# 1. Overview

# 2. Literature Review

## 2.1. Introduction

The aim of this literature review is to examine and compare Behaviour Trees and Goal Oriented Action Planning in the creation of videogame AI. The literature review will look at how both work and the advantages and disadvantages of using one over the other.

## 2.2. Behaviour Trees (BT)

### 2.2.1. Introduction to BTs

A Behaviour tree is a plan of execution of tasks that was made to shape the behaviour of Artificial Intelligence (AI) Non-Player Characters (NPCs) in video games. While it was developed for video games it has also seen use in other industries such as robotics. It has been used in many big games such as Halo and Bioshock. One of the highest profile implementations of it has been built into Unreal Engine (UE) which is one of the biggest game development engines in the world. A BT is comprised of two components. The first is a graphical representation of a hierarchy of different nodes. These nodes store the logic of the behaviour tree and are connected in a tree like structure, hence the name behaviour tree. The second component is a data structure called a Blackboard which stores any information used by the behaviour tree such as player location. The tree executes its logic going through the nodes from left-to-right and from top-to-bottom.

BTs are very modular which allow for easy change in the design of AI behaviour in turn allowing the developers to make the NPCs work and compliment the designers works. It allows NPCs to be manually tweaked to fit an area specifically to get the best result out of that area and to make use of the weapons or environment to give the best experience to the user.

### 2.2.2. Nodes

The nodes are split in to three main types, the root node, control flow nodes and execution nodes.

**Root Node -** The root node also known as the start node is a unique node with only one being able to exist in a behaviour tree. It is used as the starting point of a behaviour tree and it can only have one node connected to it.

**Control Flow Nodes -** These nodes control the child nodes under them that they are connected to. There is no limit to how many of these nodes there can be and they can be connected to other control flow nodes. They can decide whether and how to run the child nodes under them based on conditions and they return a success or failure based on the result of those nodes. There are four main types of control flow nodes but as different behaviour tree implementations are being made and behaviour trees evolve more are being made and the existing ones can change based on which implementation you might look at.

**Sequence Control Flow Node -** Returns success if all its child nodes return success otherwise returns failure if any child node returns failure.

**Fallback/Selector Control Flow Node -** Fallback nodes run their child nodes until they find a child node that returns success then they return success otherwise if all child nodes fail the fallback nodes return failure.

**Parallel Control Flow Node -** Runs all child nodes constantly on a tick and returns success if one or more of its child nodes returns success.

**Decorator Control Flow Nodes -** Can only have one child node. Works as a conditional node deciding whether its child nodes should be run or whether the tree should continue. It has multiple versions that control then how it runs the nodes and how it handles the return value. Some newer versions exist of decorators such as UEs version which turns the decorator node from a node to a condition that you can add to other control flow nodes to control when they should run.

**Timeline

Description automatically generatedExecution Nodes -** Execution nodes also known as task nodes or action nodes carry the main logic of the behaviour tree. They run custom code to execute the specific actions the AI needs to perform.

*Figure 1: Behaviour Tree Example* (Haytam, 2020)

### 2.2.3. Blackboard

While blackboards are not necessary most implementations of BTs have them as BTs nowadays and the AI made through them tend to be very complex and blackboards allow for easy sharing of data between the nodes in the tree and each NPC using that BT. At its simplest form the blackboard is a key-value storage system which allows you to store values and assign a name to them which is then used to call them in the BT when required. These values can be easily used and edited by the nodes of the BT and each NPC using that BT.

Graphical user interface, application

Description automatically generated

*Figure 2: Unreal Engine Blackboard example* (Unreal Engine, 2020)

## 2.3. Goal-Oriented Action Planning (GOAP)

Goal-Oriented Action Planning (GOAP) is a decision-making architecture that takes the next step, and allows characters to decide not only what to do, but how to do it (Orkin, 2002). It is a STRIPS (Stanford Research Institute Problem Solver) architecture used to design the behaviour of NPCs in video games originally created by Jeff Orkin which was first implemented in the game F.E.A.R. (First Encounter Assault Recon) which was published on October 17, 2005.

GOAP allows the NPC to make plans for itself allowing it to adapt to any situation and area it is put in without its behaviour needing to be modified. This makes the NPCs very unpredictable and thus making the game less repetitive and more enjoyable for longer and subsequent playthroughs of it. This also reduces the potential workload on larger games where the NPC would have to be modified to fit each area correctly.

At its base GOAP works like a BT or Finite State Machine (FSM) transitioning from one node to another. However, while BTs and FSMs have many nodes they can transition to and get more as the AI grows. In comparison the initial implementation of GOAP in F.E.A.R. has an FSM with three states. It has Go to, animate and Use Smart Object states with Use Smart Object being used for animations just like Animate so it can be really considered as two. Jeff Orkin in his paper talks on F.E.A.R. talks about this stating “As much as we like to pat ourselves on the back, and talk about how smart our A.I. are, the reality is that all A.I. ever do is move around and play animations! Think about it. An A.I. going for cover is just moving to some position, and then playing a duck or lean animation. An A.I. attacking just loops a firing animation”. He says all NPCs do is move and animate which means that you do not need a complex FSM and you just need to pass it the correct animations and values for it to work.

Diagram

Description automatically generated

*Figure 3: F.E.A.R. GOAP FSM* (Orkin, Three States and a Plan: The A.I. of F.E.A.R., 2006)

Diagram

Description automatically generated

*Figure 4: regular FSM example* (Owens, 2014)

This is where GOAP differs from FSMs and BTs. While FSMs and BTs add more nodes, transitions, and conditions for said transitions GOAP instead make use of a planner, a set of goals, a set of actions and the A\* search algorithm. Instead of making more nodes and transition to those nodes for the actions an AI does instead for GOAP you can get rid of the transitions between these nodes altogether and just creates a set of nodes for the actions to be performed with costs assigned to them and a set of nodes for the goals an NPC can fulfil. NPCs in the game can then be assigned specific sets of actions and goals they can do. The NPC can make use of the actions in a specific chain to complete the goals but with the goals and actions being unconnected they need to be told which to do. For this the planner and search algorithm are used. First an NPC needs a goal to work towards. To give an NPC a goal the NPC calls out to planner and the planner figures out the priority of the goals based on the state of the world such as in F.E.A.R. where if the player is not spotted the patrol goal will have a higher priority than the attack enemy goal until the enemy is spotted. the planner then uses the goal with the highest priority.

Diagram

Description automatically generated

*Figure 5: GOAP planning process visualized* (Orkin, Applying Goal-Oriented Action Planning to Games, 2002)

Once an NPC has a goal to work towards it needs to formulate a plan to complete the goal. Once again then it will require the help of the planner which will look at this list of actions that NPC can perform such as shoot, move, throw grenade, etc, it will use their scores the A\* search algorithm to create a chain of actions to perform the goal with the lowest cost possible. So, for example an NPC may need to move to a location. The planner will assign the goal of moving to the location and then based on the state of the world make a plan. It will check if the location and path to the location is clear then it will plan to move there but say there is a door in the way than it will need to plan to move to the door than open the door and then move to the location again but what if again now there was another path without a door blocking. This is where the scoring system comes in and the A\* search algorithm. The path will be scored based on the actions which in this case will be distance to travel and opening the door. Then the algorithm will go through all choices and pick the least costly one to perform. Then once all actions are picked they get assigned in the correct order working from the end of the goal to the start giving the NPC the plan they need to fulfil their goal.

Once the plan is made each action is gone through one by one through the FSM allowing the NPC to move or play an animation such as opening a door and interacting with that object.

GOAP just like BTs is very modular with how it is made. If the NPCs in the game ever get new content allowing them to perform new actions all that needs to be added for the NPC to work is new actions and goals to their set along with the animation that action performs and scores for the algorithm to use. Even then this is only required for unique actions as if you were to just add new weapons than all that needs to change is the animation assigned to the shooting action when the NPC has that weapon.

GOAP can also be used very easily between all NPC in the game as the actions are unconnected they allow all the NPC to just use the same planner with their set of actions. For example, in F.E.A.R. there is mice that use the same planner as regular combat enemies except the set of actions and goals they use only require them to move.

## 2.4. Comparison

### 2.4.1. Advantages of BTs over GOAP

* **Low Complexity –** Easy to start up and use especially with big game engines like Unreal Engine having high-quality built-in implementations that are free to use. Many Blueprints implementations also use visual nodes making them easy to look at and understand with Unreal Engines version being able to be done with their visual scripting so you do not have to even touch code. GOAP on the other hand is complex to set up and only really benefits the game when the AI has many actions it needs to perform making it difficult to set up and not worth using for most smaller games.
* **More Control Over Design of AI –** GOAP AI makes the m be able to plan out their own moves and be very unpredictable which while it has its advantages it also makes it so that the AI may not make full use of the design of the game making it pointless while with BTs you have full control over what the AI does and they can be setup to compliment the design of the game and allow the designers to craft the most enjoyable experience possible.
* **Easier Debugging –** GOAP unlike BTs is a lot more difficult to debug because of how unpredictable it is and because of everything being unconnected while BT debugging is a lot easier because of its lower complexity and control the developer has.
* **Smaller Computational Overhead –** GOAP is a lot more computationally expensive as while in BTs you have a defined set of transitions to check and follow while the BT runs GOAP on the other hand must make plans for every NPC constantly checking over all the possible actions and calculating the best one and whenever there is a change they all must be recalculated again.

### 2.4.2. Advantages of GOAP over BTs

* **Easy Scaling –** While GOAP is more difficult to implement and start up for bigger games it can be used to scale the game easier as when big games are made and have more content added BTs and FSM need a lot of reworking for all the transitions to work with each other correctly while with GOAP you would need to just create the appropriate actions and costs assigned with them and they will work with new content a lot easier because of its adaptability.
* **Adaptability –** GOAP allows NPC made with it to adapt to most situations. The AI can plan for most changes made and any new areas added to the game that does not make use of new mechanics and can make effective use of the area without needing any changes while BTs will require a lot of extra work for the AI to make use of the new area or mechanics properly. This also allows for user generated content to work with the game better as the user will not have to edit the AI for it to work with what they make and allow them to make more creative content with less restraint.
* **Replay Ability –** The unpredictability of the AI as they make their plan and player interaction changing that plan in an infinite number of ways leads to the game being infinitely more replay able than as it would be mathematically impossible to get the same encounter twice while Ais made with BTs can only have so many ways they can act especially when designed to take the map into account. This make games developed with GOAP able to be played many years and playthroughs down the line and still lead to interesting encounters and can still challenge the player while with BTs you will be able to predict most things ahead of time and the experience will get less enjoyable and more tedious a lot faster.

## 2.5. Literature Review Conclusion

In conclusion both methods have their strengths however behaviour trees are more common and have become the industry standard because of their modular design, ease of use and low complexity while goal-oriented action planning is a lot more complex, more difficult to use and start up making it terrible for most games but very good for giant games where the Ai gets very complex and well-done implementations can make a game endlessly replay able. So, for most games behaviour trees are usually the best option and allow for very creative and designer friendly AI but bigger games that can make proper use of goal-oriented action planning can make for timeless and endlessly replay able games.

The fact that each method has its own strengths, weaknesses and there is no clear comparison between the methods as the games made using these methods are not the same and therefore cannot give a clear comparison between the two methods making the answer as to which is better not clear which is the reason for this project. The projects aim is to get more clear information by creating a small game and implementing two identical versions of AI NPCs into the game with the use of behaviour trees and goal-oriented action planning and tracking the process to try to get a better first-hand comparison of the two AI creation methods.