Procedurally Generated Background Characters

by

Vanya Eccles, B.A.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

(Interactive Entertainment Technology)

University of Dublin, Trinity College

August 2017

Declaration of Authorship

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Signed: Vanya Eccles

Date: 31st August, 2017

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Signed: Vanya Eccles

Date: 31st August, 2017

Acknowledgements

I would like to offer my sincerest gratitude to my supervisor Dr. Mads Haahr, for his support throughout the project. Dr. Haahr's insight and advice on research directions led to great improvements to the project's development during its duration.

In addition, I would like to thank my family for their continuous support during my studies.

Vanya Eccles

University of Dublin, Trinity College
August 2017

Procedurally Generated Background Characters

Vanya Eccles
University of Dublin, Trinity College, 2017

Supervisor: Dr. Mads Haahr

Non-player characters in games and simulations often rely on scripted behaviour or simplified decision-making architectures. This can result in behaviour that is not coherent or believable for the player. Similarly the scope of large open-world games with many non-player characters (NPCs) denies designers the ability to extensively hand-craft differences in behaviour in order to convey agent personality.

This project presents a model for NPC behaviour where actions are chosen in a hierarchical manner through the evaluation of functions that map the agent's current state to utility values. Each action is chosen on the basis of specific parameters representing the state of the agent or the environment, and performing actions has an effect on the agent's state. Agent personalities are specified on the basis of the five-factor model, and through procedural generation have unique and intuitive effects on how agents weigh decisions and are effected by actions and events.

The implementation showcases a model that allows for easy specification of game characters that efficiently display unique behavioural preferences. Agents can be easily stored in a compressed state to be generated in real-time and simulated at full-detail, offering potential for use in the next generation of procedurally generated worlds.

Contents

Acknowledgements iii

Abstract iv

List of Figures viii

List of Tables ix

1 Introduction 1
- 1.1 Open Worlds and Procedural Content Generation . . . 1
- 1.2 Motivation . . . 2
- 1.3 Objectives . . . 4
- 1.4 Roadmap . . . 5

2 State of the Art 6
- 2.1 Believable Behaviour . . . 6
  - 2.1.1 Approaches from Computer Science . . . 7
  - 2.1.2 Perception of Believability . . . 8
  - 2.1.3 Believability for Game Agents . . . 10
- 2.2 Architectures for Agents in Games . . . 12
  - 2.2.1 Note on Background NPC Architectures . . . 14
  - 2.2.2 Scripting and Scheduling . . . 14
  - 2.2.3 Finite State Machines . . . 15
  - 2.2.4 Behaviour Trees . . . 15
  - 2.2.5 Planning Architectures . . . 16
  - 2.2.6 Machine Learning as Applied to Agent Behaviour Architectures . . . 17
  - 2.2.7 Utility Systems . . . 18
- 2.3 Utility Theory for Believable Behaviour . . . 19
  - 2.3.1 Infinite Axis Utility System . . . 19
  - 2.3.2 Dual-Utility Reasoning . . . 21
  - 2.3.3 Hierarchical Utility Reasoning . . . 21
- 2.4 Procedural Content Generation . . . 22
  - 2.4.1 Procedural Content Generation for Game Agents . . . 23
  - 2.4.2 Randomness Techniques for Game Agents . . . 24
  - 2.4.3 Generating Randomness . . . 25
    - 2.4.3.1 Filtered Randomness . . . 25

2.4.4 Generating from a Gaussian Distribution . . . 26
2.4.5 Level-Of-Detail Systems for Game Agents . . . 27
2.4.6 Personality Modelling . . . 27

3 Design **31**
3.1 Previous Work . . . 31
3.2 Model Definition . . . 31
3.2.1 Agent and Environmental State . . . 32
3.2.2 The Action Hierarchy . . . 33
3.2.3 Utility-Based Action Selection . . . 34
3.2.4 Performing Actions . . . 37
3.2.5 Personality Modelling . . . 39
3.2.6 Procedurally Generated Agents . . . 40
3.2.6.1 Level of Detail Systems . . . 41
3.2.7 Agent-Environment Interaction . . . 41
3.3 Illustrating the Model . . . 42
3.3.1 The Neurotic Blacksmith . . . 42
3.3.2 The Conscientious Woodcutter . . . 45
3.4 Model Architecture . . . 46
3.5 Manual Creation of Actions . . . 47
3.6 Automatic Addition of Actions . . . 47
3.7 Conclusion . . . 48

4 Implementation **49**
4.1 Platform Review . . . 49
4.1.1 Demo Environment . . . 50
4.1.2 Agent Representation . . . 50
4.1.3 Agent Movement . . . 51
4.2 Utility-Based Decision Hierarchy . . . 52
4.2.1 Interruption System . . . 55
4.2.2 Implemented Actions & Hierarchy . . . 56
4.3 Performing Actions . . . 58
4.4 Procedural Generation . . . 58
4.4.1 Gaussian Pseudorandom Number Generation . . . 58
4.4.2 Decision Weight Generation . . . 60
4.4.3 State Modification Vector Generation . . . 61
4.4.4 Level of Detail System . . . 62
4.5 User Interface & Debugging Tools . . . 63

5 Evaluation **65**
5.1 Generation of Believable Agents . . . 65
5.1.1 Comparison of Similar Personalities . . . 65
5.1.2 Believable Behaviour . . . 68
5.2 Efficiency of Implementation . . . 71
5.2.1 Procedural Content Generation . . . 71
5.2.2 Memory Requirements . . . 73
5.2.3 CPU Requirements . . . 74

5.3 Impediments 76

6 Conclusion **78**
6.1 Main Contributions 78
6.2 Future Work 79
6.2.1 Agent Specification 79
6.2.2 Animations and Speech 80
6.2.3 Probabilistic Action Selection and Interruptions 80
6.2.4 Goal-Oriented Actions 81
6.3 Perspective 81

A **82**
A.1 State Modification Vectors 82

B **84**
B.1 Generated Weights and Action Modification Vectors 84

C **86**
C.1 Top Actions for Extreme Personalities 86

D **88**
D.1 Reference Materials 88
D.2 External Assets & Plug-ins 88

Bibliography **90**

List of Figures

2.1 The Uncanny Valley as detailed by Masahiro Mori, adapted from [1] . . . 9
2.2 Descriptions of the FFM, adapted from [2] 28
2.3 Correlations between traits and socioeconomic outcome, adapted from [3], [4] 30

3.1 Agent Emotional Representation as specified by Mullally, adapted from [5] 33
3.2 Agent 'Needs' Representation as depicted in the Sims [6] 33
3.3 An example tree structure for action selection, Adapted from [7] 34
3.4 Action-specific Utility Functions for agent 'Energy' 35
3.5 Action hierarchy, each action with its own considerations 36
3.6 Example State Modification Vectors for actions 'Work Hard' and 'Sleep' . 38
3.7 High Level Representation of the Proposed Model 47

4.1 The Implemented Demo Environment 50
4.2 NavMesh (blue) Visualisation within the Demo Environment 52
4.3 The AnimationCurve component in Unity 54
4.4 The Implemented Action Hierarchy 56
4.5 The Implemented User-Interface for Assessing Agent State. For every agent running there is a button automatically created in the left hand panel. The right hand panel can be used for toggling between the selected agent's history, current state or current action 64

5.1 Memory Requirements Per Agent 74
5.2 CPU Requirements Per Agent 75

List of Tables

4.1 Actions with Considerations & Locations 57
4.2 Seeds Generated for Different Personality Vectors 59
4.3 Personality Weight Influences 60
4.4 Personality Modifier Influences 61

5.1 Personality Vectors for Agents Fido, Mello and Bingo. 66
5.2 Generated State Modifiers 66
5.3 Generated Consideration Weights 67
5.4 Top Actions for Fido, Mello and Bingo over 2 in-game days 68
5.5 Mundo, a conscientious woodcutter 69
5.6 Mundo's Day 69
5.7 Average CPU Usage of Generating Agents *Average value over two frames of execution 72
5.8 Average CPU Usage of Generating Agent, Using Per Frame Coroutine . . 72
5.9 System Memory Usage Per Agent 74
5.10 Average CPU Usage of Agent Update 75

A.1 State Modification Vectors for leaf actions in the demo environment 83

B.1 Weights Generated for Personality Extremes. Balanced (+0.0001 for all personality dimensions) shown for comparison. Values in Bold correspond highly to the personality dimension (i.e. Openness contributes to 'timeofday' and 'relationship' parameters) 85
B.2 Action Modifiers Generated for Personality Extremes. Balanced (+0.0001 for all personality dimensions) shown for comparison. Values in Bold correspond highly to the personality dimension (i.e. Openness contributes to 'timeofday' and 'relationship' parameters). 'timeofday' has no modifier as it is a global state parameter 85

C.1 Actions Performed by Agents with Personality Extremes, Over 2 In-Game Days. Personalities with extreme values on a single parameter: +0.999 for High, −0.999 for Low, 0.0001 for all other personality parameters . . . 87

Chapter 1

Introduction

This dissertation aims to explore a novel method for the creation of background characters for interactive environments such as digital games. The research focuses on background non-player characters (NPCs) that are inessential to core gameplay or story, but add detail to the backdrop of populated areas. The model developed during the project employs a hierarchical model for behaviour selection, where actions are evaluated and chosen on the basis of calculated utility. The primary hypothesis of the project was that a generalised model for personality could be used to procedurally generate believable manifestations of personality through differences in NPC behaviour. A demonstrative environment was developed to examine the model, and an exploration of its benefits and drawbacks is discussed. A primary intention was that this general model could be used to improve existing approaches to modelling background NPC behaviour and provide a tool for the easy generation of large numbers of NPCs with interesting and believable individuality.

1.1 Open Worlds and Procedural Content Generation

Whether attempting to capture real-world phenomena or to provide novel experiences, games are increasingly equipped with the tools to expand the space of possibilities. One of the avenues of games that has proved to be increasingly popular has been the theme of 'Open-World' game design. Games that fall under this category are characterised by offering player freedom in navigating a large virtual world, commonly with a focus

on nonlinear gameplay where there are multiple ways for players to achieve objectives. Popular open-world games include Skyrim [8], with its large medieval fantasy world populated with small towns and explorable dungeons, and Assassin's Creed Unity [9] where the vast city of revolution-era Paris was simulated in detail.

Manual content creation has been presented with scalability challenges as open-world games offer increasingly large, complex environments for players to explore. Procedural Content Generation for Games (PCG-G), the automated production of game content by computers, has gained the interest of many in the game industry as a tool to address these challenges. PCG-G techniques commonly focus on the application of algorithms that operate on a few parameters and a pseudorandom process to programmatically generate an unpredictable range of game content [10]. Techniques of PCG-G have been used at least since the popular space trading game Elite (1984) used pseudorandom number generation to create eight explorable galaxies, each with 256 explorable planets. More recently, the action-adventure game No Man's Sky used PCG-G techniques to generate a universe of 18 quintillion planets, each claiming unique content [11]. Informative surveys of PCG-G can be found in [12] and [13].

1.2 Motivation

This project aims to tackle one of the core issues for these large environments: how can these worlds become populated with interesting and believable NPCs? Various techniques are used to generate responsive or intelligent behaviours in game characters; traditionally borrowing from approaches within the field of artificial intelligence (AI) to form the subfield of 'Game AI'.

There exists confusion over the role of AI in games, to the point where many developers feel the need to underline their position. As Kevin Dill puts it: "Game AI should be about one thing and one thing only: enabling the developers to create a compelling experience for the player" [14], an attitude shared among many industry professionals and researchers alike [15]. Artificial intelligence as applied to video games has traditionally been placed in the 'narrow' AI bin [16]. An irony perhaps is that most AI agents in games are trying to emulate human ability, an exceedingly complex phenomena. To take a common example, a stealth based video game opponent effectively needs to perform

the tasks that present the greatest challenge to areas of academic AI. These include interpretation of subtle sensory stimuli and bodily control within a complex spatial environment. For a non-combat character in a role-playing game you might expect the agent to hold a human conversation. Clearly the hard limit for realism is to recreate the human intelligence in such agents, no mean feat for the small percentage of the CPU budget normally allocated to AI in games; typically between 5 and 50% [17].

The focus of this work is to specifically address the perceived lack of good approaches to generate background NPCs that are believable [18]. Background game agents can be thought of as analogous to extras in film; they give the semblance of a dynamic, character-driven environment, but the details of their behaviour are not designed to be essential to the narrative nor to the immediate gameplay. Importantly in real-time interactive applications such as games where hardware resources are limited, their AI must be computationally inexpensive.

For the discussion that follows the termss 'game AI' and 'game agent' to describe an agent that manifests as a character in the game. Broad descriptions of what we mean by AI often vary along two main scales: a concern with thought-processes and reasoning versus behaviour, and a measure of success in comparison to human performance versus a comparison to *ideal* performance. This ideal concept of intelligence is referred to as rationality; a rational agent is defined according to Russell and Norvig as a system that "does the 'right thing', given what it knows". According to the views of Kevin Dill, AI characters for games are most usefully placed somewhere in the bottom left of this scale. Indeed behaviour of the agent is the primary way of conveying information about the agent to the player. All aspects of behaviour are considered to have the primary motive of creating an experience to the player, and often the most effective methods of conveying intelligence are in fact the least nuanced. This is perhaps most striking in the use of short spoken lines by game agents (known as 'barks') to clue the player into what is occurring within the agent's internal world; several commentators have noted how this is often more effective than necessarily making the AI better [14], [19].

Particular focus is placed on background NPCs that display believable behaviour, however the study of believability in this context is not yet well-defined. In most game experiences players do not want agents that will behave perfectly; it would be trivial for instance to program opponents in a first-person shooter game that would consistently

score a perfect head shot against the player. Therefore for games we rarely want perfectly rational agents, but rather agents that display degrees of rationality. Humans are beset by myriad cognitive biases, and after 50+ years of scientific work into decision-making we now know that human choices are rarely as rational as we may expect [20]. Should we therefore be most receptive to agents that are irrational? Perhaps not. The game journalist Mark Brown outlines the opinion that good game agents should be predictable [19]. Chris Butcher, engineering lead for the AI in Halo 1 and 2, elaborates on this: "We don't do things by random chance very much. The goal is not to create something that is unpredictable. What you want is an artificial intelligence that is consistent so that the player... can do things and expect the AI will react in a certain way" [21]. He envisages the fun and re-playability of the player's experience as relying on "predictable actions but unpredictable consequences", stating that ideally "the grunt will always run away, but you don't necessarily know where he'll run away to". This opinion is contested, and others state that there is a strong need to focus on non-determinism in virtual character actions [22], [23]. A useful model for making rational decisions is to make decisions on the basis of perceived 'utility', or usefulness to the agent. Degrees of irrationality can be introduced simply by specifying how certain agents weigh up the considerations for a decision over others.

When attempting to specify the qualities of believable characters, a ubiquitous request is that the agents display personality [23], [24]. Whilst it is possible to hand-author character quirks within a game AI framework, doing so for the massive crowds in games such as Assassin's Creed Unity or the myriad alien-populated planets of No Man's Sky suggests a need for methods that allow the automatic generation of characters. Such methods would need to be general, fast and modular if they are to be applicable to the many needs of modern interactive environments.

1.3 Objectives

The challenges for good quality and believable game agent behaviour are unique, any model must balance several constraints:

- **Realism**: how well it results in realistic agent behaviour, however that is defined within the game

- **Performance**: how efficiently it can run in real-time (usually secondary to other aspects of a game such as rendering)
- **Complexity**: how easy it is to adjust
- **Ease of use**: how intuitive it is for non-technical designers to work with

With these in mind, the dissertation attempts to address these constraints for the development of a system for generating agents that can express personal behavioural differences through easy specification of the agent's personality.

1. Offer a general purpose utility-based decision-making architecture that is easy to adjust and add to.
2. Implement the procedural generation of factors that influence agent decision-making from a compact and intuitive representation of the agent's personality.
3. Focus on an efficient implementation that offers a good standard of CPU and memory use.

1.4 Roadmap

The dissertation begins in earnest with a review of relevant literature in Chapter 2. Particular focus is placed on exploring the meaning and study of believable behaviour, the state of the art for NPC behaviour models in the industry and approaches to procedural content generation in the context of agent behaviour. Chapter 3 presents a proposed model for a general PCG approach to NPC behaviour, influenced by the work covered during research. Chapter 4 presents the prototype of the model, with a focus on how the model design was implemented. This is followed by an evaluation in Chapter 5 of the implementation with regards to the criteria outlined in Section 1.3, with a discussion of the benefits and drawbacks of the model. Chapter 6 concludes the project with a summary of the work done and contributions made, in addition to suggestions for future work on the model.

Chapter 2

State of the Art

The purpose of this chapter is to cover the relevant literature on areas pertaining to the creation of a model that addresses the focus of the project outlined in Section 1.3. First, a discussion of 'believable behaviour' and the approaches made by various disciplines to define and measure it. Following this is a discussion of popular architectures used for the development of game agents, with a particular focus on those offering dynamic and fuzzy decision making. Lastly there is a look at procedural content generation, with emphasis on the possibility of generation of individual differences in game agents through personality modelling.

2.1 Believable Behaviour

If an imperative of game agents is that they must behave in a believable manner, we must consider what 'believable behaviour' could mean. Breaking this down into useful lines of approach, there are two immediate facets to consider. The contents of behaviour and the details of behaviour that we as observers are predisposed to finding believable. Study into these elements of behaviour span disparate academic fields, from psychology to computer science to insights from storytelling media. This section is intended to trace work on this area so that a useful appreciation of believable behaviour can be used in the development of believable agents.

What can be considered believable in a game agent depends greatly on how we would expect the agent to behave. The challenge is multifaceted in cases where the agent is

supposed to play as an equal to the human, such as an opponent in a first person shooter or real-time strategy game. Do we want the agent to play similarly to a human or to role play within the game world? The believability of AI for non-player roles operates under different constraints. In these cases, the agent must instead act like an intelligent character or creature within the game world. Grading believability in this case depends greatly on what kind of being the agent is simulating. Game characters can vary from human villagers [8] to super-intelligent computers [25] to bivalves or bacteria [26].

2.1.1 Approaches from Computer Science

In his 1950 paper 'Computing Machinery and Intelligence', Alan Turing laid out the concept for what would become known as the now famous 'Turing Test' [27]. In this he begins with the question of whether machines can think. Seeking to avoid the issues inherent with defining the words 'machine' and 'think', he instead replaced the problem with a thought experiment, proposing that we should instead ask if the machine can win an *imitation game*. Suppose a man A and a woman B are in communication with a person C. C does not know the identity of A or B (we can imagine them as being located in separate rooms) but can only communicate with them through written notes. Through asking questions to A and B, and without any other hints C is asked to determine which of the two is the man and which is the woman. Both A and B have the aim of convincing C that they are the woman. Understandably one can imagine the difficulty of C's task; what would constitute sufficient evidence on which a decision could be made?

Turing follows this illustration by asking "What will happen when a machine takes the part of A in this game?". This question poses a key challenge to the idea of machine intelligence, however for many AI researchers the constraints are unsatisfactory; their goal is to understand intelligent behaviour rather than merely provide a façade. For applications in video games however, the façade is exactly what the developers want. Just as there is little appetite for unbeatable AI in games, providing *believable* behaviour in game agents is generally sufficient for entertainment and story-telling purposes.

2.1.2 Perception of Believability

Before we determine the attributes of behaviour that we can describe as believable, it is worth considering exactly how it is that we are in a position to do so. To address this we must consider the human mind, a 1300-1400g, 20 Watt organ that is capable of transforming sparse sensory information into complex cognitive representations. As a product of natural selection it is highly adapted for this purpose, and it has specialized structures for tasks of varying complexity related to processing our evolutionary environment, ranging from interpretation of the visual information [28] to encoding spatial relations in memory [29]. Understanding and interpreting the movements and actions of others is uniquely important for tasks related to survival, such as pursuing prey, avoiding predation and dealing with other individuals [30].

Modern perspectives on the evolution of the human brain suggest that much of its unique and extraordinary complexity reflect adaptation to the immense cognitive demands of living in social groups and interpreting behaviour [29]. A unique consequence of this may be the propensity to anthropomorphize; to ascribe human attributes (such as agency or social meaning) to non-human objects [31]. This tendency was first demonstrated in 1944; in a classic study it was shown that there is a tendency for virtually all people to make up a social plot on being shown hand-drawn animated scenes with interacting geometric shapes, and that this is proportional to specific cues such as the speed and movement of the 'agents' [32]. There is evidence that the propensity is discriminative across several dimensions; for instance the presence and arrangement of facial features heavily influence the perception of humanness in robot heads [33], and that newborns display a unique, unlearned preference for human faces [34].

Clearly the design of believable game agents will be based heavily on the details of our perception. The relationship between mimicking human features and achieving success in convincing observers may be not be linearly proportional. Trying to make artificial agents look as human as possible, peoples disposition towards the agent can become increasingly positive. Further increasing the human-likeness of agents can soon lead to the well-documented phenomenon of the 'uncanny valley', where people suddenly experience a feeling of eeriness and disconcert. The perceptual reasons behind the phenomena are not immediately clear, but have been hypothesized to be caused by a violation of the

brain's predictions where detailed human-like behaviour (such as movement kinematics) is expected alongside human-like appearance (a structurally identical limb or hand) [30]. Studies have been undertaken to determine the physiological underpinnings of the uncanny valley, Saygin *et al* performed neuroimaging of the human action perception system (constituting the lateral temporal cortex, inferior frontal/ventral premotor cortex and anterior intraparietal cortex). They found greater responsiveness for human and non-human robot movement, but less responsiveness for android (human-like movement), suggesting the possibility that overcoming the uncanny valley could be achieved through neurophysiological experiment [30].

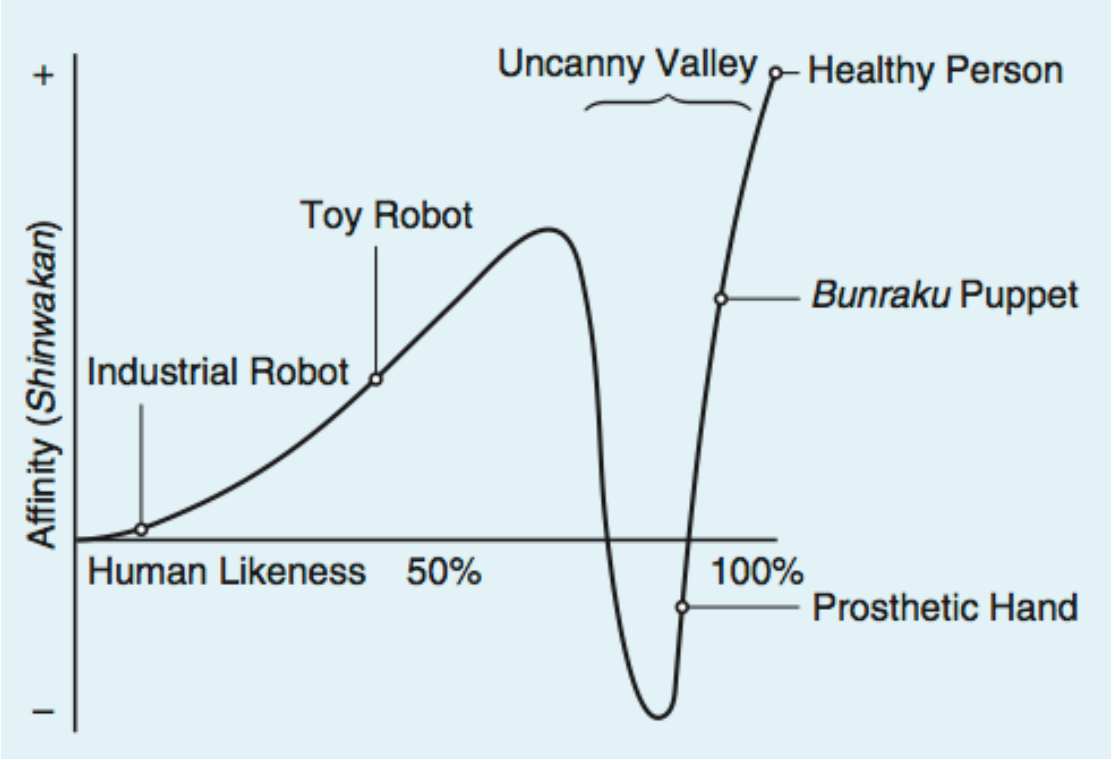

FIGURE 2.1: The Uncanny Valley as detailed by Masahiro Mori, adapted from [1]

The process of designing believable agents may not be successfully applied to all members of society. Although it has been shown that most are quick to attribute human-like mental states to arbitrary objects given the right cues, some of us may be particularly unable to qualify believability. For example, 'theory of mind': the ability to infer the mental states of others is considered to be a cornerstone of human adaptation to complex social situations. Components necessary for theory of mind (such as the attribution of belief and internal mental states on others) has been demonstrated to be impaired in

individuals with autism spectrum disorder (ASD), affecting a substantial portion (1%) of the human population [35].

2.1.3 Believability for Game Agents

How could believability in game agents be measured? As with the Turing test, past approaches to quantifying this have relied on the opinion of a human judge. To this effect research into measuring perceived believability in game agents is largely restricted to the use of questionnaires [36]. However perceived believability has been seen as having a highly subjective component [37]. Alternatives that measure physiological attributes as highlighted above can involve much costly research equipment and support, and may be impractical for those reasons. Togelius *et al* note that participatory observation (where the player judging the believability is playing the game) is prone to distortion effects, and that the best approaches involve an external observer [18]. This has been observed by industry professionals, such as Chris Butcher and Jaime Griesemer of Bungie [38]. When play-testing levels of Halo, they gave players a standard level populated by enemies with different hit-points and attack damage. They observed that players considered the tougher enemies more intelligent, even though the internal agent AI was identical.

Alternative approaches to user-feedback questionnaires have explored automated behaviour analysis, by storing past actions made by an agent in a suitable data structure and analysing them for 'believability attributes', possibly through comparison to the action data generated from real human players [39]. These have for the most part been applied to create or evaluate AI-controlled players. Kemmerling *et al.* developed a 'believability calculator' from user surveys and incorporated this into an AI player for the table-top strategy game Diplomacy [40]. Automated testing for believability attributes was employed by Pao *et al.* [41], where an algorithm comparing the trajectories of AI-controlled Quake 2 players to pre-recorded human and AI player data with the goal of classification.

Several attempts have been made to define believability in the context of virtual agents. These approaches can be traced back to the idea of a 'believable character' in the arts, and have been defined as agents that provide 'the illusion of life, and thus permits the audience's suspension of disbelief'. This principle has been underscored for the creation of believable characters in the arts, quoting the words of pioneering Disney animators

Frank Thomas and Olie Johnston in their book 'The Illusion of Life' [24]. They highlight the particular importance of characters that convey personality, emotional reactions and social relationships.

Principles for building believable agents in games have been proposed. Some are particularly broad, such as the list quoted by Umarov: (a) the agent should have the same basic data processing structure as human players; (b) it should have the same basic sensory and motor systems as human players; (c) it should have the same knowledge as humans players [39]. As mentioned, Togelius notes that any model for optimizing believability should not be from designers *a priori* assumptions on believable cognition, but data-driven. He notes that supervised learning techniques offer promise in creating a model from game agent configuration to believability [18].

It is also conceivable that the best approach for game agents is in optimizing the environment, Togelius and colleagues quote Hebert Simon's book *The Sciences of the Artificial*, where Simon considers the path of an ant on the beach. Although the ant's control system is exceedingly simple, the path it takes is complex. This is potentially as much a product of the environment as the ant, and therefore perhaps the most interesting agent will derive most of its uniqueness from its interactions with an environment. This technique has been employed in commercial games, notably with the 'smart object' system in the Sims series, where the agent derives animations and behaviours from components stored in objects instead of the agent [6].

The recent book 'Metrics for Character Believability in Interactive Narrative' sought to provide a succinct list of objective aspects to agent believability [23]. These are:

- **Behaviour Coherence**: audiences will judge behaviour as they cannot usually observe agent state
- **Change with Experience**: agents should show that they change with story events
- **Awareness**: agents should show that they perceive the environment around them
- **Behaviour Understandability**: audience should be able to create a model of the agent's behaviour motivations
- **Personality**: audience should be able to identify the aspects of an agents behaviour that define it as an individual

- **Emotional Expressiveness**: extent to which an agent expresses its emotions
- **Social**: relationships between agents should be identifiable
- **Visual Impact**: amount by which an agent draws our attention
- **Predictability**: both extreme predictability and extreme unpredictability harm believability

2.2 Architectures for Agents in Games

It is through behaviour that game agents are noticed by a player, though where does behaviour come from? For this one needs an agent architecture; something that can receive input from the environment, output behaviour through actuators and has some program that can map from the former to the latter. This Sense-Think-Act cycle is inspired by academic AI research, and several decision-making architectures have seen use in the design of game agents. Decision making architectures for game agents have often been impacted from fields of research such as psychology or economics. Each of these may be appropriate for different uses; there is no universal standard of the perfect game agent architecture.

If agents are to behave realistically, we must expect them to choose how to behave according to some internal logic. The study of the reasoning underlying an agent's choices is a field known as decision theory. Decision theory offers a game-theoretic approach to formulating a decision making system. In decision theory, the normative approach is concerned with finding the best decision to make, assuming a rational agent deriving choices from an accurate description of the game environment. The descriptive or positive approach attempts merely to describe what decisions are made in what circumstance, that is agents respond passively to input. This is in some ways akin to the behaviourist approach taken by many psychologists in the 20th century, where behaviours are thought of as learned reflexes to stimuli. This is in contrast to the cognitive approach, placing greater importance on mental processes [42]. As we will see, some of the approaches to game agent decision making focus merely on designing a static descriptive model of behaviour in response to stimuli or conditions. Others attempt to model mental processes in cognitive terms to dictate agent behaviour. Others still combine these approaches to

provide decision making that takes agent history and personality into account to provide some interesting level of irrationality.

Several forms of logic are used for decision making models. These include boolean logic, a method of reasoning with logical expressions describing membership of sets, the only caveat being that each logical expression evaluates to a *true* or *false* value. An example of this would be an agent that chooses to attack an enemy IF the enemy is within a threshold radius. Here the action 'Attack Enemy' is dependant on the condition 'Distance To Enemy $<$ Tolerance Threshold Distance'. Boolean decisions can be made arbitrarily specific by depending on any number of conditions.

A notable issue with boolean conditions is that membership of some sets is not clearly binary. When considering what constitutes a 'pile of sand', most would agree that a single grain could not be considered a pile; 10,000 grains on the other hand could. For these cases, where *uncertainty* or *vagueness* apply, it may be beneficial to use a many-valued logic such as fuzzy logic. In fuzzy logic, reasoning leads to descriptions of memberships in fuzzy sets. This allows for degrees of truth; the collection of sand grains being denoted a pile/not a pile can now be described with a continuous function. Fuzzy control has been implemented in control systems such as automatic transmissions and electric razors, the key being to provide a 'concise and intuitive way to specify a smoothly interpolated, real-valued function' [43]. Fuzzy logic has been posited as offering a closer representation of human reasoning [44], where probabilistic and incomplete information (such as the outcome of an action) must be used as the basis of a decision. For the creation of game agents, both forms of logic have been employed. The following section is intended to provide an overview of the most popular architectures for decision-making game agents, with some commentary on the advantages/disadvantages of each framework.

In many cases the actions it can perform will only have a probability of achieving a desired goal. In the case that the agent resides in a non-deterministic environment it must have preferences between the expected outcomes of an action, where some actions are determined to be more preferable than others for a given environment. These observations on game environments have led to further architectures that make predictions on the outcome of certain actions. These include models where actions are chosen based on the merits of their perceived outcome; these include agents where plans are made to

accomplish desired goals given certain conditions and systems where state-action pairs are estimated by the agent or learnt through training.

2.2.1 Note on Background NPC Architectures

Background NPCs are most commonly used to provide detail to lived-in environments such as towns and cities, but are generally not the focus of gameplay. As a result, the architectures used to manage their behaviour are rarely afforded much focus or CPU time [45].

When giving background NPCs the illusion of life, the approach commonly employed is to use 'looping idles': a repeated animation such as a blacksmith hitting an anvil repeatedly. This is commonly paired with other NPCs running random walk cycles where they walk to-and-fro randomly. Both of these approaches offer minor background detail, but do not stand up to any scrutiny; the NPCs will repeat the behaviour *ad nauseam*. They do however have the benefit of being cheap to run, and some have decried the use of complex (and computationally expensive) AI for managing background agents [45].

2.2.2 Scripting and Scheduling

The simplest architecture possible is perhaps *scripting*, where the exact timing and sequence of behaviours is precisely defined by the designer. This may include simple sensory input, such as situations that trigger execution. In the case of characters that populate an environment, this leads naturally to the concept of scheduling as a computationally cheap method of driving the behaviours needed. NPC schedules are treated as a black box, with current time as an input and an action as output. The actions can usually be summarized as 'go here and run this animation', once the action has ended the NPC will be given another action by the schedule. In order to avoid peak times where all agents finish working at 6pm, often the scheduler will randomize the schedule update time with a Gaussian distribution as a means to stagger agents swapping behaviours [45]. Unfortunately the simple solutions offered by scripting and scheduling mean that agents are highly nonreactive to external events, if there is to be any dynamic behaviour (through player-interaction for example) a more sophisticated system is required.

2.2.3 Finite State Machines

A step-up in complexity from the scripting approach is the finite-state machine (FSM), one of the earliest and most widely adopted behaviour modelling approaches. In the FSM model an agent is broken down into a number of discrete *states*, each representing a specific behaviour or internal configuration, with only one state active at a time. States connect to other states through transitions, usually predicated by some conditions. This approach is simple, easy to implement and effective for simple agents. The primary issue with FSMs is scalability, this become most apparent in complex agents with more than 10 or 20 states. Each new state added needs to fit within the conditional framework, within all relevant transitions, and as a result can be fragile to extension. Similarly its structure does not handle situational behaviour reuse, where we may desire actions to be performed in the context of other actions.

An approach to fitting sub-behaviours into the FSM structure in a global state that is executed concurrently with active states, or with state *blips* that can be executed at any time but will always return the agent to its prior state [46]. An extension of this idea is to have multiple FSMs working in parallel, in a Hierarchical finite-state machine HFSM. As an example consider a game agent with the states *Explore*, *Combat* and *Patrol*. The Combat state could in turn manage a state machine that manages the states *ReloadWeapon*, *ShootAtEnemy* and *FindCover*. Once out of the Combat state the agent could return to the most recent active state of the original state machine. This approach similarly addresses the issue of duplicating states for different contexts, and provides much more structural control over the arrangement of states. The approach however does not ultimately solve the issue of overloaded state transitions, and therefore might not meet design criteria for many game agents [47].

2.2.4 Behaviour Trees

The idea of organising behaviour into hierarchical structures led to the implementation of tree-like structures for the efficient navigation between tasks. An example of this is the use of behaviour trees (BTs), used famously for the enemy agents in the first-person shooter game Halo 2 [21]. Starting from a root node behaviours are perused in a hierarchical manner, with each behaviour potentially specifying any number of child behaviours.

Each behaviour defines precondition(s) necessary for the execution of the behaviour, and an action that specifies what the agent should do as part of that behaviour. The algorithm navigates down the tree, only executing one behaviour at each level of the tree if its preconditions are met. This provides an architecture that is simple to implement, highly predictable and extensible. Unlike FSMs however, BTs must potentially evaluate many nodes of the tree in one decision-making cycle and has a resultant computational cost [48].

2.2.5 Planning Architectures

Some more advanced architectures include the use of AI planning architectures, allowing agents to increase their autonomy and flexibility through the construction of action sequences to achieve their goals. Notable applications of planners in games include Goal-Oriented Action Planning (GOAP) and Hierarchical Task Networks (HTNs). GOAP is derived from the older Stanford Research Institute Problem Solver (STRIPS) planner, and was pioneered by Jeff Orkin who implemented it for the game F.E.A.R [49]. GOAP performs 'backward chaining search'; starting with a desired goal the agent can determine a particular sequence of actions that will accomplish it. The sequence of actions chosen will depend not only on the goal but also on the current state of the world and the agent [50]. There may be multiple action sequences that will satisfy the goal, allowing for dynamic behaviour in different contexts. Whereas FSMs and BTs are reasonably deterministic in their decision making, the agent-driven approach of GOAP allows for 'thinking' agents, the resultant emergence of behaviour can be a desirable feature.

GOAP is an example of a backward planner. A forward planner that has seen use in games such as Transformers: Fall of Cybertron [51] is Hierarchical Task Network (HTN) approach. Starting from the current world state, HTN works towards a plan of tasks that will provide a solution to problems. The provided set of tasks for planning solutions take the form of: primitive tasks (an action) and compound tasks (a sequence of simpler tasks). Depending on different world states, tasks will decompose into tasks that required that condition. By searching through available tasks and applying their effects to the world state the agent can build a valid plan. As with the other hierarchical architectures we have seen, the search times of HTN will be proportional to the size of the graph.

2.2.6 Machine Learning as Applied to Agent Behaviour Architectures

Several machine learning approaches have been used in the development of AI for games, including the use of neural networks, evolutionary computation and reinforcement learning [17]. Broadly speaking, machine learning algorithms perform some classification or decision based on received input data. Depending on some metric of their success they adjust their decision process accordingly; as a result they often require training on a large amount of data before they perform well on some meaningful measure. Use in games has been primarily confined to offline learning during game development or between periods of gameplay, usually for parameter tuning. Online (or in-game) learning has been deployed in a small amount of commercial games such as Black and White [52], where player teaching of a game agent was a major gameplay component. One of the major game industry concerns with machine learning approaches is the unpredictable nature of learning, sometimes denying authorial control over evolved agent behaviours. Similarly although agents trained with machine learning methods can become extremely effective game players or converge to single optimum; this is often not conducive to the human player-oriented focus of commercial games [53]. Recent interest in learning applications offer promise for the future of game agent research, for example recent research into locomotion behaviours in 3D environments allowed the emergence of fine-tuned actions such as jumping and running through reinforcement learning [22]. Integrating learning in an effective manner for commercial game agents remains a promising field.

Several noteworthy attempts have been made to improve realism in agent behaviour through machine learning methodologies based on human data; broadly known as Learning from Demonstration. This approach has been widely employed in the field of robotics, where the aim is to develop agent policies from example state-to-action mappings [54]. This has been applied in various efforts to create organic virtual agents in games though there remain issues in encoding contextual information, on which social interactions are highly dependant [55]. Jeff Orkin's 'The Restaurant Game' sought to achieve conversational virtual agents through use of a minimal investment multiplayer onine (MIMO) role-playing game. In this game, players are placed in a small but highly interactive restaurant environment, assigned the role of a waitress or customer and given objectives such as earning money or having dinner. Through an unsupervised learning methodology

based on data from over 5,000 gameplay sessions, a *Plan Network* was learned, a statistical model that encodes context-sensitive expected patterns of behaviour. Although players would sometimes exhibit aberrant behaviour, this would 'wash away' statistically given the large data set of interactions. When agents generated with this model were demonstrated to human testers there was strong agreement on the realism of the agent's behaviour, noting their responsiveness to a wide variety of interactions. The experiment was concluded after six months, and demonstrated the potential for using games to teach AI, in this case about human behaviour [56].

2.2.7 Utility Systems

Utility-based decisions, based in economic theories of human behaviour, have been used in popular multi-agent based games such as the Sims series [6] and have been highlighted as an effective and reliable methodology by game agent industry consultants [57]. The concept of utility can be described as follows: the usefulness of a given action chosen by a decision making agent can be described with a single, uniform value known as its *utility*; the quality of being useful.

The concept of utility has been used extensively in fields such as game theory or economics where it is used to characterise the idea of a *rational individual*: an agent that always chooses the option that gives the greatest utility. Such a delineation is rarely straightforward however; most decisions are multidimensional in their considerations. For example, should an agent buy an ice cream from a nearby corner shop or from the supermarket a short walk away? The corner shop is local, personal but expensive. The supermarket however will have a greater selection and cheaper ice creams, yet perhaps the agent is concerned that the impersonal experience will feel alienating. To make this decision, utility theory would have the agent weigh up these considerations to give a total preference, known as the *expected utility* for either option. The most common technique to acquire the expected utility is to multiply the probability of each outcome by the utility score and sum these weighted scores.

$$U_{exp} = \sum_{i=1}^{n} D_i P_i \tag{2.1}$$

Where D is the desire for the outcome (utility) and P is the probability that the outcome will occur. These expected utility scores must be consistent and on the same scale for comparison to make a final decision, so commonly scores are normalised. By the principle of *maximum expected utility*, this is applied to every possible option and an action is chosen that confers the greatest expected utility [43], [58]. As highlighted before, calculating the utility of a given action will generally depend on several pieces of data. Calculating the utility for a decision factor necessitates a function that will map from states to a utility value. The core of using utility theory for decision making is in understanding the relationship between the input and output, and being able to define the function that describes their relationship. Assuming the data is relevant to the decision, an arbitrary number of factors can be considered. A dragon might decide to attack and eat a nearby dwarf, but the utility of this action could depend on the dragon's hunger, the distance to the dwarf and the dwarf's perceived combat strength. A value for each of these components factor into the action's consideration, for that reason they are referred to as *decision factors* or consideration *axes*.

2.3 Utility Theory for Believable Behaviour

Utility systems offer the benefit of making fuzzy decisions, where actions aren't based on conditions so much as scores. As we have seen this offers a more accurate model of human decision making. As will be discussed in this section, utility functions can be tweaked to achieve the different decision making processes made different agents. This accommodates the ability to specify the values and personality of agents, simply by adding some variability to utility functions and scoring. Several methods have been used to develop utility-based decisions in game agents, this section will offer an introduction to the most popular.

2.3.1 Infinite Axis Utility System

In the Infinite Axis Utility System, developed by Dave Mark [57], the game agent has a number of atomic actions. These represent the agents behaviour, and once chosen the agent will do whatever the action consists of. Each action has a number of 'axes' (considerations) that are used to weight the utility of the action, and are given an input

and a number of parameters to work with. Treating each consideration as an element of a dynamic data-structure such as a list, axes can be added or removed. For example, the action "Use Health Kit" could consider the agent's health. The action "Move to Cover" could consider current threat level of the agent, its health, amount of ammunition and distance to cover. Actions could be evaluated on a per-target basis, e.g. there could be multiple sites where the agent could take cover. Each of the axes would consider the inputs and evaluate them on a response curve. Noticing that response curves often took the form of four functions (Linear, Quadratic, Logistic and Logit), Mark saw that these could be tweaked to designer specification by providing four parameters: *m, k, b, c*. Linear/quadratic curves could be specified with the equation:

$$y = m(x - c)^k + b$$

Where m specifies the slope of the line, k is the exponent, b and c specify the y-intercept and the x-intercept respectively. Logistic/Logit functions are specified with:

$$y = m(x - c)^k + b$$

Where m specifies the slope of the line at the inflection point, k is the vertical size of the curve, b provides the y-intercept and c the x-intercept at the inflection point. With this system, designers could create a fine-tuned curve by specifying only six variables, the input parameter x, the curve type and *m, k, b, c*. When the agent is deciding on an action, the agent will look at all the actions, evaluate their utility by multiplying each action's axes to create an action score. These could be then chosen based on the highest scoring action or by random selection of options that have been multiplied by a weight. In order to achieve prioritization tiers of important actions (e.g. if the agent must dodge a grenade) by providing a weight coefficient for each action. These could be multiplied by the action to denote increase or decrease its score as a reflection of its priority. This system, Mark argues offers a useful architecture for game agents as it is easily extensible, fast to process and the consideration axes system offers fine-level control [57].

2.3.2 Dual-Utility Reasoning

The strength of combining utility-based reasoning with probabilistic decision making led to the development of dual-utility reasoning, notably by Kevin Dill of the Lockheed Martin Advanced Simulation Center [22], [59]. Dill noted that there are two common approaches to utility-based game agent AI. The first, based on *absolute utility* evaluates each option and chooses the one with the highest score. The second, *relative utility* or *weight-based random utility*, chooses actions at random but uses the score of each action to influence the probability that action will be chosen. In the latter the probability of an option being selected P_o is determined by dividing the utility of that option U_o by the utility of all other options:

$$P_o = \frac{U_o}{\sum_{i=1}^{n} U_i}$$

Dill observed that both approaches have unique disadvantages. The absolute approach can result in deterministic behaviour whereas the weight-based approach can result in extremely low-utility actions being chosen, resulting in choices that might be perceived as stupid or non sequitur. Dual-utility reasoning was designed to combine both to avoid either scenario. Actions are assigned two utility values, a rank and a weight. Rank divides actions up into categories via an absolute utility approach so that only actions of the top rank will be selected. Weight is then used to provide weight-based random choosing from high-ranked actions.

2.3.3 Hierarchical Utility Reasoning

Utility-based reasoning offers rational decision making in gray-area situations as a form of fuzzy logic, offering greater nuance than many other popular AI architectures based on boolean logic decision making. When supplemented with weighting or probabilistic selection deterministic decision-making can be avoided. Utility-based architectures, such as the Infinite Axis System also offers flexibility as modular actions can be slotted into an agent's behavioural repertoire. The dual-utility based reasoning similarly offers performance boosts (only high rank actions will be considered at any time) but also a reduction in nonsensical action-selection (such as an agent choosing to reload their weapon when

its magazine is at capacity). With this in mind, a useful extension to utility-based decision making architectures can be through the implementation of a hierarchical action structure, such as a behaviour tree [7].

This approach was used by Dill not only in a commercial game [60] but also for the creation of the "Angry Grandmother" as part of an immersive training environment demonstration [22]. This was a mixed-reality character portraying the elderly grandparent of an insurgent as part of a training simulation for trainees. The trainees would be tasked with searching the house of an insurgent's grandmother for contraband and be faced with the belligerent but non-threatening grandmother character that would react to the search of her house. Highlighting the importance that the character had to be "believable, culturally authentic, nondeterminisitic and reactive within the limited scope of the scenario", Dill built an AI that combined behaviour trees with utility-based reasoning. Within the architecture, the 'component reasoner' used the hierarchical structure of a behaviour tree with dual-utility for the decision making at each level of the tree. The character had five behavioural states: Stand and Listen, Rant, Inconsolable, Surrender and Dead. Each of these, along with their substates, would be chosen on the basis of the behaviour of the trainees engaging with the simulation. For example, the Rant state had the three substates, denoting her emotional response to how close the trainees were to finding the contraband: Cold, Warm and Hot. The behaviours were limited to the selection of character animation and spoken utterances, with importance placed on the character responding to the search without repetition or determinism. The system was successful in its realism, with one squad leader quoted as saying "the angry grandmother acted exactly like women I experienced in the country", however Dill notes that the bottleneck on the system was the development of the large number of animations to maintain the suspension of disbelief [22].

2.4 Procedural Content Generation

PCG offers much to the game industry, perhaps most notably in alleviating the pressures on content creators (game artists, designers etc). However content created through PCG often results in some limitations on the ability of game developers to maintain a high level of creative control. Proponents of PCG often make the argument that the benefits

of outsourcing much of the creative work to automation will benefit lower budget projects and inspire human content creators through unexpected creations [12].

There have been several technical approaches employed in PCG algorithms, generally by creating expansive content from a much more compact representation such as a seed or multidimensional vector. PCG can be offline, where the content is generated and tailored by human designers before shipping, alternatively is can be online, where the content is generated during runtime. The latter can be employed as a form of compression (No Man's Sky shipped on the Steam store as a 2GB download, 1.5GB of which was audio [11]). Similarly content can be generated through a stochastic process or a deterministic one where the algorithms produce identical content for a given input. Approaches for PCG can be simulation based, such as in Dwarf Fortress where an initial world is given detail (specified with operators) through the simulation of natural forces such as erosion [13]. Constructionist methods piece together pre-made building blocks according to a specified algorithm, such as in design of a building where the contents of rooms are explicitly laid out by designers but the layout of the rooms changes from iteration to iteration. Generate-and-test algorithms incorporate the generation of content but then tests it according to some criteria, if it fails the test the content can be recreated in part or in its entirety.

Search-Based PCG (described by Julien Togelius in [12]) is a special case of the generate-and-test approach, characterised by a defined grading (ie beyond a simple pass/fail) of the candidate content through a 'fitness function'. High-fitness instances are used in the generation of future content, often with some form of evolutionary algorithm providing small random changes or recombining highly fit copies to successive generations. Togelius revives the useful comparison to genetics, where the 'genotype' data-structure (that encodes the content) is mapped to a 'phenotype' that is graded by the fitness function.

2.4.1 Procedural Content Generation for Game Agents

Procedural generation of content related to game characters has been employed to mixed success. Ambitious attempts by the industry have included 'Spore' [61], where fantasy animals could be generated from a small amount of data. This resulted in a large possibility space of creature morphology, complete with unique, generated animations reflecting

their anatomy. Other approaches have included generation of 3D humanoid mesh models through principal components analysis, reference fitting and genetic algorithms [62].

Although 'game content' is commonly defined to the exclusion of NPC behaviour and artificial intelligence [12], [63], the use PCG in this direction has been attempted. These attempts are usually in the form of introducing variability in behaviour without excessive hand-authoring of each individual agent. Christopher Dragert and colleagues developed a high-level *Statechart*-based system for modelling behaviour on background NPCs with the same core set of actions. This system allowed them to apply automatic or manual variation on behaviour parameters or statechart structure to create individual AIs with discernible behavioural differences [64].

Szita *et al.* describe a system for automatic macro generation, where a macro is a sequence of actions that are handled as a single unit [53]. Using a cross-entropy method (a general algorithm for combinatorial optimization tasks) to learn macro actions as defined in a rule base along with a fitness function to update the probability of a macro sequence being selected, they create a model that chooses macros based on macro diversity. They achieve this by evaluating fitness based on the macro's strength (effectiveness in the game) in addition to prioritizing macros that are significantly different to previously learned macros. By combining this with dynamic scripting, a reinforcement learning technique where the macros could be chosen based on effectiveness, they create a system that they claim this maximizes effective yet interesting adaptive behaviour.

2.4.2 Randomness Techniques for Game Agents

One mainstay of game programming is utilization of randomness. Randomness is employed in games for areas that benefit from not being too predictable, such as large generated environments or when non-repetitive behaviour is required for game agents. Randomness is notoriously difficult to define. However the mathematician's definition is in many ways the easiest: randomness is where a real number within the unit interval [0,1] has some probability p [65]. This is an example of uniform randomness, but many variables require a slightly more complex probability distribution. Randomness simply selects a value according to the distribution. Humans are famously poor at generating randomness, moreso at recognising it, claiming that randomness 'does not look random'

[66]. As will be discussed, this has significant ramifications for the use of randomness in games.

2.4.3 Generating Randomness

Whereas we are poor at generating randomness, it can be argued that computers are equally disadvantaged. Computers after all operate according to deterministic processes. For this reason, algorithms to generate sequences of 'random-looking' numbers are referred to 'pseudorandom number generators' (PRNGs). These are completely determined by an initial 'seed' value fed as an initial state, and operate iteratively. One of the earliest algorithms was the middle-square method suggested by John von Neumann, and operates as so: take any number, square it, remove the middle digits as the 'random number' and use this 'random' number as the seed for another iteration. If repeated *ad infinitum*, this and any PRNG will eventually repeat itself with some periodicity. Indeed if a PRNGs initial state contains n bits, its period can be no longer than 2^n. All procedural content generators that rely on PRNGs are underpinned by this limit. Many embrace the relationship between consecutive numbers in a pseudo-random sequence, allowing numbers to 'wander' in a smooth manner. A common example is Perlin Noise, used extensively in computer graphics to generate 'coherent' randomness. By combining random number generation with smooth functions at several scales, smooth yet grainy sequences of numbers can be generated. Perlin noise has been used to specify nuanced agent behaviour such as movement (steering, speed, acceleration), attention (responsiveness) and play style (offensiveness, defensiveness) [66].

2.4.3.1 Filtered Randomness

When people think of randomness they expect randomly selected variables to be quite evenly distributed, such as the following binary string:

101011001010101001001101

What many don't realise is that true randomness can be expected to contain 'runs' of identical (or close) values, as can be seen in the sequence generated below (generated through RANDOM.org [67]):

111100001110110110001011

This is of particular importance in games, which are centered around player experience. Therefore for randomness in games it can be useful to adapt approaches with human psychology in mind. To address this, a common technique in games is to use *filtered randomness*. Filtered randomness generally takes an authored approach to identify and rectify anomalies that will stand out to the player. Often these will involve filtering randomly generated variables for repeating sequences (such as '3**444**6' or sequences where the numbers are close together, as in '3**44741**'), sequences where numbers descend or ascend (such as '2345') or too many values at the top or bottom of a distribution range.

2.4.4 Generating from a Gaussian Distribution

Commonly, when seeking to emulate nature realistically it can be beneficial to utilize the Gaussian (or normal) distribution. In Gaussian distributions, variables are centred around some mean value μ, with some standard deviation σ (such that 68% of the values within the distribution fall within one standard deviation of the mean, 95% fall within two standard deviations), according to the equation:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{(2\sigma)^2}}$$

Normal distributions are used ubiquitously in the natural and social sciences to represent variables whose distributions are not yet known. When many contributing factors lead to the generation of phenomena (such as a myriad of genetic and environmental influences on human height), the result will often be normally distributed. This is a result of the Central Limit Theorem, which establishes that when independent random variables (such as the height of a large sample of individual people) are added, their normalised sum tends towards a normal distribution [66].

When generating random Gaussian numbers, the Central Limit Theorem states that the addition of uniform random numbers in the range [-1,1] will approach a normal distribution with a mean of zero and a standard deviation of $\sqrt{\frac{K}{3}}$, where K is the total number of input uniforms [66]. Study into algorithms for Gaussian PRNGs is extensive, and there exist variations for the needs of almost any simulation environment [68].

2.4.5 Level-Of-Detail Systems for Game Agents

Where there are large environments with many agents, it can be preferable to persistently simulate agents even when not being directly observed. To address this, Level of Detail (LOD) simulation is sometimes implemented to reduce unnecessary high detail computation. These allow lower fidelity simulations to run in the background, sometimes trained with a model for predicting the outcomes of game events [69].

Level of detail systems for game AI have implications for the use of randomness in agent behaviour. For instance it has been suggested that deviations in game agent behaviour as a result of Perlin noise may have an obvious lack of rationale, in which case it may be better to switch to a more logic-driven framework when a game agent is being directly observed by the player [66].

LOD systems for game agents similarly offer a form of compression, analogous to PCG. For either it can be beneficial to determine the minimum amount of information to specify, store and create content. For game agents, it would be useful to develop a method for specifying agents with unique personal behaviour in a small amount of information. For this we must consider ways of describing personality.

2.4.6 Personality Modelling

For background NPCs, it has been highlighted that its important to provide the illusion that they have their own personalities [45]. For this it can be useful to have means to describe and quantify the important details of personality, if it is to have any perceptible influence on behaviour. Should it be possible to describe a personality in as few variables as possible, it would enable the mass creation of characters.

For human personalities, several attempts have been made to develop a comprehensive taxonomy of human personality traits. One of the most successful attempts has resulted in the Five-Factor Model (FFM), so much so that has become the dominant approach for representing human traits today in psychological research [3]. The FFM was developed based on the association of words commonly used to describe the same person, an effect observable through factor analysis on personality survey data [2]. The FFM is a hierarchical organisation of personality traits based on five dimensions: Openness to Experience, Conscientiousness, Extroversion, Agreeableness and Neuroticism (conveniently abbreviated as 'OCEAN'. For this reason the traits are commonly referred to as the 'Big Five'. The FFM structure can be demonstrated across cultures and groups. The five factors are described the the figure below.

Factor		Factor definers		
Name	Number	Adjectives	Q-sort items	Scales
Extraversion (E)	I	Active	Talkative	Warmth
		Assertive	Skilled in play, humor	Gregariousness
		Energetic	Rapid personal tempo	Assertiveness
		Enthusiastic	Facially, gesturally expressive	Activity
		Outgoing	Behaves assertively	Excitement Seeking
		Talkative	Gregarious	Positive Emotions
Agreeableness (A)	II	Appreciative	Not critical, skeptical	Trust
		Forgiving	Behaves in giving way	Straightforwardness
		Generous	Sympathetic, considerate	Altruism
		Kind	Arouses liking	Compliance
		Sympathetic	Warm, compassionate	Modesty
		Trusting	Basically trustful	Tender-Mindedness
Conscientiousness (C)	III	Efficient	Dependable, responsible	Competence
		Organized	Productive	Order
		Planful	Able to delay gratification	Dutifulness
		Reliable	Not self-indulgent	Achievement Striving
		Responsible	Behaves ethically	Self-Discipline
		Thorough	Has high aspiration level	Deliberation
Neuroticism (N)	−IV	Anxious	Thin-skinned	Anxiety
		Self-pitying	Brittle ego defenses	Hostility
		Tense	Self-defeating	Depression
		Touchy	Basically anxious	Self-Consciousness
		Unstable	Concerned with adequacy	Impulsiveness
		Worrying	Fluctuating moods	Vulnerability
Openness (O)	V	Artistic	Wide range of interests	Fantasy
		Curious	Introspective	Aesthetics
		Imaginative	Unusual thought processes	Feelings
		Insightful	Values intellectual matters	Actions
		Original	Judges in unconventional terms	Ideas
		Wide interests	Aesthetically reactive	Values

FIGURE 2.2: Descriptions of the FFM, adapted from [2]

Importantly the FFM is based on *a posteriori* descriptions of personality traits, not on predictions made through neuroscientific experiment or theory. Therefore although it

offers an intuitive way to describe a personality in a small number of variables, it can be difficult to interpret the meaning of the Big Five, or to make predictions of their influence on behaviour. Nevertheless researchers have used the model to predict individual differences in a variety of settings. Particular focus has been on the relationship and influence of these personality traits on feelings, behaviour and patterns of thought [3], [4].

Borghans and colleagues paid particular attention to the effect that personality traits have on behaviour, in particular through their influence on economic and social outcomes. They cover this in a review of relevant psychological research into personality and its effect on behaviour [3], [4]. They document how FFM personality traits have been demonstrated to have significant influence on traditional preference parameters in economics: time preference, risk aversion and preference for leisure. It is also demonstrated that there is evidence personality traits offer equal or greater power than cognitive traits for schooling, occupation (including job performance), wages, health behaviours and crime. The correlations between FFM traits and socioeconomic outcome documented in [3], [4] are highlighted in the Figure below, notably the intelligence quotient (IQ) measure is the best predictor across most of these outcomes.

Integrating these findings into behaviour models remains challenging. As mentioned, economists often model individual preferences through utility functions, where individuals evaluate options according to the perceived value that each option would bring. Borghans *et al* note the difficulty of deriving a direct map from personality onto preference values that would effect utility calculation. For example, it has been suggested that the factors influencing time preference utility may be most usefully thought of as tri-dimensional: underpinned by *impulsivity* (tendency to act with spontaneity), *compulsivity* (tendency to stick with plans) and *inhibition*. The prospect of providing useful functions relating FFM traits and behaviour is still underway, though it is highly likely that any model for predicting behaviour based on personality descriptions will involve the FFM.

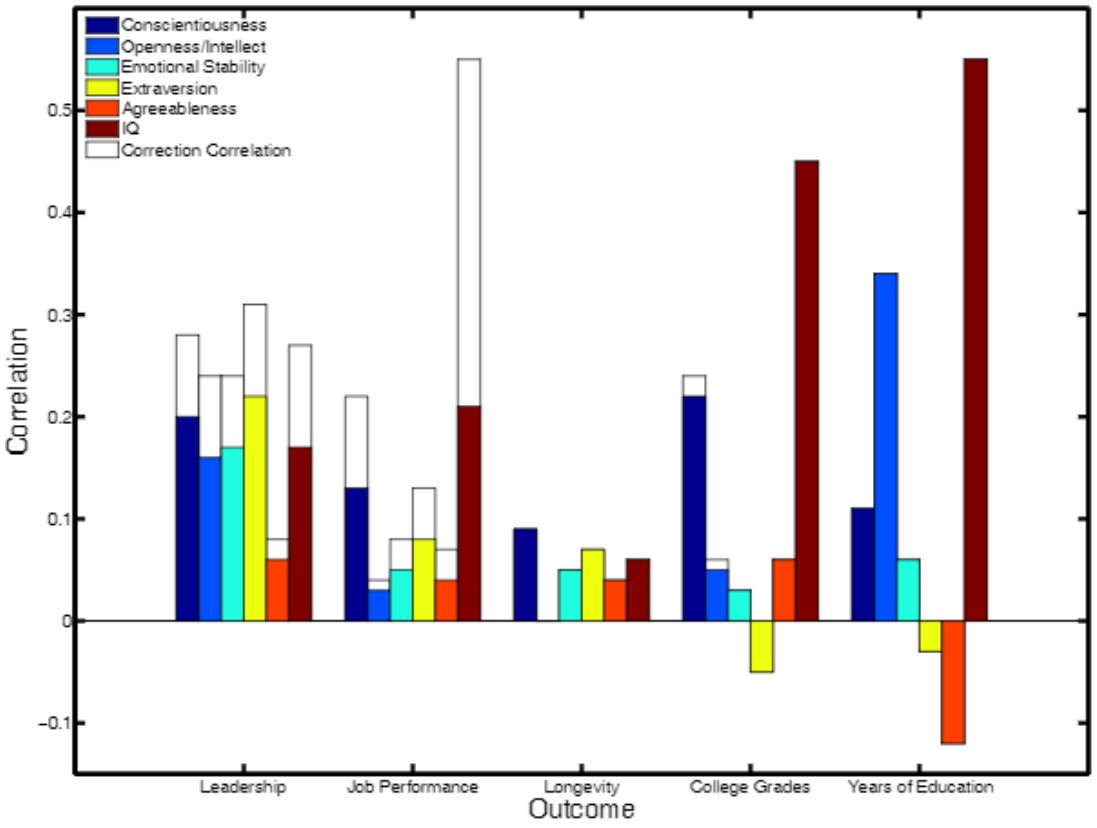

FIGURE 2.3: Correlations between traits and socioeconomic outcome, adapted from [3], [4]

Chapter 3

Design

3.1 Previous Work

The primary inspiration for this model comes from work by Fallon [70], and continued by Mullally [5]. Their model sought to achieve agent individuality through action selection based on agent state, though use of a non-hierarchical manner meant that complexity of decision making was proportional to the number of actions, and procedural generation of agents was highlighted by both as an avenue of future direction. This model seeks to address both of these aspects as a continuation of their their work.

3.2 Model Definition

Decisions are commonly made on the basis of both internal state (e.g. emotional or physiological state) and external state of the environment (e.g. temporal, meteorological or societal state). It is plausible therefore that individuals display unique preferences and display irrational behaviour on the basis of some meaningful difference in their internal cognitive state. These differences manifest in how individuals identify or choose between alternatives due to differences in values and preferences.

We have seen that for background characters to be believable, a common criteria is for them to display personality. This is effectively done by standard tools of the trade in interactive media; for the expression of a grumpy dwarf one might need gruff utterances (voice acting) and a shuffling gait (animation). In addition to these it is important that

personality shines through in behavioural preferences; if we followed the aforementioned dwarf around for a day we might see them eating greedily, working hard in the mines followed by an evening of drinking bitterly in the corner of a dark tavern.

For background agents it would be useful to have a model that allows for the creation of such characters, allowing easy specification of their personality. Similarly, a robust system that could populate a city with different characters without necessitating extensive hand-authoring would be ideal for the next generation of interactive worlds.A model for decision-making agents, whose goal is to allow for the creation of agents with different personalities would therefore need some representation of an agent's preferences. For simplicity it would be ideal if preferences could be specified for discrete aspects of an agent's state (e.g. treating hunger and energy as separate 'State Parameter' values). We have seen that the concept of simple utility functions is a robust way to treat the relative importance or urgency of certain state values. This allows for the direct mapping of individual aspects of agent/environment state to a utility score that can be used to form the basis of decision making.

This provides a convenient way to specify preferences, certain agents could value or discount the relative importance of state parameters according to some function that maps their personality to their values.

3.2.1 Agent and Environmental State

If decisions are made on the basis of the internal/environmental state, there must be useful representations of this data that can be accessed by the decision-making architecture. Internal agent state can be stored by the agent itself, as individual **State Parameters**. These can be stored as a numerical value (such as a floating point number) as a flexible representation of physiological and emotional state. Similarly they can be used for storing material state (such as the personal wealth of the agent), or for opinions (such as a simple relationship ranging from 'highly favourable' to 'highly unfavourable'). A finite number of these could be specified according to the needs of the character representation, but could follow common parameters used in academia (see Figure 3.1) or industry (see Figure 3.2)

State Name	-1	+1
Health	Injured	Healthy
Mood	Sad	Happy
Energy	Tired	Full
Satedness	Hungry	Full
Passivity	Aggressive	Passive
Sociability	Unsatisfied	Satisfied
Soberness	Drunk	Sober
Memory	Forgetful	Retentive

FIGURE 3.1: Agent Emotional Representation as specified by Mullally, adapted from [5]

FIGURE 3.2: Agent 'Needs' Representation as depicted in the Sims [6]

In addition to agent-specific parameters it could be useful to represent global parameters such as temporal factors (time of day, day of the week), atmospheric (temperature, weather) or even global social factors (community morale) with a similar data structure. These factors could be set globally and referred to by the agent. This would allow for offset or bias in how agents treat these universal factors, such as agents that either discount the time of day as important, or those who simply live on a different schedule.

3.2.2 The Action Hierarchy

For fine-detailed behaviour it become necessary to specify a wide variety of possible actions. For instance, we might want to have multiple ways a farmer character can go about their work. Possible farmer behaviours could involve tending the crops, feeding the livestock, bringing goods to the market or sleeping under a tree. These actions would differ to those the character would conduct during a social occasion, or when eating at a restaurant.

As we has been noted, a defining characteristic for believability is that behaviours are coherent; any system that regularly allowed agents to jump from one behaviour to another

totally unrelated one could appear incongruous. For this a wide-variety of actions is best stored and accessed in a hierarchy. This avoids the cost of evaluating all actions and performing direct comparisons by storing actions in a hierarchical tree structure. This is analogous to the structure we have seen in previous action-selection models such as behaviour trees (see Figure 3.3) . At each level of the tree, utility-based evaluation of each node could be used to choose a path down the tree. This reduces the need to perform utility calculations; if a character has already evaluated its hunger and decided that 'Have a Snack' is a suitable action, it wouldn't need to check its hunger levels again when choosing between two possible sandwiches.

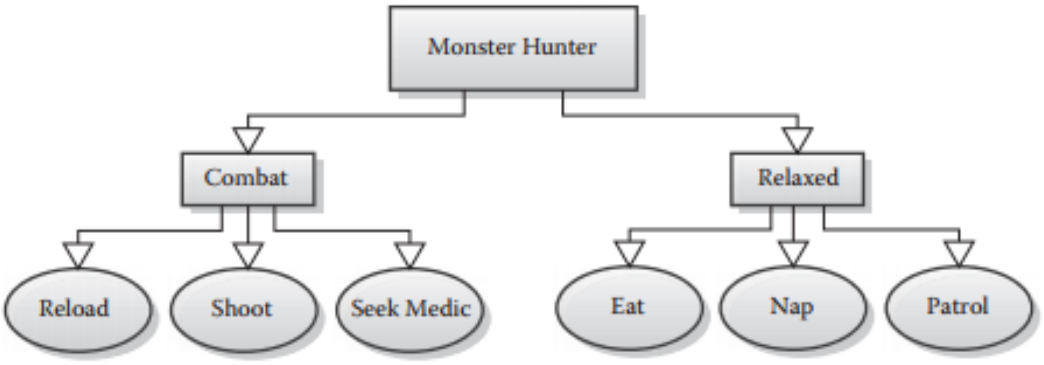

FIGURE 3.3: An example tree structure for action selection, Adapted from [7]

This would similarly allow for easy addition of new actions; designers could simply add an action (with or without subactions) to any level of the tree. The modular nature of the approach similarly offers the ability to 'teach' agents new behaviours, where agents could switch professions and acquire a whole sub-tree of associated actions. The system would also permit introduction of a 'Smart Object'-style architecture [6], agents could be given area-specific actions. This would allow an agent to walk into a tavern and temporarily acquire actions for a tavern-context such as 'Start a Brawl' or 'Sing Boorishly', only to have them removed from the hierarchy after they leave the location.

3.2.3 Utility-Based Action Selection

At each level of the hierarchy, nodes select from their child actions on the basis of each actions utility. Selecting actions could be carried out via picking the top scoring action, or by sorting the actions and picking through some semi-random heuristic. Assuming

useful functions mapping state to the utility of an action (and its effect on that state), this could offer an accurate means to model probabilistic decision-making. These functions would calculate the utility with respect to one state parameter only, so that each utility calculation could be easily specified with a simple univariate function. These functions would necessarily be action specific. For example, consider the two possible actions 'Sleep' and 'Work Hard'. Both might consider the state parameter 'Energy' as a relevant variable for the decision. Clearly the mapping from 'Energy' to the utility of choosing to 'Sleep' would differ from the mapping to the utility of 'Work Hard', indeed it would be more realistic to each function to have an opposite-sign slope (see Figure 3.4).

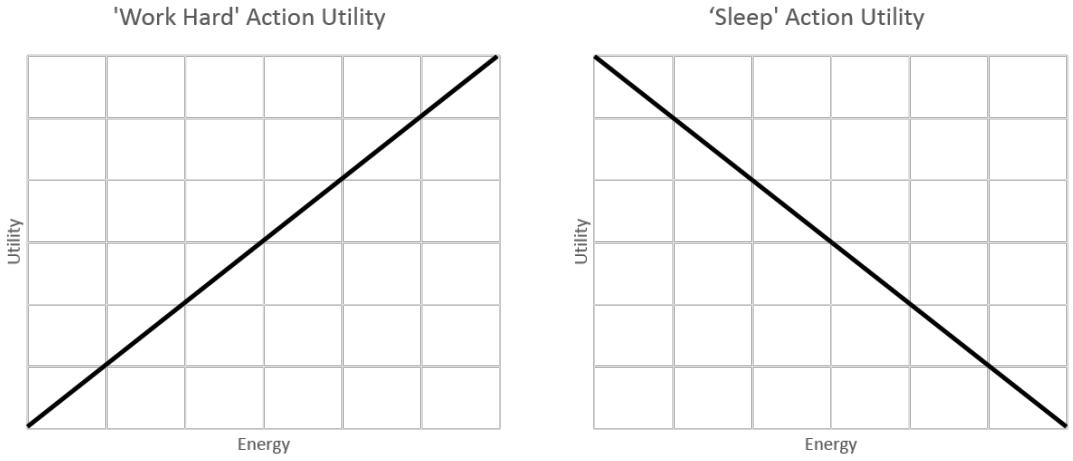

FIGURE 3.4: Action-specific Utility Functions for agent 'Energy'

Specifying utility functions as univariate permits a flexible system for utility calculation; however many decision might rely on several different parameters to consider. For example, an agent deciding whether to perform the action 'Go To Work' might want to consider its bank account, its energy and the day of the week (perhaps the agent lives in a world with weekends). Specifying multi-dimensional utility functions that map several parameters to a single utility value could be unwieldy to design and result in unintended maxima and minima in the utility space. Treating each relevant state parameter as a separate action **Consideration** as outlined in the Infinite-Axis Utility model [57], would allow for easy specification and visualization of expected utility for given agent states. An action could have an arbitrary selection of considerations for every action, overall utility scores for an action could be calculated as the mean of all utility functions or some combination thereof.

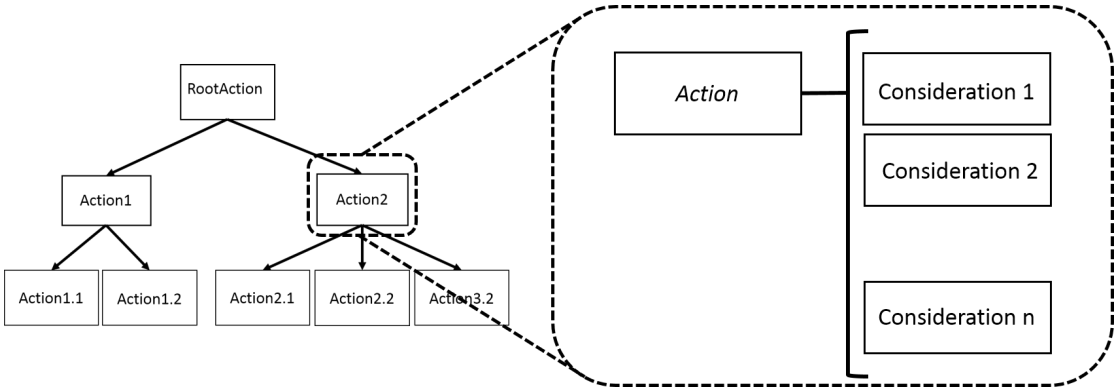

FIGURE 3.5: Action hierarchy, each action with its own considerations

In addition to offering a system for believable decision-making, this approach offers useful ways to model variation in agent preferences. This is one of ways in which personality can be applied in a meaningful way to decision-making, as different personal values can lead to unique behavioural details (one of the key metrics for believable characters [23]). This could be achieved with different weights being applied to different state parameters. For instance, a particularly miserly individual could value his personal wealth as a more important consideration than his hunger, leading him to go hungry rather than spend a penny. In order to represent these preferences there would necessarily be some function that creates specific weights for each consideration on the basis of an agent's personality. Through weights it would be possible to model meaningful variation in how agent's value certain considerations over others. With this system the procedure for evaluating and selecting child actions could be as outlined in the pseudocode below.

```
void EvaluateChildActions()
{
    Action topAction;
    topUtilityScore = 0;

    foreach (Action in ChildActions)
    {
        ActionUtility = 0;
        foreach (Consideration in Action.Considerations)
        {
            considerationUtility = Consideration.CalculateUtility();
            considerationUtility *= Consideration.Weight;
            ActionUtility += considerationUtility;
        }
        ActionUtility /= NumberOfConsiderations;

        if (ActionUtility > topUtilityScore)
            topAction = Action;
    }
}
```

3.2.4 Performing Actions

Once an action has been chosen there needs to be related behaviour, along with a defined effect on the state of the agent/environment. To imagine this we can consider a 'Clean Window' action; to convey this realistically the agent must display related behaviour (through text or animations), and signify that an aspect of the environment has changed (perhaps the window is now cleaner than it was before). If decisions are made on the basis of the agent's state, we might observe the agent repeating this behaviour indefinitely.

It is therefore useful to model the effect of particular behaviours on the agent; perhaps the agent becomes tired and hungry after a period of window washing. To address this the concept of a **State Modification Vector** was used in the previous models by Fallon [70], Mullally [5] and Noonan [71]. This is analogous to the agent's State Vector, representing a transformation on the agent's state as a result of performing an action. Each action would have a separate State Modification Vector representing the effect of the action; not all actions would effect all states, sleeping for instance would have no effect on an agent's wealth. Likewise, a State Modification Vector would have no effect on global state, such as the weather or time of day. Two example state modification vectors are shown below for two possible actions 'Work Hard' and 'Sleep' (Figure 3.6). Both vectors operate on the energy, hunger and wealth state parameters with different effects. Working hard causes a loss of energy but a gain in hunger; sleeping causes energy to be replenished but still causes hunger, less so however than when the agent is hard at work. Working could cause a rise in personal wealth, whereas sleeping has no effect.

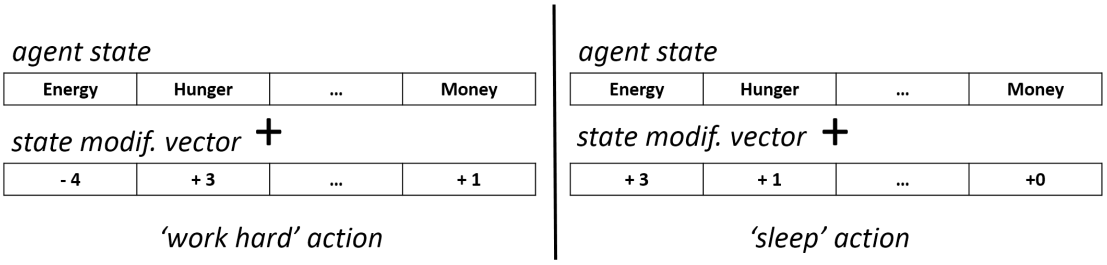

FIGURE 3.6: Example State Modification Vectors for actions 'Work Hard' and 'Sleep'

Through the effects of actions chosen by agents mediated by state modification, behaviour coherence can be achieved. An agent that works hard all day will naturally choose to sleep, given the appropriate utility modelling. This also presents a system for the effects of the environment on an agent, if it were to catch a cold, a state modification vector could apply a transformation on the agents state, affecting its future behaviour.

This model presents an additional opportunity to implement personality modelling. In this approach we consider how agents with different personalities might react to different experiences. For example, individuals that are described as highly extroverted (the 'Extroversion' trait in the Five-Factor Model) are usually described as deriving energy from stimulation, such as social activity. Introverts on the other hand are usually understood as deriving energy from low-stimulation environments, and might be more readily exhausted from social activity. This could readily be captured with the state modification

vector, for an agent with high extroversion, a 'Socialise' action could have a modification vector with the 'Energy' index scaled up. This agent would become less tired from social activity. For an introverted agent the 'Socialise' modification vector's 'Energy' index could be scaled down, modelling a more exhausting effect.

3.2.5 Personality Modelling

Given that any personality model would have multiple parameters, it may be that different aspects of personality affect/value the same agent state parameters. For instance, the effect actions have on an agent's mood might be a product of the agent's neuroticism *and* their conscientiousness. In order to represent the contributions each personality dimension has on the valuing/modifying of each state parameter, the relationships could be contained in a simple 2-dimensional array.

For the calculation of parameter weights (how much an agent cares about hunger/wealth/weather etc) this array would be n by $l + m$, where n is the number of personality dimensions, l is the number of possible external state parameters (such as the time or weather) and m is the number of internal state parameters (hunger, energy etc). For the generation of personality contributions to state modifiers (e.g. by what amount socializing detracts from an agent's energy), an array of size m by n is required as it would only pertain to internal (and therefore modifiable) values.

In order to keep the size of the modifier and weight generation tables to a reasonable minimum, there must be some representation of personality that employs the smallest number of variables. A well established model for personality factors is the Five-Factor model, based on recurrent factors from analyses of personality rating studies [2]. These are considered to be candidates for the basic dimensions of personality; the "most important ways in which individuals differ in their enduring emotional, interpersonal , experiential, attitudinal, and motivational styles" [2]. The five factors are Openness to Experience, Conscientiousness, Extroversion, Agreeableness and Neuroticism.

Importantly these factors are an *a posterioiri* model for personality, rather than being derived from some understanding of the underlying causes (such as individual differences in environment, genetics or neurobiology). Although some effort has been made to assess the predicative power of FFM personality traits with regards to behaviour and

socioeconomic outcome [4], there is no rigorous model for how these factors influence basic behaviour. Nonetheless, using these parameters and storing the influence they have on parameter weights and state parameter modifiers results in relatively compact tables (5 by $l + m$ and 5 by m respectively). Individual influences could be applied and tested as needed.

3.2.6 Procedurally Generated Agents

A key motivation for this project is to develop a model for the procedural generation of background NPCs. Understandably one of the main bottlenecks for any system where game agents are procedurally generated would be the generation of large assets (character art, such as 3D rigged meshes with associated animations), an issue that has been addressed by several industry projects [61] [11]. This model is focused on behaviour selection as the means to convey believable behaviour, as such emphasis will be on generating meaningful variation in agent behaviour.

As detailed above, the model thus far generates variation in agent behaviour through the influence of a personality model on how states are weighed in importance and how the agent is affected by performing actions. Modelling personality parameters as floating-point numbers within the interval $[-1.0, +1.0]$, there could be h by 5 different agent configurations (where h is the amount of unique numbers you can represent as a floating point number within that range). To avoid minuscule differences, personality variables could be limited to 4 decimal places, giving $h = 2 \times (10^4 + 1) = 20002$ (+1 for the upper and lower bounds, i.e. the values +1.0 and -1.0). This would result in the total number of unique personalities being limited to $5 \times 20002 = 100010$. Although this number is large, it pales in comparison to the 1.8×10^{19} unique planets of No Man's Sky. In addition many of these agents would have little difference in personality unless they were at the extremes (i.e. +1.0 on the neuroticism scale).

To achieve a greater number of agent configurations in addition to some unpredictability in their behaviour, it becomes favourable to supplement character generation with pseudorandom variability. With a suitable PRNG supplementing the effect of personality on agent weights and action modifiers, agents could have some interesting variation in individual preferences. In the natural world, when measuring single phenomena over a large sample size, the measurements often approximate the normal distribution. This is

by the Central Limit Theorem, which states that the arithmetic mean of a large number of iterates of independent, random variables will approach the normal (Gaussian) distribution. For example, phenomena such as the height of individuals in a population will be normally distributed. This is due to the large assortment of factors that contribute to an individuals height (genes, developmental environment), each of which are randomly distributed in the population. Therefore for this model, modelling phenomena within the population (such as individual values and responses) as normal distributed offers a realistic and intuitive system for interesting unpredictability.

By supplementing their weights/modifiers with pseudorandom numbers drawn from a Gaussian distribution, two agents with almost identical personalities could display unique quirks, such as one that particularly values money, or another that is particularly exhausted by socialising. This would vastly increase the number of unique agents that could be generated with the personality-based model thus far discussed.

3.2.6.1 Level of Detail Systems

It would be useful to implement a level of detail system for storing/updating agent state when they are not in the vicinity of the player. This would increase believability as the agents could 'change with experience' [23], but would reduce the computational resources associated with simulating them at high fidelity. In addition it would be useful if the generation of agents from a low-detail state to high-detail was not an overly expensive procedure. This presents a challenge, as vast numbers of agents might need to be expanded should the player enter a city from a less populated area.

3.2.7 Agent-Environment Interaction

One of the important metrics for believable agents is 'awareness': "agents should show they perceive the world around them" [23]. For agents in games, this is most often through sensing and can be utilized for object-interaction, player awareness and socialising with other NPCs. Representing these actions with utility-based decision making in the action hierarchy, it would be necessary to either store them in a deactivated state so they can only be accessed in the right environmental context. For example, should a character walk into the vicinity of another character, they could both activate their

'Social' actions and evaluate them, in the same way one might decide whether to stop and talk to an acquaintance on the street. Similarly, a possible solution for spontaneous object interaction could be to store actions in objects, these could be added/removed to agents depending on their proximity. This would also allow agents to acquire new actions, such as an agent that changes occupation and acquires new associated actions.

3.3 Illustrating the Model

As highlighted by Fallon [70] and McNulty [72], open world-style games such as Skyrim focus on character classes in order to represent different background characters. These often result in different appearance (be it a grizzled leather-clad hunter or a finely dressed courtier) and behaviours (a thief that goes pickpocketing at night or a soldier that stands guard on a watchtower). Characters within a class rarely display much individuality, a blacksmith in one town will behave much the same as the blacksmith in another.

The reasons for this are understandable, each new class requires associated art assets (voice acting, animations) and repeating archetypes saves on development costs while still resembling the specialisation of labour that one might expect from varied characters. This model is intended to offer a system whereby characters with similar backgrounds, environment and profession could still show meaningful differences in behaviour through personality modelling and procedural generation.

This section is intended to demonstrate the intended model through considering potential characters it could generate. Two characters for a Skyrim-like village are developed, each with a profession and associated actions. Through their personality and experience, they display varied behaviour through the preferential selection of some actions over others. It is through differing behaviour patterns that we can imagine an internal world driving them.

3.3.1 The Neurotic Blacksmith

Consider a blacksmith in a small village, he might have a home and a workplace where he performs his trade. Some of his behaviour is universally human: he eats, sleeps and

perhaps does something to relax in his spare time. He works as a blacksmith in order to provide for himself.

Now imagine that this blacksmith is particularly neurotic. After he gets up in the morning he evaluates his options; **work**, **eat**, **sleep**, *recreate*. He is hungry, this is a consideration for the action 'eat', therefore when he evaluates that option its calculated utility is high. The **eat** action has two options; the child actions **eat at home** and **eat food at the market**. Eating at the market will entail some social activity, he weights this quite low; he would really rather not have to talk to anyone he doesn't have to. After eating a particularly unsatisfying meal, he evaluates his options again; go back to bed, eat again, recreate or go to work. When considering going to work, he considers the time. Its standard working hours. He rates this highly and immediately proceeds to work.

On the way to work he bumps into a neighbor, although the neighbor stops and seems to want a conversation, the blacksmith chooses to ignore her. The blacksmith is quite introverted in addition to being neurotic, this leads to a discount in the weighed importance of choosing to socialise. The blacksmith reaches work and thinks about how to proceed with the days work; whether to **forge new tools**, **sell wares at the market** or **sleep in the corner**. He's feeling tired but chooses to forge new tools for a few hours, he has plenty of tools already but wants to avoid the social activity of making sales at the market. In the process of doing so he adds to the growing pile of tools on his workbench. He then chooses to sleep in the corner of his workshop, barely able to keep his eyes open. A stranger (perhaps the player) approaches his workshop; the blacksmith wakes and tells him to go away, despite the fact that the smith has many goods to sell. The player persists and the smith reluctantly sells the stranger a hammer. After the days work he trundles off to the tavern, drinking quietly in the corner, hoping that no one spots him and tries to start a conversation. He checks the time, still feeling exhausted and hungry, he trundles off to his bed at home.

From the perspective of an observer, the blacksmiths schedule over the day is as follows:

- Wake up
  (**Sleep At Home** action ends)

- Eats at home
  (**Eat →eat at home**)
- Does not stop to socialise with neighbor
  (Rejected action: **Socialise**)
- Goes to work and forges tools
  (**Work →forge new tools**)
- Sleeps in the corner of his workshop
  (**Work →forge new tools**)
- Player approaches and is told to go away
  (Rejected action: **Socialise with player**)
- Player persists and smith sells him a hammer
  (**Socialise with player →trade with player**)
- Goes to tavern and drinks alone
  (**Recreate →drink at tavern →drink alone**)
- Goes home to sleep
  (**Sleep →sleep at home**)

Over time, the blacksmiths money would run out and they would have to go to market to sell wares periodically, lest they begin stealing food (high hunger and low money).

With another blacksmith that is extroverted we might see a different day's activity. Even with the same small set of actions we can consider how noticeable behavioral preferences could manifest themselves. An imagined day could be as follows:

- Wake up
  (**Sleep At Home** action ends)
- Eats at the market
  (**Eat →buy food at market**)
- Stops to talk to neighbor
  (**Socialise**)

- Comes across another acquaintance
  (**Socialise**)
- Goes to work, brings wares to the market
  (**Work →sell wares at market**)
- Player approaches and is greeted, smith sells him a hammer
  (**Socialise with player →trade with player**)
- Goes to tavern and drinks loudly with friends
  (**Recreate →drink at tavern →drink loudly with other patrons**)
- Goes home to sleep
  (**Sleep →sleep at home**)
- Is interrupted en route, stops to talk to friend
  (**Socialise**)
- Goes home to sleep (late)
  (**Sleep →sleep at home**)

This blacksmith could over time prioritise the manufacture of new tools, having sold all wares at the market. This character might be wealthier and therefore spend money frivolously. With higher fidelity actions this could be extended, for example with additional player-interaction related actions this agent may be more receptive to gossiping activity with the player, or assigning them quests (for a discussion of procedurally generated quests see work by Noonan [71]).

3.3.2 The Conscientious Woodcutter

Another archetype, the woodcutter, would have broadly similar options. They too would have a home and options for recreation. Perhaps woodcutting is less lucrative and therefore the woodcutter has a more modest home, and frequents a more run-down tavern than the blacksmith. Through the woodcutters behaviour over a single day, we can imagine the influences of their personality. Take a particularly conscientious woodcutter for example, one that values tradition and hard work:

- Wake up (early)
  (**sleep at home** action ends)
- Eats at home
  (**Eat →eat at home**)
- Stops to socialise with neighbor
  (**Socialise**)
- Does not stop to socialise with stranger
  (Rejected action: **Socialise**)
- Goes to work, chopping wood
  (**Work →chop wood**)
- Brings wood to market
  (**Work →sell wood at market**)
- Go to church
  (**Recreate →go to church**)
- Goes home to sleep
  (**Sleep →sleep at home**)

3.4 Model Architecture

A high-level diagram decribing the model's architecture is presented in Figure 3.7. The agent is generated through personality modelling with pseudorandom number generation to create weights and modifiers. The agent evaluates its action hierarchy, supplied with actions from the environment or designer's specification. These are evaluated on the basis of consideration utility multiplied by consideration weights. When an action is selected, it exerts an effect on the agent's state with an action-specific state modification vector that has been amended with personality specific state modifiers.

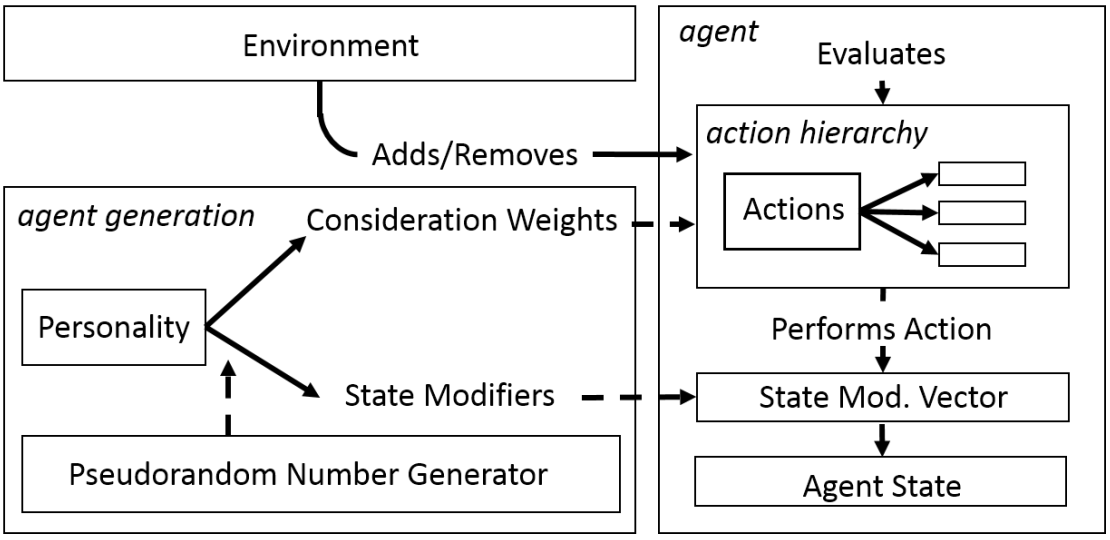

FIGURE 3.7: High Level Representation of the Proposed Model

3.5 Manual Creation of Actions

A system for designing actions would prioritize ease of use by designers. Actions could be specified with a location, associated animations and a duration. The location for actions could be supplied by the designer or possibly assigned randomly from a pool of similar locations generated for a town, such as an array of house locations. Duration of the action could be generated with a pseudorandom process, such as picking from a distribution centered around a mean action duration (for instance a sleeping time of 8 ± 2 hours). Generation of the state modification vector would be manually set and attached to the action for which personality-based variation would be generated and added.

3.6 Automatic Addition of Actions

This could be naturally extended into the idea of in-game addition of actions to agents. For this, template actions would need to get access to the receiving agent's information (such as their home or work locations). The action would need to specify its new position in the agent's action hierarchy, for instance a new occupation would need to be inserted as a child of the 'Work' action. Personality-based procedural modification on the state modification vector would need to be performed in-game, hence the PCG component would need to be able to run quickly in real time.

3.7 Conclusion

This chapter presented the details of the high level design for a background NPC model. With a combination of utility-based decision making and the personality-driven procedural generation of decision weights and state updates, the model was developed with the requirements of believable interactive agents in mind, along with the general requirements for game agents in open-world games. The concept of utility-based action selection was extended for typical behaviour of background NPCs and the method by which established personality dimensions could be turned into a framework for agent generation. Examples of generated characters that the model could accommodate were discussed, and the high level architecture of the model was outlined as a general implementation strategy.

Chapter 4

Implementation

This chapter presents the implementation of the proposed model design. It is intended to be an in-depth discussion of the design choices for the project, including choice of platform and technical considerations made.

4.1 Platform Review

For the implementation of the model a suitable development platform had to be chosen. From the conception of the project it was determined that the use of a game engine-style development platform would be prudent in order to focus on the implementation of background NPC artificial intelligence. Previous research by John Fallon [70] originally used the Skyrim Creation Kit a set of tools for custom content creation within the Skyrim game. Due to technical difficulties however, subsequent work on similar NPC models chose to focus on the Unity3D game engine [73] due to its good documentation and quick set-up [72],[5], [71].

For this reason it was decided early on that Unity would offer a suitable platform for implementing the model within an open-world game style environment. Unity primarily operates within an editor with drag-and-drop functionality, with the behaviour of components specified through scripting in the C# programming language. It similarly offers a significant amount of creative and technical control, with the ability to code at a low-level when needed.

4.1.1 Demo Environment

For the implementation of the model a 3D demonstration environment was created. For this a number of free art assets were obtained from the Unity asset store, the complete list of which are available in Appendix D.2. Environment terrain was built in the editor and 3D models for a village were positioned.

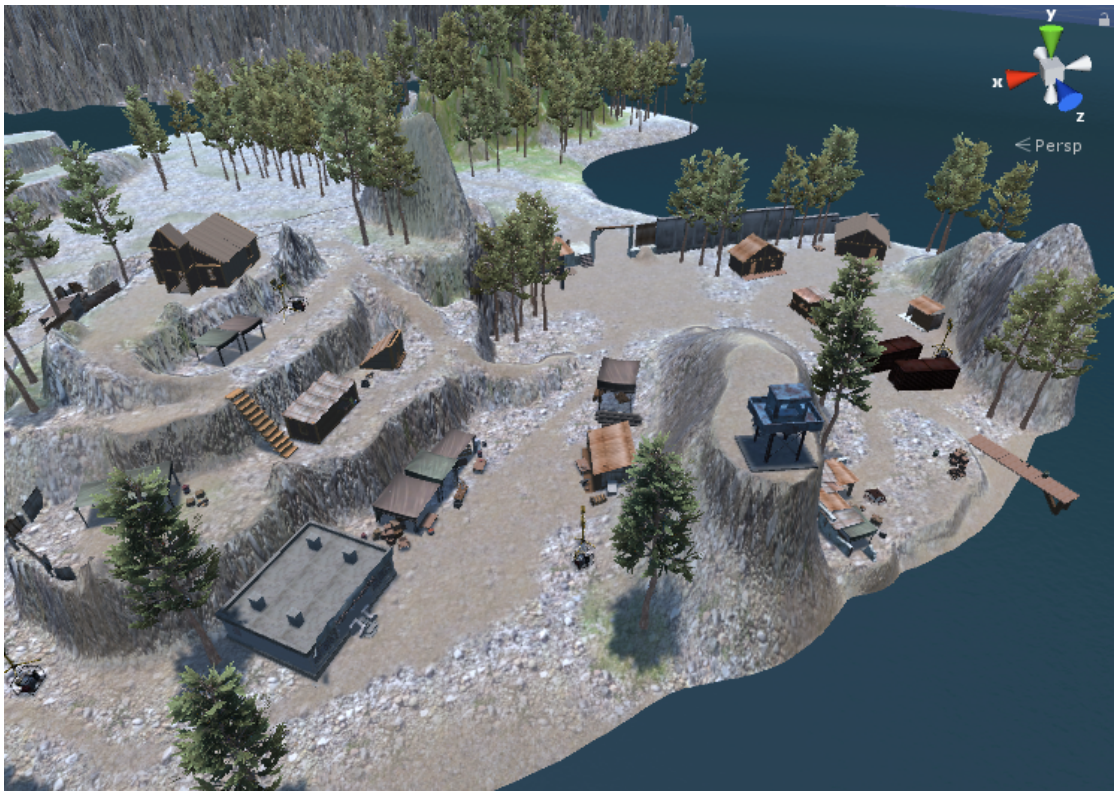

FIGURE 4.1: The Implemented Demo Environment

4.1.2 Agent Representation

Unity stores every object as a GameObject, from lights and 3D models to cameras and special effects. They represent empty base objects, and through the addition of components they can be controlled and interacted with via code. This lends unity to hierarchical organisation within the editor, the components necessary for each agent are no different. Each agent is represented with the following important objects/components:

- **Body**: A 3D rigged model with animations, a 'Speech Bubble' for identification
- **Camera**: Set behind and above the agent, for observation

- **Personality**: A class that stores the personality vector and performs procedural generation
- **Agent**: A class that updates the agent's action hierarchy and decision-making
- **Character**: A class that holds the agent's information and manages all components and behaviour

Agent state is represented with a selection of State Parameters, GameObjects containing a floating-point number representation of each state variable, bounded within the interval $[0.0, 100.0]$. The following state parameters are used to represent agent internal state, inspired by the models implemented by Fallon and Mullally: **Hunger**, **Energy**, **Wealth**, **Resources**, **Mood**, **Temper**, **Sociability** and **Soberness**.

Agent personality is specified with floating-point number representations of the Five-Factor Model dimensions (**Openness**, **Conscientiousness**, **Extroversion**, **Agreeableness**, **Neuroticism**), bounded within the interval $(-1.0, 1.0)$, with the requirement that each dimension is specified to three decimal places (the reasoning for this choice will be discussed in Section 4.4).

4.1.3 Agent Movement

For agent movement in the environment, unity's NavMesh was utilised. This is a singleton class that operates on a navigation mesh as baked automatically from level geometry. The NavMesh can be used to perform spatial queries such as pathfinding and walkability test, the setting of pathfinding costs and obstacle avoidance. In the demo the terrain was used as the basis of the navigation mesh, with sections of the grid located on roads set to a lower pathfinding cost to encourage smooth navigation.

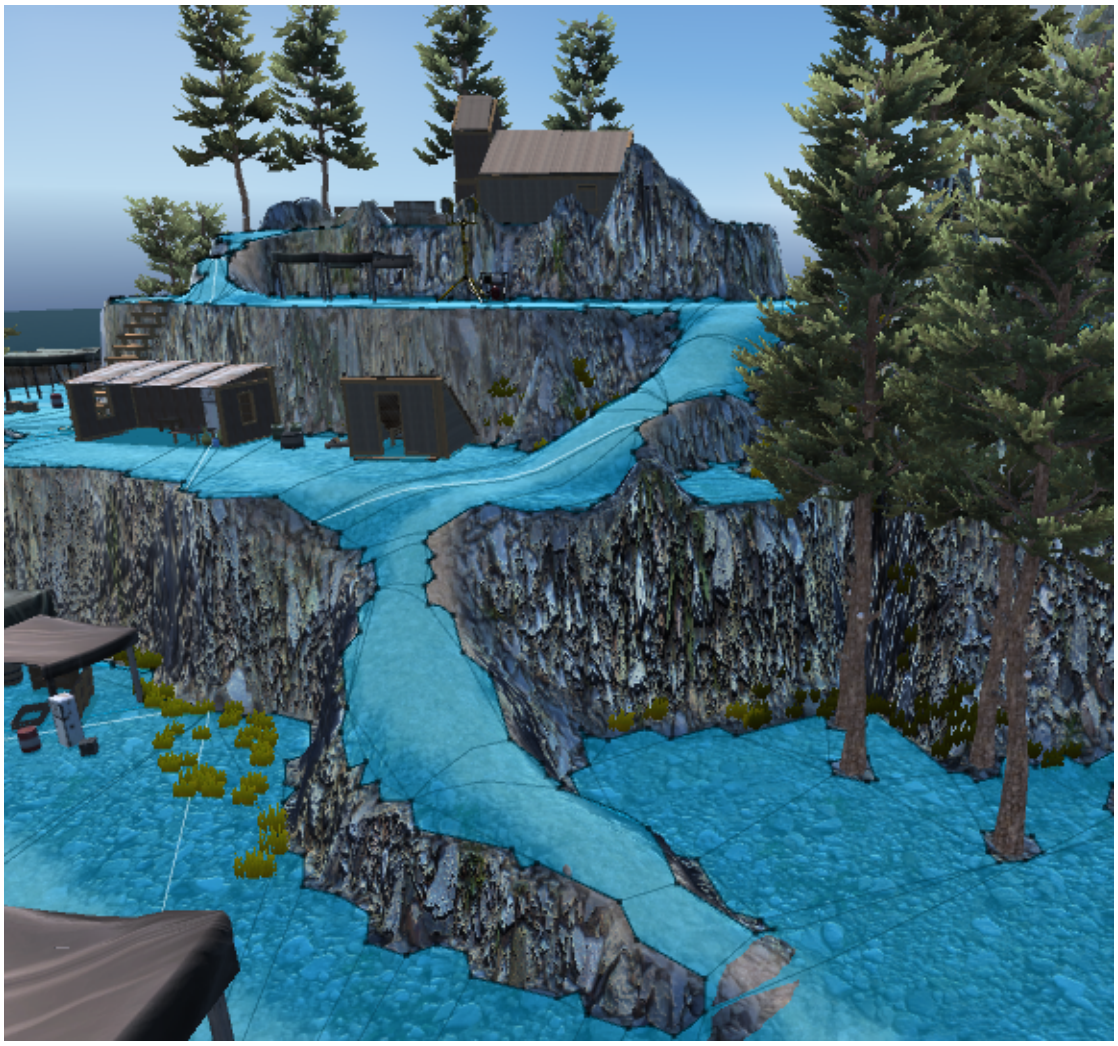

FIGURE 4.2: NavMesh (blue) Visualisation within the Demo Environment

Through the addition of the NavMeshAgent component to the agent, it is accessed by the **Character** component when an action has been selected. Every action has an associated location and once the action is selected the NavMeshAgent is told to navigate there while playing the 'Walk' animation. Once the agent reaches the destination the animation is turned off.

4.2 Utility-Based Decision Hierarchy

When an agent is activated, the `Update()` function of the Character component is run every frame, this sets the Agent component to update the action hierarchy. As mentioned, the **Agent** component is responsible for operating the decision-making of the agent. The component has access to the root node of the action hierarchy: the 'Exist' action. The

Exist action is the core of every agent's decision-making, it is always updating. Exist, and all other actions are examples of the **ActionBehaviour** class. ActionBehaviours are either the root action (i.e. the 'Exist' action), a Leaf action (an ActionBehaviour at the bottom of the hierarchy) or neither.

During a traversal down the hierarchy (i.e. Exist →Sleep →SleepAtHome), all actions are running. All actions have a duration, and this is used by the action's parent to set its own timer. Once a child action's duration has ended, the parent must pick another child action until *its own* parent stops updating it. So if the 'SleepAtHome' action ends, but the timer for the 'Sleep' action is still running in the 'Exist' action, 'Sleep' must evaluate, choose and run one of its child actions. All ActionBehaviours have an `UpdateAction()` function, this evaluates and picks from its linked child ActionBehaviours if none is currently running. Timers for currently running child actions are only activated once the agent is at that action's location (see Section 4.3).

As detailed, every action (unless it is a Leaf action) has a list of possible child ActionBehaviours to choose from. These are evaluated according to the algorithm sketched out in Section 3.2.3; namely by obtaining a utility score for each of them and picking the top action. The utility score for each child action is calculated by obtaining the mean of all **ActionConsiderations**; essentially prespecified utility functions that convert a single State Parameter into a normalised Utility Score.

ActionConsiderations have a single State Parameter, a utility function and a weight. The utility function is specified as a Unity Animation Curve, a component usually used for storing animation keyframes that can be evaluated as a function of time. By evaluating the Animation Curve with the normalised State Parameter, the utility at that point on the curve will be output. This allows for hand-crafting of specialized utility functions by tuning the curve as needed.

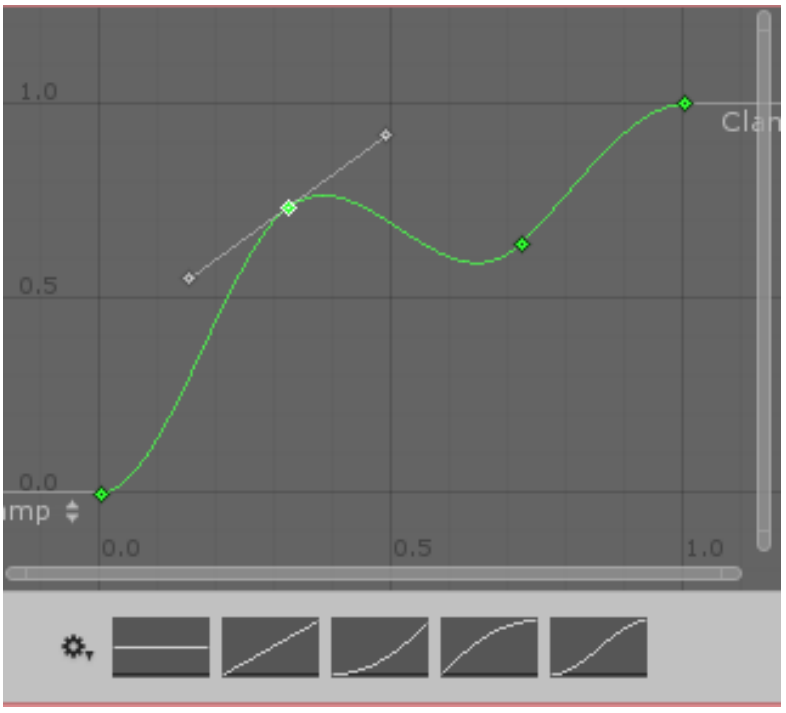

FIGURE 4.3: The AnimationCurve component in Unity

Actions have have an arbitrary number of child ActionBehaviours and ActionConsiderations and so a C List data structure is used to contain them, Lists having the ability to add elements during runtime. As many actions and considerations are supplied to every agent, a system was implemented for the automatic setting of an action's owner by traversing up the hierarchy until reaching the character GameObject (tagged as "Agent"). This is used to acquire agent-specific action locations such as the agent's home location. This system extends to set the owner for every consideration, allowing each consideration to set the relevant, agent-specific state parameter and the relevant consideration weight. This is implemented by using a hard-coded string representation of the consideration parameter (such as 'Hunger'), and accessing a hashtable (C# Dictionary) within the agent to access the agent's 'Hunger' parameter, and relevant weight.

This implementation allows for a drag-and-drop interface where generic actions can be added into an agent's action hierarchy, including during runtime, with automatic integration within the agent. This system is used during the procedural generation of agents, where actions are added according to agent details (Section 4.4). It could also be used for the addition of actions through the proposed SmartObject-inspired feature, such as an agent acquiring a hammer object along with associated actions (Section 3.2.7).

4.2.1 Interruption System

Whereas the addition of actions into the hierarchy allows for dynamic agent-environment interaction, a system for spontaneous actions is required for many scenarios. This is implemented through the **Interrupt** system. If an agent comes into contact with an object or individual containing an Interrupt, an interruption is flagged. This is achieved through an intersection check between bounding spheres associated with the Interrupt component. Interrupts are flagged and the agent pauses from their current action and evaluates the Interrupt. This could handle many possible interruptions (combat, assisting another agent) but to showcase the system a 'Socialise' interrupt action is implemented.

The Socialise action is a micro-hierarchy that exists outside the agent's normal action hierarchy, and is evaluated on the basis of two considerations: the agent's Sociability state parameter and the agent's Relationship with the recipient. Relationships are stored as State Parameters; if two agents haven't met, each will create a new Relationship tagged to the other agent. Currently relationships are represented by a single state parameter, but could easily be represented by several dimensions (i.e. Love or Respect) Relationships are non-mutual, that is agent stores a separate parameter. This allows for scenarios where agents may have vastly different opinions of each other.

When an agent Bob comes into contact with agent Alice, they both flag an Interrupt and evaluate the utility of the Socialise action (with the appropriate relationship as a consideration). If Bob determines that the utility of socialising is greater than the current chosen action utility, he will signal this to Alice. If she also accepts then they both set their current action to 'Socialise' and treat it as a normal action. They can each then choose between two child actions: 'Socialise Nice' and 'Socialise Mean'. If Bob chooses the 'Mean' option, this will be signalled to Alice which will deteriorate their opinion (i.e. the Relationship) of Bob. Although this is a one-dimensional view of relationships, it presents a system where social dynamics can arise over the course of a simulation. Similarly, the system whereby agents can manipulate the state of another agent presents several interesting possibilities, whereby agents could help/hinder each other. These are left as possible extensions of the model (further discussed in Section 6.2).

4.2.2 Implemented Actions & Hierarchy

A diagram of the implemented action hierarchy is shown below in Figure 4.4. Some actions are generic; the 'Work' action sub-hierarchy for instance can be set to a number of professions. A table of the considerations for each action is presented below in Table 4.1

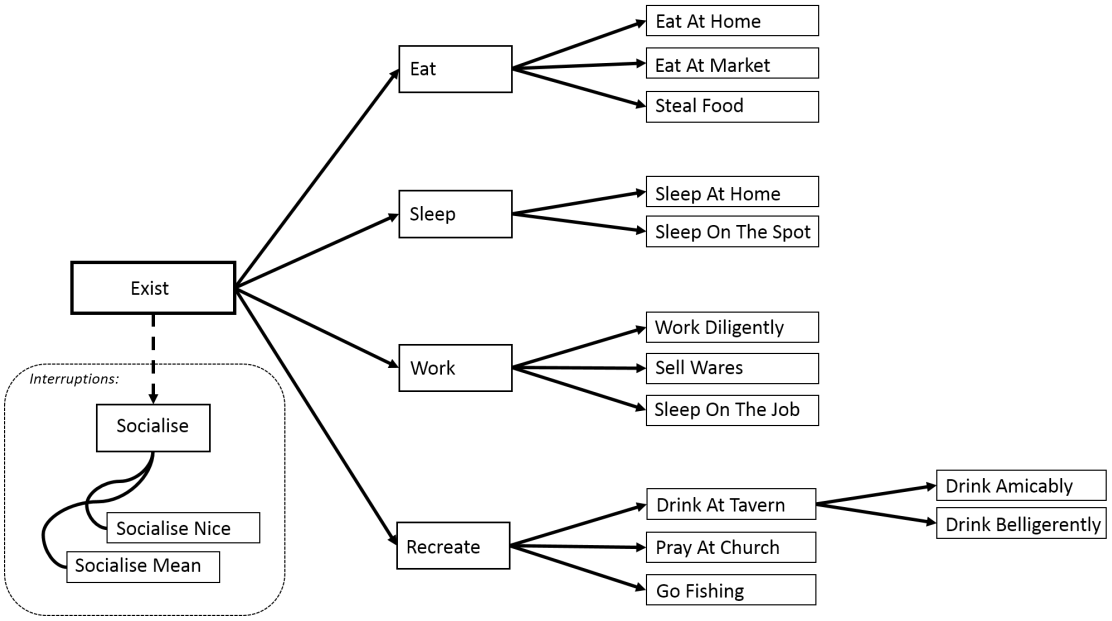

FIGURE 4.4: The Implemented Action Hierarchy

Table 4.1: Actions with Considerations & Locations

Action	**Considerations**	**Location**
Eat	Hunger	Self
EatAtHome	Wealth	Self.Home
EatAtMarket	Wealth	MarketLocation
StealFood	Wealth	MarketStealLocation
Sleep	Energy TimeOfDay	Self
SleepAtHome	Energy	Self.Home
SleepOnTheSpot	Energy Soberness	Self
Work	Wealth Energy	Self.WorkLocation
WorkDiligently	Energy	Self.WorkLocation
SellWares	Sociability Resources	Self.WorkMarketLocation
SleepOnTheJob	Energy Mood	Self.WorkLocation
Recreate	Energy Wealth TimeOfDay	Self
DrinkAtTavern	Sociability	TavernLocation
DrinkAmicably	Temper	TavernLocation
DrinkBelligerently	Temper	TavernLocation
PrayAtChurch	Sociability	ChurchLocation
GoFishing	Resources	FishingLocation
Socialise	Relationship Sociability	Self
SocialiseNice	Mood	Self
SocialiseMean	Mood	Self

4.3 Performing Actions

As has been mentioned, one component of performing an action consists of navigating to the action's location. Sometimes this destination is wherever the agent is currently standing (i.e. for the 'Sleep on the Spot' action), but agent much reach the location before the timer for the child action is started. For example the 'Sleep' action can be picked anywhere and 'Exist' will only run it for its duration, at which point the agent will decide on 'SleepAtHome' and navigate to its home.

An additional component to performing an action is the modelling of state changes with State Modification Vectors. When a leaf action is being executed (i.e. the agent is at the action's location), the action-specific State Modification Vector is added to the agent's state vector on a per-frame basis. The State Modification Vector can be scaled according to a specified frame rate. The execution of state modification is achieved using C# delegates; when an action is executed the delegate state modification function is run. Although originally it was envisaged that each action would have an associated animation, due to time limitations actions are limited to a location and state modification vector only.

State Modification Vectors used for the action in the demo environment are listed in Appendix A.1.

4.4 Procedural Generation

As outlined in the project design, procedural generation was to be used in order to increase individual agent detail. This was implemented primarily with behavioural detail in mind, through the generation of individual consideration weights and state modification vectors. For this, a combination of personality-based influences and pseudorandom Gaussian number generation was implemented.

4.4.1 Gaussian Pseudorandom Number Generation

The pseudorandom number generator used was adapted into C# from an implementation in the C language by Bourke [74]. The Gaussian PRNG operates via a ratio of uniforms

method by A.J. Kinderman augmented by J.F. Monahan with quadratic bounding curves [75]. This requires a uniform random number generator to supply random numbers within the interval $[0, 1]$, the algorithm for which originally appeared in work by George Marsaglia and Arif Zaman [76] and was later modifed by F. James [77]. The uniform PRNG is a combination of a Fibonacci sequence and an arithmetic sequence, has a period of 2^{144} and gives bit identical results on all computers with at least 24-bit mantissas in the representation of floating point numbers.

The generator needs to be initialised with two seeds, from which pseudorandom numbers can be generated in a deterministic fashion. The first seed must be an integer in the interval $[0, 31328]$ and the second must be an integer in the range $[0, 30081]$. These seeds are supplied from the agent's personality representation. Each personality value (based on the FFM) is represented as a decimal number between -1.0 and $+1.0$ with at least three significant digits after the decimal point, through string operations the first and second digits after the decimal point are concatenated to provide the seeds. If either seed is above 30,000, division operations bring it below that threshold. For an example of this process, please see Table 4.2.

Table 4.2: Seeds Generated for Different Personality Vectors

Open.	**Cons.**	**Extro.**	**Agree.**	**Neuro.**	**Seed 1**	**Seed 2**
0.345	0.8546	-0.535	0.3911	-0.999	17315	18097
-0.432	-0.991	0.999	0.001	0.349	20956	20226
-0.111	0.221	-0.001	-0.988	0.032	23202	20950

Once these seeds have been set for an agent and the PRNG initialisation function has been run, the PRNG can be used to generate numbers drawn from a Gaussian distribution with a requested mean and standard deviation. This can be used to generate pseudorandom numbers to supplement generated weights and action modifiers. As the process is deterministic given the same set of seeds, the PRNG function can be repeated to generate the identical arrays of weights and modifiers for agents with identical personality vectors. This allows for agent individuality to get collapsed/rebuilt as needed for simulation purposes.

4.4.2 Decision Weight Generation

The utility score of each action is calculated from the utility curve of each action consideration. To represent agent preferences these scores are multiplied by weights, procedurally generated from agent personality. The process by which these are generated is relatively straightforward; for a given State Parameter, the contribution of each personality dimension to its weighting is stored in a Weight Influences Table. Each personality dimension contribution is multiplied by the agent's value for that dimension. For example if the state parameter is 'Wealth' and the personality dimension is 'Conscientiousness', the influence value of conscientiousness (+3) is multiplied by the agent's conscientiousness (say 0.345) to give 1.035. Likewise, if the agent has a negative value for a given personality dimension that will result in a negative weight. The contributions of each personality dimension are then summed and added to a pseudorandom Gaussian number with a mean of 1.0 and a standard deviation of 0.3 to give the final weight.

Due to the lack of a rigorous model for mapping personality to preferences, in the current implementation the influences of personality are treated as independently influencing two specific state parameters. These are loosely inspired by the suggestions in [4], but could be reset as required by changing the values in the weight influence table (Table 4.3). This system allows for agents that are highly conscientious to weigh 'Wealth' as a highly significant consideration, whereas a highly non-conscientious agent would discount the importance of wealth in its decision-making.

Table 4.3: Personality Weight Influences

Personality Dimension	**+3.0**	**+3.0**
Openness	TimeOfDay	Relationship
Conscientiousness	Wealth	Resources
Extroversion	Energy	Sociability
Agreeableness	Temper	Soberness
Neuroticism	Hunger	Mood

4.4.3 State Modification Vector Generation

This system is extended to determine procedurally generated agent responses through the addition of individual variation from the standard State Modification Vectors shown in Appendix A.1. Due to personality contributions each agent will have unique responses to performing certain actions through the addition of State Parameter Modifiers. For example, an agent that is highly Neurotic will generally be less energised by sleep (an activity that normally increases energy) and more exhausted by working diligently (an activity that decrease energy).

These modifiers are generated through the Modifier Influence table (Table 4.4), supplemented with pseudorandom Gaussian numbers (mean of 0.0 and standard deviation of 0.5) and added to an agent's State Modification Vectors. Note that this table is similar to Table 4.3, however there are two important differences. Agreeableness has no influence on the State Parameter 'TimeOfDay' as this is external to the agent, and Neuroticism has a negative influence on Mood. Therefore we can expect that a highly neurotic agent will improve its mood slower than usual and worsen its mood faster than usual.

Table 4.4: Personality Modifier Influences

Personality Dimension	**+3.0**	**+3.0**
Openness	N/A	Relationship
Conscientiousness	Wealth	Resources
Extroversion	Energy	Sociability
Agreeableness	Temper	Soberness
Neuroticism	Hunger	Mood (-3.0)

Both Table 4.3 and Table 4.4 are implemented as 2D arrays, these are mostly empty (personality dimensions have no influence on state parameters other than the two specified) but can easily be altered with new values to achieve any particular personality influence desired.

4.4.4 Level of Detail System

In order to provide support for agent creation/generation during runtime, a preliminary level of detail system was implemented. This allows for the storage of agents in a compressed form; once the player enters a specific area (the borders of a settlement, for instance) the agents can be generated and simulated at full detail.

Within this system, agents are represented as **AgentKernels**. An AgentKernel represents the core of the agent. Namely:

- **Name:** the agent's name as a string
- **Home Location:** a vector representing the location of the agent's house
- **Occupation:** an enumerated type representing occupation
- **Personality Vector:** a specific float value for each of the 5 personality dimensions

In this implementation, the occupations are **Blacksmithing**, **Farming**, **Woodcutting**, **Guard Duty** and **Hunting**. Due to time constraints the main difference between these is location; individual animations could not be implemented.

AgentKernels can be specified through the graphical interface of the editor, and an arbitrary number of them can be supplied to the Town manager class. This has an associated radius, once the player object enters this radius and triggers a Sphere Collision, the agents are 'unzipped'. For each AgentKernel, the town instantiates an agent GameObject (consisting essentially a generic version of the agent representation listed in Section 4.1.2) but without any actions. Town sets the agent variables (name, personality) and attaches a generic version of the action hierarchy, subsequently activated to become specific to the agent. The relevant occupation action is set and the procedural generation of weights and modifiers is commenced. Finally, the Character component is activated and the agents action selection is initiated.

The entire process of agent generation/activation can be repeated as needed during a simulation, with identical weights/modifiers thanks to the PRNG. In the current AgentKernel representation of agents, agent state (including relationships) are not stored, neither are the agents updated in lower fidelity. These features could be added as needed, but

would necessarily require the AgentKernel to hold more information in addition for a model for low-fidelity state update.

4.5 User Interface & Debugging Tools

In order to assess the model implementation and provide visualisation of agent decision making, User Interface (UI) and debugging tools were developed. Initially this was through the addition of the public boolean variable *isDebugging*. This could be activated on a component-specific level and could be used to print out relevant details to the console. This system was supplemented with the development of a UI that allowed for a detailed glimpse at the internal world of each agent. To aid the UI, every time the agent choses an action the action is stored in an AgentLog, a list of chosen actions alongside relevant details: action start time, end time, utility score.

The UI consists of two main panels overlaying the viewport: an AgentSelection panel and an AgentInfo panel. The primary camera is attached to a first person controller, and has an associated button in the AgentSelection panel. Once an agent is unzipped and generated by the Town script, a button is added to the AgentSelection panel and is automatically linked to the agent. When this button is clicked, the main camera switches to the camera above and behind that agent. Three buttons in the AgentInfo panel then allow for visualisation of the agents decision making. These three buttons are:

- **Agent History:** a list representing the AgentLog: previously chosen leaf actions
- **Agent Action:** the current chosen leaf action shown through its position in the hierarchy
- **Agent State:** The current values for each of the agent's state parameters (shown alongside the generated weights), the agent's personality vector and the agent's current relationship values

With automatic addition of an agent button for every instantiated AgentKernel, it is possible to track the state and behaviour of any currently active agent.

FIGURE 4.5: The Implemented User-Interface for Assessing Agent State. For every agent running there is a button automatically created in the left hand panel. The right hand panel can be used for toggling between the selected agent's history, current state or current action

Chapter 5

Evaluation

This chapter aims to present an evaluation of the project. First the efficacy of the agent procedural generation is assessed by considering the results of the pseudorandom number generation and the resultant schedules of different personalities, this is followed by a discussion of the model with regards to the project aims. Lastly there is a brief summary of the issues encountered during development.

5.1 Generation of Believable Agents

One of the primary goals of this project was to implement a system for the generation of agents that display believable behaviour. This implementation decided to focus on personality, and whether procedural generation of agent parameters from a personality seed could manifest in noticeably different characters. In addition, this model sought for agents with similar but not identical personalities to display differences in their behaviour through the introduction of pseudorandom number generation. This section will explore both of these aspects with respect to the goals of the model.

5.1.1 Comparison of Similar Personalities

The two agents Fido and Mello have broadly similar personalities. Both are fairly open to experience, mildly conscientious, slightly introverted, quite disagreeable and mildly

neurotic. Bingo on the other hand has a thoroughly balanced personality, he sits somewhere near the center on each personality dimension. The personality vectors for all three agents are shown in Figure 5.1.

TABLE 5.1: Personality Vectors for Agents Fido, Mello and Bingo.

Agent	**Openness**	**Conscientious**	**Extrovert**	**Agreeable**	**Neurotic**
Fido	0.4567	0.3001	-0.1119	-0.6681	0.2654
Mello	0.4578	0.2981	-0.1121	-0.6701	0.2701
Bingo	0.0001	0.0001	0.0001	0.0001	0.0001

Note that although similar in specified personality, Fido and Mello do not have identical state modifiers (Table 5.2) nor consideration weights (Table 5.3). They are however on similar extremes when compared to Bingo, whose modifiers are closer to 0 (drawn from a Gaussian distribution with a mean of 0 and a standard deviation of 0.5) and weights are closer to 1.0 (Gaussian distribution with mean: 1.0, standard deviation 0.3). The differences are a result primarily of the influence of personality vector, however even slightly different personalities will lead to different seeds. These manifest as different weights and modifiers due to the pseudorandom number generation. A complete list of the influences of extreme personalities on weight and modifier generation are given in Appendix B.1.

TABLE 5.2: Generated State Modifiers

State Par.	Fido	Mello	Bingo
timeofday	N/A	N/A	N/A
relationship	1.836	1.194	0.184
wealth	0.880	1.267	-0.304
hunger	0.069	0.634	0.420
energy	0.943	-0.071	0.756
mood	-0.510	-0.859	-0.632
temper	-2.677	-1.937	-0.439
sociability	0.012	0.377	0.1742
soberness	-2.407	-1.235	0.226
resources	0.821	1.576	-0.018

Table 5.3: Generated Consideration Weights

State Par.	Fido	Mello	Bingo
timeofday	2.476	2.004	1.058
relationship	1.955	2.192	1.116
wealth	1.413	1.709	0.902
hunger	1.600	1.582	1.049
energy	0.701	1.130	1.205
mood	1.717	1.646	0.753
temper	-0.920	-1.012	1.607
sociability	0.369	1.532	1.378
soberness	-1.141	-1.384	0.791
resources	1.373	1.784	0.764

These differences manifest in their behaviour. Presented in Table 5.4 are the top actions over two simulated days with the percentage of time spent performing them. As can be seen the behaviours of Fido and Mello diverge from each other significantly. Top actions chosen by other agents with extreme personalities are available in Appendix C.1.

Although differences between Fido and Mello are stark, the system can be used to heighten or lessen the effect. A primary way to lessen differences would be to adjust the Gaussian number generation to have a smaller standard deviation, resulting in greater convergence of their behaviour. The second way would be to adjust the personality influences in Tables 4.4 and 4.3. For instance, the influences could be kept discrete (that each, with each dimension affecting only two state parameters) but simply scaled down. Either of these approaches would facilitate straightforward adjustment of differences in behaviour.

Table 5.4: Top Actions for Fido, Mello and Bingo over 2 in-game days

Action	Fido	Mello	Bingo
buy food at market	∼5.56%	∼1.49%	∼11.11%
eat food at home			
steal food			
sleep at home	∼11.11%		
sleep on the spot			
drink amicably			
drink belligerently			
pray at church		∼5.97%	∼22.22%
go fishing	∼22.22%		
sleep on the job	∼38.89%	∼8.96%	∼22.22%
work diligently		∼35.82%	∼22.22%
sell wares		∼25.37%	∼11.11%
socialise nice	∼27.78%	∼7.46%	∼11.11%
socialise mean			

This system also supports flexibility for agent specification. Should designers of particular agents want the personality vector to have a more deterministic effect on behaviour, they could merely adjust the standard deviation of the pseudorandom number generator. This could be adjusted globally or on a per-agent basis. For large worlds populated by hundreds of agents to be expanded and simulated on the basis of player proximity it may be preferable to allow for variation so that unique characters can be created automatically.

5.1.2 Believable Behaviour

The implemented framework presents a generic way to implement background agent decision making. Though there are few quantitative models for believability we can still make a comparison to some common metrics (such as those outlined in Section 2.1.3) and to approaches to non-player character AI made by the industry. First we can examine the actions chosen by one of the archetypes imagined in Section 3.3.2: the conscientious woodcutter. To test this an AgentKernel is specified as follows:

Table 5.5: Mundo, a conscientious woodcutter

Name	Mundo
Occupation	Woodcutting
Home Location	House2
Openness	0.0005
Conscientiousness	0.6542
Extroversion	0.1121
Agreeableness	-0.0031
Neuroticism	0.0027

Mundo's personality is specified as very conscientious, mildly extroverted and otherwise quite balanced. If we follow him for a day we can observe him performing the following actions:

Table 5.6: Mundo's Day

Eat →BuyFoodAtMarket
Work →SellWares
Socialise →SocialiseNice
Work →SellWares
SocialiseNice
Work →SellWares
Socialise →SocialiseNice
Work →SellWares
Recreate →GoFishing
Recreate →PrayAtChurch
Socialise →SocialiseNice

We see that his first action is to find something to eat. As his extroversion lends an extra weight to social actions, he chooses to buy food at the market. He then goes to work, choosing to sell his wares. Work has a wealth consideration which he weighs highly, and selling wares has sociability as a consideration. He then proceeds with a highly social working day getting periodically interrupted from his work to chat to passers by. In the evening he goes fishing, this action is recreational but is a means to acquire

resources (positive resource modifier on the state modification vector) and hence has resources (strongly weighted due to his conscientious nature) as a consideration. He then proceeds to go to church, where he is interrupted from his praying by socialising with an acquaintance.

Whereas there is no immediate way to assess whether this leads to believable behaviour as a manifestation of his personality as specified, it could be argued that his behaviour is roughly what might be expected of a hard-working but sociable woodcutter. He displays several of the metrics outlined in Section 2.1.3, awareness and social relationships (by socialising with passers by), change with experience (his hunger is reduced by eating in the morning, for instance) and personality (he consistently socialises nicely with other characters). He shows a unique schedule when compared to other agents such as Fido, Mello and Bingo (see Section 5.1.1) or the top actions of extreme personalities (see Appendix C.1).

This can be compared to approaches used in other open-world style games populated with background NPCs. In Skyrim, non-player characters are driven by predefined scripting for their daily activities in addition to the Radiant A.I. system that handles interaction with the player and other NPCs [72]. Whereas the scripting approach offers stability, it would be hard for the character to change with experience. If the player suddenly stole all the money from the NPC, the daily behaviour of the agent wouldn't be affected. The model presented in this thesis however could allow for this. The player could have ample opportunity to affect the agent by creating some temporary or permanent effect on the agent's state representation; should the player cast a laziness spell on the NPC, we would see the agent choosing actions where low energy led to high utility, such as sleeping. If the player stole from the agent, we may see the agent working hard to compensate for their loss or desperately stealing from the market to feed themselves.

In order to allow for large numbers of NPCs (up to 30,000 on-screen) the developers of Assassin's Creed Unity chose a slightly different approach to that in Skyrim [9]. They chose to implement a level-of-detail system that simulates agents at a distance with lower quality animation and meshes; agents within 12 meters of the player were relatively high fidelity and would react to the players actions, those between 12 and 40 meters away were less complex and those more than 40 meters away were animated in low resolution with only simple navigation behaviours. This approach reduced the cost of

agent AI, but believability was left as a responsibility of their animation content rather than particularly complex decision making, which mostly relied on bulk states to define the behaviour of large groups. Arguably this approach suits different needs; the Assassin's Creed NPC system sought to render large crowds with little need for the expression of individuality as presented in this model. That said, the developers approach to expressing NPCs through animation and LOD simulation offers much that could be explored as an extension to this utility-based approach which presently is limited to action selection.

5.2 Efficiency of Implementation

One of the key model aims was to achieve an efficient way to procedurally generate and simulate background NPCs. Both of these aspects, agent generation and simulation, are required to have low memory and processing requirements. This section aims to evaluate the performance with regards to these metrics. First the procedural generation of agents from their compressed form will be assessed, followed by the memory and CPU requirements for simulating them in full detail. Tests were conducted using Unity's built-in profiler for the analysis of CPU and memory performance.

5.2.1 Procedural Content Generation

The first component to be assessed is the performance of the agent procedural generation. As mentioned in Section 4.4, once the player enters the radius of the town the agents are generated from their compressed AgentKernel representation. The initial approach was to expand all agents in a single function call. Due to the number of operations required for the generation of a single agent (instantiation and linking of agent and action hierarchy objects, pseudorandom number generation), this led to heavy usage of the CPU. The results of agent generation through this approach are presented in Table 5.7. At 32 agents, generation is executed over two frames in succession.

Table 5.7: Average CPU Usage of Generating Agents
*Average value over two frames of execution

No. of Agents	Time	Percentage of CPU
1	12.3 ms	68.40%
2	15.3 ms	79.90%
4	18.3 ms	79.30%
8	24.5 ms	88.50%
16	86.1 ms	98%
32*	428.13 ms	88.35%

Due to the heavy CPU usage, such an approach would necessitate a pause in rendering in order to avoid the dropping of frames. This could be implemented through the use of a pre-baked loading screen or animation during agent generation, however a different approach was implemented through the use of coroutines in Unity. Coroutines effectively operate as a background thread, allowing user-defined conditions for a function to continue execution. This is most easily applied in the execution of a loop, where the function can be paused for a frame at the end of every iteration. This quick optimisation allows for every AgentKernel in the town to be generated one at a time on a per-frame basis. The results of this approach are presented in Table 5.8.

Table 5.8: Average CPU Usage of Generating Agent, Using Per Frame Coroutine

No. of Agents	Time	Percentage of CPU
1	0.1 ms	1.1%
2	5.36 ms	31.55%
4	8.37 ms	43.07%
8	10.69 ms	51.03%
16	10. 19 ms	52%
32	11.14 ms	48.85%

This approach offers a benefit in that it is capable of generating large numbers of agents with less performance costs than the former method. As can be seen in Table 5.7,

generation of 16+ agents requires more than 100% of the CPU capacity and results in drops in frame rate.

The per-frame approach could still be optimized via several methods. Several subfunctions called during agent generation require expensive operations, including the instantiation of objects and string concatenation for seed generation. Therefore it is highly possible that replacing these methods with a more efficient implementation could reduce CPU consumption. Additionally the current implementation generates a single agent per frame out of the entire town population as specified by the number of AgentKernels upon a flagged intersection test with the player. Whereas for smaller towns this does not present any issue, for a larger city of characters this would result in agent generation taking upwards of n frames to complete (where n is the number of agent kernels). In order to avoid this it could be preferable to contain large populated areas in a bounding volume hierarchy so that area populations could be generated on demand.

5.2.2 Memory Requirements

Unity's memory profiler allows for the analysis of the memory allocated from a managed heap to the contents of the game scene, including scripts being run. It does not offer a breakdown of the memory allocation, nonetheless by comparing the total system memory usage of the environment with agents running to that with no agents the cost of running individual agents can be ascertained. To test the memory cost of agent simulation, the average memory cost of the first 15 frames after agent creation were measured. The results are presented in Table 5.9 and plotted in Figure 5.9. The tests were performed on a compiled build of the project in order to gain the most accurate evaluation.

The memory cost is per-agent and therefore includes information pertaining to all associated objects including the action hierarchy, animated mesh model and UI overhead. As can be seen, the memory demand is approximately linearly proportional to the number of agents. Although there is little information on current memory budgets for game agents, McNulty extrapolates from older estimates and states that today's budget could be close to 128MB of memory, though games that focus on character AI such as Skyrim may be even higher [72]. Assuming a hard limit of 128MB, this model could therefore be used to simulate up to 198 agents at full detail. In addition, this budget could be increased via the removal of non-essential debugging tools such as the agent behaviour log and UI.

Table 5.9: System Memory Usage Per Agent

No. of Agents	System Memory Usage
1	3.06 MB
2	4.5 MB
4	5.69 MB
8	7.25 MB
16	13.13 MB

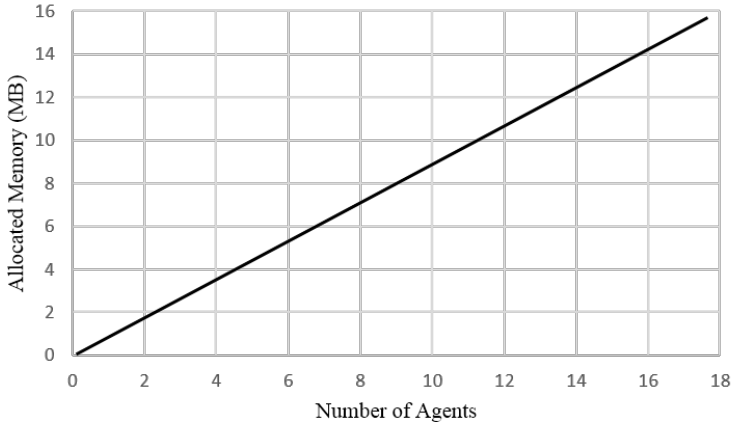

Figure 5.1: Memory Requirements Per Agent

5.2.3 CPU Requirements

Unity's profiler offers greater analysis opportunity for CPU performance by displaying the time and percentage consumed by every function call in the game scene. Though generating an agent requires several function calls, the majority of agent simulation is spent updating the decision process. By analysing the computational demands of this function the general performance of simulating agents can be assessed. The average CPU use over 15 frames after agent creation was measured for a number of different agents and the results are displayed in Table 5.10 and plotted in Figure 5.2. It is important to note that the bulk of CPU usage is taken up through navigation through pathfinding by the NavMeshAgent and controlling animation. The decision update itself is a small component of the values presented, though other aspects to agent simulation such as interruption processing are not contained in the decision process and therefore are not included as part of the estimate. Similarly, on occasions where all agents have to choose

a new action there can be spikes in computational load. In reality, due to the spacing out of actions it is highly rare to observe significant spikes above 10% of the mean values presented.

TABLE 5.10: Average CPU Usage of Agent Update

No. of Agents	Time	Percentage of CPU
1	0.054 ms	0.51%
2	0.073 ms	0.88%
4	0.126 ms	1.28%
8	0.236 ms	2.35%
16	0.409 ms	3.27%
32	0.788 ms	5.64%

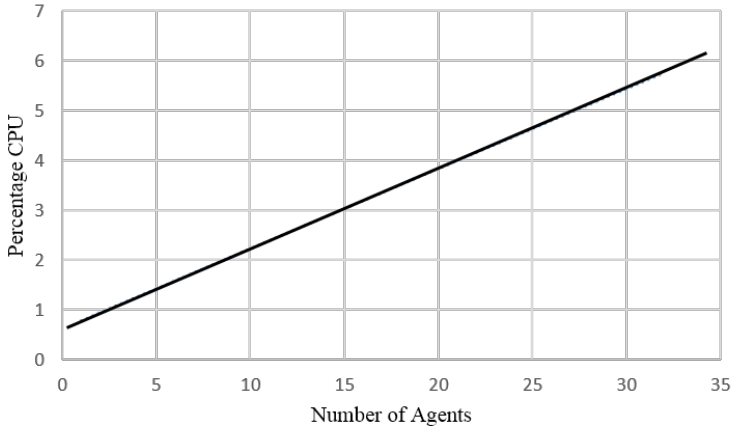

FIGURE 5.2: CPU Requirements Per Agent

A common estimate of CPU allowance for game agents is between 5 and 50% [17]. By this account the model could be comfortably used to simulate up to 32 agents at a cost of 6%. This could be extended to the larger budget associated with open-world role-playing games where additional agents could be run at the same time. This could be supplemented with level-of-detail approaches to agent simulations, where agents in nearby vicinity could be simulated in full detail and a model could be developed to simulate further agents with less behavioural fidelity. An alternative approach to reducing the cost of each update would be to implement a coroutine system (as in Section 5.2.1) to allow for action selection to be executed over several frames.

5.3 Impediments

The following presents a brief summary of the more significant issues encountered during development. Many of these stemmed from issues related to tuning and parameter selection of numerical values that could still be optimised:

- **Action Modification Vectors**: these contain a set of hardcoded representing the average effect (i.e. pre-personality based procedural content generation) performing particular actions has on the agents. These required some debugging initially to avoid overly strong effects on agent state.
- **Action Modifier Generation**: representing the effect that personality has on action modification vectors. Agents were most balanced when these effects were low and resulted in extreme behaviours when they were high.
- **Weight Generation**: Similarly, the effect that personality has on consideration weights resulted in behaviour that was either indistinguishable at low values or overly dramatic and repetitive at high values.
- **Pseudorandom Gaussian Number Generation**: These represented a level of non-determinism for agent personalities. The standard deviation of the pseudo-random number distribution had strong effects on this non-determinism, where at high values the personality effect was insignificant in comparison to the pseudorandomness.

These all presented challenges in the form of tuning the parameters to achieve desirable results, however further work on this would be required to improve the model.

An additional issue presented itself when attempting to model personalities using the Five-Factor Model. As few models exist for mapping these personality types to everyday behavioural preferences as required for background NPCs, it was not possible to rigorously examine the model for accuracy. Specifically, it is highly unlikely that each personality dimension would have an isolated effect on individual preferences and values. These are likely to contribute to behaviour in a synergistic manner, though initial attempts to model this resulted in a model that was too hard to achieve predictable

results with. A far more accurate model would indeed represent the effect of each personality value (openness, conscientiousness etc) on values (state parameter weights) and responses (modification vectors).

Chapter 6

Conclusion

This chapter provides a summary of the contributions made by this project, examines possibilities for future work that could improve or extend the model followed by some final remarks in the context of a wider perspective.

6.1 Main Contributions

Although the current model implemented offers several avenues for further improvement, it is possible to consider its main contributions. The project model as implemented offers a system for the background NPC behaviour that addresses common criteria for believability. The model was specifically designed with the criteria presented in Section 2.1.3, specifically those metrics proposed by Gomes [23]. These include agents changing with experience as they make new relationships, awareness (including social awareness) as they react to passers-by using the interruption system. Agent behaviour is relatively understandable; they get tired after a days work for instance. Through the utility-based action selection they display some degree of predictability as they choose actions to address their current state. The opportunity to express emotional state is available thanks to the action hierarchy; they can choose between drinking in a friendly or aggressive manner or between socialising in a nice or unpleasant way.

Similarly they display personality through their behaviour. Through the procedural generation of consideration weights and state modifiers, agents show specific preferences automatically from a broadly universal set of actions. These are largely predictable from

the agent personality specification, and the level of predictability can be adjusted through pseudorandom number generation that adds noise to achieve unique personalities, as presented in Section 5.1.1.

The model was intended to be as general as possible, with model components (actions, considerations and state parameters) following a generic structure that can be easily added or removed to suit requirements of any game agent implementation. Parameters for generated values such as the influence tables (Tables 4.3 and 4.4), state modification vectors (Section A.1) and pseudorandom number generation mean and standard deviation (Section 4.4.1), can be adjusted as needs be. Personality specification results in readily generated agent parameters according to a deterministic process; allowing agents to be collapsed and re-generated with no loss of personal behaviour quirks.

An additional focus was to achieve an efficient implementation. As discussed in Section 5.2.1, the procedural generation of agents can be run in real-time, albeit at an increased performance cost above the current CPU budget for AI in industry games. Effort was made to reduce the cost of this process, and several directions for further optimisation were discussed. In Sections 5.2.2 and 5.2.3 the memory and CPU requirements for running the agents were found to be acceptable by industry standards, though additional work could offer improvements.

6.2 Future Work

Upon consideration of the project as a whole, there are several directions for improvements of existing features or for further extensions. These include specific adjustments to suit different applications and general extensions to the model, including possible optimizations in the current code base.

6.2.1 Agent Specification

It would be possible to replace or extend the FFM personality specification to achieve more specific results. Although the FFM was developed as a result of the most commonly used personality descriptors, some designers may find it unintuitive to work with. The Big Five descriptors were originally chosen as a compact representation of personality,

for a given implementation it may be desirable to provide more specific descriptors. One of these could be the alignment system used for specifying characters in the role-playing tabletop game Dungeons and Dragons [78], where personalities are specified on a moral axis (good-neutral-evil) and an ethical axis (lawful-neutral-chaotic). The model could be extended to include these additional features to suit the needs of a given application.

Similarly, the current implementation requires designers to hand-specify agent personalities. Whilst this may save on extensive hand-authoring, it might become desirable for a more automated system. This could involve random specification of personality variables, possibly based on a model for their distributions within a population. With such a system it could be possible to specify that a town should have a population of 100 with a large number of conscientious characters, with automatic generation of personalities.

6.2.2 Animations and Speech

Currently performing an action entails going to a location and activating a state modification vector. Whilst the agent is animated while walking, the agent mesh does not perform additional action-dependant animations. Due to time constraints this could not be addressed, however a suggested approach would be the implementation of either an animation hash table structure, where actions have an associated key that runs an animation. Alternatively animations could be stored in a generic manner within actions, so that if a new action is added during run-time the agent could run novel animations when performing that action. Both of these approaches could be improved with the addition of a speech component (either by running sound bytes or loading text into the speech bubble) as a means to convey the agent's internal state.

6.2.3 Probabilistic Action Selection and Interruptions

Agents always choose the highest scoring action, this means that given a specific state vector the agents behaviour will be fully deterministic. There are tow immediate ways to address this: one would be to implement a probabilistic model for picking actions, another would be to extend the interruption system. The former could implement dual-utility reasoning as presented by Kevin Dill [59]. In his scheme the probability of actions being selected are proportional to the score of the action, with higher scoring actions

gaining a higher likelihood of being picked. This would add some additional cost to action selection but would allow for less deterministic behaviour. The latter approach could extend the interruption system to allow for smart objects/agents to interrupt the agent. Currently if an agent is on their way to perform an action, they can get interrupted if they choose to socialise with another agent. By extending this to allow for interactable objects such as tools or pets more spontaneous behaviour could be achieved.

6.2.4 Goal-Oriented Actions

Although this model concentrated on believable action selection, how the actions are performed is relatively simple. An interesting extension would be to integrate with a goal-oriented action selection system such as that implemented by Tiarnan McNulty in his memory-driven model [72], where scheduled actions had to be solved by an agent accessing a memory graph and choosing appropriate actions. This system could allow for utility-based action selection to achieve goals, where agents solved goals according to their personality. For example, a particularly introverted agent could choose to steal a hammer instead of having to ask other agents.

6.3 Perspective

As open world games become larger and employ more automation in their design, it is imperative that they maintain the quality of hand-authored content. This opinion is repeated by Warren Spector, creator of the popular immersive simulation game series Deus Ex, as he stated that he'd "rather do something that's an inch wide and a mile deep than something that's a mile wide and an inch deep" [79]. Arguably out of all the directions for interactive entertainment, the development of interesting game agents is an area with the greatest potential. The presented model is intended to add to the discussion, offering promise to the possibility of procedurally generated worlds.

Appendix A

A.1 State Modification Vectors

TABLE A.1: State Modification Vectors for leaf actions in the demo environment

State Parameter →	Hunger	Energy	Wealth	Mood	Temper	Sociability	Soberness	Resources
Agent Action ↓								
buy food at market	-4	-1	-3	0	1	-1	0	0
eat food at home	-4	-1	-2	0	1	3	0	-2
steal food	-2	-2	0	-1	-1	0	0	1
sleep at home	1	3	0	1	3	2	4	0
sleep on the spot	1	1	0	-1	1	0	4	0
drink amicably	2	-2	-2	1	0	-2	-4	0
drink belligerently	2	-2	-2	-1	2	-2	-4	0
pray at church	2	-1	0	1	3	-1	0	0
go fishing	2	-1	0	1	2	1	0	3
sleep on the job	1	2	0	1	1	0	2	0
work diligently	3	-3	0	-1	0	2	0	4
sell wares	3	-3	4	0	0	-2	0	-3
socialise nice	2	-1	0	-1	0	-2	0	0

Appendix B

B.1 Generated Weights and Action Modification Vectors

TABLE B.1: Weights Generated for Personality Extremes. Balanced (+0.0001 for all personality dimensions) shown for comparison. Values in Bold correspond highly to the personality dimension (i.e. Openness contributes to 'timeofday' and 'relationship' parameters)

Personality	timeofday	relationship	wealth	hunger	energy	mood	temper	sociability	soberness	resources
High Open.	**4.274246**	**4.348126**	0.5864747	1.254535	0.8230042	0.149406	1.078944	4.348126	0.917039	0.5808272
Low Open.	**-2.127605**	**-2.032512**	1.446642	0.4435185	0.2330744	1.38766	1.147811	0.764088	1.189896	1.200465
High Consc.	1.780697	0.7027894	**3.860236**	0.8893774	1.413402	1.001562	0.5428305	0.4865735	1.415092	**3.85503**
Low Consc.	1.146125	0.9975657	**-2.067918**	0.6688548	0.7611411	1.526049	0.9930927	0.6611223	0.8480416	**-2.123158**
High Extro.	0.9929922	1.176021	0.8829618	1.857234	**3.4302**	0.7816705	0.7756071	**3.892439**	0.5765208	1.561704
Low Extro.	1.30498	0.0854138	0.3806954	1.093314	**-1.851864**	1.441583	1.278199	**-1.961203**	1.011207	0.9182175
High Agree.	1.068957	1.05844	0.6029491	1.335177	1.044319	0.4939741	**3.628189**	1.115748	**3.834869**	1.180529
Low Agree.	1.305971	1.462326	1.193942	0.4944355	1.553634	1.054598	**-2.242844**	0.8735078	**-2.212897**	0.8881808
High Neuro.	0.1031219	0.408812	0.8611405	**4.124572**	0.888689	**4.010927**	1.089951	1.405353	1.652561	0.3343816
Low Neuro.	1.440323	1.137836	0.9377076	**-2.130183**	1.670937	**-1.895141**	0.925671	0.7208557	0.8682309	1.151603
Balanced	1.058381	1.116753	0.902178	1.049284	1.205032	0.7531227	1.607918	1.378694	0.7916266	0.7641723

TABLE B.2: Action Modifiers Generated for Personality Extremes. Balanced (+0.0001 for all personality dimensions) shown for comparison. Values in Bold correspond highly to the personality dimension (i.e. Openness contributes to 'timeofday' and 'relationship' parameters). 'timeofday' has no modifier as it is a global state parameter

Personality	timeofday	relationship	wealth	hunger	energy	mood	temper	sociability	soberness	resources
High Open.	**NA**	**3.398438**	-0.4917269	-1.184577	0.6891901	-0.7100334	0.7269089	0.1806427	-0.0815509	-0.2215635
Low Open.	**NA**	**-3.395821**	-0.1859288	0.2306053	0.1748247	-0.4316881	-0.2348692	0.3786029	0.4118736	-0.7463762
High Consc.	NA	0.4592186	**2.738993**	-0.0136223	0.3199452	0.1337161	-0.1963486	0.3148609	-0.3514865	**2.901304**
Low Consc.	NA	-0.3891387	**-3.355897**	0.2107405	-0.9298503	-0.1367756	-0.3685372	-0.5931904	-0.0759725	**-2.465957**
High Extro.	NA	0.6972977	0.0258557	0.4076903	**3.019837**	-0.453944	-0.5788076	**3.885782**	1.452414	-0.5363991
Low Extro.	NA	0.8747627	-0.301857	-0.1478227	**-2.624751**	-0.1229291	0.2096263	**-2.793609**	-0.5875171	-0.4272076
High Agree.	NA	0.0877169	0.6509014	-0.6827785	0.0671234	0.0076591	**2.54048**	-0.5792258	**2.347613**	-0.3477879
Low Agree.	NA	0.4904407	0.2949674	-0.2186747	0.7884422	0.4210389	**-3.459588**	-0.0113067	**-2.744385**	-0.5169434
High Neuro.	NA	-0.736424	0.4738002	**3.062581**	0.1380568	**-2.713147**	-0.3157212	0.0699023	0.3303235	0.3394741
Low Neuro.	NA	0.0873108	0.2656144	**-3.405974**	0.5457014	**2.315638**	-0.867269	-0.0896144	-0.2699846	0.0176239
Balanced	NA	0.1848748	-0.3049066	0.4202359	0.7566903	-0.6323114	-0.4390486	0.1742762	0.2266459	-0.018325

Appendix C

C.1 Top Actions for Extreme Personalities

The top actions chosen by agents with personality extremes over 2 in-game days are shown below in table C.1. These extreme personalities are signified by an extreme value (i.e. $+0.999$ for High, -0.999 for Low) on a single personality dimension (openness, conscientiousness etc). For each extreme personality only a single dimension was made extreme, all others were balanced at a neutral value (0.0001).

TABLE C.1: Actions Performed by Agents with Personality Extremes, Over 2 In-Game Days. Personalities with extreme values on a single parameter: +0.999 for High, −0.999 for Low, 0.0001 for all other personality parameters

Personality	Actions	Percentage
High Openness	Buy Food At Market	9.09%
	Sleep At Home	20%
	Pray At Church	45.45%
	Work Diligently	9.09%
	Socialise Nice	18.18%
Low Openness	Buy Food At Market	25%
	Pray At Church	25%
	Sleep On The Job	25%
	Sell Wares	25%
High Conscientious	Pray At Church	6.25%
	Go Fishing	12.5%
	Work Diligently	31.25%
	Sell Wares	43.75%
	Socialise Nice	6.25%
Low Conscientious	Buy Food At Market	25%
	Eat Food At Home	5%
	Sleep At Home	25%
	Sleep On The Spot	25%
	Socialise Nice	20%
High Extroversion	Buy Food At Market	7.14%
	Pray At Church	21.43%
	Work Diligently	57.14%
	Socialise Nice	14.29%
Low Extroversion	Buy Food At Market	50%
	Steal Food	50%
High Agreeableness	Buy Food At Market	6.67%
	Pray At Church	26.67%
	Work Diligently	33.33%
	Sell Wares	13.33%
	Socialise Nice	20%
Low Agreeableness	Pray At Church	22.73%
	Sleep On The Job	31.82%
	Work Diligently	36.36%
	Socialise Nice	9.09%
High Neuroticism	Buy Food At Market	47.62%
	Steal Food	47.62%
	Socialise Nice	4.76%
Low Neuroticism	Sleep At Home	12.5%
	Pray At Church	43.75%
	Work Diligently	25%
	Sell Wares	6.25%
	Socialise Nice	12.5%

Appendix D

D.1 Reference Materials

A disc with all of the files used or developed for the project are attached to the back of the dissertation.

D.2 External Assets & Plug-ins

- *Relaxed Man Character*, available at https://www.assetstore.unity3d.com/en/#!/content/32665
- *Concrete Blocks (Pack)*, available at https://www.assetstore.unity3d.com/en/#!/content/25961
- *Old Soviet Shop*, available at https://www.assetstore.unity3d.com/en/#!/content/54767
- *The Wasteland LITE*, available at https://www.assetstore.unity3d.com/en/#!/content/73054
- *Dumpster*, available at https://www.assetstore.unity3d.com/en/#!/content/655
- *Modular Road Block*, available at https://www.assetstore.unity3d.com/en/#!/content/87597
- *Fridge Old and New*, available at https://www.assetstore.unity3d.com/en/#!/content/24196

- *Crate and Barrels*, available at https://www.assetstore.unity3d.com/en/#!/content/73101
- *Stylized Vegetation* , available at https://www.assetstore.unity3d.com/en/#!content/59916
- *Survival Starter Kit.* , available at https://www.assetstore.unity3d.com/en/#!/content/17899
- *Medical Box*, available at https://www.assetstore.unity3d.com/en/#!/content/26967
- *Container*, available at https://www.assetstore.unity3d.com/en/#!/content/658
- *Tower The Last*, available at https://www.assetstore.unity3d.com/en/#!/content/70663
- *Lighting Generator*, available at https://www.assetstore.unity3d.com/en/#!/content/28173
- *Barrels*, available at https://www.assetstore.unity3d.com/en/#!/content/63623
- *Mega Fantasy Props Pack*, available at https://www.assetstore.unity3d.com/en/#!/content/87811
- *Tools and Logs*, available at https://www.assetstore.unity3d.com/en/#!/content/43971

Bibliography

[1] Masahiro Mori. The uncanny valley. *Energy*, 7(4):33–35, 1970.

[2] Robert R McCrae and Oliver P John. An introduction to the five-factor model and its applications. *Journal of personality*, 60(2):175–215, 1992.

[3] Sonia Roccas, Lilach Sagiv, Shalom H Schwartz, and Ariel Knafo. The big five personality factors and personal values. *Personality and social psychology bulletin*, 28(6):789–801, 2002.

[4] Lex Borghans, Angela Lee Duckworth, James J Heckman, and Bas Ter Weel. The economics and psychology of personality traits. *Journal of human Resources*, 43(4): 972–1059, 2008.

[5] Niall Mullally. An Emotional Model For Background Characters In Open World Games. Master's thesis, University of Dublin, Trinity College. Department of Computer Science, 2014.

[6] Maxis. The sims. [CD-ROM], 2000.

[7] Bill Merill. Building utility decisions into your existing behavior tree. In Steve Rabin, editor, *Game AI Pro*, chapter 10, pages 127–137. CRC Press, 2013.

[8] Bethesda Game Studios. Skyrim. [Digital Download], 2011.

[9] Francois Cournoyer and Antoine Fortier. Massive crowd on acu: Ai recycling.

[10] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1, 2013.

[11] Building worlds using math(s). https://gdcvault.com/play/1024514/Building-Worlds-Using. Accessed: 2017-07-12.

[12] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation. pages 141–150, 2010.

[13] Gillian Smith. Procedural content generation, an overview. In Steve Rabin, editor, *Game AI Pro 2*, chapter 40, pages 501–519. CRC Press, 2015.

[14] Kevin Dill. What is game ai? In Steve Rabin, editor, *Game AI Pro*, chapter 1, pages 3–11. CRC Press, 2013.

[15] Kimberly Voll. Less is more: Designing awesome ai. http://www.gdcvault.com/play/1022104/Less-is-More-Designing-Awesome. Accessed: 2017-07-20.

[16] Nick Bostrom. *Superintelligence: Paths, dangers, strategies*. OUP Oxford, 2014.

[17] Leo Galway, Darryl Charles, and Michaela Black. Machine learning in digital games: a survey. *Artificial Intelligence Review*, 29(2):123–161, 2008.

[18] Julian Togelius, Georgios N. Yannakakis, Sergey Karakovskiy, and Noor Shaker. Assessing believability. In *Believable Bots: Can Computers Play Like People?* 2012. ISBN 9783642323232. doi: 10.1007/978-3-642-32323-2_9.

[19] Mark Brown. Game makers toolkit - what makes good ai? https://www.youtube.com/watch?v=9bbhJi0NBkk. Accessed: 2017-08-10.

[20] Laurie R. Santos and Alexandra G. Rosati. The Evolutionary Roots of Human Decision Making. *Annual Review of Psychology*, 2015. ISSN 0066-4308. doi: 10.1146/annurev-psych-010814-015310.

[21] Robert Valdes. The artificial intelligence of halo 2. http://electronics.howstuffworks.com/halo2-ai3.htm. Accessed: 2017-08-01.

[22] Kevin Dill. A game ai approach to autonomous control of virtual characters. *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, 2011.

[23] Paulo Fontaínha Gomes, Ana Paiva, Carlos Martinho, and Arnav Jhala. Metrics for character believability in interactive narrative. In *ICIDS*, pages 223–228. Springer, 2013.

[24] Frank Thomas, Ollie Johnston, and Frank Thomas. *The illusion of life: Disney animation*. Hyperion New York, 1995.

[25] Ocelot Society. Event[0]. [Digital Download], 2016.

[26] David O'Reilly. Everything. [Digital Download], 2017.

[27] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[28] Kalanit Grill-Spector and Rafael Malach. The human visual cortex. *Annu. Rev. Neurosci.*, 27:649–677, 2004.

[29] Timothy Spellman, Mattia Rigotti, Susanne E Ahmari, Stefano Fusi, Joseph A Gogos, and Joshua A Gordon. Hippocampal-prefrontal input supports spatial encoding in working memory. *Nature*, 522(7556):309, 2015.

[30] Ayse Pinar Saygin, Thierry Chaminade, Hiroshi Ishiguro, Jon Driver, and Chris Frith. The thing that should not be: Predictive coding and the uncanny valley in perceiving human and humanoid robot actions. *Social Cognitive and Affective Neuroscience*, 2012. ISSN 17495016. doi: 10.1093/scan/nsr025.

[31] Dirk Scheele, Christine Schwering, Jed T Elison, Robert Spunt, Wolfgang Maier, and René Hurlemann. A human tendency to anthropomorphize is enhanced by oxytocin. *European Neuropsychopharmacology*, 25(10):1817–1823, 2015.

[32] Fritz Heider and Marianne Simmel. An Experimental Study of Apparent Behavior. *Source: The American Journal of Psychology*, 57(2):243–259, 1944.

[33] Carl F DiSalvo, Francine Gemperle, Jodi Forlizzi, and Sara Kiesler. All robots are not created equal: the design and perception of humanoid robot heads. In *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 321–326. ACM, 2002.

[34] Teresa Farroni, Mark H Johnson, Enrica Menon, Luisa Zulian, Dino Faraguna, and Gergely Csibra. Newborns' preference for face-relevant stimuli: Effects of contrast polarity. *Proceedings of the National Academy of Sciences of the United States of America*, 102(47):17245–17250, 2005.

[35] Atsushi Senju. Spontaneous theory of mind and its absence in autism spectrum disorders. *The Neuroscientist*, 18(2):108–113, 2012.

[36] Daniel Livingstone. Turing's test and believable ai in games. *Computers in Entertainment (CIE)*, 4(1):6, 2006.

[37] Brian Mac Namee. *Proactive persistent agents-using situational intelligence to create support characters in character-centric computer games*. PhD thesis, University of Dublin, Trinity College. Department of Computer Science, 2004.

[38] Chris Butcher and Jaime Griesemer. The illusion of intelligence. http://halo.bungie.org/misc/gdc.2002.haloai/talk.html?page=1. Accessed: 2017-06-10.

[39] Iskander Umarov and Maxim Mozgovoy. Believable and effective ai agents in virtual worlds: Current state and future perspectives. *International Journal of Gaming and Computer-Mediated Simulations (IJGCMS)*, 4(2):37–59, 2012.

[40] Markus Kemmerling, Niels Ackermann, Nicola Beume, Mike Preuss, Sebastian Uellenbeck, and Wolfgang Walz. Is human-like and well playing contradictory for diplomacy bots? In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 209–216. IEEE, 2009.

[41] Hsing-Kuo Pao, Kuan-Ta Chen, and Hong-Chung Chang. Game bot detection via avatar trajectory analysis. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(3):162–175, 2010.

[42] Roger Schnaitter. Behaviorism is not cognitive and cognitivism is not behavioral. *Behaviorism*, pages 1–12, 1987.

[43] Stuart Russell and Peter Norvig. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[44] Lotfi A Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy Sets and Systems*, 90:111–127, 1997.

[45] DR Graham. Breathing life into your background characters. *Game AI Pro*, 1:550, 2013.

[46] Mat Buckland. Programming game ai by example. *Jones Bartlett Learning*, 2005.

[47] Luke Dicken Troy Humphreys Michael Dawe, Steve Gargolinski and Dave Mark. Behavior selection algorithms. In Steve Rabin, editor, *Game AI Pro*, chapter 4, pages 47–61. CRC Press, 2013.

[48] Jakob Rasmussen. Are behaviour trees a thing of the past? http://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php. Accessed: 2017-08-04.

[49] Monolith Productions. F.e.a.r. [CD-ROM], 2005.

[50] Brent Owens. Goal-oriented action planning for smarter ai. https://gamedevelopment.tutsplus.com/tutorials/goal-oriented-action-planning-for-a-smarter-ai--cms-20793. Accessed: 2017-07-15.

[51] High Moon Studios. Transformers: Fall of cybertron. [CD-ROM], 2012.

[52] Lionhead Studios. Black & white. [CD-ROM], 2001.

[53] István Szita, Marc Ponsen, and Pieter Spronck. Effective and diverse adaptive game ai. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):16–27, 2009.

[54] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 2009. ISSN 09218890. doi: 10.1016/j.robot.2008.10.024.

[55] Brad Knox. Building character ai through machine learning. https://medium.com/mit-media-lab/building-character-ai-through-machine-learning-7a3159dc4940. Accessed: 2017-07-19.

[56] Jeff Orkin and Deb Roy. The restaurant game: Learning social behavior and language from thousands of players online. *Journal of Game Development*, 3(1):39–60, 2007.

[57] Dave Mark. The infinite axis utility system. https://archive.org/details/GDC2013Mark. Accessed: 2017-08-02.

[58] David Graham. An introduction to utility theory. In Steve Rabin, editor, *Game AI Pro*, chapter 9, pages 113–127. CRC Press, 2013.

[59] Kevin Dill. Dual-utility reasoning. In Steve Rabin, editor, *Game AI Pro 2*, chapter 3, pages 23–27. CRC Press, 2015.

[60] Blue Fang Games. Zoo tycoon. [CD-ROM], 2001.

[61] Maxis. Spore. [CD-ROM], 2008.

[62] Esb Bach and Andreas Madsen. Procedural Character Generation Implementing Reference Fitting and Principal Components Analysis. Master's thesis, Aalborg Universitet. Department of Computer Science, 2007.

[63] Georgios N Yannakakis and Julian Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2(3):147–161, 2011.

[64] Christopher Dragert, Jörg Kienzle, Hans Vangheluwe, and Clark Verbrugge. Generating extras: Procedural ai with statecharts. Technical report, Technical Report SOCS-TR-2011.1, 2011.

[65] Scott Aaronson. *Quantum computing since Democritus*. Cambridge University Press, 2013.

[66] Jay Goldblatt Steve Rabin and Fernando Silva. Advanced randomness techniques for game ai. In Steve Rabin, editor, *Game AI Pro*, chapter 3, pages 29–47. CRC Press, 2013.

[67] Random.org. https://www.random.org/. Accessed: 2017-08-01.

[68] David B Thomas, Wayne Luk, Philip HW Leong, and John D Villasenor. Gaussian Random Number Generators. *ACM Comput. Surv. Article*, 39(11), 2007. doi: 10.1145/1287620.1287622. URL http://doi.acm.org/10.1145/1287620.1287622.

[69] Patrick O 'Halloran. A Multi-tiered Level of Detail System for Game AI. Master's thesis, University of Dublin, Trinity College. Department of Computer Science, 2016.

[70] John Fallon and Mads Haahr. Believable Behaviour of Background Characters in Open World Games. Master's thesis, University of Dublin, Trinity College. Department of Computer Science, 2013.

[71] Sarah Noonan. Side Quest Generation using Interactive Storytelling for Open World Role Playing Games. Master's thesis, University of Dublin, Trinity College. Department of Computer Science, 2015.

[72] Tiarnán Mcnulty. Residual Memory for Background Characters in Complex Environments. Master's thesis, University of Dublin, Trinity College. Department of Computer Science, 2014.

[73] Unity game engine. https://unity3d.com/. Accessed: 2017-08-12.

[74] Uniform random number library. http://paulbourke.net/miscellaneous/random/. Accessed: 2017-08-01.

[75] Algorithm 712, collected algorithms from acm. this work published in transactions on mathematical software, vol. 18, no. 4, december, 1992, pp. 434-435. http://www.netlib.org/toms-2014-06-10/712. Accessed: 2017-07-12.

[76] George Marsaglia, Arif Zaman, and Wai Wan Tsang. Toward a universal random number generator. *Statistics & Probability Letters*, 9(1):35–39, 1990.

[77] Frederick James. A review of pseudorandom number generators. *Computer Physics Communications*, 60(3):329–344, 1990.

[78] Wizards of the Coast. Dungeons & dragons. [Tabletop Game], 1974.

[79] Warren spector on life after mickey, going 'no weapons'. https://www.rockpapershotgun.com/2013/05/14/warren-spector-on-life-after-mickey-going-no-weapons/. Accessed: 2017-07-15.