Implementing Artificial Intelligence Perception and Decisions in a Stealth Game

Author: Kyle Thomas
Student Number: 1307669

A dissertation submitted in partial fulfilment of the requirements for the degree of Bachelor of Science Honours in Games Programming

Institution: The University of Bolton Place: Bolton

GAM6001 – Major Project

Date: 2nd May 2017

Supervisors: Philip Carlisle & Tim Ash

Abstract: This dissertation comprises an account of the work that the author did on a team whilst developing a game with stealth and puzzle elements in, for his final year Major Project module. The dissertation explains and the author's approaches and presents the author's reflection on the project as an individual and as a team. The report concludes with a discussion on the implementations used on the project as well as the preparation that the author underwent in the five semesters prior to the major project module and a reflection on the suitability of this preparation.

Table of Contents

Table of Figures

1.0 Background

The aim of the project was to create a game with a team of people from different aspects of the games industry made up of designers, artists and programmers, that was fun and playable for gamers.

A personal aim was to learn about the different artificial intelligence (AI) techniques used to create perception and decision logic, within the stealth genre and to implement it accordingly. General gameplay aspects were also tackled during development to give the player extra interaction with the world.

When the 'Major Project' module began, the prototype was at a point where the team was happy with the results and could progress further in development. Issues on the prototype stage did occur but was quickly fixed with the use of narrative, which helped to further develop the game that the team had envisioned.

2.0 Introduction

Previous to studying for this module, a literature review was written noting multiple methods that the author could use to implement the AI perception for the final year Major Project module (refer to Appendix A). Implementation methods of decision making wasn't studied in the literature review but mentioned in this dissertation as the author thought it was an important aspect and relates closely to the discussion of AI perception. Throughout this dissertation, methods of implementing AI perception and decision logic will be discussed, along with any problems that occurred and solutions to them. These techniques will be discussed in detail, to give the reader an idea of how they can combat it in their project.

3.0 Artificial Intelligence

As the fidelity and realism of game worlds have increased, AI has also become more complex (Anderson, 2003), including elements of perception. 'A mixture of techniques' can be used to give the AI a believable sense of illusion, in terms of intelligence (Anderson, 2003).

The two main perception methods that was focused on during writing the literature review and during the development cycle, was the sight and hearing perception. This was based off other stealth games that used the same perception methods (Champandard, 2007b) such as 'Thief' (Looking Glass Studios, 1998). This dissertation will outline the techniques used to implement the perception, and will refer back to the literature review.

Although not written in the literature review, AI decision making was also touched on in the dissertation, explaining techniques and why they were used within the game. These methods were later studied due to finding them as a potential issue during development.

3.1 Guard

As the project was based around the stealth genre, the game relied on having a non-player character (NPC) that can potentially detect the player. The guard is responsible for patrolling and potentially chasing and killing the player, upon being spotted. The guard has many states that it can transition to, see figure 1; patrol, chase, search alert and attack. Perception was used to give the AI some 'awareness of the world' (McIntosh, 2015) which is useful in terms of the stealth genre. The sensory systems main two senses that are important for a stealth game are sight and hearing (Champandard, 2007b) and will be discussed in this section.

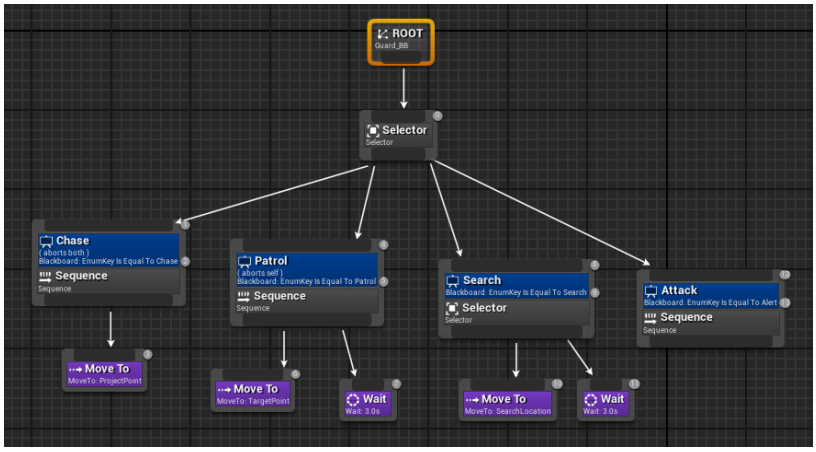

Figure 1 a diagram from the game showing the Behaviour Tree setup with the AI states

3.1.1 Sight perception

The first perception that was implemented on the guard NPC was sight, which deals with how the NPC will 'see' and interact with the player. This section will go on to discuss how the sight perception was implemented, along with any problems that occurred and solutions to them, during the development process.

3.1.1.1 How was the guard's sight perception implemented?

Rather than using a standard view cone, peripheral vision was used, see figure 2 for an example of this. The peripheral vision is more 'sensitive to sudden movements' (Khatib, 2014). It can also be adept when detecting movement (Rabin and Delp, 2008) which is why this technique was implemented because the AI needs to be able to see the player in a realistic manner and needs to feel fair, in a stealthy situation. This technique was implemented based off what was discussed in the literature review (refer to Appendix A).

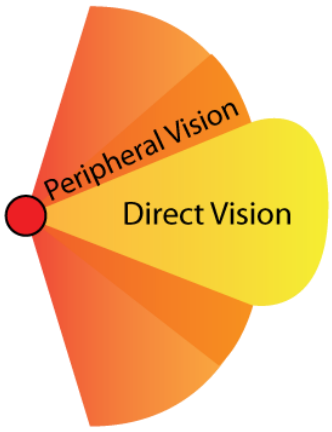

Figure 2 diagram showing a top-down view of peripheral vision on an AI, (Khatib, 2014)

With the stealth genre, you want to give the player the ability to hide behind an object undetected, which is why the raycasting technique was implemented. Raycasting is a technique used to check if a line of sight exists to a certain object (Millington & Funge, 2009) and was implemented to let the AI check the level for any obstructions in front of the player. In McIntosh (2015) Game Developers Conference (GDC) spoke about 'vision raycasts', a good open ended question was highlighted, 'how do you know that there's something in the way' of the player? To deal with this, three raycasts were fired towards the player in order to test for objects in the way. As mentioned in the literature review (refer to appendix A), this was implemented using one raycast which fires in the direction of the player, upon entering the peripheral vision, as seen in figure 3.

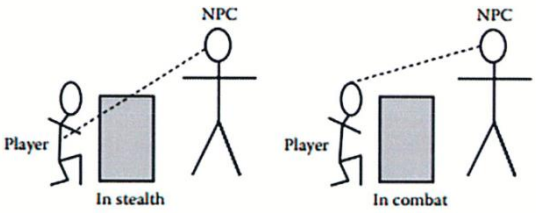

Figure 3 showing a raycast between AI and player behind an object, (McIntosh, 2015)

View distance was implemented similarly to Miles (2015, pp. 413-421), Miles mentions that 'only agents within that radius are tested to determine whether they can detect the interest source', agents out of the view distance will not be perceived by the AI at all, helping to increase the performance of the game. This links to how raycasting and performance is dealt with (refer to 4.1.1.2) and gives the AI realistic sight along with the peripheral vision. This method was discussed during the literature review (Appendix A).

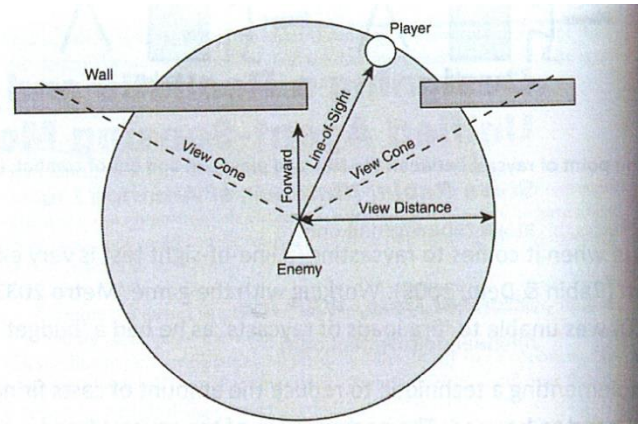

Figure 4 a top-down image showing an NPC with view distance, raycasting and view cone towards the player, (Rabin & Delp, 2008)

Unreal Engine 4 (UE4) (Epic MegaGames, Inc., 2014a) has a system in place that deals with the perception on the AI called 'AI Perception Component'. A sight configuration was used to implement the sight perception which uses a combination of the view distance, peripheral vision and raycasting. Half-life (Valve Corporation, 2004) also uses a combination of the view distance, cone of vision and line of sight to achieve the same type of sight perception (Moore, 2014) which can be seen as an example in figure 4.

3.1.1.2 Problems and solutions

A potential problem when implementing raycasting is performance. Decreasing the view angle and distance of the AI was performed so the raycast doesn't need to be cast so far (Buinitskiy, 2010) making sure only necessary checks are performed at certain intervals rather than constantly, which helped boost the performance of the game.

For the sight perception, there are many factors taken into account, one being the size of the object (Champandard, 2007b). As the player character is small an issue that occurred was the peripheral vision. If the drone was at the NPC's feet, the AI would totally lose the player and immediately change to the search state. This was fixed by increasing the peripheral vision angle, and by keeping the guard at a certain distance away from the player so it wouldn't lose sight of the player.

When the player left the sight of the AI, a way for the NPC to check the last location that the player was seen became an issue. The solution was to leave an entity behind at the last point the player was seen at (McIntosh, 2014) which stopped the AI from cheating in order to find the player. 'Metal Gear Solid games have an "Alert" concept, which puts all guards in the area into a state to search for the player' (Moore, 2014, p.6). 'Halo' (Bungie, 2001) uses this technique, when the player is out of view it has some information of where they were last seen and is unaware of the exact position

(Butcher & Griesemer, 2002). 'Thief' (Looking Glass Studios, 1998) also uses this technique as part of their sensory system as they 'store details such as the time, location' and if the guard had line of sight to the player or not (Moore, 2014, p.2). If the AI is in a search state and is looking for the player, the NPC will need to find somewhere that is near the targets last known location. (Butcher & Griesemer, 2002), which stopped the NPC from knowing exactly where the player is at any given time but gives the AI a general idea of where to look next. Storing the last location of the player and switching the AI's state to search were both used within the project to fix this problem.

It can be very wasteful to check for a condition every frame (Champandard, 2007a). If there are multiple NPCs within a level, this can create performance issues. Instead, stimulus behaviour can be used, which is part of the AI Perception Component in UE4, which will only fire an impulse dynamically to a specific point in the Behaviour Tree (Champandard, 2007a).

Though, checking for events a when stimulus is recognised brought up another problem. The 'UpdatePerception' function is only called when the 'AI Perception Component' receives new stimuli (Epic MegaGames, Inc., 2016b) for example, when the player enters or exits the AI's peripheral vision. Performing a technique like the detection system was a problem, which was fixed by doing a simple Boolean check, to see when the player is in view or not and was executed within the 'Tick' function, which runs every frame.

3.1.1.3 Future improvements

'Thief' (Looking Glass Studios, 1998) being a stealth game, implemented features such as lighting to play a part in their stealth system (Moore, 2014). Tracking the areas of shadow will help the character move in stealth (Millington & Funge, 2009). This is an aspect that can be brought to the game so the AI could react with low light situations, as seen in figure 5. However, the levels were well lit so this didn't come up as a potential feature.

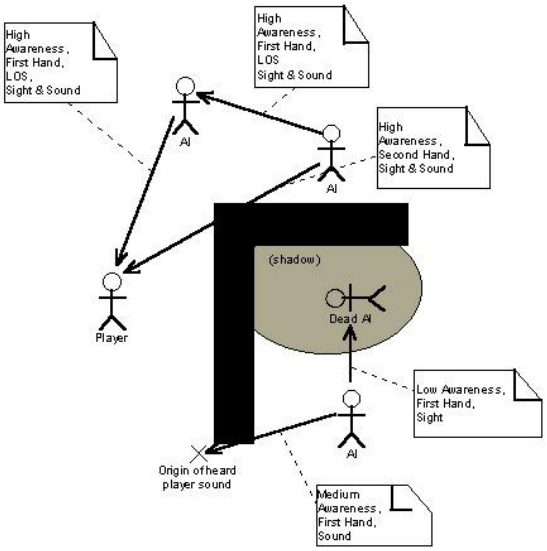

Figure 5 a diagram showing the awareness of the AI in a stealth game, (Leonard, 2003)

3.1.2 Hearing perception

The second sensory system of the guard involved hearing the player (Moore, 2014). This gave the AI the ability to 'hear' certain sounds within the scene. Although in the literature review (Refer to Appendix A), it was mentioned that 'there is a possibility that it would be slightly unrealistic for this team's project', it was still implemented to see how it would interact within the game. This section will discuss how the hearing perception was implemented, along with any problems that occurred and solutions to them, during the development process.

3.1.2.1 How was the guard's hearing perception implemented?

When the hearing perception was first implemented in the project it was devised using spheres around the AI (Barrera, et al., 2015) with a random chance of detection. This helped create a quick prototype for the hearing perception but didn't seem to feel very effective and was implemented differently, using the UE4 'AI Perception hearing configuration'.

Instead, using the UE4 AI Perception System, a hearing configuration was used to implement the hearing perception, an in-game image showing this implementation can be found in figure 6. A single-shot sound event was used which travelled a particular distance (Rabin & Delp, 2008). This system can also quite easily have a certain sound intensity that is reduced over a distance (Sommeregger, 2013), but it wasn't needed for this project because the hearing distance was small on the guards, to give the player enough time to get in and out of the hearing radius undetected when trying to grab a certain item from the guards.

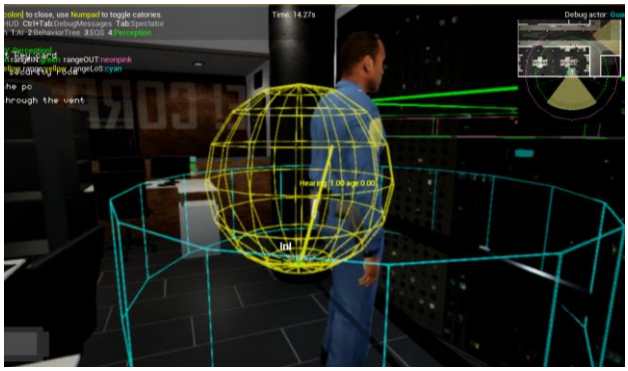

Figure 6 screenshot from the game showing hearing perception in debug mode, sound location (yellow sphere), AI hearing radius (blue circle)

3.1.2.2 Problems and solutions

When the player was damaged, it emitted an electrical surge sound and noise events were fired based on this, alerting the guard every time the player was low on health. Testing helped to spot this as a problem as players were unaware on why the guards were turning around. Instead the hearing perception was changed so that it was based on a timer, to simulate reaction times (Champandard, 2007b) and fired when the player is within hearing range (Rabin & Delp, 2008).

Another complication which came about during testing was that users didn't know why the guards were turning around when they tried to get the key card, making the decision obvious helps the player to know when a decision is made (Champandard, 2007a). 'Thief' (Looking Glass Studios, 1998) use voice recordings to tell the player exactly what the AI is doing, for example "what was that noise" (Champandard, 2007b), read under 'easily understandable actors' sub-section. In 'Halo' (Bungie, 2001) this is done in three ways (Butcher & Griesemer, 2002):

- Language (audio queues)
- Posture (sneaking or running animation)
- Gestures (Specific movements)

Following these examples, the language technique was used in the project to tell the player they are about to be seen. If you can give feedback to the player based on these examples, you can tell them what they have just performed, was either a good or bad action (Butcher & Griesemer, 2002). This was implemented using an audio queue which played upon entering the hearing range and an icon which is increased overtime whilst the player is within range, similar to how 'The Elder Scrolls V: Skyrim' (Bethesda Game Studios, 2011) detection system is done. See figure 7 on how 'Metal Gear Solid' (Konami Computer Entertainment Japan, 1998) achieve this.

Figure 7 screenshot taken from 'Metal Gear Solid' (Konami Computer Entertainment Japan, 1998) showing an alerted guard, (Hadley, 2008)

3.1.3 Detection system

In terms of a stealth game, it is 'simply defined in terms of presenting a challenge for the player to get from one location to another' while remaining undetected by NPCs (Tremblay, et al., 2013). This brought about the idea of introducing a detection system, depending on where the player is in the peripheral vision, depends on how the NPC has spotted the player. This section goes onto explain how this system was implemented along with any problems that occurred and solutions to them.

Figure 8 screenshots showing the guard detection system, semi-noticed (left), fully noticed (right)

3.1.3.1 How was the detection system implemented?

In 'Tom Clancy's Splinter Cell: Blacklist' (Ubisoft Toronto, 2013), when the player is within the vision zone, a timer 'kicks off' which is based on distance between the NPC and player and when the timer hits zero, the AI will perceive the player (Walsh, 2015).

'The Last of Us' (Naughty Dog, 2013) uses a similar idea although they use two vision cones, direct vision and peripheral vision for the AI (McIntosh, 2014). Doing this, gave them the ability to have a meter/detection system that is increased based on certain factors that they had specified (McIntosh, 2014), such as if the player is at the edge of the view cone, the AI is less likely to see.

The detection system was implemented based on both of these ideas. When the player was inside the peripheral vision zone, the timer started ticking and when they got out of the zone, the timer reset and the player wasn't detected. If the player was too close, the AI would spot the player instantly, refer to figure 8 to see the detection system. This was implemented based off feedback given from testing as players wanted to have a timer to escape vision before the guard notices them, which simulates some type of reaction time on the guards.

3.1.3.2 Problems and solutions

During testing, a common problem occurred. The player didn't know if they were seen, or not, by the guards. In the game 'Splinter Cell' (Ubisoft Toronto, 2013) the developers arrived at a solution after 'many hours of playtesting and designer tweaking' (Walsh, 2015, pp.316-317) which was to use a detection heads-up display (HUD), to display feedback to the player. This worked out well because further testing showed that the players understood more.

3.2 Decision making

When it comes to AI, decision making is a major area, as without one, 'there can be no reasonable behaviour' (Johansson, 2012, p.3). Decision making is responsible for working out what the NPC is going to perform next, which can be from a range of set behaviours such as: Attacking, patrolling and hiding (Millington & Funge, 2009).

3.2.1 Behaviour Tree

In terms of implementing behaviour on the AI, Behaviour Trees (BT) are an approach that can be used (Moore, 2014), which replaces the growing mess of state transitions used by FSMs (Knafla, 2011), see figure 9 to observe a simple BT. BTs help provide a framework to design more of an understandable and easier to read AI, which makes it nicely organised and easier to visually debug (Rasmussen, 2016). Champandard (2007c) explains under the section named 'states or behaviours' that the BT focuses 'on increasing the modularity of states by encapsulating logic transparently' for example, nested states which can introduce behaviour by adding in state transitions.

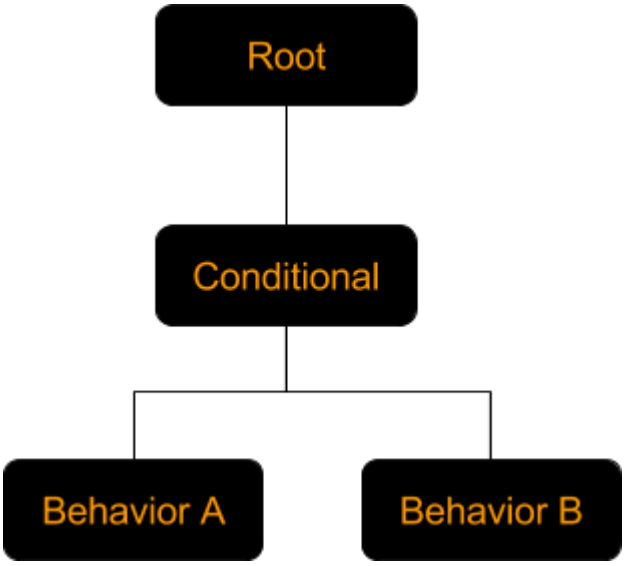

Figure 9 a diagram showing a simple Behaviour Tree, (Rasmussen, 2016)

The "modular" BT allows the developer to describe behaviours 'on a high level without having to think about pointers, memory, compilers, classes etc.' (Crytek, 2013). With the use of modular behaviours, the behaviour tree implementation is scalable and extensible because additional behaviours can be added to perform certain AI decisions (Moore, 2014).

3.2.2 Finite State Machine

A Finite State Machine (FSM) is a set number of states that control the AI behaviour by transitioning from one state to another (Moore, 2014) and are simple to use for AI implementation (Rasmussen, 2016). The FSM is used to tell the AI exactly how to behave in every situation (Orkin, 2006). The FSM will continue running its current state until a condition tells it to transition to another (Rasmussen, 2016). The FSM gives you more control over individual transitions (Champandard, 2007c) making it easier to switch between them. Knafla (2011) mentions under the section named 'behaviour trees vs. finite state machines' that 'the transitions between FSM states give a finite state machine creator great freedom'. 'Thief' (Looking Glass Studios, 1998) uses a 'standard AI logic' (FSM), in which the behaviours are 'cleverly' used to support gameplay (Champandard, 2007b). The FSM will get what

current goal is and what the completion rules for that goal will be (Butcher & Griesemer, 2002). A downside to the FSM is that the more complex the AI behaviour needs to be, the more states are required to control it' (Moore, 2014), making the FSM large and cluttered.

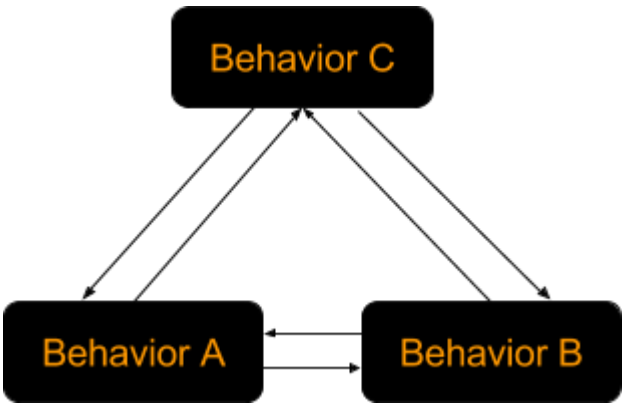

Figure 10 a diagram showing a typical Finite State Machine configuration, (Rasmussen, 2016)

3.2.3 Visual debugging

UE4 offers a useful 'Gameplay Debugger', as shown in figure 11, which lets the developer watch real-time data within the game (Epic MegaGames, Inc., 2016d, p.1). The most important debugging features that was provided for the project are; 'basic data from AIController', 'information about Behaviour Tree and Blackboard data', 'information from perception system' and navigation mesh 'around player or selected pawn' (Epic MegaGames, Inc., 2016d, p.1). This gave the programmer, and possibly other team members, the ability to check that the BT is returning the correct variables that have been passed and to make sure that the AI is actually perceiving the player or not.

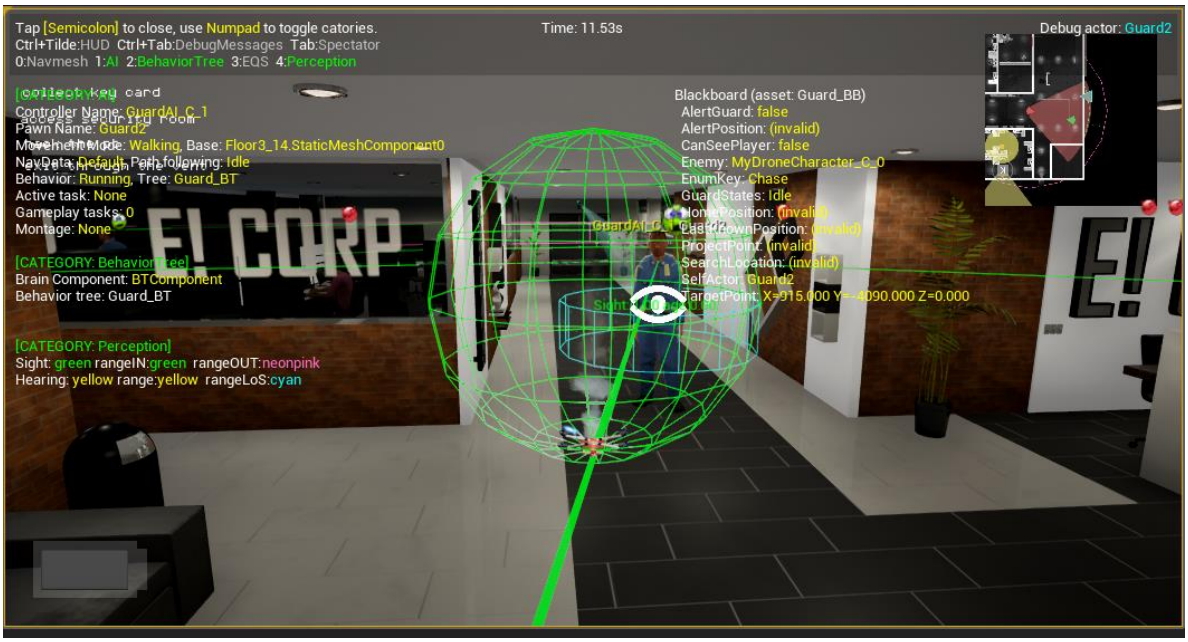

Figure 11 screenshot showing the visual debugger within the game

3.2.4 How was it implemented?

The AI behaviour was implemented using a combination of the BT and FSM. The states were passed over to the BT in order to tell the AI what to perform next, see figure 12 for a diagram of these.

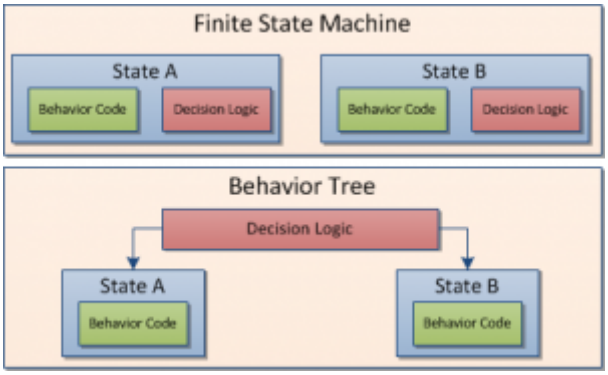

Figure 12 a diagram showing differences between the FSM and BT logic, (Mark, 2012)

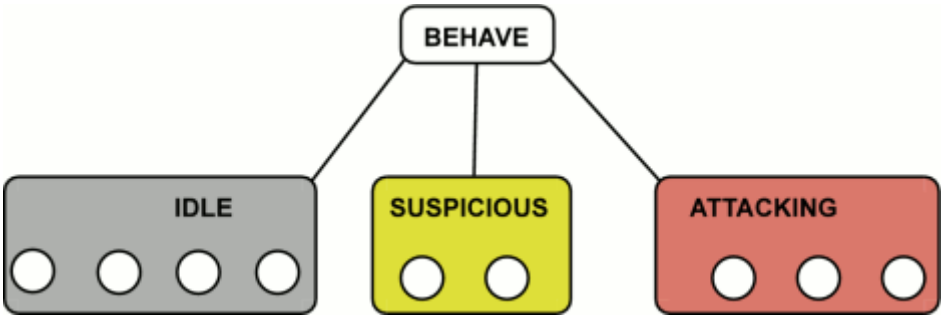

Figure 13 a diagram of a Behaviour Tree made up of states, (Champandard, 2007c)

UE4 uses a nice Behaviour Tree implementation (Rasmussen, 2016) and according to this, the BT was used and explored to make use of certain features that it offers. A decorator node was used to run certain decisions that the AI was told to make. The decorator node is used to add functionality to a node within the BT (Crytek, 2013). This was done by using a simple, and common, category used with the decorator node, which is whether or not to run the child behaviour. This is based on the result of whether the 'filter' is successful or not. If the 'filter' is successful the child behaviour will run, but if it isn't successful the decorator node will return as failed and won't run the behaviour (Millington & Funge, 2009, p.346). These decorators were attached to the 'sequence' nodes.

A 'sequence' node was used in the BT which visits each child node in order, from left to right, as seen in figure 14 and 15, when the child node returns as succeeded, it will go to the second node and so on (Simpson, 2014). This helped keep the BT organised and helped visually see what the AI would perform next.

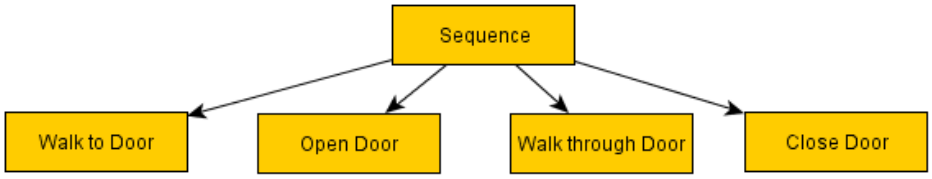

Figure 14 a diagram showing the Behaviour Tree sequence node, (Simpson, 2014)

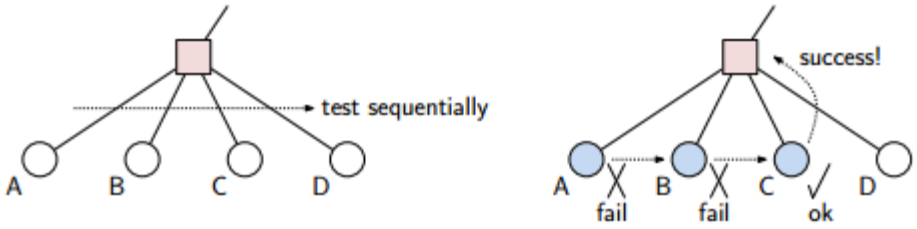

Figure 15 a diagram showing how the sequence node is run in the Behaviour Tree, (Mount, 2013)

A 'selector' node was used in the BT as the main node. Before each of the sequences ran, it had to first pass through the selector node. This differs from the sequence because it doesn't have to wait for the nodes to return success, instead it will return success when any of its child nodes succeed and can't process further (Simpson, 2014). A selector will start by selecting the first, left most task and try to execute it. If it succeeds, then the decision is run, but if it fails, it will attempt to execute the next task along (Mount, 2013).

3.2.5 Problems and solutions

Having looked back at the FSM and BT implementation, it doesn't make much sense to integrate both methods into one system. To improve upon this in the future, adding custom decorator nodes will help to avoid using FSM, which will also make the BT implementation better in terms of visual implementation, allowing the developer to spot any issues that might be occurring on the AI.

4.0 Reflection

As the project and module was a learning experience, this section will reflect on the project as a whole, from a personal and a team's point of view. Current features and improvements that can be made to the system are also explained, discussing why this was suitable to the genre.

4.1 Personal reflection

The author wanted to implement some new techniques during the development cycle of the project so he focused on studying many AI techniques in terms of implementing perception, during the literature review (refer to appendix A), one being sight. The current implementation, as discussed in section 3.1.1, works very well for the stealth elements and felt well balanced. A future improvement would be to adjust the peripheral vision, or use two vision cones, so that the AI will have less chance to see the player at the edges of the vision. This would seem more realistic in terms of vision and the detection system could also benefit from this implementation.

A potential improvement mentioned in section 3.1.1.3, discussed taking light into consideration when using the sight perception. In 'Splinter Cell' (Ubisoft Toronto, 2013) this can be done by determining the concealment of the player against 'dynamic shadows, mist and other hiding effects' (Millington & Funge, 2009, p.817), see figure 5.

As the current hearing system doesn't have use 'falloff' radius (see figure 16), an improvement would be to us a single-shot sound event that can be fired at a certain radius (Rabin & Delp, 2008). This would improve the hearing system as the hearing radius could be increased based on this. This would add in extra stealth mechanics for the player and require them to be extra cautious.

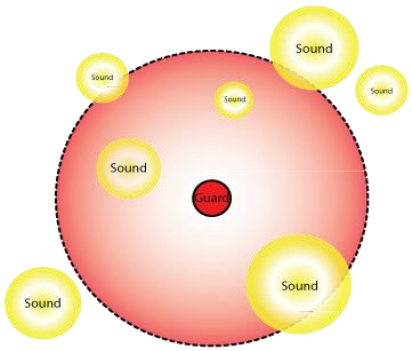

Figure 16 a top-down diagram showing sound traveling at a distance towards the guards hearing range, (Khatib, 2014)

During development, it was important that the AI had some type of decision logic. This was achieved by using FSM to pass states to the UE4s Behaviour Tree system. Although the AI decision system worked well for the project, a utility-based system could be used to replace the FSM implementation. The utility-based system is 'robust and powerful' and will give the AI the ability to pick a task based on the 'single, uniform value' (Graham, 2013, p.113), as seen in figure 17. The utility system will rate each scenario to see if it will achieve a certain criteria (Mills & Stufflebeam, 2005), in which the AI will decide on an action to perform based on the highest number. For example, an enemy AI in an RPG game might have two different outcomes when attacking the player, hit or miss (Graham, 2013).

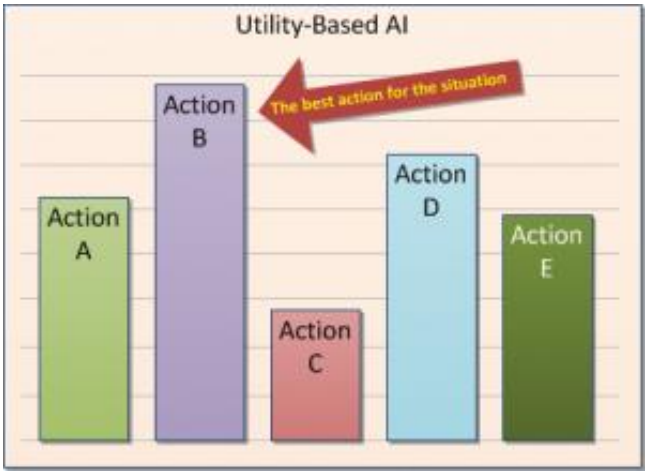

Figure 17 utility-based diagram showing all potential actions the AI can select, (Mark, 2012)

If the author was to tackle a project similar to this again, re-thinking the design implementation would be performed before writing code. During the development, a prototype method was used to create the AI, though by the end of the project this meant that refactoring had to take place and the author felt that the code was quite messy, as the prototype method is used to reduce the number of classes (Mattsson, 1996) as it is used for fast development. A better approach to this would be to create a Unified Modeling Language (UML) class diagram beforehand and introduce a design pattern, such as the factory pattern, which will give a more flexible approach by creating a single class with instances that represent a different type of object (Nystrom, 2014), refer to figure 18. An example of this would be to have a base 'Monster' class with a 'Dragon' instance that derives from the base class 'Monster' (Nystrom, 2014).

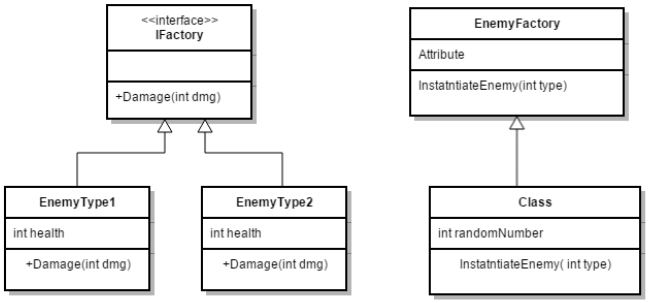

Figure 18 a UML diagram showing an implementation of a factory pattern, (Rojek, 2016)

4.2 Team reflection

One team member unfortunately left the project from the start, which left the team with four members, two designers, one artist and one programmer. This was a potential problem, but because it happened early on in development (in the prototyping stage) the project manager was able to assign the tasks differently and have a smaller scope. Ansimuz Games (2015) came to a conclusion that, working with 'a smaller scope will allow you to stay in control of your project'.

As it was the first time working alongside designers and an artist, it became a great learning experience for the author. It gave insight on how different techniques are implemented with each role, which helped the author to improve his knowledge. The author was also able to assist in

making their job easier, by exposing variables to the blueprint, and details panel, within the UE4 engine to help improve upon their development and to avoid tasks taking longer than they should.

At first, the prototype wasn't where the team wanted it to be, which is where the project manager stepped in. It was clear to the team that they needed narrative to get the feel of the game, which did indeed help to improve the project. The project manager handled certain tasks such as; scheduling weekly meetings, tasks for each member, documentation and arranging testing sessions. This turned out to be beneficial to the team, because it helped them maintain great workflow throughout development. The project manager needs to have an overall picture of the project and be able to communicate why that picture is important to everyone on the team, they need to have some awareness of how the pieces will bind together by the end of the project (Wyman, 2010). Figure 19 shows the prototype level compared to the finished level.

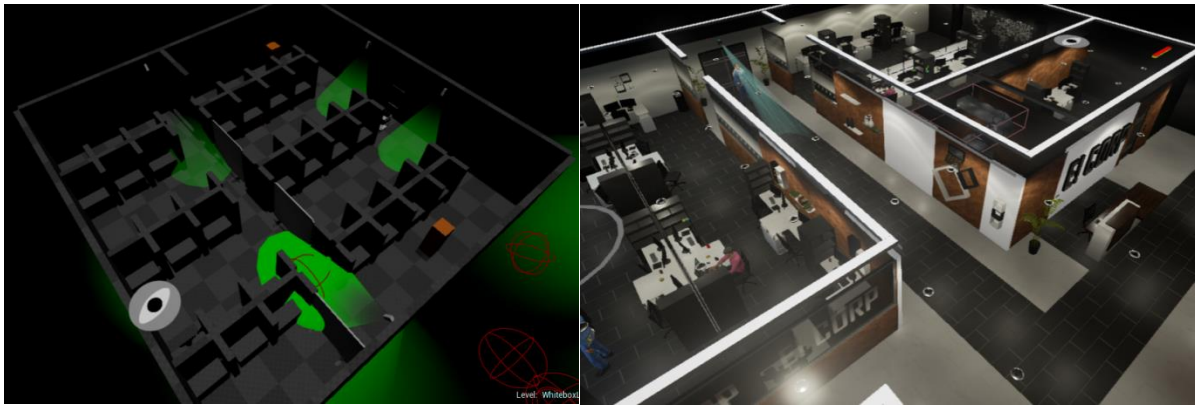

Figure 19 screenshots showing the initial prototype (left) and finished project (right)

Source control was also used for workflow which gave the team the ability to back up the project and share the project easily between each team member. Before a 'commit' was sent to 'GitHub' (GitHub Inc., 2008) the team agreed on a standard to use which is similar to how the games industry perform their commits, by having a proper 'summary' and 'description' detailing what the change was. A commit is how the project is shared between the team, each commit made is stored on a 'repository' which contains all the saved data from previous commits (Valdez, 2013), as seen in figure 20.

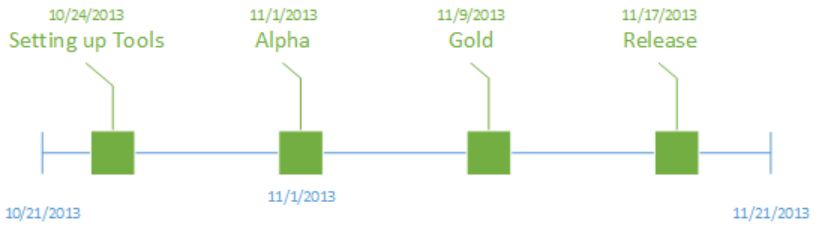

Figure 20 a diagram showing an example of source control commits, (Valdez, 2013)

After the task was committed, the developer would then open 'Trello' (Trello Inc., 2014) and move the task card from the 'in progress' section to the 'testing' section, where one of the other developers would vigorously test the system before moving it to the 'done' section.

With the use of Trello good workflow was achieved throughout development, because it had five main sections: To-do, in progress, testing, bug report and done (refer to figure 21). These sections

helped organize the board so each member could see which tasks needed to be worked on, which tasks are already being worked on, which tasks are complete and requires testing, and which task is tested and complete. Labels were placed on the task cards which helped to see if the task was for a programmer, designer or artist. This practise is used in the industry which is also known as a 'scrum' which is used as a framework to focus on project management. The 'scrum' is used to coordinate tasks between team members so that they can be complete (Godoy & Barbosa, 2010).

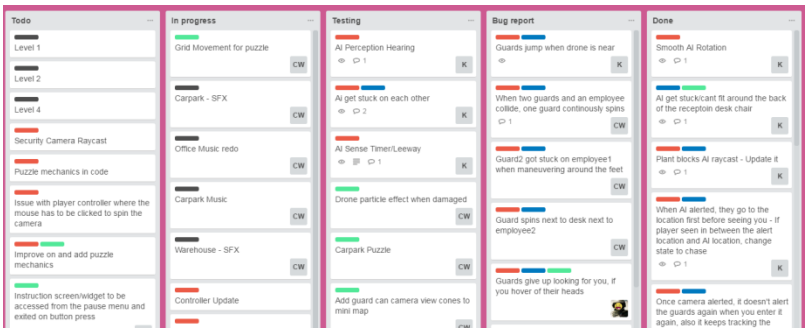

Figure 21 screenshot showing the Trello board after development

4.3 Project reflection

Nearer to the end of the development, the project needed to go through some refactoring and organizing. Having a refactored project helps with the 'removal of "bad smell" code for improving quality without changing its semantics' (Chu, et al., 2011). If the author was to do this project again, he would refactor the code better along the way, rather than letting it get to a state where it is harder to. Keeping the project well organized and structured during development would also be very beneficial not only for the programmer, but for the team. This will help to keep track of where each element is, potentially increasing development time.

Narrative gave the team the ability to have an idea of the game. Before the project had this, the team didn't really have an idea of where the project was going to be by the end of it. It's critical that everyone on the team knows what the game is about (Kershaw, 2016). When the narrative was laid out, the game was clearer and by the end of development this was very close to what was aimed for.

Playtests helped to improve the project by getting feedback from people that haven't seen or played the game before. This helped spot any bugs that needed fixing or features that needed to be implemented. Butcher & Griesemer (2002) mention that when testing 'Halo' (Bungie, 2001), they brought in between 20-25 non hard-core game players and gave them a survey to fill out for feedback. This let them act on the feedback and improve on the elements that lacked. They then go onto mention that 'if you are not doing playtests, you should be doing playtests', which shows that playtesting the game is a very crucial part of the development cycle.

5.0 Conclusion

This dissertation set out to address how AI perception and decisions were implemented within a stealth game, using certain techniques. The sight and sound implementation methods executed when the player is perceived, increasing the performance, making the game run smooth. The combination of view distance, raycasting and peripheral vision helped have a realistic guard AI for the stealth genre, so that the player could hide being undetected. This stopped the AI from feeling like it was cheating as when the player broke line-of-sight, as a last known location was stored for the AI to search around for the player, and would only notice the player upon entering the vision distance. The decision logic helped to switch task quickly, making it feel fluid. To improve upon the implementation in the future, a utility-based system can replace the FSM system to let the AI decide on a task to perform, based on the values given to it although, at its current state the system works and had no issues switching to its task. Each implementation was carefully chosen based on methods used in other stealth games and, in terms of optimization.

The author felt that overall the major project assignment was successful in terms of using the implementation methods discussed. In the three years of attending University of Bolton, the author was able to put forward a strong game, meeting the expectations of what the course was providing. Most of the skills learnt were incorporated within the project, which ranges from team working skills to techniques that can be implemented for optimization.

6.0 References

Anderson, E. F., 2003. *Playing Smart – Artificial Intelligence in Computer Games.* Braunschweig, s.n.

Ansimuz Games, 2015. *How to Scope your game.* [Online]
Available at: http://ansimuz.com/site/archives/579
[Accessed 01 05 2017].

Barrera, R., Kyaw, A. S., Peters, C. & Swe, T. N., 2015. Implementing Sensors. In: *Unity AI Game Programming - Second Edition.* Birmingham: Packt Publishing, p. 43.

Bethesda Game Studios, 2011. *The Elder Scrolls V: Skyrim,* Bethesda, MD: Bethesda Softworks.

Booth, M., 2009. *The AI Systems of Left 4 Dead.* [Online]
Available at: http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf
[Accessed 21 04 2017].

Borovikov, I., 2011. *Navigation Graph Generation.* [Online]
Available at: https://www.gamedev.net/resources/_/technical/artificial-intelligence/navigation-graph-generation-r2805
[Accessed 15 04 2017].

Botea, A. et al., 2013. *Pathfinding in Games,* Dagstuhl: Leibniz Center for Computer Science.

Bourg, D. M. & Seemann, G., 2004. Waypoint Navigation. In: *AI for Game Developers.* Newton, MA: O'Reilly Media, p. 120.

Bungie, 2001. *Halo,* Redmond, WA: Microsoft Game Studios.

Butcher, C. & Griesemer, J., 2002. *Creating the Illusion of Intelligence.* [Online]
Available at: http://gdcvault.com/play/1022590/Creating-the-Illusion-of-Intelligence
[Accessed 15 04 2017].

Champandard, A., 2007a. *Teaming Up with Halo's AI: 42 Tricks to Assist Your Game.* [Online]
Available at: http://aigamedev.com/open/review/halo-ai/
[Accessed 12 04 2017].

Champandard, A., 2007b. *Sneaking Behind Thief's AI: 14 Tricks to Steal for Your Game.* [Online]
Available at: http://aigamedev.com/open/review/thief-ai/
[Accessed 12 04 2017].

Champandard, A., 2007c. *Understanding Behavior Trees.* [Online]
Available at: http://aigamedev.com/open/article/bt-overview/
[Accessed 18 04 2017].

Chu, P., Hsueh, N., Chen, H. & Liu, C., 2011. *A test case refactoring approach for pattern-based software development,* New York, NY: Springer Science & Business Media, LLC.

Crytek, 2013. *Modular Behavior Tree,* Frankfurt: Crytek.

Cui, X. & Shi, H., 2011. A*-based Pathfinding in Modern Computer Games. *IJCSNS International Journal of Computer Science and Network Security,* 11(1), p. 127.

Epic MegaGames, Inc., 2016b. *UAIPerceptionComponent.* [Online]
Available at:
https://docs.unrealengine.com/latest/INT/API/Runtime/AIModule/Perception/UAIPerceptionComponent/index.html
[Accessed 21 04 2017].

Epic MegaGames, Inc., 2016c. *AI & Navigation.* [Online]
Available at: https://docs.unrealengine.com/udk/Three/AIAndNavigationHome.html
[Accessed 22 04 2017].

Epic MegaGames, Inc., 2016d. *Gameplay Debugger.* [Online]
Available at: https://docs.unrealengine.com/latest/INT/Gameplay/Tools/GameplayDebugger/
[Accessed 26 04 2017].

GitHub Inc., 2008. *GitHub,* San Francisco, CA: GitHub Inc..

Graham, D. R., 2013. An Introduction to Utility Theory. In: S. Rabin, ed. *Game AI Pro: Collected Wisdom of Game AI Professionals.* Boca Raton, FL: A K Peters/CRC Press, pp. 113-126.

Hadley, N., 2008. *OCD: METAL GEAR'S INATTENTIVE GUARDS.* [Online]
Available at: http://uk.ign.com/articles/2008/02/19/ocd-metal-gears-inattentive-guards
[Accessed 30 04 2017].

Johansson, A., 2012. *Affective Decision Making in Artificial Intelligence,* Linköping: University Electronic Press.

Kalani, C., 2008. *Fixing Pathfinding Once and For All.* [Online]
Available at: http://www.cs.uu.nl/docs/vakken/mpp/other/path_planning_fails.pdf
[Accessed 12 04 2017].

Khatib, Y., 2014. *Examining the Essential Building Blocks of Stealth Play.* [Online]
Available at:
http://www.gamasutra.com/blogs/YoussefKhatib/20140619/218797/Examining_the_Essential_Building_Blocks_of_Stealth_Play.php
[Accessed 30 04 2017].

Knafla, B., 2011. *Introduction to Behavior Trees.* [Online]
Available at: http://altdevblog.com/2011/02/24/introduction-to-behavior-trees/
[Accessed 19 04 2017].

Konami Computer Entertainment Japan, 1998. *Metal Gear Solid,* Tokyo: Konami.

Leonard, T., 2003. *Building an AI Sensory System: Examining The Design of Thief: The Dark Project.* [Online]
Available at:

http://www.gamasutra.com/view/feature/131297/building_an_ai_sensory_system_.php
[Accessed 30 04 2017].

Looking Glass Studios, 1998. *Thief,* Wimbledon: Eidos Interactive.

Mark, D., 2012. *AI Architectures: A Culinary Guide.* [Online]
Available at: http://intrinsicalgorithm.com/IAonAI/2012/11/ai-architectures-a-culinary-guide-gdmag-article/
[Accessed 30 04 2017].

Mattsson, M., 1996. *Object-Oriented Frameworks A survey of methodological issues.* Ph.D: University of Karskrona.

McIntosh, T., 2014. *The Last of Us: Human Enemy AI.* [Online]
Available at: http://www.gdcvault.com/play/1020338/The-Last-of-Us-Human
[Accessed 11 04 2017].

McIntosh, T., 2015. Human Enemy AI in The Last of Us. In: S. Rabin, ed. *Game AI Pro 2: Collected Wisdom of Game AI Professionals.* Natick: A K Peters/CRC Press, pp. 419-429.

Miles, B., 2015. How to Catch a Ninja NPC. In: S. Rabin, ed. *Game AI Pro 2: Collected Wisdom of Game AI Professionals.* Natick: A K Peters/CRC Press, pp. 413-421.

Millington, I. & Funge, J., 2009. *Artificial Intelligence for Games.* 2nd ed. Boca Raton, FL: CRC Press.

Mills, F. & Stufflebeam, R., 2005. *Introduction to Intelligent Agents,* Washington: CCSI.

Moore, L., 2014. *Improving Guard Behaviour in a Top-Down 2D Stealth Game.* BSc: University of Derby.

Mount, D., 2013. *Artificial Intelligence for Games: Decision Making.* College Park, MD: s.n.

Naughty Dog, 2013. *The Last of Us,* San Mateo: Sony Computer Entertainment.

Nystrom, R., 2014. *Game Programming Patterns.* 1st ed. s.l.:Genever Benning.

Orkin, J., 2006. *Three States and a Plan: The A.I. of F.E.A.R.* Kirkland, WA: Monolith Productions.

Pinter, M., 2001. *Toward More Realistic Pathfinding.* [Online]
Available at:
http://www.gamasutra.com/view/feature/131505/toward_more_realistic_pathfinding.php
[Accessed 18 04 2017].

Rabin, S. & Delp, M., 2008. Designing a Realistic and Unified Agent-Sensing Model. In: S. Jacobs, ed. *Game Programming Gems 7.* Clifton Park: Delmar, pp. 217-228.

Rasmussen, J., 2016. *Are Behavior Trees a Thing of the Past?.* [Online]
Available at:
http://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thin

g_of_the_Past.php
[Accessed 20 04 2017].

Rojek, J., 2016. *C# Factory Pattern.* [Online]
Available at: https://indiedevart.wordpress.com/2016/06/23/c-factory-pattern/
[Accessed 30 04 2017].

Simpson, C., 2014. *Behavior trees for AI: How they work.* [Online]
Available at:
http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php
[Accessed 16 04 2017].

Trello Inc., 2014. *Trello,* Manhattan, NY: Trello Inc..

Tremblay, J., Torres, P. A., Rikovitch, N. & Verbrugge, C., 2013. *An Exploration Tool for Predicting Stealthy Behaviour.* Montreal, AIIDE.

Ubisoft Toronto, 2013. *Tom Clancy's Splinter Cell: Blacklist,* Rennes: Ubisoft.

Valdez, A. A., 2013. *How To Reverse Time - Introduction to Git, Cloud Computing, and Version Control.* [Online]
Available at: https://www.gamedev.net/resources/_/technical/game-programming/how-to-reverse-time-introduction-to-git-cloud-computing-and-version-control-r3434
[Accessed 30 04 2017].

Valve Corporation, 2004. *Half-Life,* Bellevue, WA: Valve Corporation.

Walsh, M., 2015. Modeling Perception and Awareness in Tom Clancy's Splinter Cell Blacklist. In: S. Rabin, ed. *Game AI Pro 2: Collected Wisdom of Game AI Professionals.* Natick, MA: A K Peters/CRC Press, pp. 316-317.

Wang, J.-Y. & Lin, Y.-B., 2012. Implementation and Comparison the Dynamic Pathfinding Algorithm and Two Modified A* Pathfinding Algorithms in a Car Racing Game. *International Journal of Computer Theory and Engineering,* 4(4), pp. 551-555.

Wyman, M. T., 2010. *Making Great Games: An Insider's Guide to Designing and Developing the World's Greatest Video Games.* 1st ed. Waltham, MA: Focal Press.

7.0 Appendix A

Literature Review: AI Perception in Stealth Games

Kyle Thomas - 1307669

University of Bolton

GAM6000 – Research Methods

Tutor: Andrew Williams

Date: 05.12.2016

Table of Contents

Table of Figures

1.0 Introduction

The aim of this literature review is to investigate which methods of AI perception will be useful to use in the team project.

Our team, made up of three designers, one artist and one programmer, is creating a 3D stealth game, with a drone as the playable character. You are an ex-employee of a security firm and the goal is to take back the firmware that makes any server unhackable.

As the sole programmer of the project, the main focus points will be artificial intelligence, hackable systems and character control. Issues could arise from the artificial intelligence, due to the stealth mechanics. To combat this, research will be carried out to review how different perception techniques will work with the genre of gameplay. The research will be coming from industry professionals ranging from books to academic papers, based on artificial intelligence perceptions.

2.0 AI Perceptions

AI perception is how the enemy perceives the world and acts on this information given to it. The idea is to give the player a sense of reality through unique and unpredictable decisions that the AI will make. It is similar to how humans perceive things where senses such as sight, sound and touch are used.

2.1 Sight

2.1.1 View cone

The view cone is a simple vision check which will let the AI see if an object or player is in its range of sight, see figure 1. A benefit to using this technique is that it has good performance on the CPU, due to it using simple mathematical operations such as the dot product. If the result of the dot product is more than zero the player is in the AI view cone, doing this allows calculations to be quicker by potentially eliminating two square root calculations (Rabin & Delp, 2008).

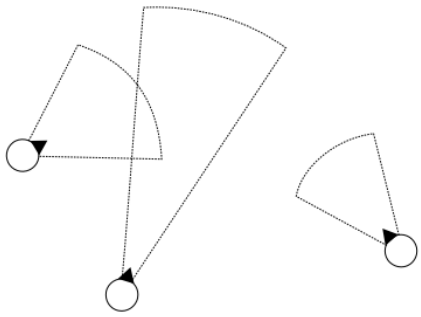

Figure 22 example of simple view cones, (Millington & Funge, 2009)

Millington (2006) and Sommeregger (2013) agree that the vision cone technique has a limitation of the cone shape. Sommeregger (2013) mentions that 'our vision is limited to a cone shape in front of us', whereas Millington (2006) indicates that 'due to the shape of the view cone agents can't see game objects right next to them'. Multiple view cones, shown in figure 2, can be used to give more of a realistic feel to the AI. Each cone will 'output a discrete awareness value' (Leonard, 2003). Sommeregger (2013) supports having multiple cones, as he found out that by experimenting, the cone shape limitation can be 'solved by using a second cone to be able to provide a broader field of view at close distance' or to 'use an ellipse instead of a cone to model the field of view'.

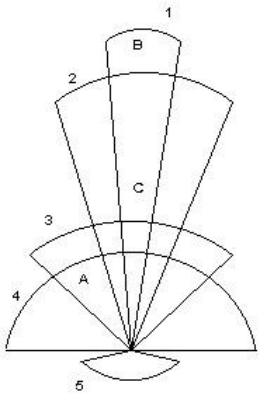

Figure 23 top-down view of multiple view cones, (Leonard, 2003)

Although the methods mentioned are great, a more complex approach to this, called peripheral vision, figure 3, will benefit the AI as it is more 'sensitive to sudden movements' (Khatib, 2014). Rabin and Delp (2008) agree with this as they mention that 'it is adept at detecting movement', which makes it a perfect technique to use for the stealth system. With real life vision, if an object is out of the focal point 'we will start to see them less clearly' (Walsh, 2014). See figure 4 for an example of this real life peripheral vision.

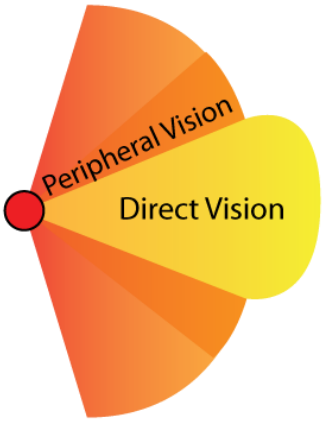

Figure 24 a top-down example of an AI peripheral vision, (Khatib, 2014)

A problem with the vision cone technique is that it doesn't take objects into consideration. If the game is populated with 'vision-obstructing objects', and the player was to hide behind said object, there isn't a way for the cone to test if there is an object in the way of it, which forces 'additional visibility checks' (Khatib, 2014). In the project, the method which will be used for this visibility check will be raycasting, refer to section 2.1.2, which can check if there are objects in the way of the player or not. Raycasting will only be enabled upon the player entering the view cone for optimization reasons.

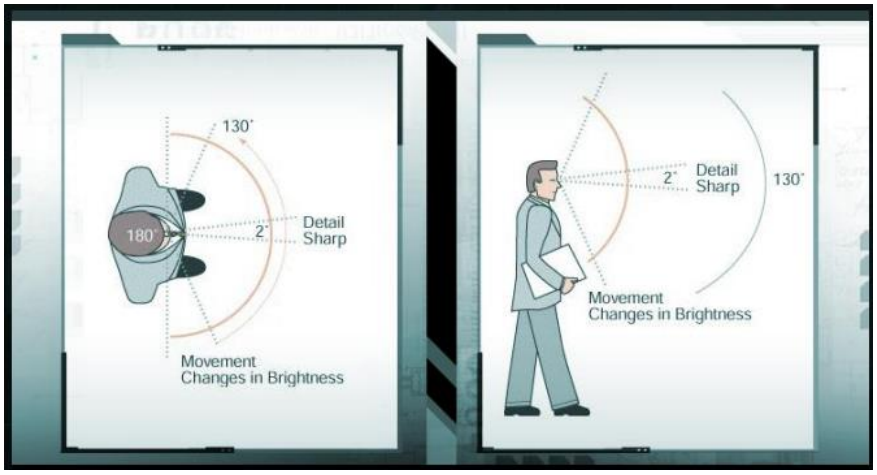

Figure 25 peripheral vision of real life vision, (Walsh, 2014)

2.1.2 Raycasting

Raycasting, also known as line-of-sight, works by plotting two points in the world and then draws a line between them, see figure 6. Games such as 'The Last of Us' (Naughty Dog, 2013) use this technique as it is a useful for spotting objects and characters around the level.

The raycast will point from the AI to the player. McIntosh (2015), a lead programmer at Naughty Dog, found out that after experimenting with the game, 'The Last of Us' (Naughty Dog, 2013), he 'could use a single point' on the players body instead. 'If the player was in stealth, then the point is located in the centre of the player's chest. If the player has engaged an NPC in combat, the point is moved to the top of the player's head', see figure 5 for a detailed example of this. Although this method would be useful for a full scale model of a player, it won't benefit this project as the player is small, which would make it pointless to switch the raycast pivot point.

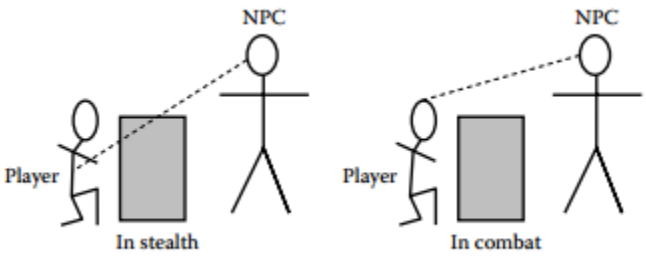

Figure 26 showing the point of raycast between the NPC and player, in and out of combat, (McIntosh, 2015)

2.1.2.1 Performance

Performance is an issue when it comes to raycasting, a line-of-sight test is very expensive and is the most costly to perform (Rabin & Delp, 2008). Working with the game 'Metro 2033' (4A Games, 2010), Buinitskiy (2010) was unable to 'fire loads of raycasts' as he had a 'budget' to follow.

To use the raycast, implementing a technique to reduce the amount of casts firing or have a set distance to fire at, will need to be used. The performance of the raycast 'can be optimized by testing against bounding boxes, instead of testing against individual polygons' (Rabin & Delp, 2008). However, a more complex approach can be used. This is done by decreasing the view angle and distance, so the raycast doesn't have to be cast so far and by limiting the operations per frame (Buinitskiy, 2010). This technique will be useful for the project as hit tests can be performed to see if any objects are blocking the player, without causing any performance issues.

2.1.3 View distance

The view distance will give the AI a realistic feel to the sight, see figure 6 for an idea of view distance. The computation of a view distance check is very simple, though it would be 'more efficient to test against the distance squared instead' (Rabin & Delp, 2008). Miles (2015) agrees that the view distance increases performance as he goes on to say that the sight radius defines 'a maximum radius, and only agents within that radius are tested to determine whether they can detect the interest source'. This will be useful so that the view of the AI won't be infinite, because it will be unfair for the player and will cause performance issues.

Murray (2014) backs this up by continuously checking the distance between the AI and the target (player), and if the 'distance is below a set chase distance, then the AI uses raycasting to check the line of sight'. This shows that the view distance will block the raycast from firing unless the player enters it, improving the performance of the game. Although, the method that Murray (2014) mentions, will only be used for the security camera system and not for the AI because the vision raycast will only fire after two checks, when the player enters the view distance and the peripheral vision.

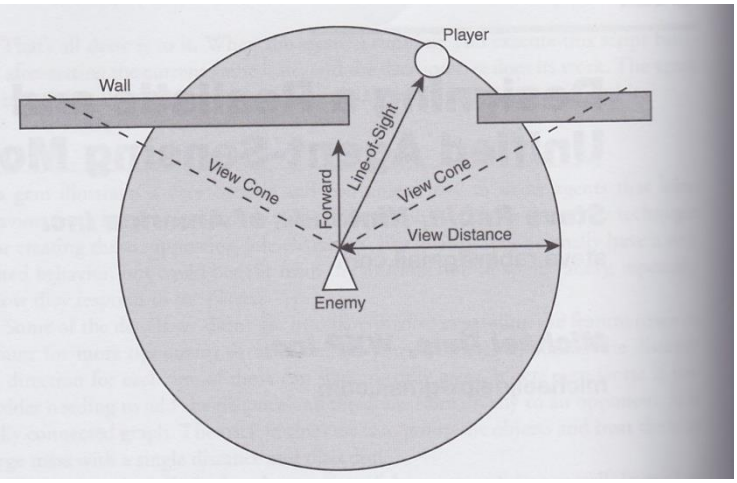

Figure 27 diagram showing view distance, view cone and line-of-sight, (Rabin & Delp, 2008)

2.2 Sound

When it comes to stealth games, sound can be an important factor of the AI perception as you would want to achieve a realistic feel to the game. Leonard (2003) and Sommeregger (2013) concur that implementing sound is similar and simpler than vision. Managing sound with the distance check, performance won't be affected as it is similar to how vision is calculated.

2.2.1 Distance check

A simple way to tackle sound perception would be do it by distance (Sommeregger, 2013), similar to figure 7. A single-shot sound event can be used which will travel a particular distance (Rabin & Delp, 2008). This will enable experimentations with the stealth genre, because if the player is behind the AI, there isn't a way for them to notice the player, so introducing this element could encourage the player to be extra cautious.

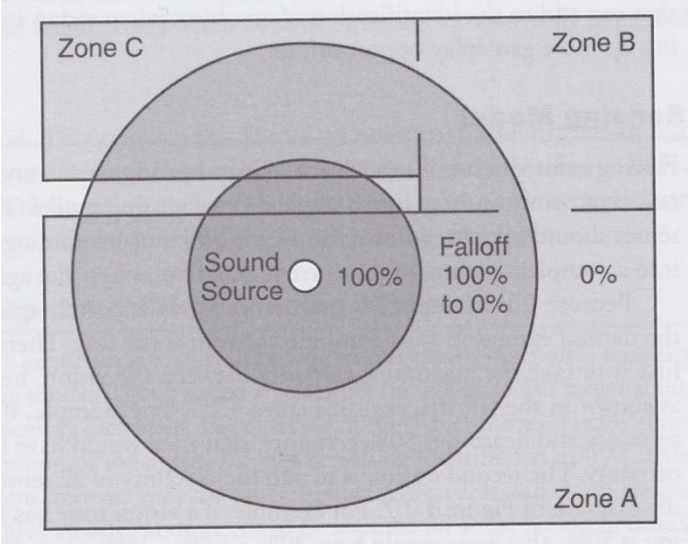

Figure 28 example of the sound radius with a falloff zone, (Rabin & Delp, 2008)

2.2.2 How to improve upon this?

Although this method would work in other games, there is a possibility that it would be slightly unrealistic for this teams project. An issue with using the hearing distance is the 'distance cut-off' point (Rabin & Delp, 2008). Instead of using the simple system, calculating the 'sensory identification as percentage' (Rabin & Delp, 2008) will give the sound perception more fluidity rather than having that cut-off point. A similar technique would be to 'assign a sound a certain intensity at its source that is reduced over distance' (Sommeregger, 2013), see figure 8. It is rather similar to what Rabin and Delp (2008) suggest, but slightly simpler by not dealing with the percentage calculation. The method that Sommeregger (2013) brought up would probably work better as the audio levels can be adjusted to get an accurate system in place and will still be simple enough to implement into the game.

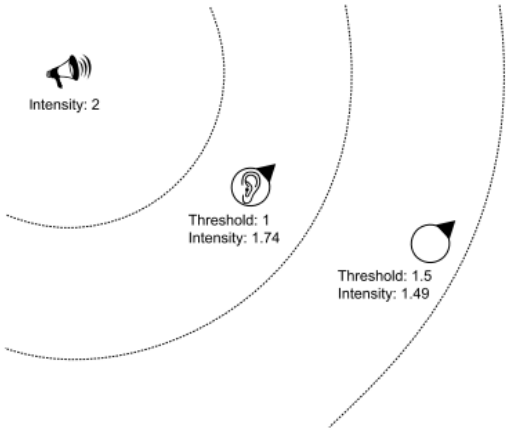

Figure 29 diagram showing the sound intensity over a distance, (Millington & Funge, 2009)

2.3 Other Senses

Away from the main vision and sound perception, there are two more which will be briefly mentioned. Touch and smell.

2.3.1 Touch

Implementing touch perception can be simple and can be done with the 'use of spheres' (Barrera, et al., 2015), see figure 6, 7 and 8 for a general idea of this. This is similar to a technique that Sommeregger (2013) talks about, 'collision detection'. Both methods are fast, as the collision detection can be a simple sphere on the player and when it collides with the AI, an alert message will be sent to it. This method could be convenient in the game, as it is a fast collision check that is being performed, although merging sound perception with touch perception might give wrong results.

2.3.2 Smell

Smell perception can be a little tricky to implement. Smell can be done with the 'use of spheres' too (Barrera, et al., 2015). Kehoe (2009), an instructor for New Jersey institute of technology information, suggests that smell perception can be added by 'each entity in the game a distinct smell number and strength', this can give off 'the radius of the smell and the size of the scent trail left behind'. This is similar to the sound perception, as keeping 'track of' these smells can be used to give the location of the smell to the enemy to search around the area. For a stealth game, smell doesn't seem like it would be a perception to use, although other games could make great use of this idea to make the AI seem human. An example of smell used in the game, 'Half-Life' (Valve, 1998) uses this to simulate 'fictional smells emanating from nearby corpses' (Leonard, 2003)

3.0 Memory

To give the AI perception a feeling of reality, memory can be added. This will enable things such as, remember the last location that the player was perceived at and scout the area when player is out of the line-of-sight. To achieve a greater degree of realism, an agent must have short term memory (Rabin & Delp, 2008), see figure 9 for a diagram of short-term and long-term memory.

3.1 Storing memory

Storing the memory is important, but it might cause performance issues. Diller, et al. (2004) feel that the memory is a simple data structure that represents the current game state. Though the best way to implement a memory model, would be to store each object that enters the sensing model with its own unique identification number and its last known position. Once it's not sensed for several seconds, it can be removed from memory (Rabin & Delp, 2008), improving the games performance, by not keeping unnecessary information stored. This is similar to how human memory works as 'the human memory can be treated as a process and has an ability to store items' (Kowalczuk & Czubenko, 2013) which shows that the method that Rabin and Delp (2008) bring up, would be the best case scenario according to these similarities.

3.2 Short-term

Short-term memory is a popular technique of implementing AI memory. Doherty and O'Riordan (2008) describe that the 'short-term memory of a human is like a temporary store of significant information that has just perceived'. This breaks down how to tackle short-term memory with the AI in a simpler form. This technique has a 'limited capacity and deteriorates quickly' which will help improve the games performance. Without using short-term memory (or in fact any memory at all), an agent is incapable of considering potential opponents that lie outside its sensory horizon which can result in silly looking behaviour (Buckland, 2005). This will be very noticeable to the player so a form of memory on the AI system will be beneficial to use. After some time has passed, 'information is either forgotten or passed to long-term memory' (Doherty & O'Riordan, 2008). This technique would be used within the game as cleaning up on the information that isn't important and passing the most important ones, will boost the performance of the game.

3.3 Long-term

Long-term memory will allow important information to be passed to the AI which will be remembered for a longer time or for the duration of the level/game. Doherty and O'Riordan (2008) point out that 'in a realistic setting, more important information should be sent to an NPS's long-term memory rather than being forgotten'. In such games like 'Mario Tennis' (Camelot Software Planning, 2000) the AI will need to have 'consistent tracking of the ball over the entire game' (Chen & Yi, 2016). Which gives the AI the capability of being human feeling, rather than just having short-term memory, where the AI will forget quickly.

Performance issues might occur as long-term memory has a 'much larger capacity and duration than short-term memory'. Though considering the 'NPC should forget its oldest or least-significant memories', the performance won't be affected too much (Doherty & O'Riordan, 2008), making this technique safe to use within the game.

Cothran and Champandard (2009) handled storing long-term memory in a different way. They were able to use an 'SQL database' in order to store the long-term AI memory. They mention that 'in terms of performance, reading and writing to the database are currently millisecond operations', giving them the ability to pass messages to the game and AI very efficiently. Though, due to the complexity of this method, it won't be included as a method of implementing long-term memory.

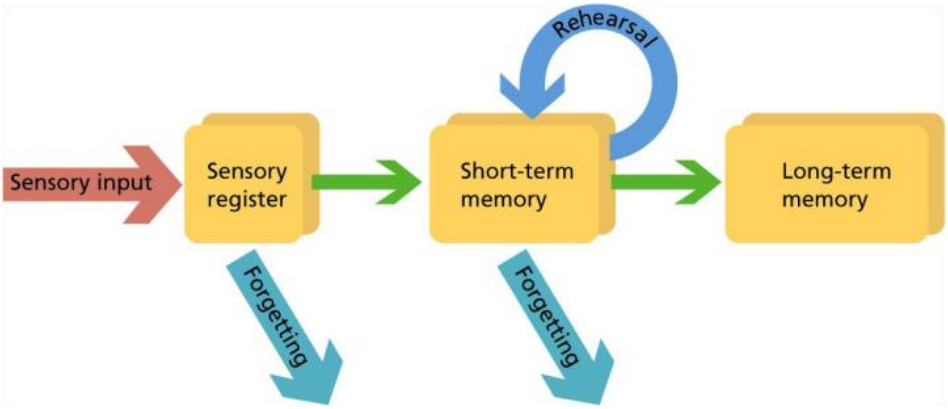

Figure 30 diagram showing human short-term and long-term memory, (Alban, n.d.)

4.0 Summary

AI can be adapted to have a perception system which can be configured to be similar to how humans act. AI perception can be very beneficial to certain game genres to improve the gameplay and to add some sense of immersion. This literature review has explored these perception techniques and points out different ways to implement them, including ways to improve the performance.

5.0 Conclusion

For AI vision, the project will use a combination of raycasting with peripheral view and a distance check. Before firing the raycast, the first check will be the view distance, if the player is within this, the second check, to see if the player enters the view cone will then be performed. Once both of those are true, the raycast can then fire to spot the direction and location of the player.

The sound will be handled by giving objects a sound source and reducing it over a distance. If the sound happens to reach the enemy, a message can be sent to the AI, telling him to be alerted. Although, a touch perception could work here too, as colliding can alert them.

Finally the AI a memory system which will give the perception a better feeling. If the player manages to escape the AI and he loses sight of the player, the AI won't search around as it didn't hold that memory. The memory system will tackle this issue and will include both short-term and long-term memory. The performance will be handled by storing important information in the long-term memory and discarding the other information from short-term.

Using a component in 'Unreal Engine' (Epic MegaGames, Inc., 2014) called AI Perception, incorporating all these senses will be much simpler as it provides the calculations. This will allow the development cycle to go much quicker.

6.0 References

4A Games, 2010. *Metro 2033,* Agoura Hills, CA: THQ.

Alban, D., n.d. *How to Improve Short-Term Memory.* [Online]
Available at: http://bebrainfit.com/improve-short-term-memory/
[Accessed 28 November 2016].

Barrera, R., Kyaw, A. S., Peters, C. & Swe, T. N., 2015. *Unity AI Game Programming.* Second ed. Birmingham: Packt Publishing.

Buckland, M., 2005. *Programming Game AI by Example.* Plano: Wordware Publishing, Inc..

Buinitskiy, A., 2010. *AI for Surviving an Apocalyptic Future: Sensory Perception in METRO 2033.* [Online]
Available at: http://aigamedev.com/premium/interview/metro2033-perception/
[Accessed 20 November 2016].

Camelot Software Planning, 2000. *Mario Tennis,* Kyoto: Nintendo.

Chen, Z. & Yi, D., 2016. *The Game Imitation: A Portable Deep Learning Model for Modern Gaming AI,* Stanford, CA: Stanford University.

Cothran, J. & Champandard, A., 2009. *Winning the 2K Bot Prize with a Long-Term Memory Database using SQLite.* [Online]
Available at: http://aigamedev.com/open/article/sqlite-bot/
[Accessed 27 November 2016].

Diller, D. E. et al., 2004. *Behavior Modeling in Commercial Games.* Sundance, UT, In Proceedings of the 13th Conference on Behavior Representation in Modeling and Simulation.

Doherty, D. & O'Riordan, C., 2008. Toward NPCs Shooter More for Humanlike First-/Third-Person Games. In: S. Rabin, ed. *AI Game Programming Wisdom 4.* Clifton Park, NY: Delmar, pp. 499-511.

Epic MegaGames, Inc., 2014. *Unreal Engine 4,* Cary, NC: Epic MegaGames, Inc..

Kehoe, D., 2009. *Designing Artificial Intelligence for Games (part 2).* [Online]
Available at: https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-2
[Accessed 28 Novemeber 2016].

Khatib, Y., 2014. *Examining the Essential Building Blocks of Stealth Play.* [Online]
Available at:
http://www.gamasutra.com/blogs/YoussefKhatib/20140619/218797/Examining_the_Essential_Building_Blocks_of_Stealth_Play.php
[Accessed 20 November 2016].

Kowalczuk, Z. & Czubenko, M., 2013. Cognitive Memory for Intelligent Systems of Decision-Making, Based on Human Psychology. In: J. Korbicz & M. Kowal, eds. *Intelligent Systems in Technical and Medical Diagnostic.* Manhattan, NY: Springer, pp. 379-380.

Leonard, T., 2003. *Building an AI Sensory System: Examining The Design of Thief: The Dark Project.* [Online]
Available at:
http://www.gamasutra.com/view/feature/2888/building_an_ai_sensory_system_.php?print=1
[Accessed 5 November 2016].

McIntosh, T., 2015. Human Enemy AI in The Last of Us. In: S. Rabin, ed. *Game AI Pro 2: Collected Wisdom of Game AI Professionals.* Natick: A K Peters/CRC Press, pp. 419-429.

Miles, B., 2015. How to Catch a Ninja NPC. In: S. Rabin, ed. *Game AI Pro 2: Collected Wisdom of Game AI Professionals.* Natick: A K Peters/CRC Press, pp. 413-421.

Millington, I., 2006. Sense Management. In: *Artificial Intelligence for Games.* Boca Raton, FL: CRC Press, pp. 742-758.

Millington, I. & Funge, J., 2009. *Artificial Intelligence for Games.* Second ed. Boca Raton, FL: CRC Press.

Murray, J. W., 2014. AI Manager. In: *C# Game Programming Cookbook for Unity 3D.* Abingdon: Routledge, pp. 169-171.

Naughty Dog, 2013. *The Last of Us.* San Mateo: Sony Computer Entertainment.

Rabin, S. & Delp, M., 2008. Designing a Realistic and Unified Agent-Sensing Model. In: S. Jacobs, ed. *Game Programming Gems 7.* Clifton Park: Delmar, pp. 217-228.

Sommeregger, B., 2013. *How to build a robust and reusable AI Sensory System.* s.l.:s.n.

Valve, 1998. *Half-Life,* Bellevue, WA: Valve Corporation.

Walsh, M., 2014. *Modeling AI Perception and Awareness in Splinter Cell: Blacklist.* [Online]
Available at: http://www.gdcvault.com/play/1020436/Modeling-AI-Perception-and-Awareness
[Accessed 28 November 2016].