a distance based relocator.

The scent tracking system was inspired by a similar mechanic I had developed during a previous project, where an object would follow the exact path left by a player. In this instance I created an invisible collider that would follow the players path after a delay by creating an array of Vector3 points as the player moved which it could then replicate. This collider could then interact with the creature should they cross paths and trigger a "tracking" state which would allow the monster to follow the scent object until it was within a set distance of the player, at which point it would begin to search for the player in an alert state. This meant the

player had to be careful with how they moved around the creature and gave the monster another tool with which to track the player around the map. As the player became more injured or coated with other scent-based effects, the range of the collider could then increase, drawing the monster in from further afield and creating a more present sense of danger.

The distance based relocator was a failsafe to ensure that the creature was never so far from the player that it was no longer a threat. Using the Vector3.Distance method I had previously instituted in the AI, the monster would sense if it was too far from the players location, and then pick a random point around the players current location to move to. This allowed it to move to the players general location without directly moving to them, giving the sense it was tracking the player rather than simply adjusting to follow them.

As the monster's toolkit expanded, the player needed an option to defend themselves, especially at low health, and so a 'retreat' state was added for situations where the monster was injured by the player, either by being shot or by an explosion. This would simply turn off its ability to attack and force it to rapidly move to a location within the Unit Sphere that was at least 50 units away, giving the sense that it was fleeing to recoup itself and giving the player some breathing room.

The final addition to the monster's core behaviour was an aggression meter that operated in a similar manner to Julia's Agitation meter. As events unfold, and the player interacts with the creature, the monster's aggression increases, making it faster, better at tracking, and reducing the radius in which it will hunt for the player after a chase, making escape more difficult. The Aggression value also reduces the chance required to smash doors, making it harder to hide from, or delay the monster's passage within the house. The value was again increased by static values during certain events and increased over time during chases by a set value that ticked over for each second the creature was following the player. These values were adjusted according to tester feedback to ensure the creature did not become a source of frustration.

The aggression meter helped the creature to become a part of the reactive narrative, as its behaviour matches the intensity of the story, reacting to how the player treated it in a visual manner. Rather than it following a set of static rules, it allowed the creature to create more paths with the increased chance to smash doors, and makes it somewhat unpredictable to gauge how quickly it will find you. This makes later segments of the game much more intense as a result, especially when the player may be low on health.

Cinematic Events:

With the core AI for the creature finished, it was time to begin working on its involvement in specific scripted events. Initially I had planned to simply disable the AI creature and use a facsimile as an animated avatar to retain control over its movement, but this created static and restricted events that the player could have no impact on.

This made me reconsider my approach to many of the narrative events in the game, as it was unrealistic for the player to not be able to interrupt certain instances if they wished. For this

to work the creature had to be able to revert to its standard behaviour in an instance as the event changed direction, and so a facsimile would not work in this instance.

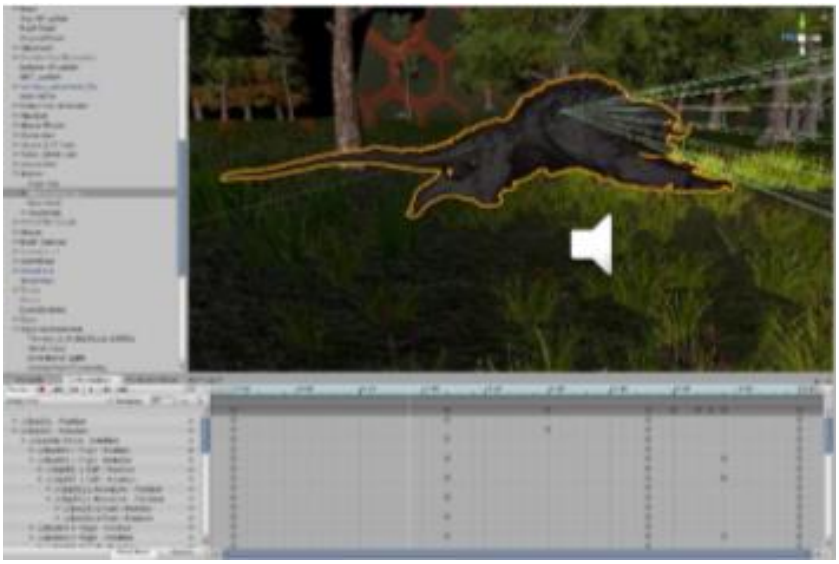
To allow interruptions I created a dummy object that could dictate the monster's movement, while the creature itself handled its own animations with a special inbuilt cinematic tree that could be triggered with a bool toggle. When an event activates, the monster can then have its active AI temporarily disabled and be forced to follow the dummy object exactly, while performing its own motions to make the event believable. Initially, I tried parenting the monster to these dummy objects, but it presented a host of issues, from disrupting the monster's animation due to Mecanims hierarchy not accepting parental swaps, to accidentally disabling the monster completely if the dummy object was turned off as part of the animation.

Instead, using the MES, the creature would have its position and rotation manually transformed each frame to match the sphere, allowing it to accurately follow the object, and be detached at a moment's notice if needed.

Interruption states were then created to allow an event to be halted by player action, either by crossing the path of the monster forcing a chase, or by shooting it and causing it to retreat. This would detach the monster from the dummy object, and re-activate its AI immediately allowing it to take appropriate action, while triggering the divergent event animation so that it could continue without it.

Whilst this method was not perfect, it made the events a lot more dynamic and realistic, giving the player a method to directly influence what was happening in the game, something rarely seen in this genre thereby increasing immersion. The main difficulty arose from having to animate the monster separately from the event, using time codes and a lot of trial and error to ensure that the monster performed specific actions at the right time. It also drastically increased the event workload, as multiple branches then had to be implemented to ensure the system worked.

Regular animations were also developed for the AI's actions and general gameplay setting, with a blend tree being used for movement to ensure the transition between walking and running was smooth as oppose to jumping between several animations, with a speed float being used to dictate how far into the blend the animation had passed.



As reference I used run and walk cycles developed for lions and dogs to help animate a fluid and realistic movement for the creature based on its composition, as well as filming my own dog in his daily activities for inspiration on how other movements might take place. As previously mentioned, I had limited experience with animation, but I believe I achieved some effective results with the tools at my disposal. However, the animations could be improved should development continue.

Conclusion:

Whilst the nature of the creature's AI continues to expand with the games content, the state-based AI allows for easy expansion and continue development, with reliable and realistic results. Whilst there are potential areas for improvement, overall the system was successful in creating a dangerous predator that could act as the primary antagonist for It Will Find You, and keep the player in a tense battle for survival.