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

Your are aiming for no more than 50 pages of report content, this count starts at Chapter 1 and does not includes your references or appendices.

The abstract is the summary of the project report within one page (aim for about 500 words). Unnumbered chapter headings, as above, are entered using the ‘Heading (Unnumbered)’ style, which automatically starts a new page.

This template starts the page numbering at the foot of this page. That is, the first page does not have a number.

It is suggested that the abstract be structured as follows:

Problem: What you tackled, and why this needed a solution

Objectives: What you set out to achieve, and how this addressed the problem

Methodology: How you went about solving the problem

Achievements: What you managed to achieve, and how far it meets your objectives.

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

Acknowledge anyone who has helped you in your work such as your supervisor, technical support staff, fellow students or external organisations. Acknowledge the source of any work that is not your own.

Table of Contents

The table of contents below is automatically generated from the paragraphs of style ‘Heading 1’ to ‘Heading 3’ and ‘Heading (Unnumbered)’. To update this after revisions, right-click in the table and choose ‘Update Field’ for the entire table. It is also possible to left-click the table and press F9 to update the entire table. Delete this paragraph before submission.

Sometimes Word does not automatically update references to other parts of the document, such as captions, Table of Contents and cross-references. To update everything, select the whole document by pressing Ctrl+A and then either press F9 or right-click anywhere and click on ‘Update Field’. Confirm all prompts by selecting ‘Update entire table’ and clicking OK. Sometimes the font changes on update so check it is the correct font face before submitting your work.

[Abstract i](#_Toc2963907)

[Attestation ii](#_Toc2963908)

[Acknowledgements iii](#_Toc2963909)

[Table of Contents iv](#_Toc2963910)

[List of Figures vi](#_Toc2963911)

[List of Tables vii](#_Toc2963912)

[List of Listings viii](#_Toc2963913)

[1 Introduction 1](#_Toc2963914)

[1.1 Background and Context 1](#_Toc2963915)

[1.2 Scope and Objectives 1](#_Toc2963916)

[1.3 Achievements 1](#_Toc2963917)

[1.4 Overview of Report 1](#_Toc2963918)

[2 Literature Review 2](#_Toc2963919)

[2.1 Introduction 2](#_Toc2963920)

[2.2 AI Programming Techniques 2](#_Toc2963921)

[2.2.1 Behaviour Tree 2](#_Toc2963922)

[2.2.2 Finite State Machine 2](#_Toc2963923)

[2.3 Game Engines 2](#_Toc2963924)

[2.3.1 Unity 2](#_Toc2963925)

[2.3.2 Other Common Game Engines 2](#_Toc2963926)

[2.3.3 Summary 2](#_Toc2963927)

[3 Project Planning 3](#_Toc2963928)

[3.1 Introduction 3](#_Toc2963929)

[3.2 Methodology 3](#_Toc2963930)

[3.3 Requirements 3](#_Toc2963931)

[3.4 Potential Solutions 3](#_Toc2963932)

[3.5 Tools and Techniques 3](#_Toc2963933)

[3.6 Legal and Ethical Issues 3](#_Toc2963934)

[3.7 Summary 3](#_Toc2963935)

[4 Design 4](#_Toc2963936)

[4.1 Introduction 4](#_Toc2963937)

[4.2 System Design 4](#_Toc2963938)

[4.3 User Interface Design 4](#_Toc2963939)

[4.4 Summary 4](#_Toc2963940)

[5 Implementation 5](#_Toc2963941)

[5.1 Introduction 5](#_Toc2963942)

[5.2 Decision Tree 5](#_Toc2963943)

[5.2.1 Console Proof of Concept 5](#_Toc2963944)

[5.2.2 Real Situation Unity Build 5](#_Toc2963945)

[5.3 Section 5](#_Toc2963946)

[5.4 Summary 5](#_Toc2963947)

[6 Test Strategy 6](#_Toc2963948)

[6.1 Introduction 6](#_Toc2963949)

[6.2 C# Unit testing 6](#_Toc2963950)

[6.3 Live testing 6](#_Toc2963951)

[6.4 Summary 6](#_Toc2963952)

[7 Evaluation, Conclusions and Future Work 7](#_Toc2963953)

[7.1 Project Objectives 7](#_Toc2963954)

[7.2 Self-Evaluation 7](#_Toc2963955)

[7.3 Project Evaluation 7](#_Toc2963956)

[7.4 Applicability of Findings to the Commercial World 7](#_Toc2963957)

[7.5 Conclusions 7](#_Toc2963958)

[7.6 Future Work 7](#_Toc2963959)

[References 8](#_Toc2963960)

[Appendix 1 – Project Proposal 9](#_Toc2963961)

[Appendix 2 – Technical Plan 10](#_Toc2963962)

[Appendix 3 – Title of Appendix 11](#_Toc2963963)

List of Figures

Similarly, you can automatically generate a list of ‘Figures’. Right-click a figure (e.g. image or diagram) and add a caption labelled ‘Figure’ and ‘below selected item’. To update this after revisions, right-click in this table and choose Update Field (or use F9) and then choose to update the entire table. Delete this paragraph before submission.

[Figure 1 - Highly Technical Diagram 3](#_Toc476373580)

List of Tables

Similarly, you can automatically generate a list of ‘Tables’. Select a table, right-click it and add a caption labelled ‘Table’ and ‘above selected item’. To update this after revisions, right-click in this table and choose Update Field (or use F9) and then choose to update the entire table. Delete this paragraph before submission.

[Table 1 - Test Results 6](#_Toc476373581)

List of Listings

This list only applies to you if you use code snippets in your report. If you don’t have any listings, remove this whole section including the heading ‘List of Listings’.

You can automatically generate a list of ‘Listings’. After formatting your Code, move the cursor to the first line below your code block and click ‘References’ -> ‘Insert Caption’ in the ribbon menu. Select the label ‘Listing’ or add a new Label called ‘Listing’ if it does not yet exist. To update this after revisions, right-click the table and choose ‘Update Field’ (or use F9) and then choose to update the entire table. Delete this paragraph before submission.

[Listing 1 - [Main.java] The main class of the program 5](#_Toc476373582)

# Introduction

For editorial consistency, it is important to use Word styles properly. Word 2003 onwards has so-called ‘quick styles’. If the styles referred to below are not visible on the Home ribbon in the Styles category, choose ‘Apply Styles’ from the down arrow at the bottom right of the Styles category. Styles can then be applied from the drop-down box. To make a style visible as a quick style, choose Apply Styles, then click Styles (the AA icon) or use ‘Alt + Ctrl + Shift + S’, then right-click on the style and then ‘Add to Style Gallery’.

Chapters are entered using the ‘Heading 1’ paragraph style. The Heading 1 style automatically moves to the start of a new page, and supplies the next chapter number. Pressing enter on a ‘Heading 1’ heading automatically inserts a ‘Heading 2’ heading underneath.

There should not be any text between a parent heading and its first sub-heading. For example, when you want to write an introductory section for the following sections, give that introductory section an own ‘Introduction’ heading instead of writing it between the heading preceding these sections and the first sub-heading.  
As an example: This comment text is incorrectly placed between a ‘Heading 1’ (‘Introduction’) and a ‘Heading 2’ (‘Background and Context’).

Most text uses the ‘Normal’ paragraph style with 12 point Arial, 1.5-line spacing, single-sided pages.

In general, use the default spacing that headings and paragraphs give you. Avoid using new-lines or spaces to format text. If you need to use quotes, preferably use single curly quotes ‘…’. If you wish to emphasise something, use the ‘Emphasis’ style. In addition, also a ‘**Strong’** style is preconfigured.

Remember to Save frequently while you are working! Check that AutoSaving is enabled under options -> save -> 'Save AutoRecover information every 5 minutes'.

## Background and Context

Give the background to your project and context of what you have done. Sections are entered using the ‘Heading 2’ paragraph style.

## Scope and Objectives

Define the scope and objectives of your project.

## Achievements

Summarise what you have achieved.

## Overview of Report

Briefly overview the contents of what follows in the report. Overview (1-2 lines per chapter):

'Chapter 3 describes the investigation of the problem and presents the top-level analysis as a Yourdon dataflow diagram. ... Chapter 4 contains an overview of the design architecture and examines the key design issues’.

# 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 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.

## 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

Each of your chapters should have an introduction to tell your readers what they will find in the chapter.

## 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

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. 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, 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

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.

### 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.

### 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. 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 $100k 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

Code copy write

Licencing

## Summary

Write a short summary at the end of each chapter.

# Design

## Introduction

Each of your chapters should have an introduction to tell your readers what they will find in the chapter. Remember to change the sub chapter headings to some that are suitable to your project.

## System Design

Text goes here.

## User Interface Design

Text goes here.

## Summary

Write a short summary at the end of each chapter.

# Implementation

## Introduction

Each of your chapters should have an introduction to tell your readers what they will find in the chapter.

## Decision Tree

### Console Proof of Concept

Code can be formatted using the ‘Code’ style. An example is shown below. It can be a little bit tricky to keep the formatting when pasting from an IDE but the following works for most IDEs: Copy the text from the IDE, paste it in Word, select the pasted code and change the style to ‘code’. It is worth noting that spell checking is deactivated for the ‘Code’ style.

public class Main {  
  
 public static void main(String[] args) {  
 System.*out*.println("Hello World!");  
 }  
}

Listing 1 - [Main.java] The main class of the program

Captions are entered through the ribbon menu under ‘References’ -> ‘Insert Caption’. Select ‘Listing’ (or add a new Label called ‘Listing’ if it does not already exist) and add the caption text in the white box, separated with a dash as the example above shows. Think about a naming convention for listings and stick to it throughout the report. For example, as seen above,  
‘[ClassName or Filename] Description’.

In case you are mixing multiple programming languages: Consider stating the language name in the caption if it is not obvious from the file name or when there is no file name to refer to. For example, when you use XML and HTML, JavaScript and TypeScript or other languages with similar syntax. A suggestion might be to add the language in parenthesis at the end.

It is also possible to use the ‘code’ style “inline” to highlight commands in normal text by selecting the words to highlight and choosing the ‘code’ style. For example:

This example demonstrates the ping 127.0.0.1 command.

Make sure to write the whole text first and select the part you want to highlight afterwards. When there is no selection, Word applies the selected style to the whole paragraph.

### Real Situation Unity Build

## Section

## Summary

Write a short summary at the end of each chapter.

# Test Strategy

## Introduction

Each of your chapters should have an introduction to tell your readers what they will find in the chapter.

## C# Unit testing

To add a caption to a table, either select the whole table (e.g. by clicking on the + symbol in the upper left corner of the table), right-click it and choose ‘Insert Caption’ or click in any table cell and select ‘References’ -> ‘Insert Caption’ from the ribbon menu. Choose ‘Table’ as label and ‘above the item’ as position. Add the caption text in the box, separated with a dash as the example below shows.

Table 1 - Test Results

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Live testing

Text goes here.

## Summary

Write a short summary at the end of each chapter.

# Evaluation, Conclusions and Future Work

## Project Objectives

Summarise what you have achieved.

## Self-Evaluation

This section is about yourself. Be honest. Look at where you were situated at the beginning of the project and where you are now. What have you learnt on a personal level, what have you found out about yourself? Try to reflect upon individual goals, experiences, and incidents. No one is perfect, and it is very likely that you will recall both good and bad experiences.

The purpose of the evaluation process is to highlight strengths, correct performance weaknesses, and develop unused skills and abilities. In order to do this, you must be willing to recognise areas that need improvement or development.

## Project Evaluation

Stand back and evaluate what you have achieved and how well you have met the objectives. Evaluate your achievements against your objectives in section 1.2. Demonstrate that you have tackled the project in a professional manner.

(The previous paragraph demonstrates the use of automatic cross-references: The ‘1.2’ is a Cross-reference to the text in a numbered item of the document, it is not literal text but a field. The number that appears here will change automatically if the number on the referred-to section is altered, for example if a chapter or section is added or deleted before it. Cross-references are entered using Word's **Insert** or **References** menu. Cross-references are set to update automatically when printed, but may not do so on-screen beforehand; you can update a field manually on-screen by right-clicking on it and selecting Update field from the pop-up menu or by selecting the whole document and pressing F9.)

## Applicability of Findings to the Commercial World

Summarise what you have achieved.

## Conclusions

Summarise what you have achieved.

## Future Work

Explain any limitations in your results and how things might be improved. Discuss how your work might be developed further. Reflect on your results in isolation and in relation to what others have achieved in the same field. This self-analysis is particularly important. You should give a critical evaluation of what went well, and what might be improved.

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).