

**This is the work of a student with specific learning difficulties.**

**Please mark in accordance with the guidelines.**

**Name/Student No. (whichever to be used to identify this assignment):**

**Stuart Andrew Leslie Hayes - 20363714 / CE**

**………………………………………………………………………………………**

**Course:**

**BSc (Hons) Comp Games Development**

**……………………………………………………………………………………….**

**Subject (if Combined Honours)**

**…………………………………………………………………………………………**

Getting AI to behave, Behaviour Trees for AI

Stuart Hayes

14 April 2019

A project report submitted in partial fulfilment for the degree of

Bachelor of Science in Computer Games Development

School of Physical Sciences and Computing

University of Central Lancashire

Abstract

Creating Artificial Intelligence (AI) can grow out of size and unorganised causing time and money during the development process. AI can grow enormously for very complicated system and require good organisation to continue working on that system.

The aim of this project is to create a behaviour tree that can help developers create an AI system that is modular, organised and has tools to simplify the process of creating and developing an AI to increase the quality produced within a smaller time span.

The project that I have developed makes it easy to import a behaviour tree into an existing project and with a few lines of code create a tree as complicated as the developer needs. The project makes it easy for the tree to execute the developers own code elsewhere within the project making easy to organise and keep code modular for reuse. The project also contains a graphical interface that helps the developer debug the behaviour tree and also produce something graphical to make it easier for other less technical people to understand what the behaviour tree is doing.

The behaviour tree works well, though I do have some ideas that would help the behaviour be easier to use by including some extra functions that allow a developer to do the same process but in a different way.

Attestation

I understand the nature of plagiarism, and I am aware of the University’s

policy on this.

I certify that this document reports original work by me during my University project.

**Signature** **Date**

Acknowledgements

I would like to thank Dr. Gareth Bellaby from the School of Physical Science and Computing at the University of Central Lancashire as my personal advisor for this project and for feedback throughout.

I would like to acknowledge Callum Leithes, David Barcley and John Green for personal support throughout this project.

Table of Contents

[Abstract i](#_Toc6070026)

[Attestation ii](#_Toc6070027)

[Acknowledgements iii](#_Toc6070028)

[Table of Contents iv](#_Toc6070029)

[List of Figures vi](#_Toc6070030)

[List of Tables vii](#_Toc6070031)

[List of Listings viii](#_Toc6070032)

[1 Introduction 1](#_Toc6070033)

[1.1 Background and Context 1](#_Toc6070034)

[1.2 Scope and Objectives 1](#_Toc6070035)

[1.3 Achievements 1](#_Toc6070036)

[1.4 Overview of Report 1](#_Toc6070037)

[2 Literature Review 3](#_Toc6070038)

[2.1 Introduction 3](#_Toc6070039)

[2.2 AI Programming Techniques 3](#_Toc6070040)

[2.2.1 Behaviour Tree 3](#_Toc6070041)

[2.2.2 Finite State Machine 6](#_Toc6070042)

[2.2.3 Hierarchical Finite State Machine 6](#_Toc6070043)

[2.2.4 Subsumption Architecture 6](#_Toc6070044)

[2.2.5 Teleo-Reactive Programs 7](#_Toc6070045)

[2.2.6 Decision Trees 7](#_Toc6070046)

[2.3 Game Engines 8](#_Toc6070047)

[2.3.1 Unity 8](#_Toc6070048)

[2.3.2 Other Common Game Engines 8](#_Toc6070049)

[2.3.3 Summary 8](#_Toc6070050)

[3 Project Planning 10](#_Toc6070051)

[3.1 Introduction 10](#_Toc6070052)

[3.2 Methodology 10](#_Toc6070053)

[3.2.1 Waterfall 10](#_Toc6070054)

[3.2.2 Software prototyping 11](#_Toc6070055)

[3.2.3 Incremental build model 12](#_Toc6070056)

[3.2.4 Agile software development 13](#_Toc6070057)

[3.2.5 Method 14](#_Toc6070058)

[3.3 Requirements 14](#_Toc6070059)

[3.4 Potential Solutions 15](#_Toc6070060)

[3.5 Tools and Techniques 15](#_Toc6070061)

[3.6 Legal and Ethical Issues 15](#_Toc6070062)

[3.7 Summary 16](#_Toc6070063)

[4 Design 17](#_Toc6070064)

[4.1 Introduction 17](#_Toc6070065)

[4.2 System Design 17](#_Toc6070066)

[4.2.1 Nodes 17](#_Toc6070067)

[4.2.2 Behaviour tree 20](#_Toc6070068)

[4.3 User Interface Design 20](#_Toc6070069)

[4.3.1 Creating the tree 20](#_Toc6070070)

[4.3.2 Debugging 21](#_Toc6070071)

[4.4 Summary 23](#_Toc6070072)

[5 Implementation 24](#_Toc6070073)

[5.1 Introduction 24](#_Toc6070074)

[5.2 Decision Tree 24](#_Toc6070075)

[5.2.1 Console Proof of Concept 24](#_Toc6070076)

[5.2.2 Real Situation Unity Build 25](#_Toc6070077)

[5.2.3 Final product 25](#_Toc6070078)

[5.3 Summary 25](#_Toc6070079)

[6 Test Strategy 26](#_Toc6070080)

[6.1 Introduction 26](#_Toc6070081)

[6.2 C# Unit testing 26](#_Toc6070082)

[6.3 Live testing 29](#_Toc6070083)

[6.4 Summary 31](#_Toc6070084)

[7 Evaluation, Conclusions and Future Work 32](#_Toc6070085)

[7.1 Project Objectives 32](#_Toc6070086)

[7.2 Self-Evaluation 32](#_Toc6070087)

[7.3 Project Evaluation 33](#_Toc6070088)

[7.4 Applicability of Findings to the Commercial World 33](#_Toc6070089)

[7.5 Conclusions 34](#_Toc6070090)

[7.6 Future Work 34](#_Toc6070091)

[References 35](#_Toc6070092)

[Appendix 1 – Project Proposal 36](#_Toc6070093)

[Appendix 2 – Technical Plan 37](#_Toc6070094)

[Appendix 3 – Title of Appendix 38](#_Toc6070095)

List of Figures

[Figure 1 - Flow diagram of the waterfall model (Rovce, 2019) 10](#_Toc6070019)

[Figure 2 - Diagam of Incrementak Model (Tilloo, 2016) 12](#_Toc6070020)

[Figure 3 - Node structure 17](#_Toc6070021)

[Figure 4 – Graph of a running behaviour tree 22](#_Toc6070022)

[Figure 5 - Results of Unit Testing 28](#_Toc6070023)

[Figure 6 - Prototype Demo Screenshot 29](#_Toc6070024)

[Figure 7 - Screenshot of the agent in the final product 30](#_Toc6070025)

List of Tables

[Table 1 - Inverter Node Test Results 28](#_Toc6070009)

List of Listings

[Listing 1 - [Program.cs] Example of creating a behaviour tree 20](#_Toc6070014)

[Listing 2 - [DecoratorNodeTesting.cs] The testing for the invert node 27](#_Toc6070015)

# Introduction

## Background and Context

With AI there are several approaches to programming and organising code. Especially when it comes to video games AI predominantly present and can range from simple to extremely complex and used a few times to many per frame.

While AI is present it is not the only process running and is often only assessed when it is broken or when they do something that amazes the user.

This project investigates behaviour trees, their advantages and disadvantages, alternatives and when they should be used and produce a working one that can be used in future products.

## Scope and Objectives

* Create a behaviour tree that is simple to use and can just be imported and used to help developers’ program for AI
* Simple interface that allow the developer to create nodes to make up the tree structure for the behaviour tree.
* Graphical interface for the state of the behaviour tree for debugging and reports.
* The behaviour tree can run as often as the developer wants and can be run multiple time per frame for games.

## Achievements

The completed project shows a behaviour tree working and easily implemented into a game engine that is used commercially and show it running. The behaviour tree is simple to import and use. A behaviour tree is also a technique used within the game industry for AI and code organisation.

## Overview of Report

**Chapter 2** reviews literature around AI programming techniques on behaviour trees including how they work, advantages, disadvantages and when they should be applied, and a look at some alternatives to behaviour trees. This section then has a quick overview of game engines which behaviour trees get commonly used.

**Chapter 3** looks at methodologies and which is best appropriate for this project, then looks at the project’s requirements, tools wanted and any problems to consider.

**Chapter 4** looks at the design of the behaviour and its nodes. It then goes onto talking about the user interface for creating a behaviour tree and then debugging it.

**Chapter 5** show the implementation process of the project breaking it down into sections and clearly marking what will be done.

**Chapter 6** breaks down the testing into sections and explains what was done and report of findings.

**Chapter 7** evaluates the behaviour tree created from this project and the project itself. I also reflect on what was learn from this project through its process and then investigate future work that could be done with the produced behaviour tree.

# Literature Review

## Introduction

This chapter explores the available literature surrounding this project. This literature review will cover the relevant topics for this project, for example, AI Programming Techniques (mainly focusing on Behaviour trees the subject of this project) and Game Engines, development platforms where this technique may be used.

## AI Programming Techniques

There are many techniques that can be used to program AI, this report will look at some and focus on behaviour trees. Each technique has it befits and its negatives as well, further research it to these techniques will help developers use the correct technique for the appropriate problem.

### Behaviour Tree

As games developed over time, computer games designers where making larger and more complicated NPC’s in their games. At the start of the 2000’s Dromey (2003) developed a behaviour management system to manage individual components in a tree like structure, the behaviour tree. This new system was quickly picked up by game developers and used in the AI systems for NPC, most notably in Halo 2 (Isla, 2005).

Scheper, Kirk Y. W. (2014) claimed; notable traits of a behaviour tree are the use of hierarchy simplifies the code, aides in its reuse and increase in character behaviour. Programming the character by task for the nodes in the behaviour tree helps divide the entire character’s behaviour into simpler tasks. Recursive actions by the character are easier to create. Graphical charts give users a good overview with little complexity.

#### Behaviour Tree Structure

“BTs are depth-first, ordered Directed Acyclic Graphs (DAGs) used to represent a decision process.” (Scheper, K. Y. W., 2014) Each node can return a single status; success, failure or running, these are the minimum required status, but some systems may contain extras.

The three major node types are; composite, decorator and action nodes (Epic Games, Inc 2019). Composite nodes contain a list or sequence of child nodes. Composite nodes will process the list of nodes it contains and react properly given the result of its children, for example if a child returns success the node may return success, failure or process the next child. Decorator nodes are similar to composite nodes with the exception of it only contains one child node. The child node’s result influences what the decorator node returns similar to how a composite node reacts. The last standard node is action node, a leaf node, which the developer write code, that node will then run the given code to get a result and return it, usually to its parent.

#### Running A Behaviour Tree

Behaviour trees are flexible and can be ran as much as the developer wants, a tree can be designed to run per frame or just to be run when an update is required. A behaviour tree simply processes the dedicated root node for that tree. The tree calls the root node and that node then does what it needs to do. That node may contain children and those nodes will be called so the currently running node runs down the chain like a tree until it hits an action node, which where the action node will run its code. The node will then return a result from the function it calls and pass it back, usually up the chain to its parent, which is when the parent will process that result and may call the next child in the sequence until it gets a result it wants off a child or runs out of children. This process repeats until the root node gets a result.

Behaviour trees use a status called running which is when a process requires more than one tick to complete, in this time the tree returns running, the tree can be called again to check if the current process should still be running. If something else with a higher priority in the tree should be running, then the tree will call that function giving a hierarchy to its processes which if programmed correctly can interrupt what it was doing to do the more important task. Given this feature of behaviour trees, a tree wants to be called as often as possible to check if an update is needed. Updates are up to the developer, this can be as often as every tick, frame, on a timer or when a change has taken place. Reducing the number of times, the tree is updated reduces how reactive the tree can be, but this is program dependant.

#### Advantages

* Modular, code is written in parts so a single node can run that small part of the system. This means that the code is broken down into small sections, making it very modular and it can be swapped in and out or moved around very easily.
* Human readable, since the code is written for singular node, code is written in chunks and breaks down the entire behaviour into sections making it more readable. Visual representations of the behaviour tree are also more human readable since the diagram of the tree is depth first search the tree can be easily followed. If an instance of a tree has been taken you can see what the tree has done and by colour coding the nodes based on that nodes status and you can follow from the tree top of the tree down and see what choice was made at each node, which nodes where ran and which path the tree took.
* Reactive, behaviour tree’s can react to situations thanks to its running status. If a tree runs and returns a running command it means that it is in the process of completing its task, the tree can be run again before the task is complete and check if it should still be running that task, if not it can interrupt itself and run the new task that has a higher priority with the given variables.
* Hierarchical, the tree is processed in order. Starting off with the root node, then it calls its children in order, where the first one has priority.
* Reusable code, since the code has been broken down into snippets or smaller sections, some code could be reused across multiple objects in the project or even cross projects.

#### Disadvantages

Complexity, creating the tree into nodes makes each section much more readable and understandable but it can be complex to break down a larger system into smaller chunks and arrange it into a behaviour tree. This is another way of thinking of the same system which can be tricky to get out of your current mindset into a mindset that works with a behaviour tree.

Behaviour tree can also be expensive, each node has its own overhead, each node is its own class which is inherited from the node type which can take up memory which could be avoided with well programmed code.

Sometimes a behaviour tree is not needed, small AI or systems are just simple enough that the benefits of a behaviour tree would not help that system.

### Finite State Machine

A finite state machine (FSM) is another approach to AI, key characteristics are it has defined states, there is a finite amour of inputs, outputs and events that cause state transitions, the system behaviours are based on events at any given time, each state has its own defined behaviour based on inputs or events and there is an initial state (Wright, D. R. 2005), behaviour trees also have these characteristics as well.

This approach is simple and can be efficient but starts to become difficult to manage, as the system grows large, it starts to get difficult to organise, starts to get less readable and harder to reuse. A large finite state machine becomes a hard task to understand so progress on the system slows down.

### Hierarchical Finite State Machine

Hierarchical finite state machines (HFSM) add another later to a state machine helping reduce logic. Adding this layer reduces the number of transitions and in turn helping modularity in code which helps with ease of readability (Girault et al, 1999). While this does help with some of the problems of FSM they still struggle with organisation while the system has been cut down into section it’s still an FSM that can still be hard to track what is doing what and is not easily readable.

### Subsumption Architecture

Subsumption architecture is a reactive robotic architecture but is heavily associated with behaviour-based robotics. This architecture takes in information via sensors and relays that to multiple controllers, these controllers run their own task and output the results which in turn operate actuators. Yufeng (2010) states that the controllers run in parallel to each other and have a hierarchy so that the top layer can override those below it, giving benefits as robustness, concurrency support, incremental construction and ease of testing. This means code is much more modular and uses a hierarchy give an order priority but unfortunately this architecture struggles with scalability as actions are more complicated designing the system can become difficult and a lack of structure means changes can have unintended consequences. While this is a more specialised AI system for robotics this is another approach and could be implemented in code running parallel tasks with techniques like multi-threading can help reduce the performance cost of running multiple task simultaneously.

### Teleo-Reactive Programs

Teleo-reactive (TR) programs use a list of prioritised condition-action rules that drive an agent to achieve its goal (teleo) based on the current environment (reactive). Nilsson (1994) in his article introducing TR programs, he claims they are intuitive to program as all condition-actions are in an ordered list and thanks to the monitoring of environment, reactive. Issues arise with maintainability as the list of rules grow large since consequences of editing a condition-action may affect others in the list.

### Decision Trees

Decision tree are very similar to behaviour trees, but they are essentially if else statements, each node represents a ‘test’ which has at least two outputs. The tree has a root node this will be its first test and gets given an output which sends the tree down another branch, this continues until a leaf node is hit and it runs a task. This is very similar to behaviour tree in many ways, they gain some of the same advantages, for example modularity, code is written into sections independent of the tree, it is hierarchical as the top of the tree get higher priority nodes and the tree is human readable, Bacciu (2015) claims that a behaviour tree is had a strong structure and easy to understand graphical representation, which will help with other that are not as literate with the software. The main drawback of a decision tree is the lack of information in the nodes. These nodes do not contain any status so debugging becomes a lot harder when something does not go correctly.

## Game Engines

This project uses Unity, which is a game engine, to test and demonstrate a running behaviour tree. This section will elaborate on game engines, unity the chosen engine and other possible game engines.

A game engine is a software that is designed to assist the development progress of video game development. A game engine abstracts common game-related task, for example, rendering, physics and input so developers can focus on their game. (Ward, J. 2008)

### Unity

Unity is a game engine created by Unity Technologies. Unity allows the creations of games quickly and on multiple platforms. Unity is the chosen engine for this project as it can use C# as a programming language, its simplicity in starting a project and it is a game engine that is used in game studio like Ubisoft (Unity Technologies, 2019). In this project unity will be taking care of most of content not relevant to this project, for example rendering and sound.

### Other Common Game Engines

Unreal is another game engine that was up for consideration. Unreal is another engine that is currently relevant at the time of write which is also used by current game studios but was not selected due to no prior experience with this engine and would reduce development time by requiring to learn how to use that engine.

Other notable game engines are;CryEngine, GameMaker Studio, Godot, Leadwerks and Source. There are many other game engines not listed.

### Summary

From this literature review there are many ways to program AI, this report mainly looks at behaviour tree. While behaviour trees are a useful technique for the AI development process due to is organisation and readability, they are not always the go to way of programming AI as sometimes the AI is simple enough that the use of a behaviour tree doesn’t increase readability and organisation. But bigger projects with larger AI’s behaviour trees are recommended the benefits of a behaviour trees really shine with organisation and readability but it can be complex to program and developers may struggle to get into the correct mindset to program with a behaviour tree.

# Project Planning

## Introduction

This section will consider what methodologies that are used in code development, looking at advantages and disadvantages at each approach and decide which is the most appropriate for this project and apply it. Requirements are then considered for a behaviour tree and break down what is needed. Any potential problems are flagged and tackled with this project to ensure these are solved as early as possible to encourage a smoother workflow. Planned tools to be used are then stated and then a look at any ethical or legal issues that could cause problems are discussed.

## Methodology

Multiple methodologies where considered for this project, this section will talk about the following; Waterfall model, Software prototyping Incremental build model and Agile software development and why they were considered.

### Waterfall

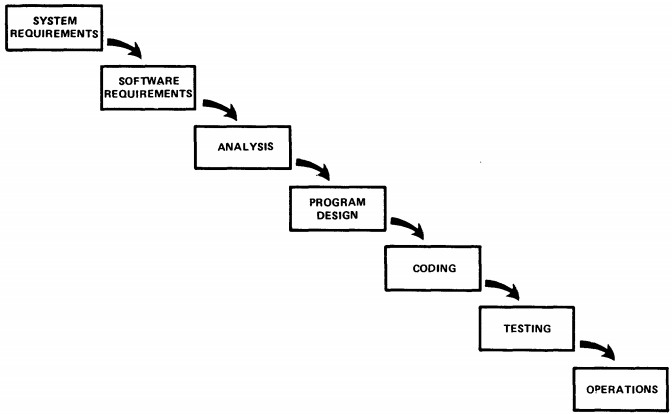


Figure 1 - Flow diagram of the waterfall model (Rovce, 2019)

Waterfall is the oldest and most well-known methodology. The basic premise of waterfall is a design than flows from one stage to the next and follows the following stages; Requirements, Design, Implementation, Testing and Deployment.

The first stage is where all requirements for a system or in this case project are documented and written up this can take time but gets all the current requirements needed. The second stage is to use the requirements and design the entire project from start to finish, this sets up the next stage, which is implementation, developers will be given the design documentation which will tell them how the system shall be programmed and program it. Once the system has been fully programmed it will then be tested for any bugs or defects and they will be fixed and then the system is complete. The project will then be deployed according to the requires/design then long-term support takes over.

This methodology has great theory behind it but can fall short in real world situations, for example, the documentation of this approach is great, but this documentation is from the start and situations can change or requirement can change meaning a lot of documentation can become obsolete and time spent on this is considered a waste. One of the core problems is that the waterfall method only flows one way and when there is a change to the system that requires a previous stage to be reworked, the next stages can’t be worked on until the previous stage has been complete which leads to a lot of waiting around and the project coming in overtime and over budget. “Once you've completed a phase, its results are frozen. You can't go back and revise anything based on changing needs or fresh insights.” (Weisert, 2003)

The waterfall method can be great for large projects where everything can be predefined so there is no backtracking and it gives great documentation but for this project it is on a smaller scale where not everything can be known from the start.

### Software prototyping

Software prototyping is a methodology in where the program is being completed in stages, the purpose of this approach is that software can be developed from the start and an early prototype of the product can be made as shown to the client so that any problems and/or missed requirements can be discovered early and solved (Komatineni, 2016), this means that client is more involved in the project which gives more trust to the developer and the clients feels like they are more part of the team and any undiscovered problems with a project can be found earlier and solved before hitting the testing phase like in a waterfall based model. While prototyping does have advantages, it does also have some problems to this approach, documentation can be lacking since aspects of the project can change documentation is left towards the end of the project and may not be as complete. Other problems can be that clients believe a prototype is the final product but just needs polish and developers can grow attached to a prototype they made since they put time and effect into making it and there may be a better method or approach but the developer will not want follow that new approach.

### Incremental build model

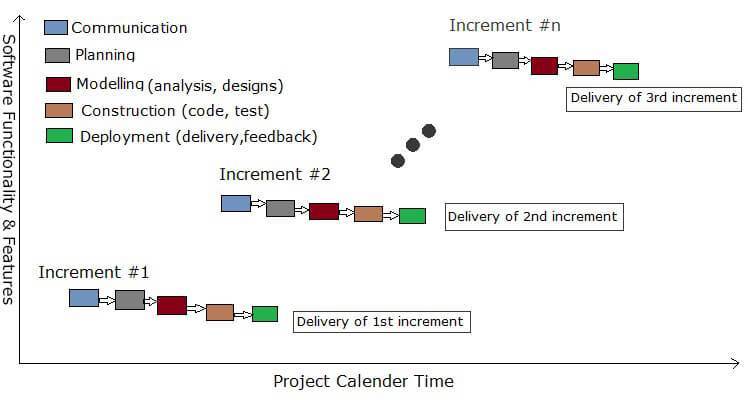


Figure 2 - Diagam of Incrementak Model (Tilloo, 2016)

The incremental build model is another approach to developing software where it takes inspiration from the waterfall model and the software prototyping. In this method the waterfall method is followed incrementally. The system is broken down into increments then each partial system of the final system is developed following the waterfall model, so it goes through requirements, design, implementation, testing and deployment. Once this increment is complete then it is shown to the client, feedback is given, and the process begins again with the user’s feedback in mind. These increments keep happening with the highest priority requirements being incremented first, until the project is complete.

This methodology takes to take the advantages of what the waterfall model gives and merges with the incremental design of software prototyping this mean there is an initial product that the customer can see and feedback can be given, documentation can be better than prototyping as design is taken in each increment and since it is incrementally developed testing can be easier as if it is testing between increment new bugs that are found can be trace back to changes between increments but with the inspiration of the waterfall method it too can also come in running overtime and over budge especially if changes are made while this method tries to reduce this problem it is still prevalent, also if the product gets additional functionality new problems can form that the system was not initially designed to have solutions for. (tryqa, n.d.)

### Agile software development

Agile software development is a popular and different style of development which still uses an iterative design philosophy. Previous development styles have a fixed scope, they then use resources and time to complete the development, agile takes another approach, it fixes resources and time and varies scope so that a product will time within time and within recourses, which previous methods risk.

There are many different types of agile but generally the agile development process follows the following; Plan, Design, Build, Test and Review in a cycle. The team will set a plan and a time frame, then go into the design the system for the allotted time given. The team will build what has been set out in the plan, then it is tested, and the end of the cycle comes up. Just before the end of a cycle the current cycle will be review on what has been done in the time frame and what needs to be done in the next cycle until the predefined end time. This ensure that the customer gets what they want and on time and in cost, then it is up to the client wants more time or the product in its current state it what is wanted.

This method comes with many advantages similar to incremental build model, where the client is close to the development team and incremental builds are made that the client can see but still has drawbacks the product may not be as far long as the client would like and like other iterative methods documentation takes a lower priority and is not as strong as the waterfall method. (Gray, 2018)

### Method

This project will use an agile method since this project has a set time and a set amount of resource that can be spent on this project. This project will use Dynamic systems development method (DSDM) to help develop the behaviour tree. Useful key features of DSDM are timeboxing, MoSCoW, prototyping and testing. The development will firstly be broken into three sections; firstly, a console version running a simple version that contains the core of the project and act as an early proof of concept and review what needs to be done to give a clearer vision of the entire project. The second stage is first test usage in a real-life situation, so version will be implemented into unity and tested. The third stage is the final product and an example of the behaviour tree running in a project.

## Requirements

The core requirements that are required for a behaviour tree are:

* Create the tree, the most obvious requirement for a behaviour tree is that ability to create the tree and trees can be created as many times as needed for the game/project.
* Create nodes, the developer needed to be able to multiple nodes and nodes of a given type.
* Nodes must return success, failure or running. For a behaviour tree once a node is executed it needs to return a status so that other nodes can react to that status.
* Feed in functions to action nodes, action nodes are nodes that give the developer the ability to add their code to the tree, the developer must be able to make code and insert it into an action node for it to be executed when the node is called.
* Composite and decorator nodes link to other nodes, these nodes react from the return status of other nodes therefor they need to be able to call other nodes to be able to get a return.
* Create the tree structure, once nodes are created and link to each other the beginning of a tree structure is there, but a tree requires a base node.
* Run the tree, the final requirement for a tree is the ability to call the tree when the developer wants to call it.

## Potential Solutions

Projects over time can have some challenges with progression especially if other projects are also running concurrently. To help with managing what tasks need to be done there is a potential solution, Kanban. Kanban is a technique where sections of a project is split up to blocks and ordered on completion status and priority, this will keep the developer organised and prioritise correctly (Gross, 2003). Another technique to help keep track of progression is weekly reviews, this helps bring the developers into the overall position of the project rather than what they are currently doing and making sure progress is being made and correctly.

## Tools and Techniques

This project is being developed with Unity in mind as the game engine, this was chosen as cost of entry is free for products under $100,000 while this product is intended to be sold the initial cost of free and if it takes of then only after a cap has been reached licencing fees take place. Visual studio is the integrated development environment (IDE) as it is a common IDE and the developer’s IDE of choice with features like intellisense. To manage version control and consistency between multiple computers, git will be used to backup code.

## Legal and Ethical Issues

As with any project there are a few considerations when it comes to legality or ethical issues. When writing any code, copy write is always considered, researching a project and if any code is shown the writer of that code or the company that person was writing code for has copy write of it and using anything that has not been done yourself without permission and/or proper crediting can lead to serious legal problems.

Another issue that this project could run into is licencing, Unity is a licenced software with requirements. This means we need to consider what the licence allows, for this project it intends to be a commercial product by one developer, under Unity’s licencing agreement there are fee that take place only one this type of product exceeds the cap of $100,000 so for the time being under this licence there is no problems but if this product becomes a commercial success then we will need to pay the licencing fee.

## Summary

This section looks look at what methodologies are available and has chosen that an agile approach is best applicable using DSDM. The core requirements are broken down and clearly stated so that they are fulling understood so better implemented into the project. Potential problems are considered at this stage and recommends solutions to them, Kanban and weekly meeting will help keep progression throughout the project. Tool and techniques state Visual Studio, the IDE of choice, Unity as the game engine and state git will be used for version control and finally any issues that may crop up in this project are considered with Unity’s licence being only a problem if this project becomes a success and a overview of code copywrite.

# Design

## Introduction

In this section I will talk about feature and the design of the behaviour tree and the nodes that make it. This section will also look at user interface of the behaviour tree and useful debugging tools and methods.

## System Design

A behaviour tree can be broken down into 2 main sections the tree and the nodes. The nodes can be sub divided into 3 sections, composite, decorator and action nodes.

### Nodes

From Figure 3 you can see that all nodes inherit from a node class which contain the status of the nodes last execution, the function to execute the node, the ability to get and set id and the ability to get and set the status. While only the run is required in all nodes these extra functions are useful for debugging while setting up the tree.

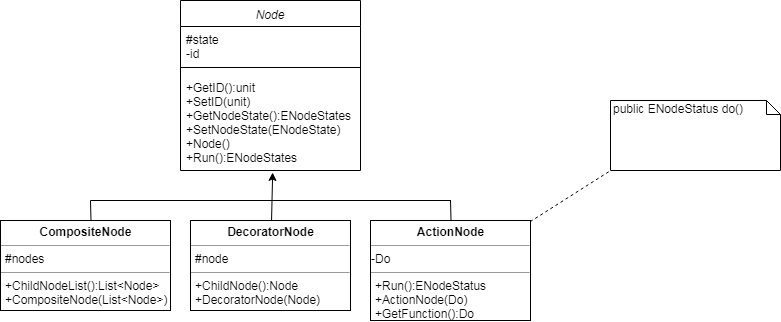


Figure 3 - Node structure

#### Action node

When the action node is ran, it needs to execute a function, one method would be to introduce a messenger in-between the tree and have that action node send out a message for that to be executed and wait for a reply, this would reduce coupling between would slow down the behaviour tree dramatically, this would work with trees that are more passive but not for a tree that is called often. The method this project took was function pointers but since unity will be used the C# equivalent will be used, delegates. Using delegate, we are able instantly call the assigned function instead of waiting for a messenger to execute and then reply, this will increase performance for trees that want to run often.

#### Decorator node

Decorator nodes require a child node to function, this node will store a reference to its child node. There are many decorator nodes for example Inverter node, which simply changes the result it gets and flips the result, so failure is success and success is failure, nothing changes the running status as we want it to pass up and return running to the root node. All decorator nodes will inherit from the decorator node and add their own functionality be overriding the run function so it calls its child node get a status then it can do what it wants.

##### Sequence node

The sequence node is a node that runs through each child until it either; runs out of children and return success itself or if it receives failure or running it just returns failure or running without running another child.

##### Selector node

The selector node is similar to the sequence node but continues while it gets the failure status from its child, it too will run through all children and if they all return failure it returns failure. If a child returns success or running at any point it will not run another child and just return success or running.

#### Composite node

Composite node is similar to decorator nodes, they are nodes that react to many children, which mean this node will contain a list of nodes which it will process sequentially. The node will firstly call a node and then decide on the result of that node if another child node shall be processed or to return a result. The two main nodes implemented are Sequence and Selector. Sequence will run through the children until a not returns fail and returns fail itself or returns success if all children return success. Selector nodes return success once a child returns success else it returns failure. All composite nodes will instantly return running if a child returns a status of running.

##### Inverter node

The inverter node returns the opposite status from its child, so success becomes failure and vice versa. If the child returns running the inverter node just returns running.

##### Repeater node

The repeater node has some extra data in it, an unsigned int, this counts how many repeats the repeater will do. The repeater will run the child as many times as the counter has been set up to. If the receives failure or running it returns that status, else if it receives success it will keep repeat calling the child till the counter runs out.

##### Repeat till fail node

The repeat till fail node keeps running its child if it returns success, if the child returns failure or running it will stop and return failure or running itself.

##### Limiter node

A limiter node is another node with extra data, stored as an unsigned int count how many runs are left. This node runs its child as long as it still has runs remaining, if it does not have any runs left it will automatically return failure without calling its child. If there are runs remaining it calls its child and returns its status.

##### Succeeder node

A succeeder node will call its child and return success if the child replies with failure or success. The only time this node does not return success is if it receives running at that point this node will also return running.

##### Failure node

A failure node is the same as the succeeder node but using the failure status, so if it receives success or failure it will return failure and if it receives running it returns running as well.

### Behaviour tree

The behaviour tree has its own class which contains all the useful functions for the behaviour tree. This contains the functions for creating the nodes since they are stored within the tree and are tree specific. The class stores all the nodes in a dictionary with unique ids for access to any node. The tree stores the root node and starts the execution of the tree from the root node. The tree also contains useful functions like setting the node to idle for debugging purposes.

## User Interface Design

### Creating the tree

TestMethods testFunc = new TestMethods();

BehaviourTree firstTree = new BehaviourTree();

BehaviourTree secondTree = new BehaviourTree();

var T3 = firstTree.CreateActionNode(testFunc.CounterInc);

var T2A = firstTree.CreateActionNode(testFunc.CounterInc);

var T2B = firstTree.CreateDecoratorNode(DecoratorNodeType.Inverter, T3);

var T2C = firstTree.CreateActionNode(testFunc.CounterInc);

var T2D = firstTree.CreateDecoratorNode(DecoratorNodeType.Repeater, T3, 3);

var children = new List<uint> { T2A, T2B, T2C, T2D };

var root = firstTree.CreateCompositeNode(CompositeNodeTypes.Selector, children);

firstTree.SetRootNode(root);

firstTree.RunTree();

Listing 1 - [Program.cs] Example of creating a behaviour tree

The interface for creating a tree is designed for a programmer so it is written in a way they could understand and easily manipulate. The first step is that the developer creates an instance of the behaviour tree, this created the base class with no data to it. The next step is the node of a tree, using the instance of the tree a developer can create a node there are 3 functions one of each type of node for organisation sake. Action nodes will need to take in a function that they will call when being executed, decorator nodes will require what type of decorator node it will be and the id of the node of its child. Composite nodes also require the type of composite node and the list of id of the child nodes. Creating each node will return the id on the node just created which can be passed into other nodes. Once all the nodes of the tree are made the root node needs to be assigned, this is the top level of the tree and the first node to be executed, by the developer using the SetRootNode function. Once a tree has been made it can be saved to a file so that the data is not in the programming file and can be externally edited if wanted. Once the file has been saved the file can be loaded in one of two ways, either give the file path of the saved file to constructor or to create an instance and use the LoadTree function and it will reconstruct all node, correctly assign then and set the root node so that the tree is ready to be ran.

### Debugging

There are multiple ways of debugging the tree, this project has added feature to help a developer debug what is happening with a tree.

#### Dynamic/Static debugging

The most basic way of debug is using static or dynamic, this is a common way to debug. A developer can dynamically step through each node of the tree or use static debugging within the action nodes of the tree.

#### Graphic user interface

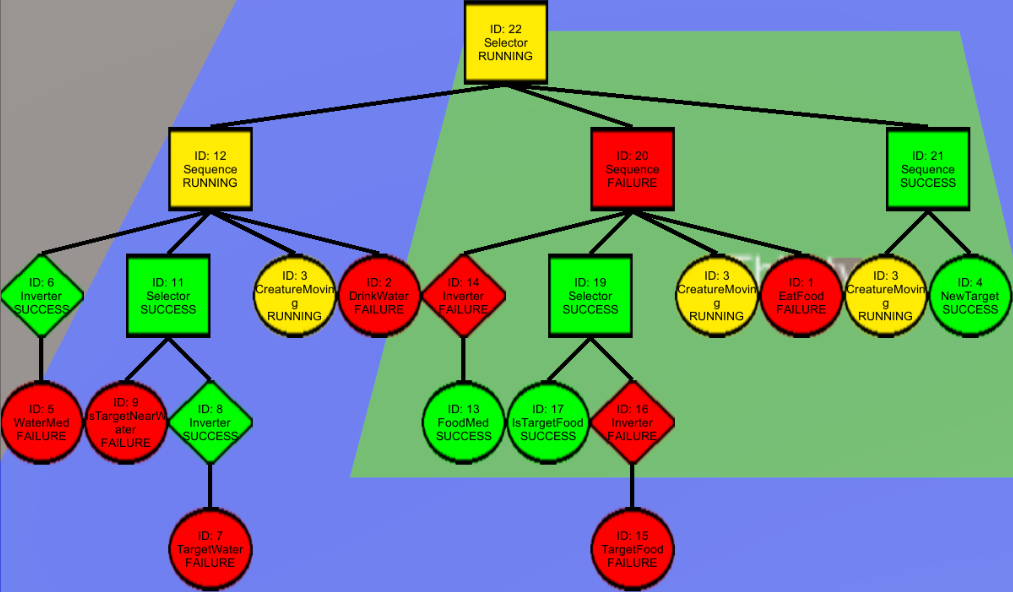


Figure 4 – Graph of a running behaviour tree

A graphic user interface will show the entire tree structure and all its nodes this will help with any misplaced node and check the logical order that the tree will run. The tree is also colour coordinated so that they are coloured by status it helps to see what the tree has done at each node, the developer can follow the tree down and across as see what each node’s result was and follow that happened at each stage to see if that was the expected result. This also is a great way to show those who are less technical with behaviour tree to understand what is being done.

#### Idle nodes

Another feature add to the behaviour tree is idle node status, this will turn all nodes in the tree to idle so that it is easier to see which node has or has not been executed and see what composite nodes touch. This does decrease performance due to unnecessary update of status in the nodes, but this feature can be turned off for a release version.

## Summary

In this chapter I have explained my design of the behaviour tree and its features. These designs will help give a prototype version and check that all features are in the final version.

# Implementation

## Introduction

In this section we will take a look at how the project was implemented. Going throw the 3 main stages of the development from a console proof of concept into a unity build prototype and finally into a more realistic and complex situation to ensure no bugs are in the code and to show what can be done.

## Decision Tree

To implement the development the behaviour tree was created in three main stages. Firstly, a proof of concept that shows that a behaviour tree can be build and start testing on with little external factors so that any problems that can turn up. The second stage implements the behaviour tree into a unity prototype and extra feature start to get implemented. Once the pervious stage is considered complete the project enters the last phase, here the behaviour tree is made more complex to further bug testing and extra functions are finished in this phase.

### Console Proof of Concept

Thanks to defining in the design stage that we would be using unity as the game engine later, that predefines that C# will be a programming language later on so it would be best use the language from the start. The author mainly uses C++ the difference in languages are minimal only a small amount of revision and research is needed to start implementation. The core difference between the two languages are function pointers and syntax for inheritance.

C# uses delegates instead of function pointers as it adds encapsulation and safety to pointers which are some of the core concepts of C#. Microsoft, the designers of C# have delegates well documented and it did not take long to research this new implementation method. The other research was on syntax for inheritance while the implementation was similar some keywords have changed that is too is well documented online.

To tackle implementation, it was a choice of which section to start the behaviour tree or the nodes. In this project, nodes where the first to be chosen as they can be tested without the behaviour tree part. A separate file for just the nodes was used to keep the project organised. The following nodes where implemented, action node, composite nodes: sequence and selector, decorator nodes: inverter, repeater, repeat till fail, limiter, succeeder and failure. These nodes require the Node abstract class this is a simple node that contains the abstract function run, which is the function that node nod0es call to be executed, it also store basic information as id and other functions to get and set the id. Once all nodes where implemented the basics of the behaviour tree where then developed, the basic version of the tree was to use the tree to create the nodes and store them, set the root node and run the tree. Once these functions where implemented unit testing began on each node and on the tree. Once all tests where passed development went onto the next stage.

### Real Situation Unity Build

In the second stage the target was to get a simple stage with some random cubes in the map, these cubes where assigned a type relative to the agent’s stats. The agent will have 3 values, which are food, water and heat, these just map to a numerical value. The behaviour tree would check against each of the three values and if a value drops below a certain threshold the agent would find the nearest cube that would increase that value and move over and pick it up.

At this stage other feature of the behaviour tree get implemented, load and save get implemented here. Further testing was done here by using the behaviour tree in a real situation. Once this prototype was considered complete the graphical user interface was loosely implementing, this is the first version of the GUI. The GUI was a simple version just to help debugging, this will layout the tree and write info to the nodes.

### Final product

The final product is an example of a game, this game is a simulation game for example, the sims and species. This game has a creature survive as long as possible; at this stage the creature will check if it has any requirements if it does it will search for them. The creature can only walk on land so it can not enter the water, the creature will need to move over the land to see what there is, the creature now has a cone of vision limiting what it can see forcing it to explore more The creature will have a water and food value, the creature can eat grass to increase its food and drink from water but I cannot enter the water to it will need to see if the grass tile has water next to it and then drink the water from that grass tile. If the creature does not have any requirement it will explore its area.

At this point the tree should be well tested but this is another scenario will help with any missed bugs. The tree visualisation gets fully implemented at this point, this shows the behaviour tree over at any point with colour coordination and information for each node to make it easy for a developer to debug and anyone to understand what the tree did last and watch it update the nodes per each run of the tree. Once a visualisation is implemented then the ability to set the nodes idle is added which will help with debugging with the use of the graphical implementation of the behaviour tree, adding idle nodes increase clarity of the nodes as untouched node displaying their old status works for the tree can cause some confusion for people when they look at a the tree and see that a node that has not been touched has a status which is the last status it had.

## Summary

This section breaks down the implementation process into three sections. The first section sets up any research that was needed before implementation and the decides on the development order, each section explains what is implemented at that stage. The first stage implements a simple console version testing that the node and trees work, the second stage implements the behaviour tree in a unity prototype level once it is implements a small test is added to check the tree, then finally the last version is made that fully implements all feature including the GUI and further test the behaviour tree to find any missed bugs.

# Test Strategy

## Introduction

Testing is a focus on this project with testing beginning, the testing is broken down into three sections, unit testing which will be used from the start as soon as a node is implemented. Once all the basics of the behaviour tree are completed the testing will go onto using the product in a real-world scenario to find anything else missed from the unit testing.

## C# Unit testing

[TestMethod]

public void FAILURE1()

{

myTestsFuncs.counter = 9;

myTestsFuncs.target = 10;

ActionNode IncAction = new ActionNode(myTestsFuncs.CounterInc);

Inverter test = new Inverter(IncAction);

test.Run();

Assert.AreEqual(NodeStates.FAILURE, test.NodeState);

}

[TestMethod]

public void SUCCESS1()

{

myTestsFuncs.counter = 0;

myTestsFuncs.target = 10;

ActionNode IncAction = new ActionNode(myTestsFuncs.CounterInc);

Inverter test = new Inverter(IncAction);

test.Run();

Assert.AreEqual(NodeStates.SUCCESS, test.NodeState);

}

[TestMethod]

public void SPEED()

{

myTestsFuncs.counter = 9;

myTestsFuncs.target = 10;

ActionNode IncAction = new ActionNode(myTestsFuncs.CounterInc);

Inverter invert = new Inverter(IncAction);

for (int i = 0; i < 10000; ++i)

{

invert = new Inverter(invert);

}

invert.Run();

Assert.AreEqual(NodeStates.FAILURE, invert.NodeState);

}

Listing 2 - [DecoratorNodeTesting.cs] The testing for the invert node

Once programming of the nodes began testing of the nodes when in mind. Nodes where tested upon completion to see if they correctly return success or failure given a situation. Once the full behaviour tree was completed that was also included in each test as it is the intended way to create and use the nodes. Each node passes its own tests with different expected results.

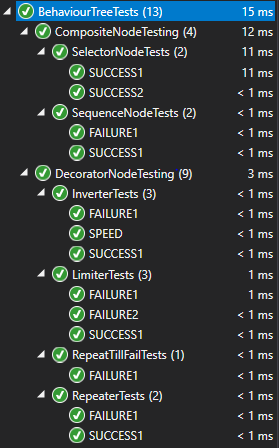


Figure 5 - Results of Unit Testing

The results shown in Table 1, the testing of the inverter node, the point of the tests where to check for accuracy rather than speed, the tests labelled success and failure just run the inverter node once to call an action function that just adds one to a value while the speeds node does the same thing but inverts the results 10,001 times. Given these results we can see that running a node has little cost and increasing the nodes used from 1 to 10,001 does add some performance but each run of a node is less than a microsecond.

|  |  |  |
| --- | --- | --- |
| Test name | Result | Elapsed time |
| Success | Pass | 0:00:00.0001159 |
| Failure | Pass | 0:00:00.0001743 |
| Speed | pass | 0:00:00.0005487 |

Table 1 - Inverter Node Test Results

Though performance has been a deciding factor on how it is programmed, nodes only require a few lines of code per node. Where performance will matter is in the action node since this where a developer can link to his code and dependent on what the node is doing it can be very fast or very slow.

## Live testing

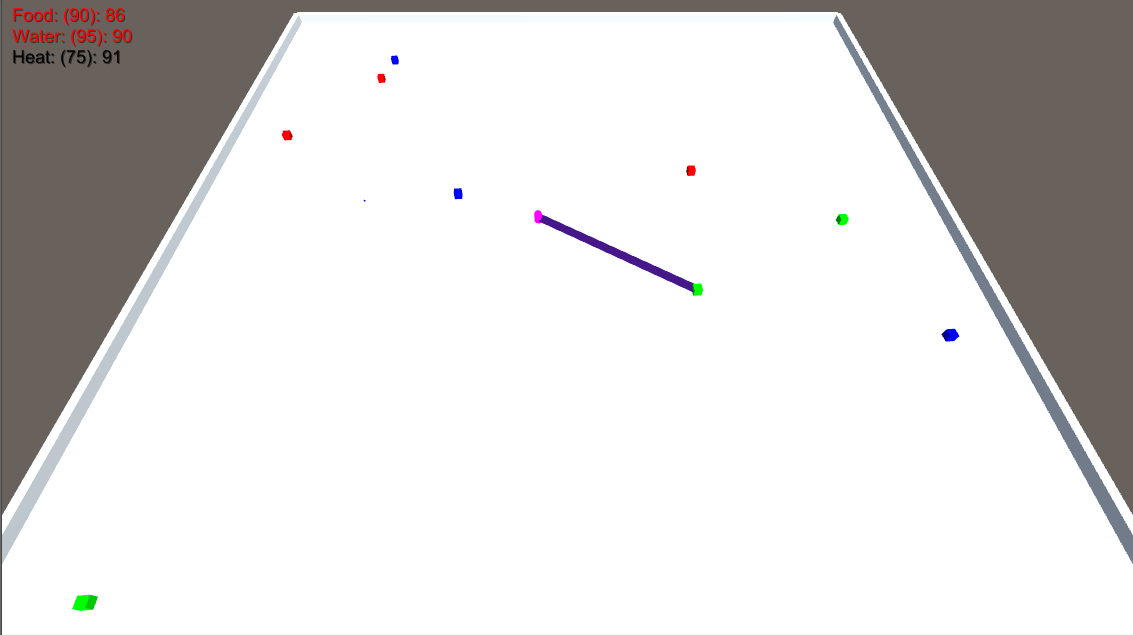


Figure 6 - Prototype Demo Screenshot

The next stage of testing is to try and use the behaviour tree in a real-world scenario. The real-world scenario is to use the prototype game, the first instance was a simple move and collect pickups scenario, the goal was the AI would check if its value of one of three attributes where low, (in figure 6, top left stated the attribute, red text meaning low and black meaning sufficient) if a stat was low it would iterate through the list of cubes finding the closest cube that would give the value that was low. The stats are in a priority order so that the food stat was always top priority, water was second, and heat was third. If the agent had two stats low it would go for the one with the highest priority, if a higher priority state became low the agent would stop going after its current target and go after its new target. The cubes where colour coordinated with green for food, blue for water and red for heat. If there were no stats needed the agent (pink) would sit idle but if the agent had a target it would create a beam between it and the target. No bugs where had making this scene so the project then moved onto the next scenario.

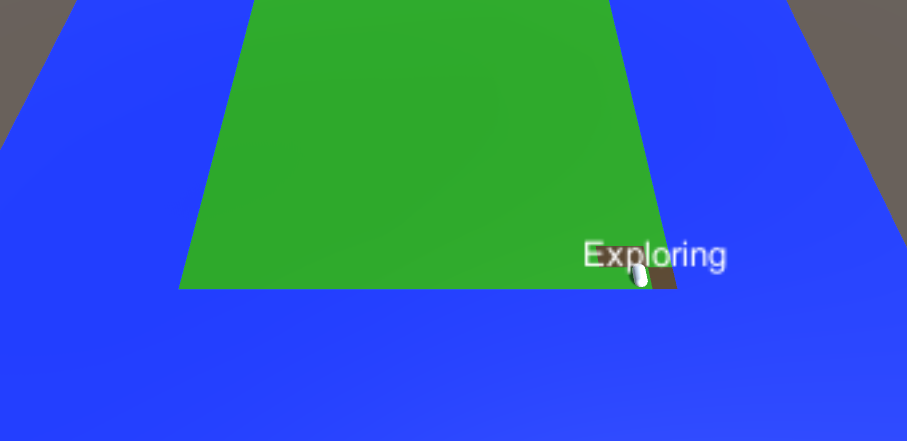


Figure 7 - Screenshot of the agent in the final product

The final scenario is containing a more advanced behaviour tree and tests different combinations. The agent has a more complicated tree as he does not know everything, it now has a vision and must search, the agent while ‘happy’ moves around exploring his land. If one of the agent’s stats falls below a threshold he becomes ‘unhappy’ and searches for something that will increase that stat. If the agent finds something, he will go to it and consume it, if it can not find anything that will help it, it will explorer the area until it finds something that increase its stat.

During making this version, a bug was discovered that some nodes where not returning running from their children and just continuing, this became a problem as extra nodes that where not meant to be running where being called and jumps where appearing in game as movement was being called multiple times per frame instead of once per frame, the fix was very simple, some nodes that was call multiple children or the same node multiple times would receive the running stats and continue as if that was a success, now these nodes when they receive running they stop and return running.

## Summary

Development went smoothly thanks to the early adoption of testing with visual studio’s unit testing making sure that each node passed and failed correctly. Testing then went onto a real-world scenario which went well until a bug was found that caused hiccups due to the behaviour tree not correctly handling the return status which was quickly solved.

# Evaluation, Conclusions and Future Work

## Project Objectives

The core requirements of this project where to create a working behaviour tree which takes two key things, the ability to create nodes for the tree and associate them and the ability to create a tree and run it on command.

Another objective was to make debug tools to help a developer program with the behaviour tree and also assist though who do not understand a behaviour tree get a better grasp of what it does.

## Self-Evaluation

I believe I am a strong programmer, but I lack knowledge and I am always finding out new, I feel like I threw myself into this topic as I have a passions for AI but the only things I knew where what I learn at the University of Central Lancashire, which at the time was is mainly state dependant like finite state machines which is a great start but it’s only a start. Throughout this project I have discovered other methods and increased my knowledgebase, besides learning about behaviour trees, I have discovered alternative to them as well including their advantages and disadvantages.

I had some experience with C# and Unity but doing a project has helped me learn more than what I’m being taught, as someone who mainly uses C++ using other languages helps broaden what I can do and how to tackle a problem. I would like to go onto using other games engines like unreal engine.

I started with a developer diary at the start of this project I felt I kept up well with it but at a certain point it fell off, my next project I’m going to put more importance into keeping progress well documented.

This project is one of the longest project that I have done, knowing this I have put a lot of effort into it and I have felt it has gone well, I feel my organisation and time management has been used well as I also balanced other smaller project alongside this one.

One part of this project that is notable to me as I have not don’t his before is that I made a prototype showing off using a behaviour tree in a simple scenario but once it proved its point and it was used to test if their where any problems integrating a behaviour tree into unity that was its job done. It felt odd that once it was done, I did not need it anymore, what I did I believe was the best way to do it and in the future if a prototype could be used to verify, I would be more likely to use one.

## Project Evaluation

I believe I have hit all targets I set out in this project within the time frame I was given. The behaviour tree can be created, have nodes added to it and create the tree structure and ran whenever the developer would like with minimal cost.

One criticism I have of this project is that creating a behaviour tree initially the nodes need to reference each other which causes the node to be created from the bottom of the tree upwards since the adult nodes need a reference. I believe developers who create the tree in this way are less like to miss out nodes, but it can be harder to plan the tree. If this product continues, I believe this is an important issue to tackle.

Overall, I am happy with this project, the development went smoothly, the product has been tested, works well as a product and runs well in a scenario.

## Applicability of Findings to the Commercial World

This project while mainly focused on games application this AI is designed to be modular and can be use in any situation where any form of AI is needed, especially if it is a large project. Other applications of a behaviour tree can include simulations, which can be seen used within emergency services. Robotics is another main target where this could be applied to, the ability to make code reusable and modular helps with iterative design.

This could be easily turned into a product, the code can be easily extracted from what I have produced, though the graphical part of the behaviour tree may need some work to make it modular and to transfer this project to other engines will only take a small amount of work for the behaviour tree and used commercially.

I believe this project shows off that I can be a valuable employee, takins something that I have little knowledge off, researching about it and producing a product with it and showing a will to develop myself.

## Conclusions

I believe this project went smoothly, thanks to the focus on debugging most problems with the tree and nodes where solved as soon as they popped up. Testing went onto making a prototype and the final product which helped discovered any other bug with the behaviour tree. The behaviour tree is well tested eliminating most bugs and can be easily adapted into other situations, future improvements are mainly for ease of use.

## Future Work

Time and development power were the key factors on the quantity that was produced in this project. There are a few areas I am aware that could do with improvement.

* Ease of use functions, the way that nodes are added to a tree are bottom first which can be hard for a developer to program with, some functions so that a node can be created and then child nodes can be added after.
* XML save file, currently the save file uses simple strings to save the data of each node, it would be better if a xml format was used so that it is more legible for developers to edit.
* Binary save file, this would help loading of files for situations where loading speed is required.
* Editable in GUI, the current gui is designed for debugging in mind, an improvement to it would be to be able to move nodes around, change the children of a node and maybe even create nodes.
* Optimisation, I have noticed that you can make redundant nodes, while this is useful for debugging this increase the size of the tree and when memory usages is a concern it would be nice to remove redundant nodes and replace and check for any duplicate node and remove them.

References

Cuevas, R. et al. (2010a) 'A collaborative P2P scheme for NAT Traversal server discovery based on topological information', *Computer Networks,* 54(12), pp. 120-122.

Cuevas, R. et al. (2010b) 'Book Section', in Kramer, J. D. (ed) *This is a another Book released in the same year,* London: PressAwesome, pp. 150-180.

Greene, D. & Williams, P. C. (1997) *'Linear Accelerators for Radiation Therapy',* 2nd edition, Bristol and Philadelphia: IOP Publishing Ltd.

Jacobson, J. & Andersen, O. (1997) 'Software Controlled Medical Devices', in *SP Report 1997:11,* Sweden: Swedish National Testing and Research Institute.

Ji, H. & Turner, K. J. (1999) 'Specification and Verification of Synchronous Hardware using LOTOS', in Wu, J., Chanson, S. T. & Gao, Q. (eds) *Formal Methods for Protocol Engineering and Distributed Systems (FORTE XII/PSTV XIX),* London: Kluwer Academic Publishers, pp. 295-312.

Turner, K. J. & Jennings, M. A. (2002) 'The Rules of Sailing Races for Hand-Held Devices', *J. Navigation,* 23(5), pp. 114-240.

University of Stirling (2002) *Computing Science and Mathematics Research,*   
http://www.cs.stir.ac.uk/research  
(accessed 01 March 2017).

The reference list above is generated by Word. Under ‘References’ -> ‘Manage Sources’ it is possible to add new references which can later be inserted in the report with ‘Insert Citation’. Example: (Cuevas, et al., 2010a)

Alternatively, you can add a new reference and immediately insert it as a citation by moving the cursor to the place where you want to insert the citation and then choosing ‘References’ -> ‘Insert Citation’ -> ‘Insert New Source’ from the ribbon menu.

You can hide parts of a citation by right-clicking on it, selecting ‘Edit Citation’ and ticking the checkboxes under ‘Suppress’. In the same dialog a reference to specific pages of the cited source can be added (which won’t be displayed in the references list). Example (same as above but with ‘Author’ suppressed and page 120 added): (2010a, p. 120)

Keep in mind that uncited sources will still appear in the references list above. Go to ‘References’ -> ‘Manage Sources’ to see which sources are cited and which aren’t. Sources in ‘Current List’ which have a checkmark are cited.

The UCLan styling for references is different than the default Harvard notation Word provides. Therefore, a custom style has been created. Download the file ‘HarvardUCLan2017.xsl’ and copy it to: %appdata%\Microsoft\Bibliography\Style.

Keep Word closed when doing this. To select the style click on ‘References’ -> ‘Style’ and select ‘Harvard – UCLan (2017)’. Tested with Word 2016 (Windows).

There might be some cases the Word bibliography function can’t handle. If you have a tool that suits you better such as ‘RefWorks’, ‘Citethisforme’ or ‘RefMe’ use that instead, then remove the list above and copy the references over. Don’t forget to use the correct Harvard notation style.

Appendix 1 – Project Proposal

Your first appendix should be a copy of your Project Proposal.

You may have one or more appendices containing detail, bulky or reference material that is relevant though supplementary to the main text: perhaps additional specifications, tables or diagrams that would distract the reader if placed in the main part of the dissertation. Make sure that you place appropriate cross-references in the main text to direct the reader to the relevant appendices.

Do not blindly include all of your code in the appendix or the body. Only include the parts you refer to in the report. You can put those parts either in the appendix or in the body (e.g. in the “Implementation” part).

Appendix 2 – Technical Plan

Your second appendix should be a copy of your Technical Plan.

You may have one or more appendices containing detail, bulky or reference material that is relevant though supplementary to the main text: perhaps additional specifications, tables or diagrams that would distract the reader if placed in the main part of the dissertation. Make sure that you place appropriate cross-references in the main text to direct the reader to the relevant appendices.

Do not blindly include all of your code in the appendix or the body. Only include the parts you refer to in the report. You can put those parts either in the appendix or in the body (e.g. in the “Implementation” part).

Appendix 3 – Title of Appendix

You may have one or more appendices containing detail, bulky or reference material that is relevant though supplementary to the main text: perhaps additional specifications, tables or diagrams that would distract the reader if placed in the main part of the report. Make sure that you place appropriate cross-references in the main text to direct the reader to the relevant appendices.

Do not blindly include all of your code in the appendix or the body. Only include the parts you refer to in the report. You can put those parts either in the appendix or in the body (e.g. in the “Implementation” part).