Improving Guard Behaviour in a Top-Down 2D Stealth Game

Lewis Moore
University of Derby
Derby, England
lewismoore@outlook.com

Abstract – This project aimed to determine if the game behaviour of a top-down 2D Stealth Game will benefit from improvements to the artificial intelligence of the guards in the game. Research and work were conducted on improving the guard’s ability to detect the player, their reaction to the player once detection had occurred, and implementing a form of squad behaviour. Implemented techniques include an awareness system for judging when backup is needed, an Occupancy Map for searching for the player once line of sight is lost, and a Behaviour Tree system to encapsulate all behaviours and provide modularity and extensibility to the implementation. The implementation improves the short-term guard behaviour, such as player detection and initial response, but for long periods of gameplay additional behaviours could be added to extend reaction once the player has been detected on multiple occasions.

I. INTRODUCTION

This project continues previously completed work on game behaviour for a top-down 2D Stealth Game. The developed game includes player path-finding in a physical environment, multiple levels to traverse, and Guards that have the ability to move between waypoints and respond to the player in a basic way. They each operate individually using Finite State Machines (FSMs) and can detect and navigate to the player’s position.

This paper will investigate, through research into the field and additional work, if the overall game behaviour will benefit from improvements made to the artificial intelligence of the guards. Work focuses on modelling the guards’ ability to detect the player more realistically, improving the guards’ detection response, and linking the guards with a squad-based AI technique. If one guard detects the player, can he inform the other Guards to come home in on the player’s position? Can the Guards “work together” in order to increase the challenge of the game and make it feel more realistic?

Research into existing work in the field will focus on guard and squad behaviour implementations in games, as well as different AI constructs and techniques that may be useful for implementing squad behaviour in the game. Planning will be researched; both STRIPS (STanford Research Institute Problem Solver) style and HTN (Hierarchical Task Network) style, in addition to Behaviour Trees and other appropriate areas.

Ultimately, appropriate constructs will be researched and then implemented, with the aim of creating a believable squad of guards for a top-down 2D stealth game. The player will have to avoid and work through these while trying to reach the goal in each level.

II. INVESTIGATION

The inspiration for the game developed as part of this project was the whole Metal Gear Solid series, the roots of which began in 1987 with Metal Gear [1]. The inspiration for the goals of this project; to improve game behaviour by improving the artificial intelligence of the guards, was Metal Gear Solid 2; developed by Konami and released on 2001 [2]. Metal Gear Solid 2 was the first stealth game to feature squad-based guard intelligence; guards could work together to find the player and organise counter attacks [3], and “this level of Artificial Intelligence has since become integral to the stealth sub-genre as a whole” [3].

This section details research conducted into existing work into the field, including Artificial Intelligence for guards and its application in games, work on the behaviour of squads of agents and its application in games, and other advanced AI techniques and constructs. All research aimed to inform the development of improved artificial intelligence and behaviour of the guards in the developed game.

A. AI Techniques and Constructs

Finite State Machines (FSMs) are typically constructed from a set of states in which certain behaviour occurs, and transitions from one state to another at particular times. The more complex an agent’s behaviour needs to be, the more states and transitions are required, and therefore; the larger the FSM becomes. For some scenarios this can be suitable, but because states are designed to perform specific roles within a specific context, they are rarely reusable [4].

Hierarchical Finite State Machines (HFSMs) are a viable approach to implementing state-based systems with some reusable components, and are described by Alex Champandard in ‘The Gist of Hierarchical FSM’ [5]. States are built in the same way and still connected with transitions, but new states can be created by grouping existing states to share transitions. However, in most HFSM implementations, states cannot be used in more than one state group. This means that while transitions are reusable, states are not. [5].
In ‘A Flexible AI System through Behavior Compositing’ [6], the authors describe a system for creating dynamic AI behaviour with modular states (as opposed to just modular transitions in HFSMs). The system is based on a series of Behaviours (in this case they represent a single item, for example a simple task or playing an animation), and a Behaviour Manager which chooses which Behaviour to run [6]. Only a single Behaviour can be run at a time, and the manager decides which to run based on the ‘run-ability’ of the Behaviour at that time and a priority value. It is built with modularity in mind and favours modular Behaviours over inheritance; “code duplication is preferred to creating unnecessary dependencies in the behaviour hierarchy” [6]. The system seems useful for certain desired behaviour; using a priority value allows the developer to easily assign which behaviour should be run over others. However, making decisions based on dynamic factors does not seem like a feature of the system – the priority values are assigned as the tree is constructed by the developer.

For a more dynamic approach, Aaron Khoo describes alternate methods of modelling behaviour and deciding between them at runtime in ‘An Introduction to Behaviour-Based Systems for Games’ [7]. As part of this method, a Behaviour has sensors and actions. Sensors return Boolean values based on whether or not particular environmental variables are assessed to be true, and actions are self-explanatory; they perform an action (again typically a simple task or animation). The author describes methods of picking behaviours as “Conflict Resolution Techniques” [7] and details several in the paper. One which would perhaps allow for more dynamic decision making is to have each ‘run-able’ Behaviour return a normalised activation level then decide the behaviour with the highest level. This activation level could be based on live environment variables, such as distances to other objects. It seems that this, in combination with the Sensor system, could more dynamic behaviour while still remaining modular and reusable.

In terms of behaviour based approaches for applying artificial intelligence to game agents, Behaviour Trees are another approach [8] [9]. Behaviour Trees use the same “single-task” Behaviour structure as in the above implementations, but Behaviours are arranged in a tree-like hierarchy, much in the same way as HTN (Hierarchical Task Network) Planners (discussed later) [10]. The Behaviours are self-contained and stored in nodes, with each node returning a success indicator to their parent. Lists of Behaviours can be added to parent nodes, and parents can execute and evaluate the success of their child nodes in different ways [8]. For example, ‘Sequence’ nodes [11] will only return successful if all nodes in the child sequence return successful. As soon as a node in the sequence fails, the entire node fails. Additionally, ‘Selector’ nodes [12] run the first node in their list whose preconditions are met. The ordering of nodes within a parent is important; for ‘Selector’ nodes this typically indicates priority, and for ‘Sequence’ nodes it is the order in which the nodes are run. Lastly, node Decorators allow nodes to be extended with additional modular behaviour. Decorators can be applied to any node and typically are used to modify the way in which the node runs, such as restricting how often it runs or forcing it to restart once it has ended [9].

B. Artificial Intelligence for Guards

Artificial Intelligence for guards exists in a variety of game types including many stealth games, action games, and more. ‘Thief: The Dark Project’ is a stealth-oriented game developed by Looking Glass Studios [13] that uses a guard sensory system for player detection, as described in “Building an AI Sensory System: Examining The Design of Thief: The Dark Project” [14]. The author states that, in game development, the term “senses” typically means the part of the AI that gathers information from the environment. In addition, senses usually refer to “vision” or “hearing” in most games [14].

Thief has a sensory system that builds on work from Half-Life, although it has been further developed to better suit the stealth genre. The system in Half-Life is built from a view distance, cone of vision, line of sight check and eye position, and the conditional checks are run in order from least expensive to most [14]. The requirements of a sensory system for Thief were quite different from Half-Life’s requirements. It is a stealth game, so elements such as lighting, hiding bodies and sneaking came into play; the sensory system of the guard had to involve hearing the player, as well as detection of hidden bodies. In addition, Thief’s sensory system contains an ‘awareness’ value stored in a ‘Sense Link’. Sense Links point to a position or game element (the player, for example), and store details such as the time, location, whether or not the guard had line of sight. The awareness of the sense link is fed back into the sensory system and modifies the behaviour [14].

While developing the guard sensory system, the developers of Thief realised an important gameplay element; players should not attract the attention of guards that they have not yet seen [14]. This, although potentially “realistic”, does not make for good game behaviour. It would be frustrating; “the player seen by an opponent they do not see often feels cheated” [14]. To avoid this problem, the developers added a check – to only allow guards to detect the player if the player had first seen the guard.

Guard behaviour includes the ability to detect the player, but also to search for the player once detection has occurred. In ‘Probabilistic Target Tracking and Search Using Occupancy Maps’ [15], the author discusses methods of tracking the position of mobile objects after line of sight is lost. This method uses a knowledge-representation technique known as “Occupancy Mapping” [15]. A grid is generated over the environment, and the probability of an object’s location is mapped. If the moving object can currently be “seen”, the probability of it being on the node it is on is 100%, but once line of sight is lost, the probability of the object’s location dissipates to neighbouring nodes. An agent’s behaviour could then be to move to the node with the highest probability at that time, “clearing” nodes as they move; visible nodes without the object have their probability dropped to 0. This gives the impression that the agent is “searching” for the object [15].
C. Squad Behaviour

Squad behaviour has been explored in many games in recent years. In "Squad Tactics: Team AI and Emergent Maneuvers" [16], the author discusses two main approaches to implementing Squad AI or teamwork in games. The first of these is a 'centralized' approach; the squad leader receives information from squad members, then issues commands based on the information he has. The second is a 'decentralized' approach. With this, squad members exchange information, requests and intentions as to what they are going to do, then make decisions based on the information they have [16].

Squad manoeuvres can emerge naturally from a decentralised approach. decentralized methods can often use a message system as described above, although another approach is to implement an AI 'Blackboard'; an area of shared information for all agents [17]. Agents can post data to the Blackboard and retrieve data from it, but their behaviour is their own and is modified based only on the information they have.

Perhaps the most famous implementation of squad behaviour in games is F.E.A.R [18]: one of the first to use Planning in its implementation [10]. F.E.A.R uses GOAP (Goal Oriented Action Planning), which closely resembles a STRIPS (STanford Research Institute Problem Solver) style planner, to plan actions in real-time, adapting AI behaviour to correspond to various factors [19]. STRIPS planners consist of goal states and actions, where goal 'states' represent the state of the world, and actions represent a set of conditions (pre action) and effects (post action). States are represented and used differently in planners than they are in state machines; in this case they are a "set of variables that collectively describe the world" [19]. In F.E.A.R, the agents use the GOAP planner to perform a search through all possible actions which could lead to their current goal state [10]. Since Jeff Orkin's work on GOAP in F.E.A.R, many other modern games are known to use GOAP or STRIPS-style planners, including those in which stealth can factor in as a gameplay style, such as S.T.A.L.K.E.R [20].

Another popular style of planner is HTN (Hierarchical Task Network), based on SHOP (Simple Hierarchical Ordered Planner). These use a tree of tasks and break down tasks with each layer, and different styles of HTN planner break down tasks in different ways [10]. HTN planning is used in Killzone 2 [21] to evaluate the relevance of goals and take actions that link to those goals. Knowledge is represented in a domain that specifies how tasks are performed and the priority of tasks. In addition, the HTN planner in Killzone 2 is not only used for control of each guard; squads of guards have actions and movement co-ordinated by the planner [22].

III. IMPLEMENTATION

Many of the researched techniques have been implemented and tested in the context of improving the game behaviour. The project was conducted in stages, from improving the base behaviour of the existing guards through to implementing forms of squad behaviour. This section is broken down into the same stages to represent the phases of the implementation.

A. Improving the Base Behaviour

In the original version of the game in which these improvements were implemented, there were several basic guard agents who could patrol, detect the player, move to the detected position, and then either "catch" the player if they saw him within their close cone of vision, or return to their patrol state. This behaviour was implemented using FSMs, which research suggested would not be very scalable or maintainable as the guard behaviour grew. As such, before implementing any additional behaviour the first goal of the project was to change the AI implementation to use a more scalable and extensible approach.

Research into methods of modelling Artificial Intelligence led to several using a 'behaviour-based' approach, where behaviours are typically defined as a single task or action [6]. Out of these approaches, a Behaviour Tree implementation seemed to be the best for this project; it would generate modular, re-usable behaviours, as well as decision making that could take into account dynamic environmental factors. The guards make many decisions based on environmental factors such as whether or not they can currently see the player, so this element was very important.

A Behaviour Tree class structure was therefore implemented based on popular structures [9], including Leaf Nodes, Parent Nodes such as Selector Nodes and Sequence Nodes, Node Controllers, and Decorator Nodes such as the Restart Node. The next task was to create a variety of Leaf Nodes, each of which would contain a task. The states that formed part of the previous guard implementation were large and encompassed many actions which triggered depending on the situation, so the Leaf Nodes were split into two key types; a Condition Node, which checks a situational condition and succeeds or fails depending on the result, and an Action Node, which makes the guard perform a small action.

The existing guard FSM included two core states: Patrol and Detected Player, and these states were converted into sequences for the Behaviour Tree. The complex part of the implementation here was configuring existing behaviour to fit into a set of modular nodes. Certain checks had to be performed before some nodes could run, and these were created as reusable Condition Nodes. For example, HasBeenSeenByPlayerCondition is a node used before detecting the player, to ensure the player has seen the guard first, as recommended by the research into existing stealth games like Thief [14]. Many actions were already contained in code methods, and these could be ported relatively simply to their own Action Node.

An overview of the initial Behaviour Tree implemented from the existing guard FSM can be seen in Fig. 1.
At this stage, a functional Behaviour Tree for all existing guard behavior had been implemented. Using this as a base, all new functionality could now be quickly and easily added as modular nodes. In addition, self-contained existing nodes such as CanDetectPlayerCondition could be enhanced with improved player detection functionality without having to add any additional code elsewhere.

**B. Improving Detection and Response**

Many researched elements from Thief and Half-Life [14] could be used in this project’s target game to improve the guard’s ability to detect the player, and by extension, the overall feel and enjoyment of the game.

The theory behind the implemented “vision” system for the guards in the game works in much the same way as in Half-Life: it is constructed from a view radius, a cone of vision and line of sight [14]. However, because the game is viewed from a top-down perspective and is 2D, eye position is not factored in. This is perhaps more important for a first-person 3D game, as it is a potential way for detecting whether or not the player has seen the guard before the guard can detect the player. As previously stated, this is implemented to “co-ordinate the player’s entertainment” and ensure they cannot be detected by a guard they have not yet seen [14]. This still seem important for the developed game as it is top-down but has independent camera control, so it is certainly possible that a guard could “see” or “hear” the player before the player is aware of the guard. Because the game is top-down, this is implemented simply by checking whether or not the guard is on-screen before allowing the guard to detect the player.

To further improve the guards’ detection ability and realism, inspiration was taken from the ‘Sense Link’ and awareness implementation in Thief [14]. The conducted research suggested that these were the core difference between the sensory system in Thief and the one in Half-Life as requirements were different for the two games. The guards in Thief contain Sense Links to game objects such as the player, or a hidden body, which provides an awareness level of that object. Many of the additional elements to be considered in Thief would not be suitable checks in this project’s target game however, as it does not contain implementations of light and shadow or the ability to attack guards and hide bodies.

The only element the guards need to be aware of is the player.

As such, an awareness mechanic was implemented; however it is a single value for each guard. The guard’s awareness of the player is increased by a variable amount as the guard “sees” or “hears” the player, with the amount factoring in the distance the player was from the guard when detected. The awareness then slowly decreases over time as the guards return to patrolling. The initial awareness implementation was then used to adjust the guard’s behaviour to feel more realistic. If the guard saw the player from a long distance and his awareness was very low, he would stop and look around on the spot, but would not act on it any further. This imitates a guard that was unsure of what he had seen and did not feel the need to investigate. As the awareness increased he would react more and more: first moving to the detected position and then the goal was to ultimately have the guard call for backup if the awareness was above a certain value, although this was not implemented at this stage.

Introducing the awareness mechanic into the guards’ behaviour had gone a small way towards improving their responses to detecting the player, but further improvements were desired. Once the guard decided to investigate a detection, the current behaviour simply moved to the detected position, “looked around” on the spot for a couple of seconds, then returned to patrolling. The key problem was that guards were not searching for the player once they had lost sight, so if the player moved any real distance away from the position at which he was detected, the guard would not catch him. As such, methods of implementing searching were researched. "Occupancy Mapping" is a researched knowledge-representation technique that had potential for providing search-like behaviour for the guards, and is a method of tracking the position of mobile objects once line of sight is lost [15]. To implement this, the original navigation grid used for guard and player path finding was used as a starting point. An Occupancy Node class was added that holds the relevant information from a tile on the grid, and each guard has a list of nodes. As a guard detects the player, a Node is added to their list which indicates the current known position. Because the guard can currently see the player, the Node is given a probability of 1 (or 100%). While the guard keeps line of sight, this Node is updated to reflect the real-time position of the player.

As the guard loses line of sight of the player, the probability of him being on any particular tile on the navigation grid cannot be guaranteed, so the value dissipates to neighbouring tiles. These tiles are given a probability between 0 and 1 that is relative to the tile from which they received the value, and then added to the guard’s Occupancy
Map. As probability values dissipate to neighbouring tiles, they are multiplied by 0.8 at each stage, which in effect reduces the probability that the player has reached that tile. The agent’s behaviour is then to move to the node with the highest probability at that time. As the agent sees nodes and confirms that the player is not located there, the probability for that node is dropped to 0 and it is removed from their map. This gives the impression that the agent is “searching” for the player by clearing the area of nodes with a probability of higher than 0. In ‘Probabilistic Target Tracking and Search Using Occupancy Maps’[15] the author recommends a slightly more complex equation to determine how the probability spreads to each node, but in gameplay testing at this stage, this seemed like a good value which produced a reasonable range of tracking and good behaviour.

This Occupancy Map is used in place of the existing “search” code which simply moved to the player’s last known position. Because a Behaviour Tree was implemented, this was simply a case of editing the SearchForPlayerAction node and adding references to a list of Occupancy Nodes; no other actions or areas of code had to be edited.

C. Implementing Squad Behaviour

In order to greatly improve the behaviour of the guards in the targeted game, the next target was to implement a form of squad behaviour between the guards. Research suggested a wide variety of approaches to this, and guard behaviour from the Metal Gear Solid series also influenced work. Research into the use of planning to produce squad based behaviour was conducted, and a potential solution was to use a real-time planner to allow guards to dynamically produce plans based on the state of the environment and the other guards’ actions. However, the lack of availability of real-time planners and engineering time required to implement one meant that using a real-time planner was largely out of the scope of this project. In addition, the other popular researched technique for implementing squad or team behaviour uses modular behaviour systems such as Behaviour Trees. Behaviour Trees have been implemented for individual guard behaviour, and as such, it was decided that the implementation would be reusable and may integrate well with the existing code base. Methods using modular behaviours were taken forward.

The initial plan was to implement a centralised squad systemsinspired by the behaviour of certain guards within the Metal Gear universe. At certain points in the games, guards “report back” to a central commander regarding their state; typically if everything is all clear in the area around them. If a guard is knocked out and doesn’t report back on time, the commander may “command” other guards to check the area and see why he hasn’t responded. Although the game for which this project is focused does not implement any form of combat or disabling of guards, this system did provide inspiration for a potential behaviour style.

A centralised squad system in which a squad leader receives commands from members, then makes decisions based on the information and issues commands [16]. With this in mind, a central Guard Manager was added. Its purpose was to receive messages from the guards in the game as to their current awareness, and when they detect the player. As the Manager received messages, a separate Behaviour Tree was implemented which made decisions based on guard awareness. If a guard’s awareness was above a certain level and they detected the player, and there was a guard nearby who was available to help, they were sent to provide backup by searching in the same area. Although this initially worked, the systemdid not link up well with existing agent Behaviour Trees. External commands were not factored into the trees, so the guards would essentially have to break out of their decisions and typical behaviour in order to provide assistance.

It seemed that a decentralised approach may better integrate with the existing guards’ Behaviour Trees. A decentralised squad system is one in which squad members exchange data, and decisions are made based on what they know and what other group members are doing [16]. This seemed that it would be more appropriate for the current guard behaviour implementation, so an idea for a decentralised approach was formed.

This approach uses a shared Blackboard between all guards within a squad, and they have the ability to access information from the board and post to it from within their Behaviours. This is an alternative to a messaging system often used within decentralised squad behaviour. The important component of the Blackboard was what information it would hold, and how the guards would use that information as part of their behaviour. At this point, the existing guard behaviour used an awareness value to determine what actions to take, and an Occupancy Map to decide where to move in order to try and find the player. This method was conceived, and the awareness and Occupancy Map components were moved to the Blackboard.

With a single shared awareness value, and a single shared Occupancy Map for all guards, squad behaviours did not require a huge amount of adjustment to existing Behaviour Trees in order to emerge. A new sequence was added to the guard Behaviour Trees to accompany the new Blackboard and promote squad behaviour. The new Behaviour Tree can be seen in Fig. 2.
Fig. 2. Behaviour Tree with inclusion of the "Provide Backup" sequence

This approach simulates the guards reporting to each other; telling each other when they thought they saw something, and when certain areas of the map were clear. If at any point a guard can see, or has recently seen the player, the shared Occupancy Map is updated, in addition to the shared awareness value. The guard then proceeds to behave as before; by searching for the player. However, as other guards process their behaviour trees, the newly added Provide Backup sequence may trigger.

The Provide Backup sequence runs as follows: if the guard hasn’t detected the player themselves, and the Occupancy Map isn’t empty (indicating that another guard has detected the player) and the awareness value is above a certain threshold, the guard “detects” the player from wherever he is and proceeds to “search” for the player using the Occupancy Map. This provides interesting behaviour, again simulating a level of communication between guards. Guards from other areas can often come to help search for the player, and although all guards aim for the highest probability node, as each guard sees nodes that do not contain the player, the Occupancy Map is updated for all (simulating an “all clear” message for a certain area from one of the guards to the squad). This means guards can often form natural formations and other interesting, seemingly realistic behaviours as they home in on the potential player position, constantly “communicating” as they do so.

IV. REFLECTION

Gameplay testing was conducted to judge the level to which game behaviour had been improved with the research and work conducted as part of this project. Guards use advanced search techniques and a dynamic decentralised squad system in order to work together to find the player when detected, which provides very interesting behaviours. In terms of short-term reaction, the guards feel very realistic; they search in a local area from which the player may have just moved, and in addition, if the player is detected multiple times in a small area, other guards will move on the player’s position to help the search. If one guard can see the player, other guards also have access to this data as it is stored on the shared Occupancy Map, so the guards will all move together in an attempt to catch the player.

However, problems still exist with longer term reactions. Once all positions in the Occupancy Map have been explored, if the player cannot be seen, the guards will return to patrolling. Metal Gear Solid games have an “Alert” concept, which puts all guards in the area into a state to search for the player. They do not require any knowledge of the player’s position at this time; they simply systematically search the environment. This is a potential method for long-term reaction that could be implemented for the game to complement the work on short-term reaction completed here.

In terms of scalability, the guard behaviour is built from modular behaviours, so additional behaviour can be easily added and additional sequences can reuse existing behaviours. This means that the individual behaviour of each guard is built to be very scalable and extensible to react to any potential situation added as part of future work. However, the implementation is not as scalable as it could be in its current state. The modular system was built on top of the existing framework, and as such, compromises had to be made for it to integrate with the rest of the codebase. The behaviour system could be far more modular if it was re-implemented from scratch to be a core part of the rest of the game’s framework, which will be a high priority piece of future work for the project.

In terms of scalability across squad behaviour, any number of guards can be added in areas and their own behaviours will work in the same way. With the current implementation, as the awareness of the guards increases, up to two guards will move to the detected position in order to assist in searching for the player. This means that, as long as large numbers of guards are not placed in very small areas, the behaviour remains believable. In addition, future work could also very easily allow for the number of guards to provide backup to be increased, and perhaps guards within certain areas can be grouped as a single “squad” with their own Blackboard. This would allow squads to work with each other in an area, and for other separate squads of guards in other areas of the environment to remain unaware.

An additional piece of future work would be to add combat to the gameplay. This would allow the player to attack guards and vice versa, which would greatly modify the behaviour of the guards. The agent Behaviour Tree size would increase in size and complexity, but the use of the modular behaviour system would help with this expansion. Adding combat means that guards can be removed by the player, for which additional behaviour would need to be added. How will guards react if they notice a squad mate is missing? How will guards react to finding the body of another guard? Also, guards will be able to
attack the player, so existing squad behaviour could be extended to support squad based attack movements, usage of cover and more.

V. Conclusion

This project continues previously completed work on gameplay for a top-down 2D Stealth Game. The aim was to investigate if the overall game behaviour would benefit from improvements made to the artificial intelligence of the guards, including improvements to detecting the player and reacting to the detection, as well as implementation of any squad based system.

Improved player detection was implemented using inspiration from games such as Thief and Half-Life [14], while improved player reaction was implemented using a knowledge-representation technique known as “Occupancy Mapping” [15] to simulate guards searching for the player. In addition, the behaviour of the guards in the game was completely re-written to use Behaviour Trees [8] and self-contained modules of behaviour, which greatly helped when adding additional behaviour or extending other areas throughout the project. Lastly, a form of squad behaviour was implemented using the implemented Behaviour Trees, as well as providing a Blackboard for the agents on which to store a shared Occupancy Map and awareness value. This simulated communication between the guards in a squad; as if they were updating each other with what they were seeing as they each moved around. Squad behaviours emerged from this system with a small amount of extra work on the existing Behaviour Tree implementation.

Overall, the produced gameplay feels good as the player moves through the environment; individual guards react in a believable way while being easy to work around without getting caught, while multiple guards reacting in an area provides a challenge which only occurs if you are spotted multiple times (thus raising the awareness). However, for extended gameplay periods in which you are seen several times, the guard behaviour can seem unrealistic. Their behaviour does not advance beyond a certain point; they do not get to the point at which it seems like they know an intruder is in the area and they have to hunt him down before returning to patrol. This is recommended future work in order to improve the artificial intelligence of the guards over long periods of time, while keeping the balance between intelligent behaviour and enjoyable gameplay.

REFERENCES


