Residual Memory for Background Characters in Complex Environments

by

Tiarnán McNulty, B.Eng. (Hons)

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

September 2014

Declaration

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.

Tiarnán McNulty

September 1, 2014

Permission to Lend and/or Copy

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

Tiarnán McNulty

September 1, 2014

Acknowledgments

I would like to offer my sincerest gratitude to my supervisor, Dr. Mads Haahr for his exceptional support throughout the duration of this project. Dr. Haahr frequently and consistently offered wonderful advice and insights which greatly improved the project at every stage of its development.

In addition, I would like to thank Niall Mullally for his help in expanding the environment created for the prototype.

I would also like to thank my family for all their encouragement and support throughout my entire time in academia.

Tiarnán McNulty

University of Dublin, Trinity College
September 2014

Residual Memory for Background Characters in Complex Environments

Tiarnán McNulty
University of Dublin, Trinity College, 2014

Supervisor: Dr. Mads Haahr

This dissertation aims to explore methods for increasing the believability of background characters in open-world games by giving them the ability to react to situations in a much more natural manner than current approaches. Background characters play a vital role in making a game's environment feel cohesive and believable, but they generally follow scripted, repetitive motions until influenced by the actions of a player. This project presents a generalised memory model that enables characters to remember and recall the state of the world around them, either in the form of larger scale events or as smaller scale interactions, and react to any changes as they occur. Over time, memories which aren't reinforced become fuzzy, may be remembered incorrectly, and are eventually forgotten.

The model is tested by developing a small prototype environment within the Unity game engine, and observing how the characters within this environment behaved. A

goal-driven system allows characters to access their memories to determine the best methods to achieve their goals, and a fallback system allows characters to attempt to resolve their own problems if they find themselves unable to recall the memories they need. Through consideration of the memories they have, a character can determine how best to gain the memories they need, for example by asking someone they know to share their relevant memories. Players can even interact with the characters, to help or hinder them.

The implementation showcases a model with significant potential, where characters naturally develop habits and make decisions with genuine purpose, making them much more interesting to observe and encounter within the environment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation explores methods for increasing the believability of background characters in complex environments by giving them the ability to react to situations and recall past events in a much more natural manner than current approaches. The research focuses on the non-player characters (NPCs) that are not considered essential to a game's story, but which still play a vital role in making the game environment feel cohesive and believable. This project proposes a hypothesis that the creation of a generalised memory model would enhance overall believability by allowing NPC behaviour to be impacted by their ability to remember and recall past events. The hypothesis would be tested through the creation of a prototype environment that showcases the new model and allows for it to be observed and compared to current approaches. It is hoped that this model would lead to more immersive, interactive and engaging gameplay experiences for players, and by keeping the model generalised it would ensure it remains applicable to many different game genres.

1.1 Open World Games

An open world game is one in which a player is given considerable freedom to roam a virtual environment and choose how or when to approach their objectives [72]. Open-world design is largely absent of the artificial barriers that are particularly common in more linear designs.

The first open-world game, *Elite*, was created in 1984 and featured a fully inhabited

universe of planets in which players could freely explore [8]. Unlike other games at the time, Elite allowed the player to choose where they went and how they behaved, providing them with a variety of different options for progressing through the game. Elite was a seminal success for its time, with an estimated 600,000 copies being sold [77].

2001 saw the release of *Grand Theft Auto 3* for the PS2 [31]. This would prove to be a milestone in open-world games as it presented players players with a fully explorable 3D city packed with dozens of different missions they could tackle using whatever weapons and vehicles they liked. It became an overnight success and would spawn an entire genre of increasingly popular 'city sandbox games' [68].

Previously open world games were mainly PC cult hits due to the technical trade-offs and excessive performance demands they required to achieve their large environments. Elite rendered its universe entirely using wire-frame, while other, such as *The Elder Scrolls: Arena* [75], used extensive fog to only show the environment directly in front of the player. GTA 3's success can be partially attributed to the fact that hardware had now reached a point where open-world games could be created without such limitations.

As hardware performance continued to improve the scale and scope of environments in even traditionally linear games continued to grow, and today most games now have some element of open-world design within them.

1.2 Challenges of Open-World Games

The most significant design challenge in an open-world game is striking the correct balance between the freedom of the environment and the actual structure of the game's story. The designers need creative ways to impose a storyline on a player even when they could be performing actions that the designer did not expect. A large number of open-world games provide a main character who is a blank slate, allowing the player to project their own thoughts and actions into the environment. However it is becoming more common for open-world games to have a complex player character who has their own development, personality and dialogue. This approach often leads to the problem of 'ludonarrative dissonance', where the actions of the character during player controlled sequences and during scripted sequences are widely different and potentially

contradictory [82]. Another common way in which this dissonance surfaces is through the actions of other background characters whom the player might ignore or treat badly during the course of the game only for them to be friendly, supportive and completely ignorant of every negative action the player did to them during the scripted sequences.

World building proves to be another challenge for open-world games. Some can contain large but empty and uninteresting environments, while others contain repetitive locations used to give a false sense of a larger scale. Creating a vast, populated and detailed open environment is a very time consuming process, so procedural generation can be used to reduce this requirement while still keeping quality high. A smart use of procedural generation for characters, environments and objects can allow game systems to build the world rather than requiring the designers to oversee every individual element of it.

Finally, open-world games must consider the concept of 'emergence', a term which defines the complex situations within a game that emerge, either intentionally or by accident, from the interactions between relatively simple mechanics [18]. Emergence puts a focus on the design of the simulation and its rules, rather than explicitly scripting individual actions or events. A good open-world simulation will allow the player a lot of scope to explore, bend and occasionally break the rules, while a poor simulation will have much less room for improvisation and personalisation.

1.3 Motivation

In regards to the area of AI research, it has been stated that "of all the technological frontiers in world-building, artificial intelligence (AI) holds the most promise of change" [15] and that "from the point of view of world design, AI promises great things. If virtual worlds could be populated by intelligent NPCs, all manner of doors would open" [4]. Immersion is directly related to the wilful suspension of disbelief, a state in which a player chooses to fully engage with the game world regardless of its virtual nature. Considering this context, the role of background NPCs is to enrich this virtual world and further entice players into immersing themselves within the game world. NPCs which are unconvincing in this role can cause players to look more closely at other flaws in the environment and potentially lose their ability to fully immerse themselves.

Ed del Castillo believes that AI is one of the most important areas for enhancing open-world gaming, and that "in order to have great AI, you need [to] create systems that simulate life" [72]. Tom Howard has a similar viewpoint, stating "it's become common for developers to be able to push lush scenery together, but creating other characters that can react to what you do in a believable and compelling way is still very difficult" [72]. Brian Baglow clarifies the issue further, stating "The environment is not merely the setting for the action, but is an active part of the overall gameplay, which affects and reacts to the player as they progress" [72].

Within the context of an open-world game, an environment refers to all the background elements that make up the overall world. That includes the geometry, buildings, vehicles, objects, etc., but also the characters that actively inhabit the environment. These characters are usually the most important factor at play when considering a 'reactive' environment, and so their behaviour is crucial to the creation of an immersive world.

However when speaking about the quality of these background characters, Howard paints a grim picture as he states "We have a long way to go" [72]. The graphical fidelity, size, and general level of detail of game environments has increased at a rapid rate with each new generation of console hardware, however improvements in Artificial Intelligence has noticeably lagged behind the other fields [21]. This ever-increasing gap between A.I. and graphics manifests itself in the form of incredibly realistic game worlds being populated by simplistic and unconvincing caricatures of real people. As realism increases, the lack of intelligence in these NPCs becomes increasingly apparent.

The effect is somewhat similar to the advancements made in film, where early films used painted backdrops, simple special effects and various tricks to get around their limitations. As skills and technology improved, so did the methods by which environments were realised. While the simple methods were acceptable during their time, and were at least internally consistent, the stagnation of character behaviour is similar to a modern film employing a lot of CGI but still occasionally featuring a poorly painted backdrop. The modern nature of the other techniques would only cause the glaring nature of the backdrop to become readily apparent, and in a similar fashion simple behaviours threaten to break a player's immersion as games continue to improve. However the new generation of console hardware, and ever improving PC hardware, have powerful CPUs with large pools of memory which provide ample opportunity for

developers to explore and implement new and greatly improved A.I. This significant increase in available memory proves to be a key motivation in why now is the right time to explore how a memory model could improve NPC believability.

The complexity of A.I. required for creating believable NPC varies greatly across different genres, and can even vary with different NPCs within the same game. As such, the rather abstract and hard to judge question of "Is this a believable NPC?" could instead be refocused as "Will the player believe this NPCs and its purpose?" For example, an NPC shopkeeper has a completely different purpose to that of an NPC that simply drinks at a bar. The triggers that make people believe that either NPC is real are different, and their expectations of how they should behave changes too. Therefore the memory that each NPC chooses to retain should be related to its purpose and help further increase its believability.

The concept of believability is not unique to games, with it being explored across a variety of mediums including film, T.V, theatre and literature. In this essence, the believability of a character comes from not just their actions, but from their thoughts and speech too. This comprises their personality, emotion, motivation and relationships, but it also comes from their ability to be internally consistent. Which is to say that they follow similar patterns of learned behaviour, e.g., an NPC blacksmith would be more believable if they were to use their favourite tools on a daily basis instead of just randomly selecting the closest tool at a given time, which allows them to maintain an 'illusion of life' [50]. These types of characters are referred to as 'believable agents', and it is now widely accepted that a character's personality and their expressions of emotion are key factors in whether agents are perceived as 'believable' or not [6].

1.4 Improving Believability

The vast majority of interactive characters currently achieve their believability through heavily controlled scripting, with pre-defined animations, behaviours and dialogue being executed at the right moments to convey an illusion of life. This static design approach means that NPCs have no ability to respond outside of their scripting, so a player interacting with them in an unexpected way can cause undesirable, and usually bizarre, responses. There has been a large amount of research done to overcome these limitations, with various procedural techniques being developed to allow for more dy-

namic approach NPC responses, such as combining pre-made animations with dynamic blends to convey additional emotion in real-time. These blends are usually focused on the face, with the eyes and the mouth being capable of manipulation independently of any animation [28].

There has been significantly less work done with regards to memory models that could improve the believability of action choices, but A.I. researchers have explored how emotional models could be used in this context. These models allow believable characters to have a wider range of responses based on a variety of external factors that have impacted their overall mood [3].

1.5 Scope / Objectives

This dissertation aims to create a new model that uses residual memory to enhance the believability and behaviours of the non-critical background NPCs that populate the towns and villages of open-world games. These background characters usually have no direct influence in the overall plot or gameplay of a game but remain crucial in creating a believable and fully realised world for the player to explore. This model will then be prototyped and the results will be evaluated. The model will aim to achieve the following objectives:

1. Offer a tangible improvement over existing approaches for background characters.
2. Support the procedural generation of A.I. behaviours by giving NPCs initial 'prior' memory of the game world.
3. Provide an efficient implementation, which makes smart use of available CPU and memory.

The project's core focus is on the first objective, which is of primary significance to this proposal's aim to create more believable NPCs. The second objective hopes to take advantage of the ability for developers to 'pre-load' NPCs with residual memory so that they have default, initial behaviours that do not necessarily need to be scripted from start-to-finish. This pre-loaded memory would contain information about the character's history and the events that happened to the world before the game has begun. Finally, it is important that the model strives for efficiency where possible, as a more efficient implementation is far more likely to have wider use across the industry

than a bloated, buggy one, but as the proposal is only implementing a prototype this objective is secondary compared to the other two.

1.6 Roadmap

The dissertation continues in Chapter 2 with a review of the current industry state of the art for NPC behaviour, focusing on current models, approaches to believability, and recommended AI architectures. Chapter 3 uses the knowledge gained through the industry review to present a design for the new model which will address issues currently impacting the perceived believability of NPCs in related games. This will be followed in Chapter 4 with a breakdown of the process for how the model presented in the design was implemented in the prototype. This implementation is evaluated in Chapter 5, with the project's successes and failures being discussed and addressed. Chapter 6 presents a final conclusion, drawing together all the work done on the project and presenting the overall contributions it has made, as well as suggesting future work that could be carried out to further expand upon the model.

Chapter 2

State of the Art

The purpose of this chapter is to carry out a comprehensive review of the state of the art for a number of areas that prove crucial to the success of this project. Initially focusing on the concept of believable agents, the aim is to discover the aspects of human behaviour these agents must capture for players to consider them "'real"'. Identifying the requirements for truly believable agents is necessary before any model for cognitive behaviour can be created. Having identified the requirements the focus then proceeds on to an investigation of the current state of NPC agents through their accompanying behavioural models, to see how believable they actually are. Finally, the potential architectures which can be used to model NPC behaviour are presented and evaluated.

2.1 Believability

Believability is difficult to precisely define; thanks to a family of different meanings that are used almost interchangeably [36]. Simply put, believability quite literally means something that a person can believe. In the realm of background characters it can be further defined as a character, or aspect of that character, that a person believes is genuine, plausible or real within the context of the environment it inhabits. (i.e., a fictitious environment inhabited by alien characters can still be considered believable given the right context.)

2.1.1 Role of Human Behaviour

The potential, both positive and negative, within the field of Artificial Intelligence has always been wildly discussed, and it is commonly defined within two brackets - Weak AI and Strong AI. Weak AI encompasses any non-sentient computer intelligence that is usually assigned a very specific task and is the most widely used category of A.I. Voice recognition software is a particularly prevalent example of it, being software that will attempt to understand and respond to human speech but has no other functional capability. Weak AI is usually recognisable by the fact it can easily break down if it goes outside of its parameters, i.e., strong accents, foreign languages, or unusual speaking patterns can cause spectacular failures in voice recognition software. A benefit to this approach is that multiple weak AIs can be combined together, i.e., voice recognition and route planning, to provide a much stronger impression of overall intelligence.

Strong AI defines artificial intelligence that is capable of behaving as intelligently as a real person, and so by extension must have a mind of its own and the ability for conscious thought. This is AI most people consider to be real AI and the holy grail of a large degree of AI research, but is also the most contentious with many disagreements over what exactly is meant by 'human-like behaviour'. The Turing Test is a popular method by which such behaviour can be tested, but some believe that the very idea of Strong AI is a fallacy. Using the example of the Chinese Room, Searle argues that a program enabling a computer to perform intelligent actions does not mean the computer is intelligent as it does not actually understanding the action. Haugeland summaries this fallacy as "the problem with Artificial Intelligence is that computers don't give a damn" [35].

When considering the believability of NPCs this philosophical argument initially seems important, since the aim is to create characters that players believe to be genuine and real. However these is no true expectation of the NPCs being 'real' and as long as the outward behaviour appears 'strong' then the systems that drive the behaviour can be 'weak'.

Considering the need to display human-like behaviour naturally leads to a study of what human behaviour actually entails. Maurice Merleau-Ponty is considered to have the greatest insight into human behaviour [70], and so his work proves to be ideal for identifying the behaviour an NPC should imitate. Once this behaviour is identified

then weak AI systems can be used to model suitable imitations that are capable of convincing players that a character is real.

Merleau-Ponty's discussion of unreflective behaviour proves useful in this context, as he believes that human behaviour is caused by a person's perception of its environment. For example, someone who changes their behaviour to suit a social situation is displaying unreflective behaviour, as is a martial artist sparring in a tournament. Although the subject will be fully absorbed in what they are doing in the tournament, the actual movements will not be guided by specific thought. In addition, the behaviour required for a given environment is determined by the current task. Compare a golfer to someone who has accidentally wandered on to the course. In this case, the ball would considered by the golfer as *to-be-putted* but for the walker it would be *to-be-avoided*. These contextual perceptions of the objects in the environment naturally drive the actions associated with them, a process known as "absorbed coping" [58].

Absorbed coping provides a starting point for explaining how a person's intentions can drive behaviour, but it only considers unreflective behaviour. Essentially, an agent observes its surroundings and determines opportunities to perform actions that are suitable for the current activity. Absorbed copying has no concept of practical reasoning, and so struggles to capture a person's ability to make conscious decisions. This is most easily demonstrated when considering intentions that are formed from thoughts, e.g., a person considers their monthly expenditure and decides to buy a new television which forms the intention to go to the shop and ultimately drives the behaviour of leaving the house [70]. If perception drives action, then what role does intention play in this process?

Dreyfus suggests that intention initiates the absorbed copying process when behaviour has reached a standstill. This could occur in two scenarios: the person has stopped perceiving and so intention is necessary to restart the process or the perception is insufficient in determining the next action and so intention gives the system a nudge to continue. However this is an inadequate solution in both instances, as the first suggests that intention can only drive behaviour if the person was unconscious, or otherwise unresponsive, which is not true. Secondly, imagine a person returns home after a long day and collapses into a chair. They sit relaxed and content, but although they are receiving perceptions from the environment there is no change in behaviour and coping

appears to have ceased [22]. However in reality, this is not true. Considering the earlier examples, the person is sitting in the chair because *it-is-comfortable*, if their perceptions became such that the chair was uncomfortable it would likely trigger movement, to find a more comfortable position. This interpretation appears to overlook the key fact that a person can independently choose to act and then do so [70].

Romdenh-Romluc provides an elegant solution to the above contradiction by making the simply claim that humans can decide to take on projects, and that deciding to carry out a project involves forming the intention to do it. Since the current task determines the contextual perceptions provided by the absorbed coping process, and the decision to take on a task is achieved by forming the intention to do so, Dreyfus' initial account of unreflective behaviour already explains how intention can drive behaviour [70].

However Merleau-Ponty provides even more insight into this process through his discussion of instincts [58]. Physical creatures must satisfy their basic needs, such as hunger, rest, companionship, safety, etc. These needs impose specific demands that must be met, and so if a creature feels hungry they will seek food. The feeling of hunger naturally imposes the task of searching for food without explicit need to form the intention. It is also possible for a creature to progress from one task to another without making any actual decision, for example a creature manages to find food, eats the food, becomes satisfied but feels sleepy so takes on a task to rest. As such, a person can form intentions to carry out actions, and also instinctively carry out actions [70].

When carrying out a specific task, the actual environment and actual task drive a person's absorbed copying, but Merleau-Ponty also claimed that humans have the power "to reckon with the possible" [58]. The 'possible' in this context represents the other tasks a person could do or the possible environments they could inhabit. For example, passing a house of a friend will cause a person to recognise the house, and perhaps consider visiting it later. However the opportunities for action that drive the person's current task will be perceived as most urgent and so will be executed, but less urgent actions that relate to 'possible' tasks will also be perceived and stored [70].

This ability "to reckon with the possible" provides people with an appreciation of the options available to them, while perception allows for the demand of a task to be evaluated, allowing for one task to be selected out of many.

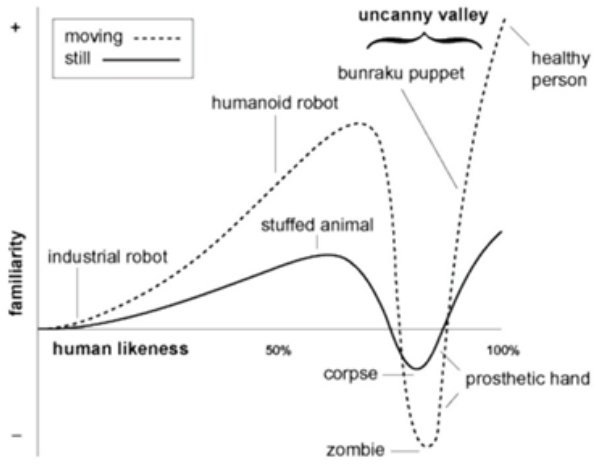

Figure 2.1: Dr. Mashario Mori's original depiction of the 'Uncanny Valley'. (Adapted from [24])

2.1.2 Believable Players

Believable players are NPCs within an environment which a player believes are real and being player by other humans, despite them actually being controlled by a computer. The aim of believable players is of interest mainly in the area of multiplayer games, where AI controlled characters take the place of human controlled ones, and so is not a focus here.

2.1.3 Believable Characters

Characters can be considered believable when a player believes that they are an actual living, autonomous entity. This belief is not truly genuine, but is achieved through the 'suspension of disbelief' and the willingness for the player to invest in the world and its characters.

As mentioned previously, believability is only loosely defined and it is often used to mean 'realistic'. That is to say that if a character is believable then they are real and so are 'realistic'. Within the context of a game, realistic characters are determined by how they visually appear and act, usually through a mixture of high quality textures, detailed animations, and plentiful dialogue. This means that a NPC can appear 'realistic' without being believable, a combination known as the 'uncanny valley'.

The 'Uncanny Valley', shown in Figure 2.1.3, refers to a temporary dip in the positive

relationship between how human a robot or virtual character looks and how comfortable people are with its appearance [24]. Additionally, Vinayagamoorthy found a strong relationship between the strength of the uncanny valley effect and the disparity of the character from its environment [86]. Realistic characters in low-resolution environments seemed less believable than low-resolution ones in the same space, and the reverse was also true. A higher resolution environment creates the expectation of a greater depth of character, which means that as game's get more detailed the believability of NPCs will be further called into question, as they increasingly seem to be "out-of-place".

Using *The Sciences of the Artificial* [73] as a basis, Togelius suggests that the believability of an NPCs behaviour could be as much influenced by the environment as the complexity of the NPC itself. To that end, optimising the environment for believability, in conjunction with an appropriate NPC, could be every bit as effective as simply optimising each NPC. This makes it conceivable that the most believable NPCs are achieved by a combination of well designed environmental and character interaction [85].

2.1.4 Creating Believable Agents

Aaron Bryan Loyall defines believable agents as "personality-rich autonomous agents with the powerful properties of characters from the arts" [51]. He makes the comparison that in the traditional arts a character is considered believable if it allows the audience to suspend their disbelief by providing a convincing portrayal of the personality they expect. A believable agent is simply an autonomous version of this type of character.

Unlike more traditional agent types, it is not a necessity for believable agents to be capable of accomplishing useful tasks or even to be effective at solving problems. Instead the focus is on less studied areas such as self-motivation, emotional drive, and social facility. A believable agent is entirely virtual and so a human actor cannot be expected to instinctively provide a lot of the more subtle aspects of believability, such as happens in film. However this disadvantage is not exclusive to games, being also shared with animated works and literature, and so these industries can provide useful insight towards solving this problem. Disney's The Illusion of Life tackled the difficulties in creating and animating real characters "out of nowhere" [84]. The interactive nature of these believable agents remains a unique challenge, which makes their cre-

ation particularly difficult. It is impossible to explicitly control a believable agent, they can only really be guided within a loose framework. This means there is an inherent uncertainty to the situations an agent will encounter, which is only compounded by the potentially chaotic interaction provided by a human player. In linear mediums the creator can control and highlight specific aspects of a character's personality, emotions and actions to push the story forward, while a believable agent needs to contain a much more comprehensive and within the context of the environment 'complete' personality to allow them to correctly react to as many situations as possible, even some which the creator might not have explicitly considered [51].

Loyall also provides a set of criteria that provides a standard for believable agents, which he derived from an analysis of character-based artists and experience gained from agents.

Personality
Personality is considered to be the most important requirement for any believable agent. It is so powerful that it can single-handedly draw the audience in and bring the characters alive. It is claimed that for a character to be real they must have an interesting personality [84]. The personality is defined as all of the particular details (i.e., behaviour, thought and emotion) that collectively define an individual.

Emotion
It is important for believable agents to have emotion reactions, and for those emotions to be conveyed in a logical way. These emotions should be inspired by the personality defined above, and artists often consider a character's emotional response to be the characteristic that brings them to life.

Self-Motivation
It is common for autonomous agents to only react to external events, not to any actions initiated of their own accord. Believable agents, however, need to be self-motivated and give the illusion that they are "really appearing to think" for themselves [84]. For example, an agent which gets hungry should begin to seek out food. These sometimes inconsequential actions have a profound impact on the believability of an agent.

Change

Egri [25] claims that bad writing is defined by characters that forever remain the same or which change without regard for their own characteristics. In the case of a believable agent, this change could be in how it reacts to a player over time based on prior interactions. Although difficult to achieve, character growth has a profound impact on believability.

Social Relationships

Humans are naturally social creatures [11]. The believability of an agent can similarly be defined by how it interacts with other agents. These interactions can be influenced by a character's own personality, the personality of the agent (or agents) it is interacting with, as well as the relationship these agents have with each other. The artists at Disney believe that these relationships are central to believability, but as no two relationships are the same this can be quite a difficult concept to realise [84].

Consistency of Expression

At almost every moment, an agent will have a variety of potential methods of expressions available to it. These can include facial expressions, posture, actions, etc. To be believable the combination of these expressions must be consistent, conveying one collective message that matches the agent's personality, current feelings and situation [78]. A contradiction of expressions can almost instantly ruin an agent's believability, although conversely a contradiction in expressions can be used to convey a different message (i.e., lying). Such contradictions must be made clearly intentional to ensure they seem sincere.

It's also important for expression to be consistent across a longer time period. A character simply cycling through animations will seem much less believable than an agent which has habitual expressions.

The Illusion of Life

The final requirement is of particular interest to believable agents as it covers an area that is simply taken for granted in other mediums. This covers some 'common sense' areas that are so completely assumed that they can be easily forgotten when it comes to designing these agents. The ability for human characters to multi-task, i.e., walk

and talk, is something that is never explicitly stated as a necessity in animated, film or literature works but almost all characters are naturally capable of doing more than one thing at a time. As such, when creating believable agents they must also capable of these 'sub-conscious' abilities.

- *Appearance of Goals* - Characters are naturally goal-driven, so agents must also appear to have goals.
- *Concurrent Pursuit of Goals / Parallel Actions* - Characters generally have multiple goals, both short term and long term, so believable agents should be also capable of working towards multiple goals in parallel. Either simultaneously, such as working on something while conversing with the player, or as a progressive, repeatable, and interruptible sequence of steps.
- *Reactive and Responsive* - Believable agents should react and respond to events as they are happening, not just at the exact moment they occur. For example, an agent crossing the street should react if a speeding car approaches, not only once it has been hit. Similarly these reactions should be responsive, but in a believable fashion. An observer would likely expect an agent to react to a gunshot faster than they would someone calling their name, and similarly their personality should impact how they respond to events. Tired agents should react slower than energetic ones.
- *Situated* - Agents should appear situated in the world around them, as described by [1], changing what they are doing and how they are doing it in response to any events unfolding around them.

  This can also include the bounding of an agent's available resources, either mentally or physically. For example, it is not believable for an 'old person' agent to be capable of lifting, or moving, heavy objects multiple times their own weight. Similarly, NPCs that instantly respond to the player could be considered to be thinking 'too fast', so delays can actually help believability.

  Agents should exist within a social context, and as such understand the social conventions, culture and environment in which they exist. Their behaviours should be consistent with those around them, unless their personality is such

that they are defying convention.

They should also be well integrated, with complementary capabilities and behaviours. If all of the agent's routines are separate independent modules then a viewer might see the boundaries in capabilities as the different modules are given control, i.e., an agent stopping when it attempts to process its next line of speech. A well-integrated agent will also prevent contradictions between actions and words, such as an agent which speaks as if it knows nothing of the world, but then confidently pathfinds across a large area.

2.2 Emotional Agents

When selecting behaviours for NPCs, the overall aim is to create as lifelike a performance as possible and so the combination of behaviours available to each NPC must be suitable for this purpose. They should also focus on areas that players would most likely use as identifiers for believable behaviour. When it comes to identifying these areas, it is best to first look at how existing psychological theories have been translated into the digital realm.

It is now well understood that believable agents must be emotional. Previous sections provided insights into this problem, but they do not provide any computational suggestions for how these believable emotional agents can be created. However, the fields of psychology and cognitive AI do provide some suggestions for tackling this problem.

2.2.1 OCC Emotional Model

When discussing believability there is one core focus that comes up repeatedly across a variety of related works - Emotion. Bates argues that "appropriately timed and clearly expressed emotion is a central requirement for believable characters" [6]. When discussing believability, Chuck Jones, animator at Warner Brothers, states that "Character always comes first, before the physical representation. Just as it is with all living things, including human beings. We are not what we look like. We are not even what we sound like. We are how we move; in other words, our personalities. And our personalities are shaped by what we think, by where we come from, by what we have experienced. And that personality is unique to each of us" [41]. The argument being

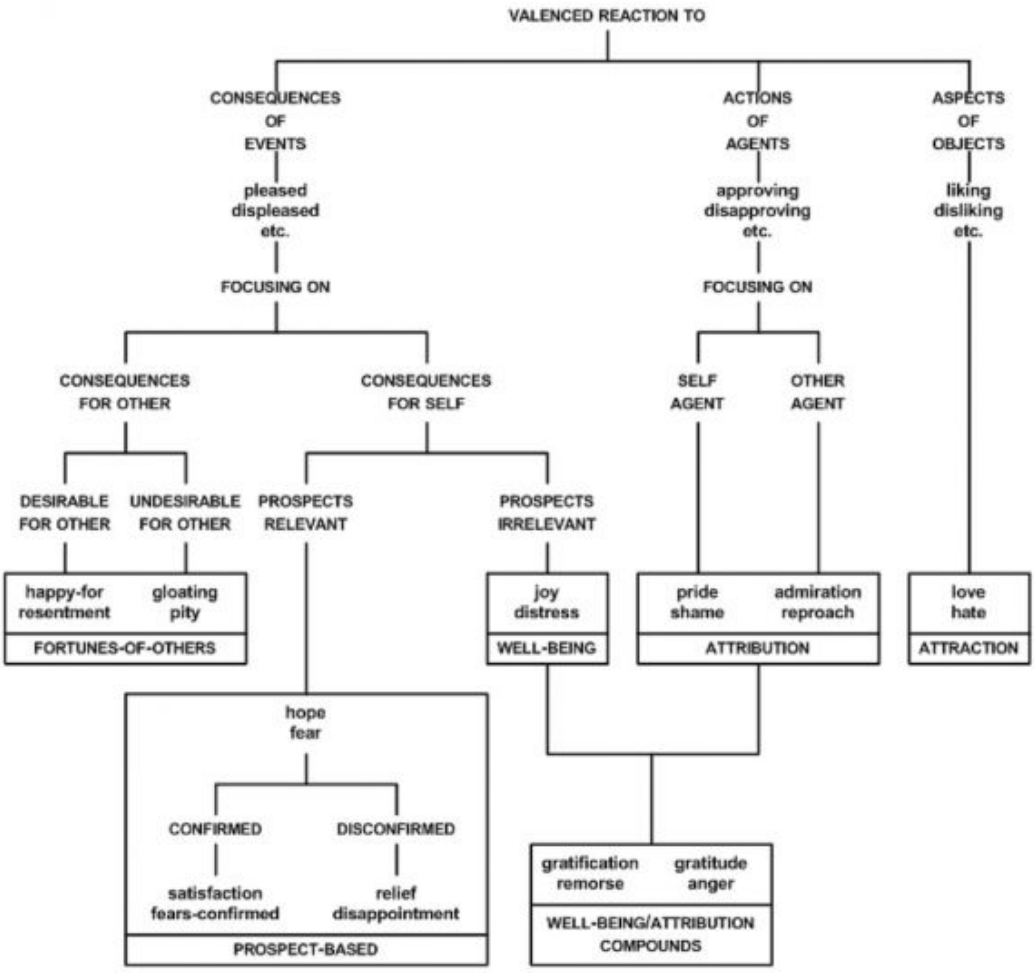

Figure 2.2: The structure of the OCC Emotional Model. (Adapted from [64])

that when a character is believable the creator's job is not to invent what the character did but actually to report their doings.

Knowing an agent's personality allows for a natural decision to be made about how they should act, and the OCC Emotional Model provides a solid framework for this approach. This model breaks down the creation of emotion in to an easy to understand structure, that can then be applied to believable agents. Within the OCC model, emotions are triggered when an event, action, or object is appraised. This appraisal reveals if an agent is pleased or displeased, and produces a consequence that is applied to either the agent or others in the environment. If applicable, the other agents will then determine if the consequence was desirable or undesirable. If the consequence had a direct impact on another agent then that agent will rate the consequence relative to its current goal, which this will elicit an emotional response dependent on it having a positive or negative impact on their ability to attain the goal. If an agent recognises the action which triggered the initial event then they will make an appraisal related to who carried out the event and if it was considered a positive or negative action. This will result in a change to the agent's opinion of the character that triggered the initial event.

The appraisal of events is key to how the OCC model captures personality, and thus powers believability, as the emotional response can then be mapped to an action or state. Bates presents a simple example of the OCC model through three OCC Agents with different personalities. Although all three agents can get in a situation which elicits the 'fear' emotion, it triggers the 'alarmed' state in one agent and the 'aggressive' state in another. Similarly, when appraising an event caused by another agent that has caused a character to fail their goal, the 'anger' emotion is triggered [6]. This combination of emotional cause and effect allows for the OCC model to easily build up a very complex web of emotional reactions from a small set of basic events.

The OCC model can also be used as an evolving model, changing an NPCs personality over time in reaction to the player's actions. Winanda presents an implementation of the model which can be used to power a variety of values that represent an NPC's opinion of a player and their personality which allows them to choose the best emotional response to player actions [88].

2.2.2 GO Emotional Model

There have been few attempts to decompose emotion-based action within the fields of AI research, probably because of the difficulty of the task. There are multiple obstacles preventing a satisfactory solution to this problem. Should the actions be merely expressive, or should they also drive the underlying system? What level of emotional complexity is most suitable - should it be as simple as making an embarrassed agent blush, or should it allow for a betrayed agent to plan revenge? If agents are already carrying out behaviours then how can emotion-based action help agents to further their goals? What is the specific purpose of an emotional system, and how can it be integrated in such a way that the new sequence is more beneficial than the old? Finally, taxonomic issues prove a significant challenge. Smiling is an emotional response, but it can be triggered by a person feeling pride or happiness, or it can be used to mask negative emotions such as sadness. Other responses are actions that are heavily associated with particular emotions, such as glaring at someone when angry. Emotional expressions form many-to-many relationships, with most emotions having multiple possible expressions, while one expression could be triggered by many possible emotions [26].

This creates a weak-theory domain, one where some relationships consistently hold but where most only occasionally do.

Figure 2.3 provides a breakdown of the action response categories and their theoretical groupings for the 'gloating' emotion. The high-level categories are arranged from the most spontaneous to the most planned. Within each category the particular response are similarly arranged. Within each action the tokens / mini-plans are arranged from least to most intense. Of course, these ratings are purely subjective.

It is important to consider that this is not a list of action words associated with a particular emotion (e.g., boasting as a glory action) and is instead a record of a purpose that would better and more generally reflect a possible internal state or action. Although 'slapping' an inanimate object is a obscure way of gloating, it is more useful to the system than saying 'crow' [26].

The model presents twenty-four emotion categories, identical in design to Figure 2.3, each with twenty action response categories, each with three actions. This gives 1440

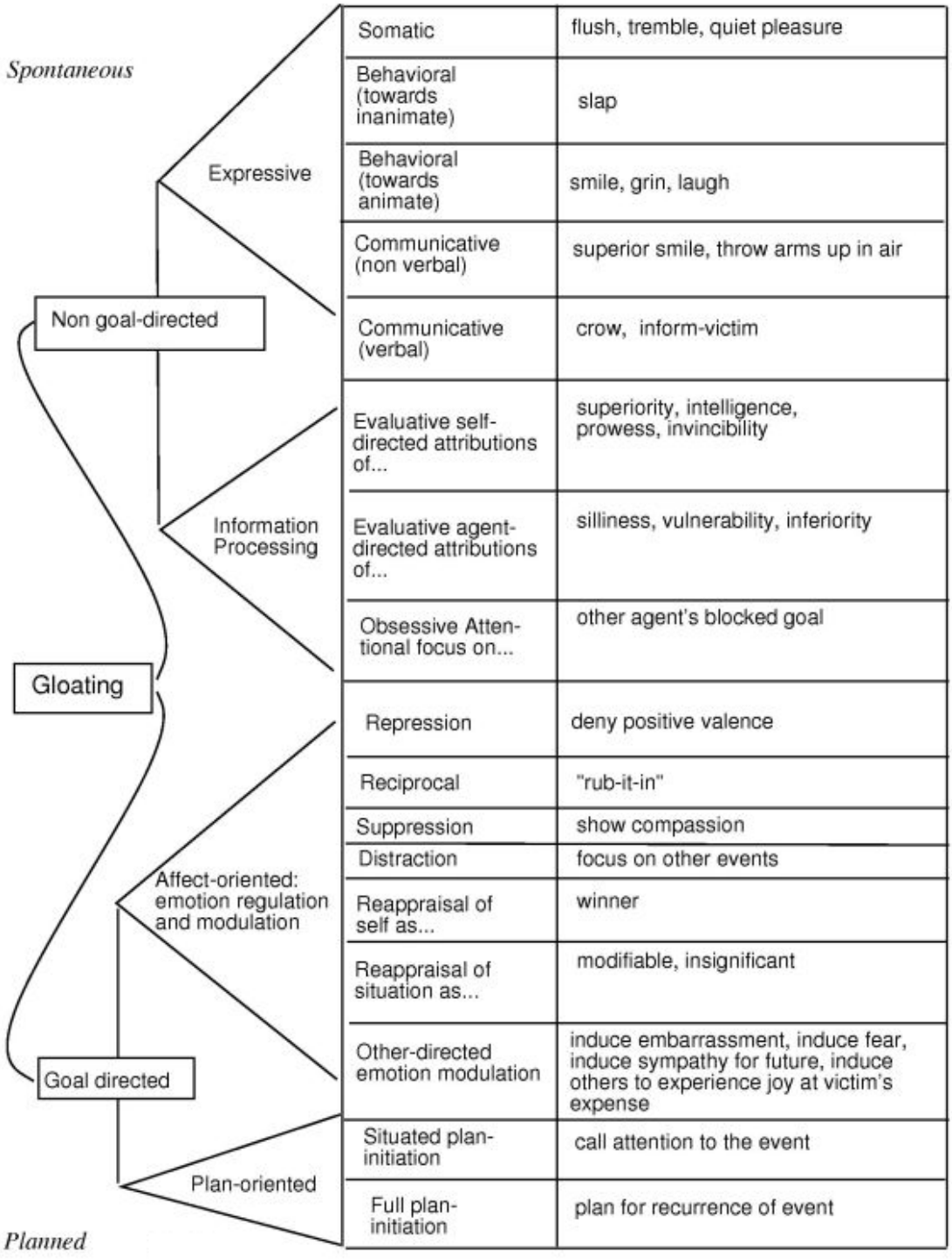

Figure 2.3: The structure of the GO Emotional Model, showing the action response categories for 'Gloating'. (Adapted from [26])

Table 2.1: The Twenty-Four Categories of Emotion. (Sourced from [63])

GROUP	*SPECIFICATION*	***CATEGORY LABEL** AND EMOTION TYPE*
Well-Being	appraisal of a situation as an *event*	**joy**: pleased about an *event* **distress**: displeased about an *event*
Fortunes-of-Others	presumed value of a situation as an *event* affecting another	**happy-for**: pleased about an *event* desirable for another **gloating**: pleased about an *event* undesirable for another **resentment**: displeased about an *event* desirable for another **jealousy***: *resentment* over a desired mutually exclusive goal **envy***: *resentment* over a desired non-exclusive goal **sorry-for**: displeased about an *event* undesirable for another
Prospect-based	appraisal of a situation as a prospective *event*	**hope**: pleased about a prospective desirable *event* **fear**: displeased about a prospective undesirable *event*
Confirmation	appraisal of a situation as confirming or disconfirming an expectation	**satisfaction**: pleased about a confirmed desirable *event* **relief**: pleased about a disconfirmed undesirable *event* **fears-confirmed**: displeased about a confirmed undesirable *event* **disappointment**: displeased about a disconfirmed desirable *event*
Attribution	appraisal of a situation as an accountable *act* of some agent	**pride**: approving of one's own *act* **admiration**: approving of another's *act* **shame**: disapproving of one's own *act* **reproach**: disapproving of another's *act*
Attraction	appraisal of a situation as containing an attractive or unattractive *object*	**liking**: finding an *object* appealing **disliking**: finding an *object* unappealing
Well-being/ Attribution	compound emotions	**gratitude**: admiration + joy **anger**: reproach + distress **gratification**: pride + joy **remorse**: shame + distress
Attraction/ Attribution	compound emotion extensions	**love**: admiration + liking **hate**: reproach + disliking

possible actions within the system, and even only allowing three simultaneous actions to represent the physical manifestation of an emotion gives 1440^3 possible combinations [26].

The twenty-four categories of emotion are shown in Table 2.1 sourced from [63], while the action response categories build upon [32], with the actions coming from [26].

Characters are represented within the GO Model with their own unique personalities, controlling how they choose which actions are most suitable for specific emotional triggers. The model splits personality into two areas: the 'interpretive' component represents how they react with respect to their interpretations of situations, while the 'manifestative' component represents their temperament and how they express or manifest their emotions. For example, a character could have an optimistic or pessimistic interpretation of events, along with a passive or aggressive manifestation of his emotions [26].

2.2.3 OCC / GO Emotional Hybrid

Elliot describes further research that combines the work of the OCC and GO model to create a hybrid model for the processing of emotions in multi-agent systems. The main contribution comes in the form of an 'Affective Reasoner' which breaks stories down into features which comprise the antecedents of emotions. The appeal comes from how a simulation unfolds as the agents interpret these antecedents in their own way to create a dynamic and appealing story [26]. Figure 2.4 shows how the Affective Reasoner can be used in combination with the OCC and GO models to generate the correction emotional action response for a given situation.

The combination of both models creates a very complex hybrid, and it is not well suited to a non-programmer. This makes it of less use to a designer, and more limited in that sense. Designing all the required action responses is also particularly complex, and some limitations necessary for implementation had to be made. For example, the Reasoner has no concept of emotional intensity or duration. An agent can be angry, but it has no way of knowing how angry an event actually is, therefore a minor annoyance is treated the same as a major one. Additionally, the model uses working memory to store basic relationship information about emotions (e.g., Tom knows that Harry was mad) but has only basic complexity and loses out on a lot of finer detail.

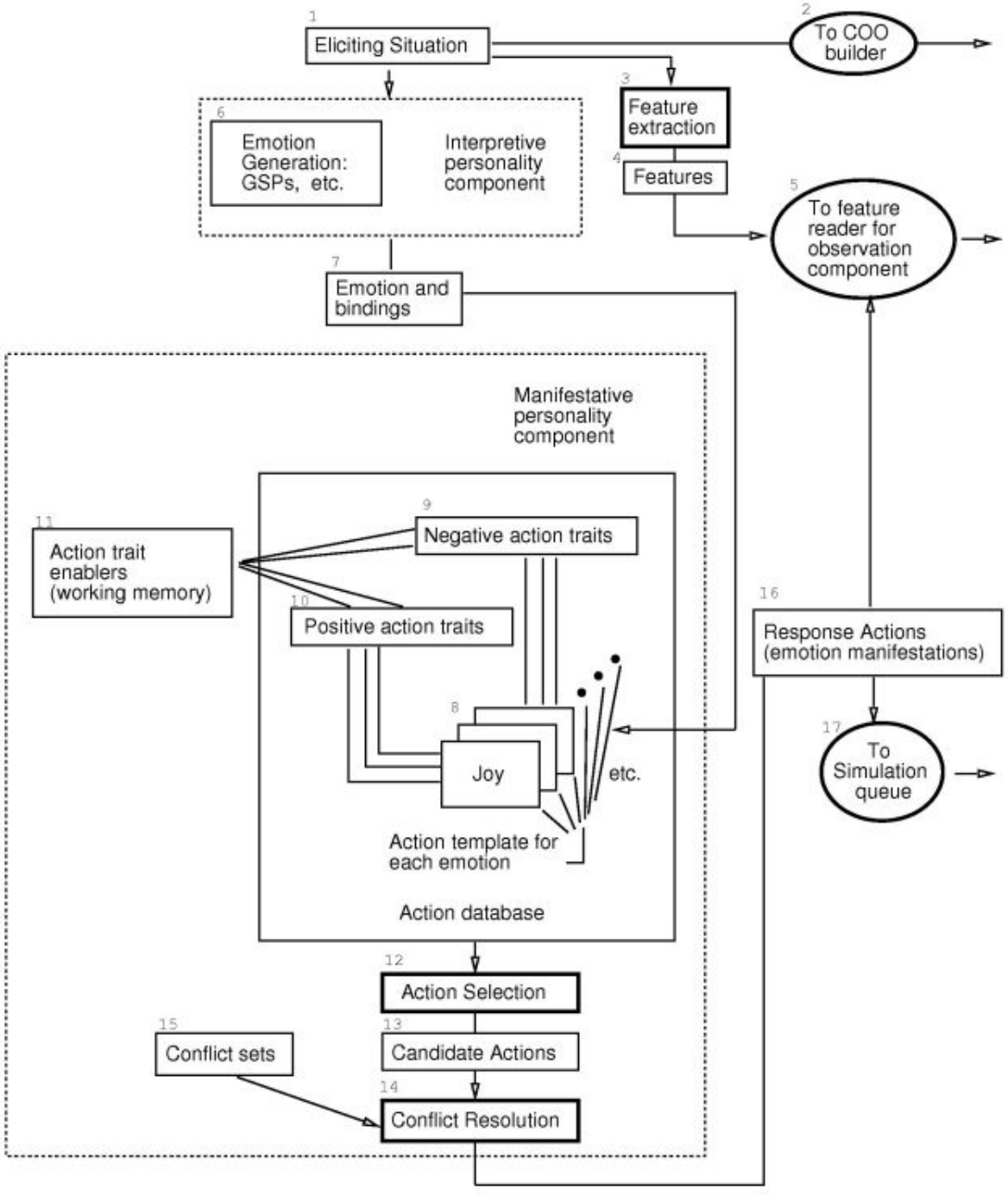

Figure 2.4: The Affective Reasoner process for generating an action response. (Sourced from [26])

2.3 Narrative Agents

When considering the research done by AI researchers, the work done to create believable agents is, of course, key. However, when considering the work done to create believable characters in other industries, the comments by Disney [84] and Warner [41] suggest that a potential area of interest could be in the area of AI research that creates stories. Good stories need believable characters, and so the tools and learnings of that field could be directly applicable to this one.

Meehan created TALE-SPIN, a program capable of creating simple stories in the style of old fables. The stories feature simple characters with interesting personalities and their own unique goals, and are generated by the program simulating the interaction of the characters in the environment over a period of time. The characters use the following scalar values: affection, competition, deception, trust, domination, familiarity, indebtedness. These values drive the interactions, which are recorded and create the story. Although the stories are not interactive, Meehan does provide an insight into how to generate believable agent to agent interactions [56], [57].

Lebowitz created UNIVERSE, a similar program which creates soap-opera style stories and features four-dimensional character relationships (positive / negative, intimate / distant, dominant / submissive, and attractiveness). Although three dimensions were sourced from Schank [71], based on the work of Wish [89], it is particularly interesting to note that Lebowitz added the dimension of 'attractiveness' because he found it necessary for the proper generation of believable stories [48], [49].

When considering the requirements necessary for believable agents, interactive or not, Reilly points out that there is a particular need to consider the social role of an agent. These roles are usually implicit when designing characters in other arts, but a computer needs the role to be made explicit so that it can act on it in a believable manner [67].

When it comes to interactive methods for story generation, the Oz Project proves to be a notable success. It blends AI technology with ideas and insight from traditional arts and media. The long term goal for Oz was to develop a popular and widespread new form of entertainment [5], [52], [53], [54].

The research of the Oz Project culminated in the creation of Façade, a fully interactive story [76]. Façade puts the player in an dynamic, ever changing environment where

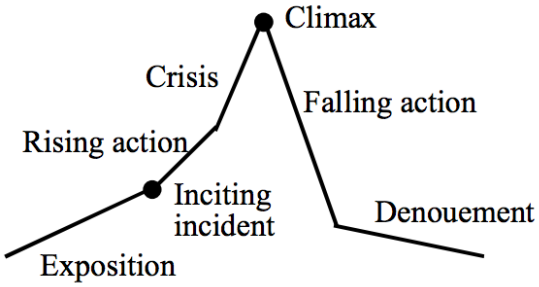

Figure 2.5: Freytag's Dramatic Arc (Sourced from [53])

two agents try to create an intriguing dramatic arc (such as in Figure 2.3) on each play. The history of the two characters is randomised at the start of each story, allowing for even identical player actions to cause entirely different stories on subsequent plays.

Façade's dynamic story is built upon the concept of beats, referencing the smallest unit of change in dramatic theory. A beat roughly consists of an action / reaction pair between characters, for example a character speaking and another reacting. Beats are selected by the drama manager to raise or lower the intensity according to follow the flow of the dramatic arc. An important part of the Façade system is that it remembers what beats have previously occurred, giving the impression that characters remember what they and the player have previously said and did [53].

However the characters themselves have no real agency or memory, simply being told exactly what to do by the drama manager. This makes them more like puppets than truly independent agents. Although to the end user these puppets appear just like believable agents, the drama engines omnipotence does limit its ability to be applied to large situations or environments.

Although narrative agents might initially seem unrelated, their similar need for believable agents creates a lot of overlap. When considering games, believable agents must also be engaging and so a consideration of the areas that narrative AI researcher focus on gives additional insight into the kind of behaviours that players would find appealing. This allows a better understanding of the role believable agents play in stories and what behaviours play in to the telling of a good story. Reilly notes that studying this field provides a better understanding of the many ways emotions can be

expressed. For example, a narrative reaction to failure can be depression. A depressed agent personality can then respond to failures by being more likely to abandon their current goal [67].

2.4 Cognitive Agents

As previously noted, a large amount of research has been done in the area of believable agents. This research primarily focuses on the creation of believable emotional responses, but little is done in the way of giving these agents the ability to remember the events that caused the emotions in the first place. An agent might remember that they are 'mad' or 'sad', but what, or who, caused them to feel that way is never stored. This can create odd and undesirable phenomenon, such as repetitive behaviours, and a general lack of learning and understanding.

In recent years, more research has been done to understand how enhanced cognition could help create better agents. Some research even goes as far as to say that truly believable agents require "histories", e.g., memories, which allows them to perceive and interpret the world in terms of their own experiences [20]. Agents must also be capable of explicitly communicating with each other, and each agent must contribute to the dynamics of an entire group, or society, and that society should also contribute to the individual. Others argue that the current lack of intelligent depth in NPCs diminishes believability, and ultimately creates a less enjoyable gaming experience for the player [50].

2.4.1 Parametrised Cognition

A parametrised memory model allows for agents to story sensory, working, and long-term memories. This model also has methods for acquiring, story, and retrieving these memories, and is aimed at long-term individual NPCs, those likely to feature as companions or recurring enemies that are central to a game's story. The focus comes from the belief that these are the characters a player examines most closely, and also from a pragmatic admission that it is technically infeasible to use such a model on a large number of short-lived characters. Focusing on long-term NPCs creates more reasonable competition and more engaging interactions [50].

The parametrised model attempts to give NPCs cognitive abilities that are roughly

those of a human, i.e., sharing similar limitations. It also incorporates elements of false memories, as well as distorted or hazy memories, to provide more human-like behaviour. 'Fuzzy' memories are a common feature in cognitive models, also being used in [10], [46], [69], and [9]. An NPC that continues to expand its memory will eventually consume all available space and cause the simulation to crash, but allowing memories to decay enables the system to reach a stable equilibrium, and [10] argues such decay is a technical necessity.

Memory is represented as a directed graph with nodes representing concepts and edges enacting links between concepts. A directed edge approach is used as humans generally archive memories in a specific sequence which does not necessarily work in reverse. This representation has the added benefit of allowing NPCs to work through memory recall in a linear sequence. For example, if an NPC is trying to remember the formula for a magic potion it could recall "To concoct this potion I need to add one more item after adding the purple potion, but I can't remember what". This recollection could then be used to recall a location where it can find the ingredient it is missing. A directionless representation would make such activities much more difficult to capture.

Two strength factors, $\text{Node}_{\text{Strength}}$ and $\text{Edge}_{\text{Strength}}$ are used to represent false memories. $\text{Node}_{\text{Strength}}$ indicates how strongly a concept is encoded within the model, while $\text{Edge}_{\text{Strength}}$ denotes the degree of ease traversing from one node to another. Essentially, $\text{Node}_{\text{Strength}}$ is how strong the memory is, while $\text{Edge}_{\text{Strength}}$ represents how familiar other memories are to the original memory. As an example, imagine an NPC builder who is trying to build a ladder. He remembers that he needs wood and nails, but also a tool to combine the two. He remembers multiple tools that could do the job, i.e., a hammer or nail-gun, but he is more familiar with using a hammer and so the hammer will have the larger $\text{Edge}_{\text{Strength}}$ and be selected as a tool to use. Essentially, $\text{Edge}_{\text{Strength}}$ can represent habits, or experience, and allow for NPCs to be consistent but also personalised as they develop in the simulation. Remembrance thresholds are used to support memory distortion, with a simple probability calculation determining if a memory below the threshold is returned correctly, misremembered, or entirely forgotten.

Sensory Memory

The Sensory Memory module maintains transient information captured by the sensory system. The paper only addresses vision, i.e., the component handles the creation of memories which correspond to the location of items in the environment. The NPC can then recall the item, and its location, when searching for it later. An upper limit is placed on the capacity of this module, to represent a human's ability to only process a certain amount of items at once [13].

Working Memory

This module stores information currently being used by the agent for a variety of cognitive activities such as reasoning, thinking and comprehending. Compared to long-term memory, the working memory has a much smaller working size and maintaining time. Essentially, to represent human ability only one graph structure is allowed in working memory at a time, with others being stored in long-term memory. An algorithm, inspired by human thought patterns [79], determines how strongly the nodes and edges will be retained after the information leaves the working memory.

Long-Term Memory

Unsurprisingly, this module contains an extensive number of concepts and maintains them for longer periods. Concepts within long-term memory will have their $\text{Node}_{\text{Strength}}$ and $\text{Edge}_{\text{Strength}}$ slowly decay, with the decaying percentage roughly following the Ebbinghaus forgetting curve [23].

Working Pattern

When cues are observed in the environment the sensory memory passes them to the working memory, where connections are made and a graph structure representation of the items is created. The cues in the working memory are then matched against long-term memory, and if a prior memory exists then using the previous $\text{Node}_{\text{Threshold}}$ and $\text{Edge}_{\text{Threshold}}$ the information is either retrieved correctly or incorrectly. For example, if an NPC is looking for his car keys, it is possible that he will remember where they keys are, or he could incorrectly remember where his house keys are instead. This correlation is possible because the two sets of keys are categorised under a meta-concept, i.e., 'keys'. If it has been too long since the NPC last saw his keys that the memory has entirely

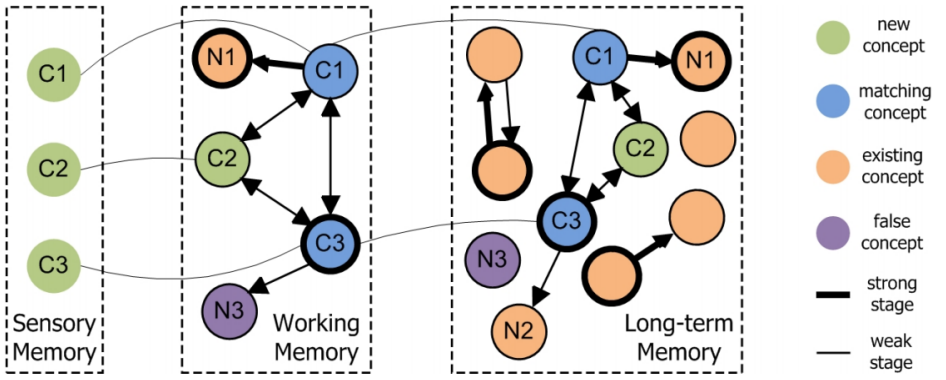

Figure 2.6: The general working pattern of sensory memory, working memory, and long-term memory. 'C' stands for *Cue* while 'N' stands for *Node* (not all nodes are labelled). (Adapted from [50])

decayed, then he will have no clue where they are. This relationship is shown in Figure 2.6.

The parametrised nature of the model allows for a lot of adhoc tuning, with tests being executed to understand what settings most closely simulate real humans with good, bad, and average recall ability and this allows for a great deal of personalisation in what NPC types are created. However the focus is mainly on the remembrance of items alone, with no mention of the models ability to remember people, places, events etc. As a generic model it provides a lot of useful information, and could certainly be used as the foundation for a more comprehensive model.

2.5 Episodic Cognition

Another popular approach to cognition is to store memories in an episodic format (such as in [10] and [9]), of which DyBaNeM is a particularly recent framework. DyBaNeM improves upon previous episode memory approaches by using a Baysesian representation to more easily model some the crucial facets of human memory, such as the hierarchical organisation of episodes, re-constructive memory retrieval, and the encoding of episodes with respect to previously learn schemata. The framework is inspired by the Fuzzy-Trace Theory (FTT) [29], and uses the Dynamic Bayesian Network (DBN) [46] as its underlying probabilistic model.

DyBaNeM has several key innovations over its predecessors. It is the first episodic

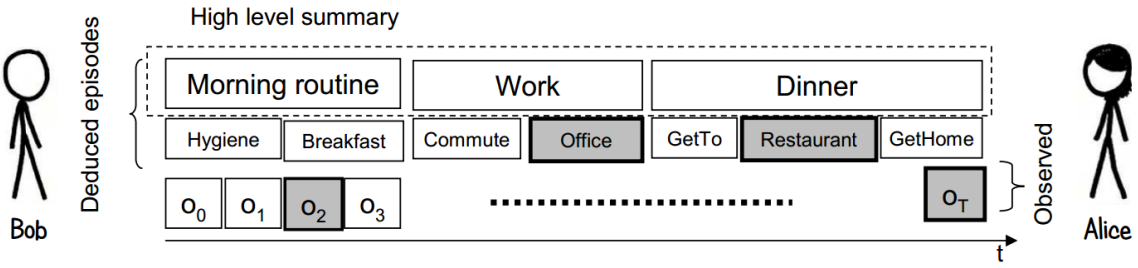

Figure 2.7: A summarisation example. Bob is monitoring Alice, and has observed the above atomic actions (Hygiene, Breakfast etc.). He deduces the higher level episodes (Morning Routine, etc.), and then encodes the day using the most salient events (the shaded mems). (Adapted from [42])

memory model to enable the reconstruction of a hierarchical episode structure (i.e., the episode to sub-episode relationship shown in Figure 2.7), so an agent can reconstruct the high-level goals of another agent to remember them. The framework also enables a probabilistic re-constructive inference process, that allows for the reconstruction of events that have not actually happened, but which are highly likely give the sequence of stored memories surrounding. Essentially, when considering a daily routine an agent can infer that another agent is most likely at work, as they always are at this time. The model only remembers the most salient events, as opposed to the comprehensive log of all events that were stored by DyBaNeM's predecessors, for a more efficient use of resources. Like most other models, but a first for episodic memory, it also allows for incorrect, or fuzzy, memories to be recalled. Finally, an agent's personal experience is encoded as part of an episode, meaning two agents may remember the same episode slightly differently [42].

The framework enables the following feature set for a player / NPC interaction.

1. An NPC can provide a high level summary of an activity. For example, when describing its day - "After getting up I went to work. In the afternoon I visited my friends, and then I returned home."

2. It can respond with further clarification, e.g., "How did you get to your friends?" - "I walked there."

3. The NPC can also convey degrees of certainty for recalled events, e.g., "Actually,

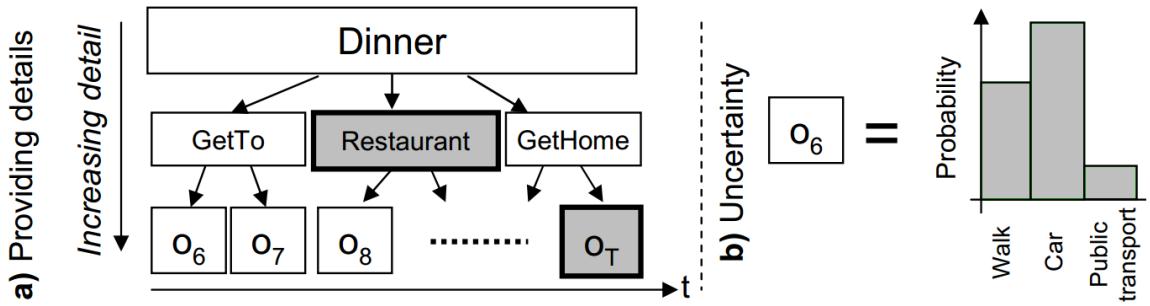

Figure 2.8: a) If Bob is asked for detail about Alice's dinner, he can break the episode into more detail. Shaded boxes represent stored mems, while white boxes are reconstructed events. b) When asked to recall the action represented by O6, Bob's uncertainty of Alice's transport method is represented by a probability graph. (Adapted from [42])

I might have went there by car. I'm not sure."

4. The NPC can also make mistakes that are believable in context, e.g., "I'm sure I used public transport."

5. The model tracks events by level of interest, so it can highlight unusual memories, e.g., "A foreign army was marching through town yesterday, but the rest of the day was just as usual."

6. NPC personality can also impact perception of events, e.g., "You seem to be enjoying the museum" could become "I can see you are scoping out security in preparation for a theft."

A Cascading Hidden Markov Model (CHMM) [7] is used to store the framework, which was chosen for its simple computation cost and ease of use. An agent stores their observations in short-term memory and then at a fixed point, e.g., the end of the day, these *mems* are encoded as episodes of the whole experience. While being encoded for storage the *mems* can undergo optional time decayed forgetting, where any that fall under a given threshold can be 'forgotten', and thus deleted. The memories can be recalled using cues, e.g., a specific time of day, where stored memories are then reconstructed, possibly misremembered, for further use.

DyBaNeM claims reasonable computation time requirements for domains of moderate

complexity, i.e., background characters in Massive Multiplayer Online Role Playing Games (MMORPGs). In addition, it has an initialisation period that can be executed offline to allow the agents time to build up their daily routines, meaning that simulations can start with agents having prior knowledge of the environment with already established behaviours. DyBaNeM, however, provides no suggested initial parameters that would create human-like behaviours, leaving it up to designers to discover good parameters for themselves [42].

In addition to the DyBaNeM framework, further research has been done into the kind of episodic structure that humans best respond to. This uncovered that general, fuzzy categories such as "morning", "after lunch", and "at night" are more believable than specific time frames, or overly detailed events. People use "socially established" time patterns, not specific clock times, and human users do not ask virtual characters for detailed depictions of their personal concerns, unless it is directly relevant to their current interests. As such, the detail that needs to be stored in episodic memory can be high-level, while still serving the same function [9].

2.6 Embodied Cognition

The CENSYS model takes a different approach towards cognitive agents, with a more detailed framework that attempts to model the human relationship between the mind and the body. Arguing that the generation of behaviours is a shared process between the mind and body, CENSYS creates individual modules which are neither body nor mind to allow for more flexibility and control [69].

Motivated by the belief that an intelligent agent is one who complies with both the physical and social rules of its environment, with the exploitation of those rules creating diverse, believable behaviour. As an agent's interaction with the environment is achieved through some virtual body, the CENSYS model believes that intelligence cannot be entirely dissociated from the body, which it is in most other models. The CENSYS approach aims to improve believability by considering the mind and body as a continuum, with the mind being a collection of cognitive processes specialised for specific functions [69].

Current approaches usually model the mind as a centralised decision-making system,

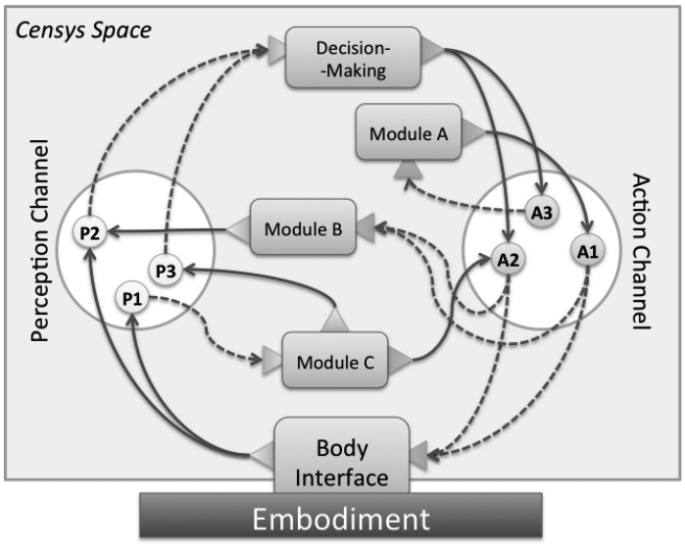

Figure 2.9: An example of a Censys agent, consisting of interfaces, modules, a perception channel and an action channel. (Adapted from [69])

which has to cope with all levels of control simultaneously, from low-level sensor and effectors to high level cognitive tasks such as reasoning and decision making. CENSYS provides a way of separating this traditional approach such that the mind can more believably emulate conscious and subconscious thoughts. Since humans subconsciously recall items of importance, and since movement is not a conscious act where each individual muscle is moved in turn, CENSYS presents a model for representing the human subconscious [69].

CENSYS is modelled as a distributed network of Modules, shown in Figure 2.6, which can have any non-zero number of connectors. Conceptually, a module is a black-box which reacts to data received on its sensors. A module can be one of four types:

- *Perception Sensor* - Subscribes to and receives perceptions of a type T.
- *Perception Effector* - Generates perceptions of a type T.
- *Action Sensors* - Subscribes to and receives actions of a type T
- *Action Effector* - Generates actions of a type T.

The main advantage of this type of architecture is that the mind does not explicitly need to know how to communicate with the body, enabling a layered approach to decision-making and cognitive behaviour. This also allows for modules to be swapped

in and out as agents need them, without any significant rewrite being needed.

CENSYS allows for various functionality to be easily implemented in an intuitive way, such as in the scenario of a virtual robot character moving through an environment. When the robot hears a noise, one module senses the noise and processes it as a potential speech command, while another module turns the robot to look towards the sound to give it more human-like behaviour. A third module can handle movement, allowing for agents to move, gaze and process input simultaneously without any complex branching requirements. Taking the role of the "body" away from agents allows the mind to be refocused on only the thought processes a human would traditionally deal with, such as memories, thoughts and emotional responses. This combination can also lead to a greater degree of believability to be implicitly programmed as emergent interactions between modules [69].

However, the distributed nature of the modules means that concurrency could become a potential concern, and a solid design framework would be necessary to ensure that modules are easy to design and change as needed.

2.7 Self-Organised Cognition

Three key requirements of realistic agents in virtual worlds can be identified as autonomy, interactivity, and personification [44]. The self-organising model attempts to provide this by proposing a brain inspired agent architecture that integrates goal-directed autonomy, natural language interaction and human-like personification [43].

Many other approaches have attempted to model dynamic environments and a user's immediate context, however they typically ignore a significant component - making the virtual world experience more intense and personal for players. This is due to the fact that they usually ignore the capability for agents to adapt over time to the environment, and to the habits and eccentricities of each player. This approach is difficult, because learning in a virtual world is unsupervised, and without an explicit teacher to guide the agent's growth. The paper presents a solution to this problem in the form of a self-organising neural model named FALCON-X. This model integrates some other previous architectures, including ACT-R [2] and fusion ART [81], with the design being motivated by the neural anatomy of human brains.

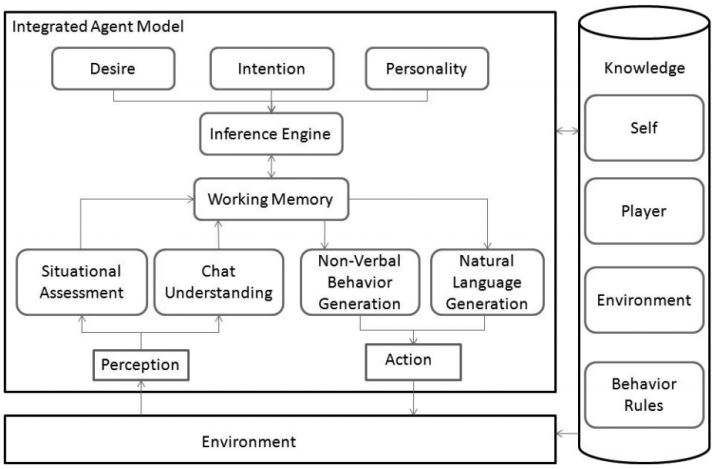

Figure 2.10: A schematic of the integrated agent model. (Adapted from [43])

Autonomy is achieved through the Inference Engine, which is based upon fusion ART, and performs a myriad of cognitive functions, including recognition, prediction and learning, in response to a continual stream of input. This allows agents to make decisions not only based on situational factors, but also from mental states characterised by desire, intention and personality.

FALCON-X adopts the Five Factor Model (FFM) [55] for improving the believability of agents. This model characterises personality in five trait dimensions, and by weighing each dimension a unique personality can be formed from the combination of all traits. It was found that players respond well to open and extrovert agents, as they are proactive and approach the player to make conversation and offer assistance.

FALCON-X has been extensively tested, with one user test presenting three versions of an environment each with a distinct type of NPC. The first environment was a control environment, where NPCs are only able to display static messages and cannot interact with users. The second environment contained interactive NPCs designed for conversation using the Artificial Intelligence Mark-Up Language (AIML). AIML is a XML-compliant language, and is considered as a rule based repository of knowledge on to which the engine performs pattern-matching to select the most appropriate output for a given input. The third environment was similar to the second, but used the paper's model to power the NPCs instead of the AIML pattern-matching approach. An

intuitive user interface is provided for interaction between players and agents, through which a player make ask questions and provide responses through button click. The agent builds an internal model of the player during these interactions, which allows for the agent to adapt to each specific player [43].

A user study tasked players with exploring the environment, solving riddles to find five check-points hidden within the world. The performance of the virtual humans was rated in five key areas, which could then be compared to see which approach was most effective [43].

The environment populated with the FALCON-X NPCs performed best, with users completing their quest in a significantly shorter time than the other two environments. When it came to analysing the other key areas, the third environment was also the highest rated for all but one, enjoyment. The paper suggests that this is due to the fact that the users exploring the second environment had the most experience with game environments, and so were naturally predisposed to having a more enjoyable time [43].

Using data analysis techniques it was also uncovered that the flow experience (i.e., how smoothly the game went) was significantly stronger in the third environment, likely as a result of the heightened NPC interaction [43].

2.8 Inference Agents

Cognitive agents can be used to generate a "theory of mind" (ToM) [87] of a human player and to use this approximated ToM to choose the best responses during interaction. One such area of research is to see if a simplified ToM is worth considering as it is less computationally expensive, so there might be trade-offs that can be successfully made between accuracy and efficiency. Such research even suggests that the formation of an incorrect ToM is not inherently damaging, and can actually prove to be beneficial. As such, cognitive agents which make decisions without having all the correct facts can actually appear *more* believable than those that do.

Agents need a theory of mind to successfully interact with people [37], and so a common approach is for an agent to use their own reasoning mechanisms as a model for the reasoning of others. The main challenge of such an implementation is to overcome the unavoidable uncertainty about the complete mental state of others, e.g., not knowing

the true goals of a player, and so the aim is to see what level of uncertainty is acceptable while still retaining believability.

To test a simplified theory of mind, researchers used a war game, based around the negotiation of territory, which pitted a human player against agents with simplified ToMs. The players had to come to an agreement over the percentage split of territory, or they could fight over it instead. This continued until both sides agreed, or one side had successfully won the war and gained all the territory.

240 participants took part in the test, of which each participant took part in four games. In each game they faced an agent using different reasoning models, either Powell [65] or Slantchev [74], but with the same ToM, and against each reasoning model the player would begin with a territorial advantage in one game, and a disadvantage in the other.

The participants had no idea they were competing against an inference agent, while the agent only had an approximated ToM so it could not be certain what a player would do each turn. To this end, it had to decide the best actions to try and win the game.

Although the approximated ToM of the agents caused them to make some strong assumptions, they still performed very well in the games (although worse than an optimal ToM would have done), so the assumptions did not significantly hurt its performance.

Players put a higher value on ending the game through negotiation than other methods as it was considered a peaceful resolution, while the agent had no such concerns. Similarly, it did not adapt its strategies based on the player, and so did not adapt to negotiations with a more stubborn player. The simplifications made to those areas hurt an agent's believability, while those in other areas did not. This suggests that the areas of approximation need to be targeted, e.g., the simplified ToM did not consider some additional "human values" placed upon the game. An unfocused ToM simplification would be detrimental to agent performance, making it necessary to explicitly design the approximations to suit the environments and the role of the NPCs.

2.9 Current Architectures

There is currently an inconclusive debate over whether symbolic or sub-symbolic representations of human cognition are most suitable for computer models. Current cog-

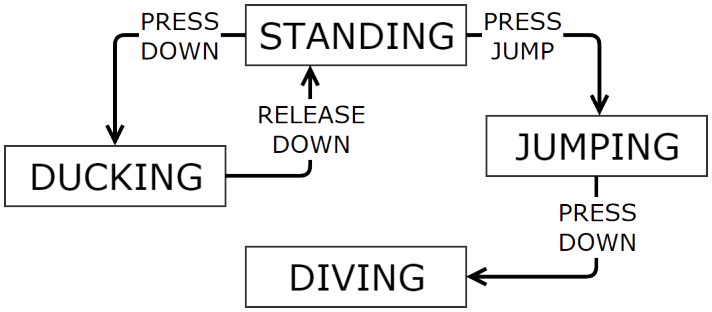

Figure 2.11: A simple Finite State Machine (Sourced from [62])

nitive models ([10], [9], [42], [69], [66], and [50]) all use their own custom architecture and the problem is that no firm answer has been found on which system is used in the human cognitive system, so it is hard to know which system to use for the closest possible emulation in a model. Although a largely unanswered question, Kelley argues for a hybrid approach as it is most effective at realising cognitive models [45].

In addition to a cognitive model, a solid AI architecture is necessary to support the complex models that have been discussed previously. When considering the architecture of AI behaviours, it is wildly regarded that they should be responsive (promptly react to the environment), interruptible (impedible by other behaviours or events), resumable (continue after being interrupted), collaborative (instigate and react to joint behaviour requests), and generative (easily implementable by non-programmers) [19]. Although many architectures fail to satisfy all five requirements.

Finite State Machines (FSMs) are the most common architecture for controlling NPC behaviour [62]. An FSM contains a fixed set of states that a character can be in, such as standing, sitting, running, and jumping. A character can only be within one state at a time, i.e., they cannot be both standing and jumping. A sequence of events are sent to the FSM, and these events might trigger a transition from one state to another state, as shown in Figure 2.11.

However, FSMs are low-level, unorthodox and are limited in their logic, i.e., they cannot handle something as simple as counting by default. In addition, they are not resumable, they do not work well with concurrency, and they scale poorly. FSMs are also not deliberative, making them particularly ill suited towards creating the goal-directed behaviours more suited to believable characters [16].

Stack-based FSMs (SFSMs) and Hierarchical Finite State Machines (HFSMs) attempt to overcome some of these issues, to varying degrees of success, but the industry as a whole is moving away from the entire FSM architecture due to its inherent unwieldiness [16], [62].

Behaviour Trees (BTs) are one alternative to the FSM, however BTs within games are very different from BTs in more traditional AI programming, being closer to HFSMs. BTs focus on increasing the modularity of states by encapsulating logic transparently, for example by using nested states. This is achieved by removing the transitions to external states, making them self-contained. This makes them essentially behaviours, or a collection of related actions. The priority of behaviours is also revealed in a much more natural way than with FSMs, as the tree can simply be traversed with the first valid behaviour being executed, although higher-priority siblings are given the ability to interrupt on subsequent ticks. Halo 2 implements a HFSM for its combat AI, specifically using a Behaviour Directed Acyclic Graph (BDAG). In addition, it uses stimulus behaviours (dynamic nodes placed in the tree based on callbacks) to allow for joint behaviours and context-specific behaviours, although these are not resumable [39]. Halo 3 upgraded their BT to support collaborative behaviours (Collaborative behaviours defines NPC interaction, while the previous joint behaviours represented simultaneous behaviours.), although this is something that is not normally supported, and is actually quite difficult to implement, in BTs [40].

Rule-based systems (or inference engines) are a particular popular architecture used in real-time strategy games. They can be used to encode behavioural rules that capture knowledge about a particular game scenario and the agents that inhabit it. These architectures comprise a database of associated rules, containing conditional program statements with consequent actions that are performed if the specified conditions are satisfied [38].

Goal-Driven Architectures provide an intuitive and cognitively motivated way for thinking about NPC problems. As an abstraction that guides behaviour, goals are often formalised as a partial description of a desired word state, such as ‘GoToTheShip’ or ‘AttackEnemy’. Unlike FSMs, goals are both interruptible and resumable, and they also save their previous state. A goal consists of a plan of how that goal can be achieved, which is represented as a sequence of actions that will achieve that goal. Multiple so-

lutions might be possible, so the goal will attempt to determine the best path. Goals also support re-planning, so if a current plan fails it can attempt to try an alternative plan or it could change its current goal. For example, if an agent's goal is to reach the castle and the entrance is locked, then its current goal would fail and a new goal would be to 'find the key'. Goals can be represented as either Hierarchical plans, where goals consist of sub-goals that decompose the goal into a series of atomic actions where actions only appear in the leaves, Hierarchical Task Networks (HTNs), And/Or Trees, or as a Behaviour Tree [12].

AI Planners are an evolution of the goal-driven architectures, essentially having a top level brain (or planner) which oversees the planning of current goals. STRIPS-based planners work backward from the goal state, searching possible situations to find the best path to get from the current world state to the goal world state. F.E.A.R was the first game to make use of a STRIPS planner, although in recent years there has been a shift away from STRIPS towards more hierarchical approaches, since it did not perform well in situations which were heavily scripted, or games with linear story-driven characters, and behaviours were not resumable. Hierarchical Task Network Planners (HTNPs) [33] work top-down, breaking the current world state down into an eventual sequence of actionable steps that represents the current ideal goal state. The method by which the plan is expanded can vary, although planners inspired by the Simple Hierarchical Ordered Planner (SHOP) are becoming increasingly popular thanks to the success of Killzone 2. Killzone's NPCs received continuous praise from players and critics, which encouraged other developers to switch to a HTNP [17].

DEMIGOD and The Sims employ utility systems, which essentially use a voting / score system that tries to evaluate the benefit of short sequences of actions and select the best one. In the case of The Sims, this utility system evaluation occurs through the object-character interaction loop. Game objects around the NPC signal interactions, and a scoring system causes the sim to make the best choice. The Sims 3 puts more focus on a top-level hierarchy, reducing the utility decisions to a lower level, creating a more consistent goal-driven behaviour that retains the rich object interactions but makes them more purposeful and consistent [17].

2.10 Conclusion

An agent cannot simply be classed as believable if they are emotional, or if they have good memory, or if they are socially aware, or even if they can interact with the environment in an intuitive way. Simply put, an agent can only really be perceived as believable if they can do all those things simultaneously, with every facet of their being feeding in to a complete whole. An agent's memory informs their social interaction, as well as how they perform in the environment. Similarly, interactions create new memories which influence which interactions, and placement in the environment presents new opportunities for interactions and events. This cycle of constant evolution is key to any believable agent, and any memory model must be designed so as to work as part of this constant feedback loop.

To create a new kind of agent it was necessary to cast a wide net of research and sample works in many connected areas, as although the structure of a memory model is key it is information gathered in these other areas that best informs what the memory model should actually be.

Chapter 3

Design

This chapter offers a comprehensive design for the project, addressing the creation of a model which will aim to create more believable behaviours for background characters in games. Starting with the model's high level definition, this chapter then presents some common character tropes that illustrate its potential to improve upon current behavioural approaches. Once a clear understanding has been achieved, an architecture diagram of the model's core components can then be defined. Following this, methods are discussed for using the model to facilitate the procedural generation of content.

3.1 Previous Work

The initial inspiration for this work comes from Fallon's research into a similar area [27]. Although focusing on emotion instead of memory, his work has many parallels with this one. From a design point of view, the processes he used to design his emotional model can be readily applied to the creation of this cognitive one.

3.2 Model Definition

When it comes to making decisions, it can generally be said that a person's current actions are a direct result of their previous ones. Prior experiences, and by the extension the memories associated with those experiences, influence the decisions that a person will make in the future. Considering this, the model should represent memory in a way which allows these prior experiences to be stored, and then queried when it comes time

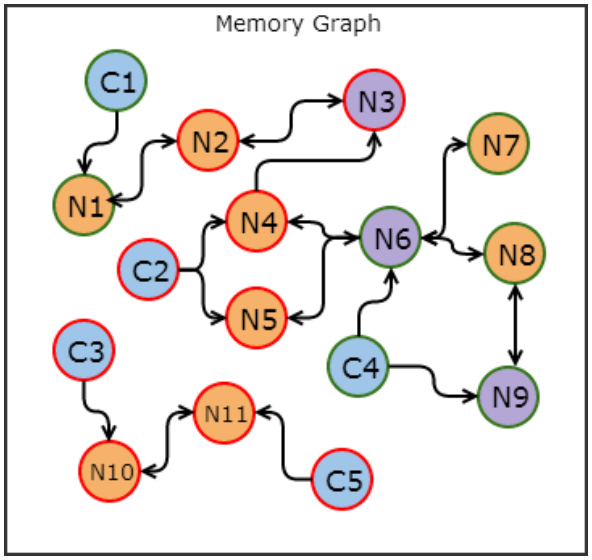

Figure 3.1: Representation of the basic memory graph. C = Concepts / Cues, N = Node. Nodes are generic, but through node flags (represented by colours, e.g., Orange = Item, Purple = Location) different thoughts can be captured. A green outline represents an active cue or node, while red represents an inactive one.

to make future choices. This is made possible through a directional graph, a tried and tested method for memory storage, with nodes on the graph representing individual thoughts. Unlike other approaches, the information stored in the nodes will be entirely generic. This might seem chaotic, but in actuality the design's potential arises from how the controlled creation and accessing of these memories allows for meaning to arise from what might at first appear to be nonsense. This is captured in Figure 3.1.

3.2.1 Memory Representation

Within the graph each individual node represents a unique memory concept, i.e., a thought. A node remains generic, so a thought could be about an item, an area, a character, or even a colour. The thoughts that are created by the system are appropriate for what that system requires, so if a game requires the characters to remember colours then they will, and similarly if they need to remember specific items then they can.

The edges between nodes, i.e., memory links, are represented in Figure 1-1 as an arrow, but in reality are actually represented by two factors: Memory Strength and Opinion

Strength. Memory Strength captures how vivid the memory is within the graph, while Opinion Strength captures a context dependant viewpoint of what the agent thinks about that specific edge relationship.

The generic nature of the graph inevitably leads to a lot of different information being stored within it. For example, a node could represent an item, character, or place. To provide the necessary context, a type flag will be contained within each node that tells the system what the memory represents at a high level. The chosen types can reflect the demands of the system, and so a system can have any number of types.

Memory Strength

Memory is represented within the graph in a 0 - 10 range, with 10 representing a very vivid memory and 0 representing an entirely faded one. Although somewhat unrealistic, there is a design need for certain memories to be permanent. This requirement is two-fold: It can capture memories that humans consider to be essentially permanent, i.e., how it is said that nobody forgets how to ride a bike, and it can also capture memories that for design specific reasons cannot be forgotten. For example, if a game's design requires that agents must always remember where they live, then that requirement can be facilitated through the application of a permanently encoded memory. These permanent memories are represented in the system with a Memory Strength of 11.

Section 2.4.1 presented an approach where the storage of conscious and subconscious memories was represented as two distinct graphs, i.e., active and inactive. When memories are in the active graph they are being thought about and strengthened, and when in the inactive graph they are stored and fading. This leads to some repeated, and redundant, data since nodes are temporarily duplicated when in both the active and inactive sets, and adds additional overhead when it comes to checking both the active and inactive graphs, as well as overhead from copying data between the two. This process is simplified by adding an 'active' flag to each node, meaning that both active and inactive memories can be represented within the one graph, allowing for lookups to be centralised and removing the need to copy data.

If memories are not reinforced then they will decay and eventually be forgotten. Memories are forgotten when their Memory Strength falls below the **Forgotten Threshold**,

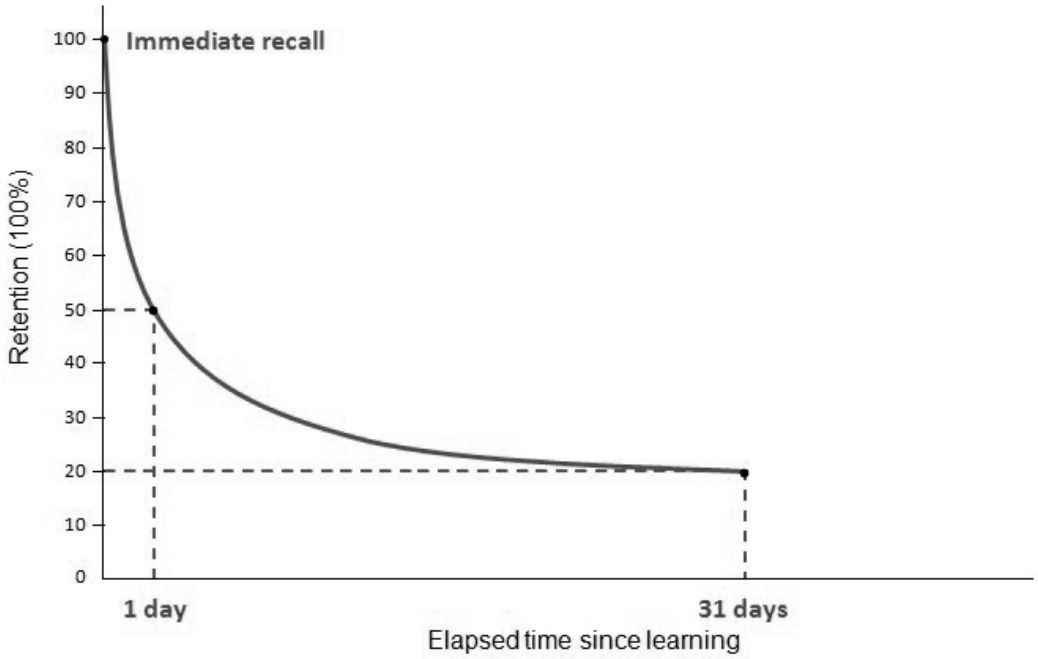

Figure 3.2: Illustration of a Forgetting Curve. (Sourced from [14])

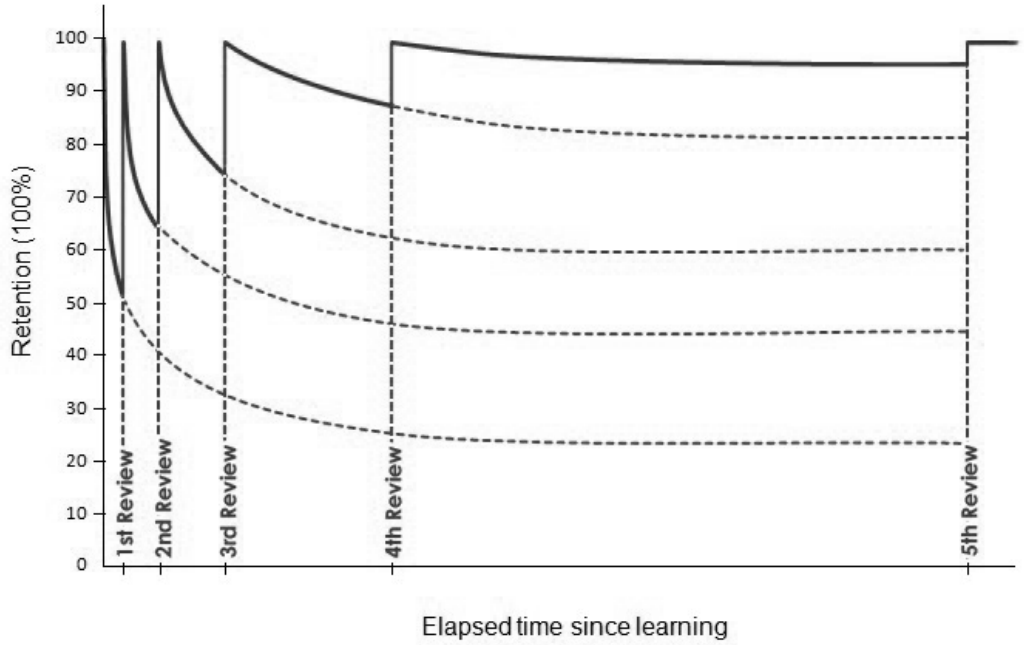

Figure 3.3: Illustration of how the forgetting curves become flatter over the course of the repetition process. (Sourced from [14])

a tunable value that can be changed in real-time to create characters with different types of memory personalities, or that can be temporarily changed as a result of outside effects (e.g., to represent the effects of alcohol).

In addition to memories being forgotten, memories will also get increasingly fuzzy as they decay. When a memory is considered fuzzy then there is a chance that the recollection process will fail and the memory will be remembered incorrectly. In this case certain aspects of the memory might be altered. For example, when considering the location of an item that has not been seen in a long time then an agent might recall it being in a different place to where it actually is. This process is not random, instead the fuzzy process will alter the memory with the values of a related neighbouring concept, so that it still appears credible to the agent. Memories are considered fuzzy when they falls below the **Fuzzy Threshold**, which means they can be categorised in two groups: **Vivid Memories** and **Fuzzy Memories**. Vivid Memories will always be remembered correctly, but the likelihood of a Fuzzy Memory being altered will increase as the Memory Strength falls. This process follows a very simple probability:

$$FuzzyProbability = 1 - \frac{MemoryStrength}{FuzzyThreshold} \tag{3.1}$$

So given a Memory Strength of 4 and a Fuzzy Threshold of 5, a memory would have a 20% of being remembered incorrectly.

Human memories do not decay linearly, and so greater control of memory decay can be provided by simulating the human process more closely. Human memories decay according to the Ebbinghaus Forgetting Curve, shown in Figure 3.2, which resembles that of an exponential curve.

Additionally, the forgetting curve actually changes based on how frequently a human revisits the memory, as shown in Figure 3.3. Considering a player is unlikely to notice this extra detail, and remembering that the focus is on believability, not accuracy, it was decided that only the basic curve would be represented in the model. However to allow for more control the steepness of the curve would be incorporated as a tunable value, allowing for different retention rates to be set on a per-system or per-character basis.

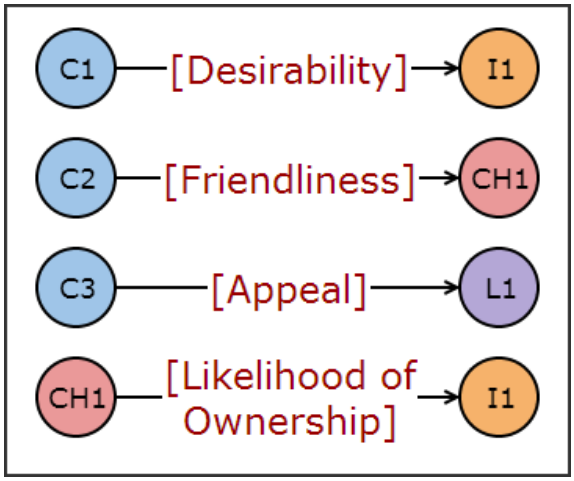

Figure 3.4: Illustration of how different opinions can be captured within 'Opinion Strength'. Edge between a Cue and Item = 'Desirability'. Cue and Character = 'Friendliness', Cue and Location = 'Appeal', Character and Item = 'Likelihood of that character having or knowing about that item'

Opinion Strength

Opinion is represented within a -1.0 to +1.0 range, capturing a 'negative' to 'positive' viewpoint. Other implementations capture the edge relationship in a similar manner to Memory Strength, but by allowing Opinion Strength to be negative creates a more general, practical design that provides greater benefits when compared to other approaches. The choice an agent might make about related memories is not necessarily just the 'strongest positive edge', but it could also be the most negative, most positive, or the opinion closest to 0.0. The neighbours of each node will be sorted based upon their opinion strength, with most positive being first and most negative being last. This means that thoughts which concern the selection of specific neighbours can be simplified to select the either the best or worst case directly in a lot of cases, while still falling back on a search if necessary to select a value in the middle. Since neighbours might not all be of a similar type of memory, i.e., item, place, or person, it will also be possible to filter neighbours to only get the information needed. For example, the most positive neighbour might be a character node when an item node is wanted, and so filtering out all non-item nodes will provide the answer in the right place.

Opinion Strength captures a context dependant view of what the agent thinks of that

specific edge, so the edges between different types of thoughts can represent different opinions. Some examples of potential opinions stored within the opinion strength value are presented in Figure 3.4.

3.2.2 Memory Interface

The graph provides an interface for the agents that consists of a series of functions that allows them to query it and gain the results they want. This keeps the actual graph model separate from any specific implementation, with only the interface needing to be adapted. At a most basic level, the agent will provide the interface with a unique identifier that maps to a node in the graph. This 'cue' node represents the concept the agent is curious about, and the edges, i.e., neighbours, of this node are the specific thoughts the agent remembers about that basic concept. For example, an agent considering "Where is my favourite hammer?" can be represented as a call to the interface to return the most positive edge under the cue node of 'Hammers'. Since the opinion captured by the hammers edge is 'desirability', an agent will desire his favourite hammer the most out of all the ones he can remember.

Of course this query interface will also invisibly handle the other aspects of the memory model, such as the process by which fuzzy memories are misremembered. If the agent has not encountered his favourite hammer in a while, then it is possible the process will return a memory that tells him the hammer is in a different location entirely.

The below pseudocode provides a brief outline of how such an interface could be implemented.

```
memory selectedMemory;

bool SelectBestMemory(string concept, out memory rememberedMemory)
{
       IF memory graph contains nothing related to \textit{concept}
              return false;

       memory memoryConcept = Node that has the ID of \textit{concept};
       rememberedMemory = Node neighbour of memoryConcept that has the most
          positive opinion value.
```

```
        IF rememberedMemory's Memory Strength < Fuzzy Threshold
                rememberedMemory's location = Random location of one of
                    memoryConcept's other neighbours.

        return true;
}
```

3.2.3 Sensing System

A sensing system is necessary to generate the memories that the graph will store, and these memories are generated based on what the agent is perceiving in the environment. This sensing is made possible through the creation of specific environmental 'cues' which are attached to objects of relevance in the environment. These cues could be attached to characters, specific locations, or items. They could also, more generally, be attached to specific events.

A virtual sensor attached to each agent would then identify these environmental cues once they enter the agent's sensory radius. As a simplification, the agent does not need to specifically see the item, with a spherical radius around the agent instead being used in place of line of sight checks. A cue provides information about the item, to help guide how appropriate memories should be generated. Generally, an environmental cue will contain information about the object itself, as well as the unique identifier of the concepts it is associated with.

When a cue crosses the threshold of this sensor it is immediately passed to the memory graph. If the memory is new then the graph creates it, while if it previously existed then it is retrieved and updated. Any concept associations are then made in the same way.

This process also marks the associated memories of the cue as active, i.e., being thought about by the agent, and so each subsequent update of the graph will strengthen the active memories. In addition, the observations made by the sensor will be used to create associations and opinions about the memory node. For example, while the memory of an item is active the sensor will be creating an associated link between the item

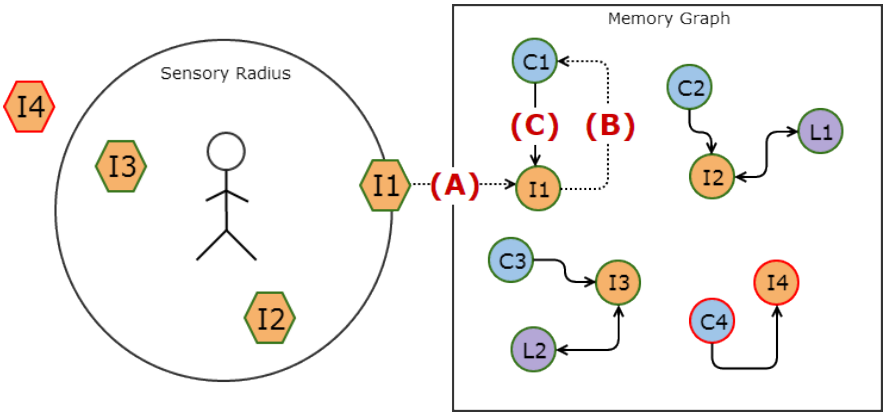

Figure 3.5: Representation of the Sensory Process. I = Items, C = Concepts / Cues, L = Locations. A green outline represents an active cue or node, while red represents an inactive one.

and the location it is being observed. It will also be evaluating the item, to create an opinion of it. If the item is new, shiny and useful then this could be quite a positive one, but if it is a broken, discarded item then it is likely to be a negative one.

At some point, the environmental cue will leave the sensory area. At this point the cue is again passed to the memory system, where it performs a final check to make sure the information stored in the node matches the object's current state. One last update of the memory is performed, and then the memory is marked as inactive. This forced update ensures that memories are always kept correct even if the model is not updated every single frame. This allows for larger time steps between model updates, while still allowing cues to enter and exit the system freely.

The first three steps of the sensing process are shown in Figure 3.5. At **(A)** a new environmental cue has entered the sensory radius, and so is passed to the memory graph where an associated memory is created. At **(B)** this process then makes the necessary associations by creating the high-level concept that represents this specific item. At **(C)** a relationship is made between the high-level concept and the specific item, so that an agent can think of the item by considering the concept. This edge will also capture the agent's initial opinion about the item. The diagram also shows how the three items within the sensory radius are considered active (outlined in green) within the graph, so are being strengthened, while the fourth item has left the radius

and so is inactive (outlined in red).

3.3 Illustrating the Model

The vast majority of games create characters from a pre-set criteria of personalities, or 'classes'. These character bases are made unique through the personalisation of their appearance, voice, habits, etc. but at a fundamental level two characters from the same class perform the same actions. A class can encompass any group of characters that share a similar core set of functions, for example an archer class, a beggar class, a shopkeeper class, and so on. The list of potential archetypes is almost endless, and will vary greatly from game to game, but the same design rules can be applied in each case.

In most games, these character classes follow the same scripts, with the difference in their behaviour coming about simply due to the differences in their respective environments. Two archers might eat different food in different places but only because the environment at the time makes do so, with the actual scripts that trigger how and when they eat being identical. As such, scripting can lead to repetitive behaviour unless the game has an incredibly diverse number of base character types.

To consider how memory can help diversify character types, as well as to identify what memories they need to store, some potential archetypes have been created.

3.3.1 The Blacksmith Type

The daily routine for a Blacksmith is quite simple. They wake in the morning, collect their hammer, and then head to work. As part of their work, they might occasionally need to go and collect new materials. At some point they are likely to go and eat, and in the evening they will relax with a hobby, such as fishing or drinking. Finally they will retiring to bed and repeat the entire routine the next day. Therefore the actions can be identified as follows:

- Sleep
- Collect Tools
- Go to Work
- Collect Materials

- Eat Food
- Go Fishing
- Go Drinking

Since 'waking up' is not actually an action, instead being a result of the sleeping action ending, it has been omitted from the above list. These can be further expanded into a series of atomic steps.

- Sleep
  - Go to Bed
    - Think of Bed Location
    - Walk to Bed Location
  - Sleep in Bed
    - Play Bed Animation
    - Play Sleeping Animation

So to go to sleep, an agent must first go to bed. To go to bed, they must access the location of their bed, and then walk to it. Once they have arrived, they must sleep in it. To do this, they can play an animation where they climb into bed, and then one in which they sleep.

This provides the first memory - *Agents must recall where they sleep.*

Applying this same breakdown to the other actions can provides the full suite of places and items a Blacksmith would need to remember to accurately do his job. These include:

- Bed
- Hammers
- Work
- Collected Materials
- Mining Spot to gather new materials / Shop to purchase new materials
- Food
- Eating Locations
- Fishing Rod / Alcohol
- Fishing Spots / Pubs ...

If it is assumed that all Blacksmiths are going to select the 'Most Positive' tool they can think of, then the value that is captured by each agent in this regard can provide greater variety. For example, consider the difference between a Frugal Blacksmith and a Perfectionist Blacksmith. They both make and store opinions about the tools they see in the world, but the Frugal one likes to save money and so puts a high value on using tools he already owns, even if they are old and worn. On the other hand, the perfectionist always wants the newest and best items, so puts a high value on owning the best type of tool available for the job.

Simply by querying their memories, the most tool for the frugal blacksmith can be the one he always uses, while the one for a perfectionist might be one they can buy new in a shop.

3.3.2 The Archer Type

Gathering classes are particularly popular in many role playing games, being the character players are most likely to encounter out in the wilderness. These characters usually patrol the areas outside of towns, capturing and killing monsters to simulate a hunter / gatherer role. The daily routine for an archer might be as follows: He awakens very early in the morning, and collects his bow and arrows. He then decides the best place to hunt for the day, perhaps based on external factors that he remembers, like where he last saw a large monster. He travels to this point and begins searching for animals. He collects a reward from each animal he kills, i.e., a pelt, and in the evening he returns home and sells his loot at the shop. He then might retire to his home and use some nearby wood to create more arrows.

Again, what the archer remembers helps to create a unique character. For example, he is going to explore the area outside of the town and begin identifying potential hunting locations. As he uncovers these locations, he will make an association between each location and the animals / plants he can find there. If he is searching for a specific item, he will head to the place it was last spotted. Similarly, if he found a particularly abundant hunting spot he will remember the next day that there was still numerous animals left there, and so will return to it. Essentially, applying memories to the archer can give him habits and power his with behaviours with genuine purpose.

A different archer, even in the same area, will develop their own habits, and might even

develop memories of areas to avoid simply because it puts him in competition with the first archer. This behaviour would be difficult to explicitly script, but can arise quite naturally from the interactions of simply memory types.

3.3.3 General Memories

The above designs feature two specific, and quite different, archetypes and so the memories that each must store initially appear quite different. However, they can be broadly summarised as belonging to three categories: Locations, Items, and Characters.

By considering the characters (animals and monsters are also considered to be 'characters') associated with a specific location, then an archer knows where best to hunt. Considering the location most strongly associated with a hammer tells the Blacksmith the best place to find it. Similarly, if one archer considers the locations most associated with another archer then he knows where to avoid if he wants to hunt by himself. As such, the associations between these three memory types can drive all of the previous behaviour.

However, more broadly, the design has implicit requirements that are not directly addressed by looking at one specific character type. For example, what if the Blacksmith is not aware of any hammers, or when he reaches his hammer it is no longer there? In the latter scenario he could update his hammer memory to indicate the location is unknown, but this still leaves him without a hammer. The solutions to this type of problem are not specific to any character type, and so are better defined broadly as part of the model.

If the above scenario will break the game design, then the obvious solution is to ensure there is no possibility for the Blacksmith to find himself in such a situation. If he is provided at run-time with a permanently encoded memory (i.e., Memory Strength = 11) of a hammer, then he will always know where to go. Similarly, if the hammer is created in such a way that only he can interact with it, then he will never find that someone else has taken it. This solution is less than ideal, since it forces the model to behave more like the scripted behaviours it is trying to surpass, but it does show how the model can be adapted for specific design limitations.

A much more suitable outcome would be achieved if the blacksmith could use additional

memories to solve this problem for himself. As such, behaviours which operate on the model should have fall-back scenarios for when the graph does not contain any supported memories for the initial query. For example, if he is not aware of a specific hammer then he could query his memories for a shop and visit it to see if they sell one. He could also consider locations where other tools are located, to see if he can find a hammer nearby. He might even query the other characters he knows, and by visiting them he could enquire if they own a hammer, or if they know where he could find one. Essentially, the design of the memory model provides agents with a way of self-resolving their own issues in a method not possible through conventional scripting, and so any behaviours should take advantage of this process. This method of self-resolution also helps prevent repetitive behaviour from becoming common place.

Combining the model with traditional behaviours, such as wandering, patrolling and searching, allows them to effectively use the thought of a specific 'location' to then fully search that location in a believable way to find an item or character they are searching for.

Considering the model is designed for interactive games, it makes it possible to include the player as part of this process. Consider the worst case scenario, where the agent has exhausted all the possible avenues of self-resolution suggested above. In this case, an agent could simply accept that they cannot resolve their own problem. For example, they might go to the Town Square and post up a notice that says "I'll pay anyone 100 gold coins if they can find me a hammer", and then perform another activity in the meantime. The player could see this notice and decide to undertake the agent's request by collecting a hammer and giving it to them. The agent is now able to resolve his problem, the player has been rewarded, and as an added bonus the dynamic interaction will come across as being quite believable.

The specifics of any such implementation would depend on the systems used for players, quests, character etc. so will be expanded upon further in the next chapter.

3.4 Event Memories

It is easy to consider how physical concepts can be captured in the graph, since they exist within the world and can be detected by the sensory system as an agent moves

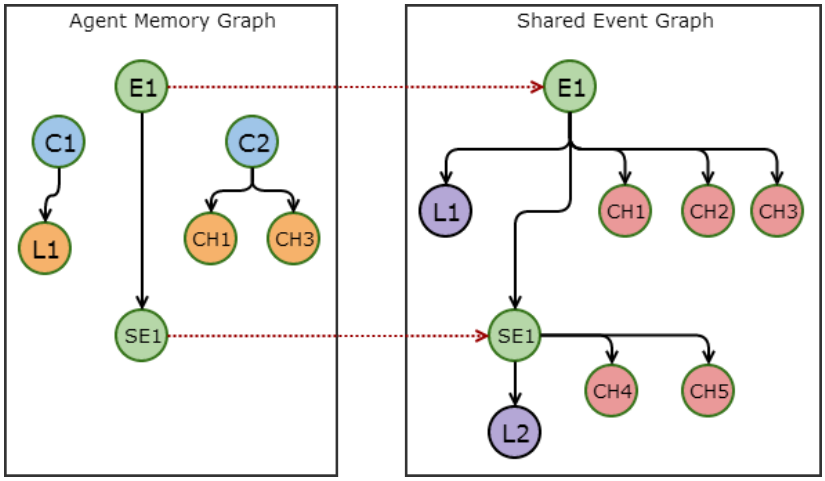

Figure 3.6: Design of the Global Events system. An event (E1) occurs at a location (L1) and involves three monsters (CH1, CH2, CH3). The agent takes part in the event, and gains a pointer memory in their own graph so they are aware of the event. In addition, they form a personal opinion about the location the event took place (C1 -¿ E1), as well as an opinion on the two monsters they saw (C2 -¿ CH1, C2 -¿ CH3) there. Finally, the monsters called for backup creating a sub-event as part of the main event (SE1) which the agent was involved in. However they had started to run away, so have no specific knowledge of where (L2) or what (CH4, CH5) was involved in this sub-event.

through the environment, but capturing memories related to specific 'events' proves to be more abstract. For example, if a dragon was to attack a village the environmental cues would automatically provide information about the dragon and where he was, but not necessarily what he was doing. To capture this extra level of detail that would allow an agent to say that a dragon 'attacked', instead of just thinking 'there was a dragon', requires a different approach.

Events can be considered global as they impact a large number of agents within a given area. This could be within a single town, but it could also be across the entire world within the game. To that end, the event system needs to be capable of handling any number of agents. Considering this, having each agent remember all the details of an event seems wasteful, as lots of potentially repetitive data would be stored in each memory graph. Since events are global, then so too should event memories be global. Essentially, each agent will have their own graph of personal memories as well as access to a shared 'event graph'. An agent will store event memories locally, but

these memories are simply pointers to the global information about the event in the event graph. That way information can be shared without being repeated.

Of course no two agents are going to see an event from the exact same viewpoint, so additional detail has to be provided in the way that agents can access global event info while still storing local, personalised thoughts about the event. For example, in the above dragon example, everyone is aware that the dragon attacked the village, but only one agent locally stores the thought that "'the dragon burned down my house"'. Such functionality also allows players to question agents about what they saw to find out what happened, although no two agents might necessarily agree on the finer details of an event. This global 'shared event' process is presented in Figure 3.6.

All games have an event system of some kind, to control specific gameplay behaviours, so centralising the event graph within this system seems like a natural approach. The systems that generate the events can then be expanded to send messages to agents involved in events, allowing them to create their own event memory pointers as well as any personalised memories they want to remember.

3.5 Goal-Driven Behaviours

A goal-driven system, such as the one presented by Buckland [12], mainly differs from more traditional state-based agents through its stack-based hierarchical architecture. This allows for context specific goals to be updated in real-time by pushing more pressing goals on to the stack. Goals can be either atomic or composite. Atomic goals define a single action, such as 'pick up item' or 'move to position', while composite goals define more complex tasks, such as 'buy a weapon' or 'build a house'. Composite goals are achieved through the application of a series of atomic goals, such as 'buy a weapon' being achieved by 'walk to shop', 'talk to shopkeeper', and 'buy sword'.

A goal-driven behaviour system captures the ability for humans to not think about something until absolutely necessary. For example, if a person decides to go to the shop they do not explicitly consider the need to open the door until they reach it. In a similar way, the goal 'go to the shop' would be pushed on to the goal stack, then expanded, and then a 'open door' goal would be pushed as a sub-goal of the main goal, making it the current active goal. This process of decomposing and satisfying goals

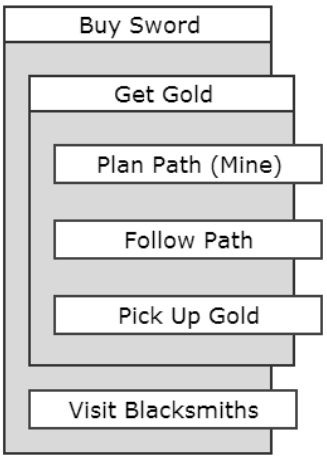

Figure 3.7: An example of goal driven behaviours, showing how the goals are expanded in a sequential manner based on the hierarchy.

by pushing and popping from the stack is repeated until the entire hierarchy has been traversed and the agent is fully satisfied.

The goal system works on a stack based hierarchy, consisting of both atomic and composite goals. A goal has a status (e.g., active, inactive, complete, or failed) which tracks its current progress in achieving its aim. In addition, composite goals contain another stack of sub-goals that must be achieved before it can be considered complete. Specific goals, such as 'Fishing', 'Patrolling', or 'Collecting Item', can be defined on a case by case basis, depending on the needs of the implementation.

When considering goal-driven systems, the top-level goal is usually a never ending 'brain' goal that drives the agents behaviour. This is able to set the agent's current main task (i.e., work, sleep, eat, fish, etc.) based on a daily routine that can be set for each agent. This allows for agent's to be easily personalised by providing them with their own unique routines. For example, the routine might list 'Fishing' as starting at 8pm, so when the in-game time matches this the scheduler goal updates the agent's current high-level goal, which previously could have been 'Working at the Shop', to make the agent think about the things he needs to do to fish instead.

3.5.1 Example Scenarios

Goal-driven behaviours provide a way for the memory model to drive behaviour that would not be easily possible using other behavioural systems. The ability to push sub-goals dynamically allows for them to be decomposed according to an agent's thoughts. This is best demonstrated through some example scenarios that consider how goals are created in response to thoughts.

The high-level goal of Blacksmith has three sub-goals: 'Get a Hammer', 'Walk to Work', and 'Be a Blacksmith'. An agent will work towards these goals in sequential order, but how he does so will depend on his memory graph.

Scenario 1 - Hammer in Inventory

When the first sub-goal becomes active, the agent will first check if he has any memory of a hammer. In this scenario he does, and its in his inventory. As such, this goal pushes the sub-goal of'Equip Hammer', which will play whatever animations are necessary to put the hammer in the agent's hand. This goal then completes, and the agent proceeds to 'Walk to Work'.

Scenario 2 - Hammer in Environment

This time the agent knows of a hammer sitting at a location in the environment. The sub-goals that will be pushed are 'Walk to Hammer', and then 'Pickup Hammer'. The 'Pickup Hammer' goal will check if the hammer is still in the area, and make any memory updates that are necessary. It is possible that it could fail if the hammer has moved since the agent last spotted it, and in that case the 'Get a Hammer' goal will send him towards another hammer he knows.

Scenario 3 - No Hammer

A tricky scenario presents itself if the agent is told to get a hammer but has no memory of ever encountering one. In this case, the sub-goal becomes 'Ask about a Hammer' and the agent will check their memory to see which other characters they are aware of who might be able to share information that could get them a hammer. The agent will select the most appropriate candidate, perhaps based on friendliness, and then push the 'Look for Person' and 'Ask Person about Hammer' goals. Once they have talked to the person the sub-goal is complete, so the 'Get a Hammer' goal will catch

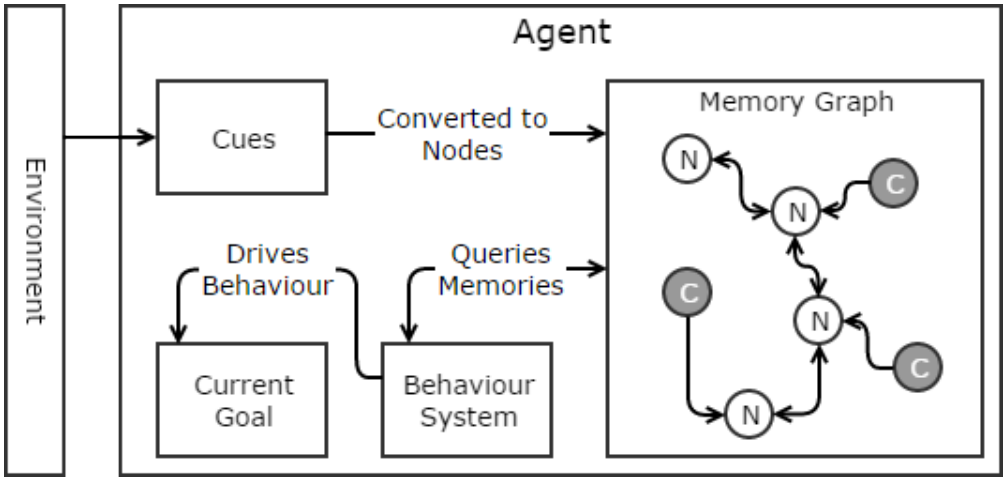

Figure 3.8: Representation of the Agent's Cognitive Architecture

this complete message and reset itself, checking to see if the agent is now aware of a hammer.

3.5.2 Fallback Scenarios

The ability for goals to gracefully fall back and expand is what allows the memory model to impact them in such a large way. For example, if Scenario 2 was to end with the agent exhausting all possible hammer locations, then the goals mentioned in Scenario 3 would automatically be pushed. Similarly, once Scenario 3 is finished, if the agent knows about a hammer then they would proceed to push the goals in Scenario 2.

3.6 Model Architecture

The high-level architecture of the model can be defined based on the requirements identified above, and is shown in Figure 3.8. Environmental Cues are passed in to the Agent via its sensor, where they are converted from Memory Cues into Memory Nodes and passed to the graph, where new nodes are added and pre-existing nodes are updated. Alongside this, an agent has a state machine, or some similar system driving its behaviour, which has a current state or goal that it is trying to achieve. This current state is, through the state machine, querying the memory graph about what the agent knows, and the results are then changing and driving the agent's current behavioural

state.

Throughout this process the information being provided by the environmental cues are marking relevant nodes as active or inactive, and the memory graph's update process is strengthening or decay nodes at frequent intervals.

One of the core objectives outlined in the introduction was to create an efficient model, and one way this can be achieved is by ensuring that tunable parameters are a central part of the model's architecture. Any value which is used as a factor in the model, e.g., update frequency, the decay rate of memories, the decay curve steepness, etc. is designed as a separate, individual parameter, never as a hard-coded constraint. If one implementation has a strong memory limitation then the overall memory requirements can be reduced by increasing the memory decay rate. Similarly, increasing the decay rate will create characters with poor recall. As such, the tunable values can be set system wide to tweak overall performance and individually for each agent to define their cognitive personality.

3.7 Procedural Content Generation

Procedural content defines any content in a game that was generated automatically, rather than manually. This is most commonly in the form of levels that are randomly created at run-time. From the point of view of game design, it can also refer to the process by which content is automatically generated on behalf of a designer. For example, rather than explicitly creating every single aspect of a character, a system can generate one given a set of high-level parameters. This process speeds up development and allows for a greater variety of content, and avoids the overheads normally associated with manually creating every single aspect of a game. As well as saving time, such systems also reduce the overall number of bugs and testing required, since issues can be traced directly to the generator itself.

For this project the procedural generation comes from the way that a character's memories drive their behaviour. Changing their memories changes their behaviour, so designers only need to consider what memories to give each character without manually needing to assign every individual behaviour. Additionally, interactions between these generated behaviour systems can create emergent gameplay that the designer might

not have explicitly designed, which adds extra layers of believability to the system.

There are two main methods by which memories can be created by a designer.

3.7.1 Manual Memory Creation

This requires support from the editor being used to implement the model, but allows for a designer to automatically assign environmental cues to agents. Essentially, a drag and drop interface could allow for cues to be placed in an agent's initialisation panel. At run-time, the cues that have been placed are instantly passed to the memory graph, creating initial memories that an agent can use to power their starting behaviour. This interface would allow for a designer to quickly assign appropriate memories to agents, to guide them along the desired path. It also prevents the scenario where agents initially perform poorly because they have no memory of the environment.

3.7.2 Simulated Memory Creation

Memories can be simulated offline by running the game for a certain period of time and saving the memory state for later use. This allows the agents to accrue memories in real-time, and when the simulation is run again these memories can be used as the initial memories, much like the manual creation above. Combining manual creation to start a simulation, and then using offline saving once the agents have spent time simulating the world provides a very easy way to quickly provide the agents with a complex set of behaviours right from the start.

3.8 Conclusion

The high level design for the model has been presented, showing how the initial idea of a memory graph was tailored towards the needs of the model. Using some example characters to define the requirements for the core memory nodes, as well as example gameplay sequences to provide scope for the more abstract event memories, allowed for the creation of a model well suited for implementation. The model's architecture diagram provides a simple look at how it could potentially be implemented in an agent, without it being necessary to change the underlying animation, movement or general control systems.

Chapter 4

Implementation

This chapter presents the implementation of the proposed prototype, illustrating the relevant technical details and the challenges faced. The choice of platform is discussed first, before launching into an in-depth breakdown of how the model was implemented based on the design specification.

4.1 Platform Review

Two different platforms were used during the creation of this project. The first turned out to be unsuitable, and so the project had to be moved to a more suitable platform during development. The reasons for this switch, and its outcome, are briefly discussed below.

4.1.1 The Elder Scrolls Creation Kit

The project was inspired by upon John Fallon's research, and so following this it was first proposed that the model be implemented within The Elder Scrolls V: Skyrim [80]. The Creation Kit (CK) is the name for a set of tools that allows a user to create custom content, i.e., modifications, for the Skyrim game. Users can use the Creation Kit to create their own characters, stories, and environments, through a scripting language known as Papyrus. In addition, a plug-in known as the Skyrim Script Extender (SKSE) further expands this by enabling modifications to be written in C++ and access the base elements of the Skyrim game engine to change anything they would like.

It was initially believed that it would be possible to use the Creation Kit to create a special version of the Skyrim world that would automatically come pre-populated with a variety of characters, personalities, jobs, and so on that could provide a lot of the foundations that the model would require. It was hoped this would save the time needed to make such an environment from scratch, and allow more time to be spent on developing the model itself. The SKSE extension would then allow for the model to be written in C++, as it required low-level access to core Skyrim systems.

However, development soon ran into problems as the SKSE proved to be more cumbersome than expected, with a troublesome setup process and no documentation. At the same time, it was discovered that the behavioural system used by the characters was much more limiting than it initially appeared, as it consists of a sequence of script packages sitting on a stack which are evaluated at set periods, and the top most package that has valid conditions is then used for the next period (usually an hour of game time). This did not suit the design of dynamic, memory driven behaviours, and so it became apparent that the A.I. systems might need to be rewritten from scratch if the model was ever to be possible in Skyrim.

A final, crippling issue was uncovered when trying to expand the characters interaction capabilities. As mentioned above, the Creation Kit allows users to create their own quests, and this also includes the dialogue that characters can say as part of this quest. Since this was possible, it was believed possible to dynamically determine what characters would say, based on their thoughts. However, it became apparent that all dialogue in Skyrim must be written externally, and cannot be passed to characters from a script. Essentially, dialogue files which contain text and voice data are called by scripts and a sequence of these files can create a conversation, but there is no way to dynamically create these files from within scripts.

As such, it became apparent that creating the model from within Skyrim was not going to be feasible, and time hoped to be gain by using Skyrim's environment would inevitably be lost again when struggling with the other issues.

4.1.2 Unity Game Engine

It was then decided that the prototype would be implemented within the Unity game engine, as its framework was particularly well suited to the quick iteration of ideas [83].

The ability to code at a low level would also prevent the issues identified above from reoccurring. In addition, the asset store provided access to an abundant resource of materials, such as environmental and character models, that, although not strictly necessary for the model, were still needed to create a fully functional prototype.

Plug-ins

The base engine was expanded through the following plug-ins, which were used where possible to expand the project and save on writing unnecessary boilerplate code.

- *NGUI: Next-Gen UI*: A powerful UI system and event notification framework, which replaces (and greatly improves) Unity's default GUI system.
- *Mobile Cartoon GUI*: A collection of image templates for quickly create appealing user interfaces.
- *Medieval Toon Village*: A large suite of models that provide everything necessary for creating a village environment.
- *Villager Boy (NPC Model)*: A model and suite of animations that can be used to bring a character to life.
- *Cartoon Food Pack*: A small selection of food models that can be used as environmental cues.
- *Stone Frog (Enemy Model)*: A mode and suite of animations for an enemy character to drive events.

4.2 Agent Movement

It was necessary to create a navigation mesh that would allow agents to move through the environment. Unity provides a tool for creating these meshes from geometry, but first each object in the environment must be flagged as walkable (i.e., floors), unwalkable (i.e., walls), or ignorable (i.e, roofs, which would otherwise block interior mesh generation).

The **Cognitive Agent** component was created to manage all aspects of an agent's behaviour. This includes the code for interacting with the navigation mesh, including selecting the correct movement animations based on how fast the agent is travelling.

Unity also provides assistance when it comes to navigating this mesh, so object avoidance settings were created to ensure agents could safely move in crowded interior spaces.

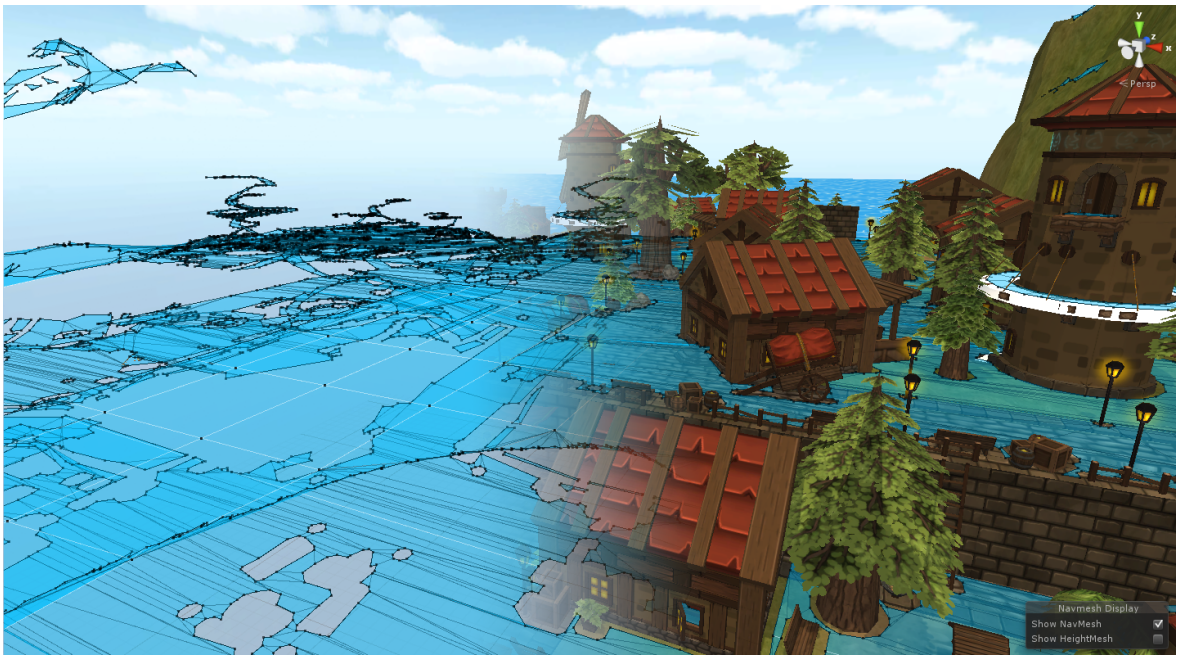

Figure 4.1: The environmental navigation mesh, shown in blue. Left shows only the mesh, while right shows it overlaid against the environment.

Finally, doors and gates were given custom behaviours to open and close as agents approached, so they did not have to worry about interacting with them or getting stuck.

4.3 Environmental Tagging

As stated in Section 3.2.3, an agent's sensor is only concerned with environmental cues, and has no other concept of the world. To that end, a special collision layer was created in Unity that was only occupied by the cues and the agent sensors. This layer system controls the objects that can interact, and so the sensor will only ever detect cues. This abstraction keeps the visual, and even gameplay, environment entirely separate from the cognitive one, meaning that an increase in the visual complexity of the real environment does not impact the model.

4.3.1 Environmental Cues

It was necessary to define the type of cues that would be placed in the environment, and after some consideration those shown in Figure 4.3.1 were selected. The choice

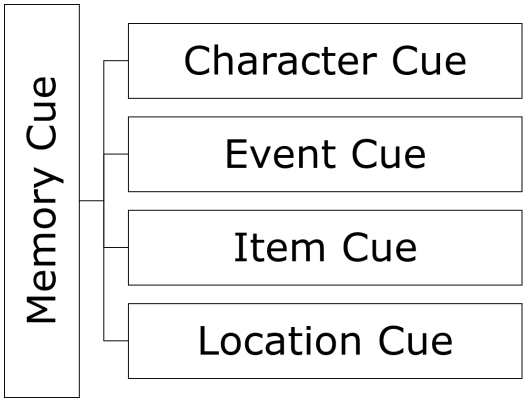

Figure 4.2: The cue types chosen for this implementation.

of cues can be considered implementation specific, but would at least be generalised across similar game genres. For example, the cues chosen here would be useful in most role-playing games, but a stealth game might be better suited to having 'sight' or 'sound' cues that would suit an agent in that environment trying to remember where they last saw or heard a character. In addition, a role-playing game with a multiple character party system might want to have an additional 'party member' cue which contains more personal information than would be stored about background characters. Thankfully, the generic nature of the graph allows for a designer to rethink, adapt, or even add any additional cues they believe they need.

The base Memory Cue contains a type flag, a unique identifier (UID), and a location. The other cues inherit from this base type and add any type specific information they require. For example, Item Cues also contain information about the item's status, if it has an owner, as well as how durable it is.

Appropriate cues can be attached to the items, characters and locations in the environment. Like with the sensors, the cue components are placed on the memory layer, while an invisible sphere collider acts as a trigger to tell the sensor when the cue has entered its radius.

Locations were created in a similar manner, with invisible markers floating above certain areas. When an agent, or environmental cue, enters the area covered by the marker's sphere collider it considers itself to be within that location. This information is stored within the cue to save on each sight event calculating the location itself.

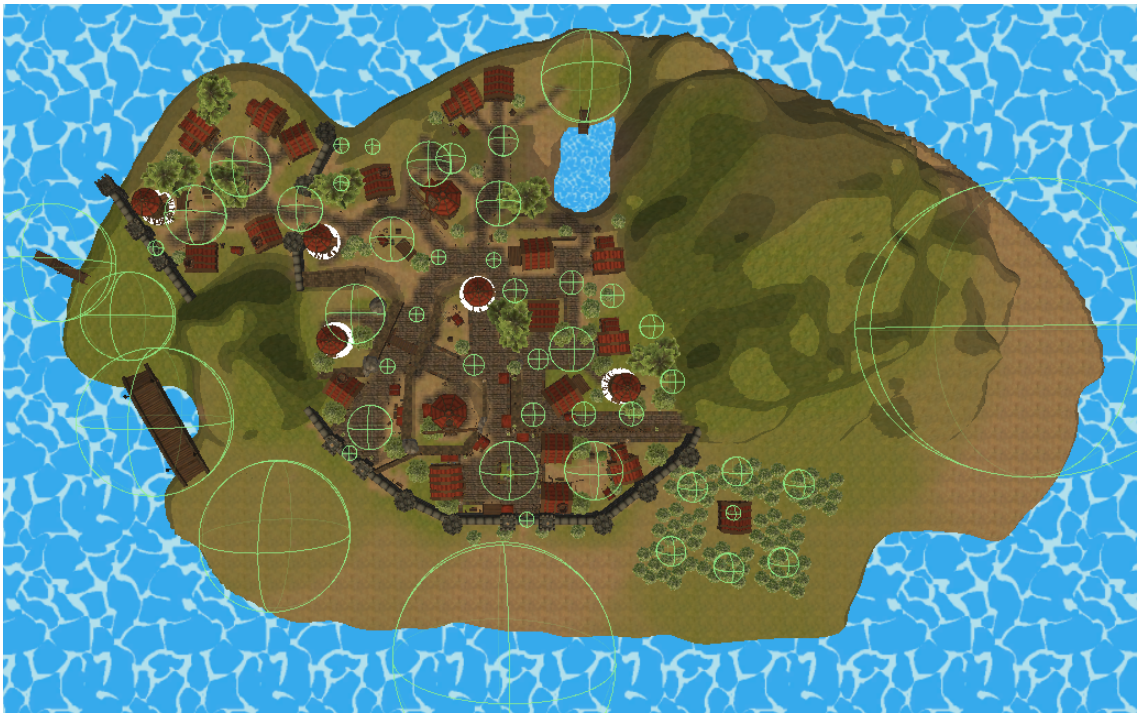

Figure 4.3: The green spheres indicate individual locations within the environment.

Once the cue system was completed it was necessary to apply it to all the relevant objects in the scene, which was quite a time-consuming task considering that over 100 environmental cues were created during the project. Thankfully, the task was made possible by splitting it between two people. Niall Mullally's research into improving emotional models for background characters shares some overlap with this project [60], as he was able to take advantage of the environment created here. Tagging objects in the environment simultaneously assisted both projects, and so sharing environmental updates made this task manageable.

4.4 Memory Representation

Once the environment was completed, and the agent's had rudimentary functionality to enable them to navigate about randomly, it was then possible to start applying the memory model. The design of the memory graph was laid out at a high-level in Section 3.2.1, but more detail concerning how this was achieved in the implementation is presented here.

4.4.1 Memory Node

Section 3.2.1 stated that thoughts are stored within the graph as generic memory nodes, with flags identifying specific thought types that allow for the nodes to be expanded. This is achieved through inheritance, so although the memory graph is full of 'memory nodes', these nodes individually could be 'item nodes', or 'location nodes', and so on, with their own category specific variables. The nodes are memorial representations of the environmental cues, so the node types are the same as those presented in Figure 4.3.1 plus one additional high-level 'Cue' category which captures the base 'concept' as discussed in Section 3.2.2.

Within the Memory Node is an update function which performs the 'thought process' an agent goes through when the nodes are active in their memory. Since the associations that agents make varies between node types, this enables the virtual function to be inherited and tweaked to suit specific needs. For example, when updating an Item Node, the function makes associations between the item and where it is located, but when updating a Character Node, the function is making associations between the character and location, as well as between the character and the items in their inventory (so they can ask them about those items later if required). Encapsulating the thought process in this way allows for new node categories to be added easily, without impacting the rest of the implementation.

4.4.2 Memory Graph Node

In addition to the core memory node, which contains the various parameters that constitute a 'thought', a Memory Graph Node (MGN) was created to store the model's graph specific information. A MGN contains one Memory Node, but it also contains a list of neighbours, as well as a list of Memory Strengths and Opinion Strengths. These three lists collectively making up the edge associations that memory node has, as shown in Figure 4.4.

Although not considered in the design, the graph-specific (MGN) and thought-specific (Memory Node) requirements were split to ease the process by which thoughts can be shared. When two agents are communicating the inquiring agent only cares about the Memory Node information, not the MGN information, so an agent can simply create their own MGN around a copy of someone else's Memory Node.

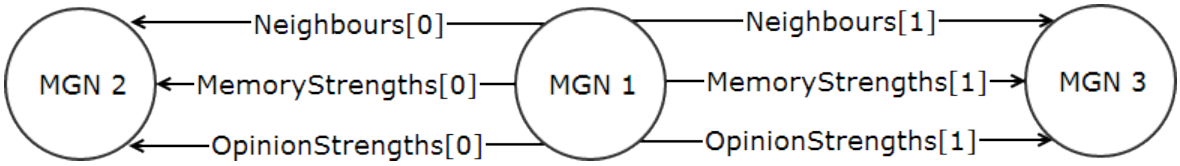

Figure 4.4: Representation of how Neighbours, Memory Strength, and Opinion Strength lists collectively make up one edge association. Edge numbers map directly to elements within each list. The association between MGN 1 → MGN 2 is Edge [0], while MGN 1 → MGN 3 is Edge [1].

The core functionality that drives the model's behaviour is also captured within the MGN. An active flag indicates if a memory is being thought about or if it should decay. In addition, the MGN contains various helper functions to provide the following features:

- *Node Activations*: Help quickly mark this node, and any necessary relations, as active or inactive.
- *Edge Associations*: Ensure that adding and removing edges is handled correctly, ensuring the three lists remain in-sync.
- *Concept Retrieval*: Search within the neighbours for a node which matches a specific UID.
- *Memory Binding*: Can permanently or temporarily make memories unforgettable.
- *Memory Strength Control*: Strengthen or decay given edge associations.
- *Opinion Strength Control*: Ensure an agent's opinion of the edge reflects what they observe in the environment.
- *Neighbour Sort*: Neighbours are sorted based on Opinion Strength, so a one-pass Quick Sort[1] is used to keep the edge associations in line

4.4.3 Memory Graph

The Memory Graph is the container for an agent's thoughts, represented as a list of Memory Graph Nodes. This list is linearly iterated during an update, but directly accessing a node within it would be costly. This cost is overcome through the addition

[1]Considered one-pass because it is known if the opinion strength has increased or decreased, so only the one value needs to be moved in either ascending or descending order, allowing the inner loop to be abolished.

of a hashtable (in the form of a C# dictionary), with the key being the node's unique identifier. This points directly to elements within the list, allowing (in most cases) a direct lookup of a specific memory node. Collectively, this list and hashtable represent the graph discussed in Section 3.2.

The graph contains various functionality to seamlessly handle the process by which memories are created, updated, and deleted. When a node no longer has any edge associations it can be considered entirely forgotten. In this case it is deleted, and so any functionality which removes edges will check if the node should also be deleted. When a node is deleted, any cue nodes pointing to it are also updated. If a cue node is left with no associations then it too can be removed.

When a node is created the graph will automatically make the correct concept associations based on the information within the environmental cue. If the cue does not exist then it will be automatically created along with its associations.

This recursive method, by which the graph adds or removes all the nodes it needs, automatically ensures that anything accessing the graph does not need to concern itself with the underlying functionality. A sensor may pass an environmental cue which will then spark multiple node creations, while an agent which breaks a link will not even be aware that the memory graph has automatically tidied up a forgotten node and its cues.

The core functionality of the Memory Graph falls under the following areas:

- *Add Nodes*: Adds a node, creating any cues (and cue associations) needed.
- *Remove Nodes*: Removes a node from the graph, ensuring any nodes pointing to it are also updated. Also removes any nodes that are made edgeless because of this.
- *Add Edge*: Can add a one-way or two-way edge association between nodes.
- *Search*: Uses the hashtable to quickly return a node with specified UID.
- *Share Information*: Copies relevant nodes from another graph, emulating agent's communicating and sharing information.
- *Update Opinions*: Updates the core opinion (between concept and specific node) the agent has about the node.
- *Print*: Prints the graph for debugging and testing purposes.

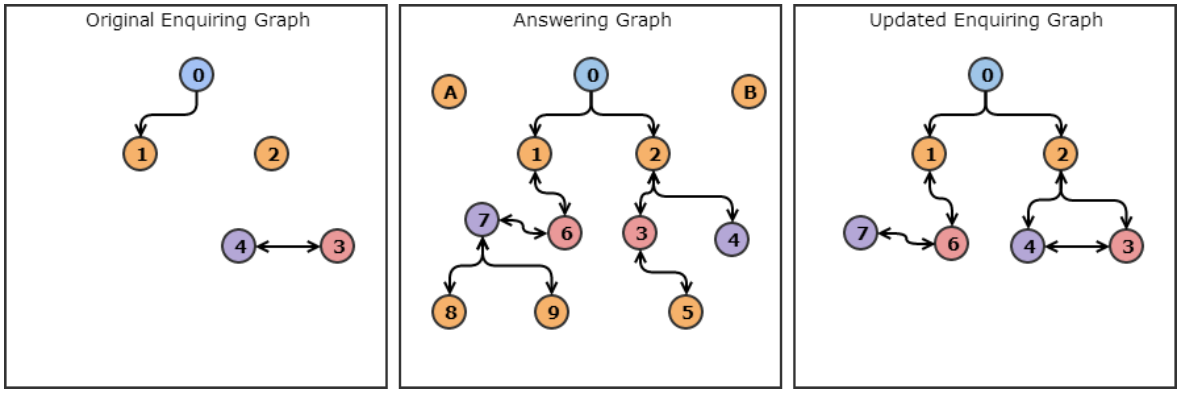

Figure 4.5: Illustration of memory sharing. The enquiring agent initially has the memory graph shown on the left, and wants to know what the answering agent (middle graph) knows about the concept '0'. Neighbours of location nodes (marked in purple) are not shared. Sharing is one way. The answering agent is unaware of the association between nodes '3' and '4' because he did not ask.

Sharing Memories

The concept of agents sharing memories was not initially given much considering during the design, being merely mentioned as possible methods of resolution in Section 3.3.3 and Section 3.5, however the process by which two agents share information turned out to be particularly complex. One agent, the enquiring agent, provides a second agent, the answering agent, with a UID that represents the concept they are enquiring about (e.g., Hammers). The answering agent provides the first agent with their version of the cue node that represents the concept, if they have one, and then a recursive copy of related memories occurs.

Starting at the cue node, the memory graph compares the current neighbours of its cue against the neighbours of the provided cue. If it finds an edge association on the provided cue that it is not aware of, it copies this association across. An association needs to have an Opinion Strength, and rather than copying the other agent's opinion directly an 'Influence Rate' multiplier is applied. This is set per character, and reflects how easily influenced they are by the opinions of others during the copy operation.

It is possible that the provided cue makes an association with a memory node that the enquiring agent is not aware of, so the copy operation will also create that node before copying the association. This reflects the idea of someone saying "Jill is down by the

lake", meaning that the agent becomes aware of both the lake as a place, as well as Jill being there.

Once these associations have been copied, the operation is recursively called on all the neighbours of the provided cue. Considering the complexity of the graph, it would be very easy for this process to continue indefinitely. A limit was put on how much information can be shared by considering what two people would likely discuss in reality. It was decided that the recursive copy can continue until it reaches a location node. Since lots of other nodes unrelated to the original concept can be neighbours of a location, it would not make sense to share them. This keeps sharing concise, and relevant, to the topic being discussed. The sharing process is visualised in Figure 4.5.

However issues arose from the fact that the memory graph can have two way relationships. If there is a connection between item $\rightarrow$ character and character $\rightarrow$ item then this could easily spiral into an infinite loop. Another hashtable is used to prevent this, with the node's UID being added at the start of the copy operation. The recursive operation then skips neighbours it has already visited to prevent an infinite loop from ever occurring.

4.5 Memory Interface & Sensing System

As discussed in Section 3.2.2, the graph needs an interface through which memories can be created and accessed. This interface would also handle the fuzzy memory process. Section 3.2.3 then discussed how a sensing system would be needed for an agent to be aware of the environmental cues around them, as well as how those cues could be converted and passed to the memory graph.

In Unity, any C# scripts, essentially classes, are considered individual components when attached to an object within the scene. Considering that any agent which has a sensor will also have both an interface and a memory graph, it seemed wise to include all three within a single script. This ensures memory functionality can be added to a character by simply providing them this single component.

This component, called the Mind's Eye, includes the Memory Graph presented in Section 4.4.3. It also contains the triggers necessary to perform the sensing duties. Unity contains functionality to track when two colliders collide, which it considers

a trigger when they have no physical representation. Essentially, a large invisible sphere collider exists around the agent and represents their sensor area, while a smaller invisible sphere 'trigger' exists around environmental cues. When this trigger enters the sensor area it fires an event within the Mind's Eye which passes the sensor the cue information. Similarly, when a cue trigger leaves the sensor area another event informs the sensor so it can act.

Between these two events the cues are considered to be within the sensor radius, so being actively thought about. When a cue enters the radius, the memory it corresponds to is marked as active, and when the cue leaves the memory is set back to inactive. Each update (not necessarily every frame, as the graph can handle longer time steps), the component calls an update of the memory graph. This update decays any inactive nodes, while updating and strengthening those that have been marked as active, as discussed in Section 4.4.1.

The memory interface wraps around the above sensor behaviour by providing functions that seemed useful for this project. The distinction between what functionality was included in the interface and what was included in the graph itself was determined by how implementation specific it was. The desire was to create a truly implementation independent node / graph situation, so any functions that would need to be rewritten for each project were instead placed in the interface. To that end, the interface within the Mind's Eye includes the following capabilities:

- *Create Memory*: Given a cue, it creates the inherited memory type (i.e., Location Node, Item Node, Character Node, etc.) that is eventually passed to the memory graph. Implementation specific node types can be added to the overall system by adding them here, without needing to update the graph functionality.
- *Get Concept*: Given a concept UID, this will get the child node with the strongest positive opinion. It will then perform the Fuzzy Check below.
- *Fuzzy Memory Check*: This performs the fuzzy check before a node is returned, and determines which fake memories to return if the fuzzy check fails.
- *Filter Neighbours*: Filters out all neighbours that do not match a given type. Can be useful if agent's do not want to consider certain types.
- *Get Node*: Access the graph and returns a specific node, if it exists. Can also handle implementation specific situations for how to proceed if the graph does

not contain the node.

- *Share Memories*: Given a series of agents, enables them all the share memories by making calls to appropriate enquiring and answering memory graphs.

4.6 Goal-Driven Behaviours

As discussed in Section 3.5, goal-driven behaviours prove to be best choice for any implementation which hopes to take full advantage of the memory model.

The goal system works on a stack based hierarchy, consisting of a base atomic Goal class and a Composite Goal class which inherits from this. A Goal contains a reference to the agent who owns the goal, as well as the goal's current status (active, inactive, complete, failed). A composite goal additionally contains a stack of goals, which are the sub-goals that must be achieved before the composite goal can be considered complete. Specific goals, such as 'Fishing', 'Patrolling', or 'Collecting Item', are created by inheriting from either the atomic or composite goal, and then scripting the specific behaviours needed to achieve the goal.

Goals contain the following functionality:

- *Activate*: Initialises the goal, performing any starting functionality.
- *Reactivate*: Called after a goal has been resumed after being interrupted.
- *Terminate*: Called when a goal is finished, either due to failing, completing, or being purged.
- *Process*: Called each update, where the core behaviour for the goal is contained. Composite goals will also process the current sub-goal here.
- *Handle Message*: Allows a goal to handle messages passed to it from the messaging system. Composite goals will also pass the message to sub-goals for handling.

When considering goal-driven systems, the top-level goal is usually a never ending 'brain' goal that drives the agents behaviour. The prototype emulated this functionality through the scheduler goal, which is able to set the agent's current main task (i.e., work, sleep, eat, fish, etc.) based on a routine loaded from an external text file. This allows for agents to have a wide variety of different behaviours by simply adding to or updating the timetable in the file. For example, the routine file might list 'Fishing' as starting at 8pm, so when the in-game time matches this the scheduler goal updates his current

high-level goal, which previously could have been 'Working at the Shop', to make the agent think about the things he needs to do to fish instead.

4.6.1 Player Interaction & Quests

The graceful fallback systems required a definite endpoint, and Section 3.3.3 suggested that this could be in the form of dynamic quests. This is implemented as a last resort, if the agent has exhausted all other possible avenues for solving their problem. When this occurs the agent gains a temporary 'quest' goal which makes them move toward the Town Square, as if to make an announcement on the notice board. When inside this goal, the agent is also able to access their Quest Information. This struct contains any quest the agent is currently offering, and so this information gets updated with whatever concept they are unable to find through normal means. A flag inside this struct indicates that the the quest is active, and the quest info is then passed to the UI where it can be displayed.

The agent will then attempt to carry out other duties that are not impacted by the quest requirements, while still displaying the Quest UI as they do so. A player can pick up and carry items in the environment within their own inventory, and can hand these items over to an agent if doing so would satisfy their current quest. If the agent's current desired action is blocked by the quest (i.e., they are trying to work, but need a hammer) then handing over the quest item causes them to instantly resume their high-level tasks. If, however, they have moved on to do something else then the quest is marked as completed, but the agent will only resume their original duties when their routine next instructs them to do so.

In addition to this, an agent is also aware of its own quest. This means it is possible for agents to self-complete quests, if they manage to resolve issues in the meantime. For example, an agent might post a quest to get a hammer in the morning, then go fishing in the afternoon. While fishing they might see a hammer nearby, or they might find someone to talk to about a hammer. When it comes time to carry out their work again, the agent will recognise that they have a pending quest related to the task that can now be resolved, so they will collect the hammer, complete their own quest, and then resume their duties. Much like with the graceful fallback behaviour of the goals, self-resolution of quests is handled at a high-level within the goal-behaviour system.

Figure 4.6: The UI used to visualise the memory and related behaviours. Top - Thought Bubble, Middle - Speech Bubble, Bottom - Quest Information

4.7 Model Visualisation

It was important to visualise the model as much as possible, to both showcase its potential and assist in debugging. While the size of the memory graph meant it was not possible to intuitively visualise specific nodes, it was at least possible to print graphs out and visualise them externally to ensure they were working correctly. Within the prototype, the memory was visualised on a smaller scale by focusing mainly on the current thought. Each character was provided with visualisation capabilities, in the form of graphics which floated next to them, that could be managed as part of their behaviours. This surfaced in two forms: Thoughts and Speech. A thought bubble above each character showed their current high-level goal (e.g., Fishing) and their current active sub-goal (e.g., Looking for Rod), so that character actions could be easily understood. In addition to this, they were given the ability to verbalise some behaviours to further clarify what they are doing. For example, when characters are sharing information the speech bubble above their head can reflect what they are talking about.

Finally, when a goal completely fails and the characters revert to creating a quest, a

special quest icon and quest goal appears. Since they are allowed to take on other goals while waiting for the quest to be completed, this allows a user to easily see if the agent is having trouble achieving previous goals, as well as suggesting how they can solve their problem. This indicates to a player what items they need to provide them, but can also indicate to a designer that there is a flaw in the environment if the character is unable to find an item they are expected to have.

4.8 Shared Event Memories

It was not within the scope of the project to create the powerful, yet complex, event systems that drive modern games, so it was decided that this would be simulated and the focus placed on the method by which event memories are created and populated instead.

The event process is simulated through an event manager, which breaks an event down into a sequence of moments which could potentially generate memories if they have been observed by an agent. Essentially, rather than a variety of dynamic events, the manager hard-codes one significant event to allow more time to be spent on the model rather than the boilerplate code necessary to code real events. The event takes the form of a large frog attack on the village, where three large and highly visible frogs attack the beach, while seventeen smaller frogs invade at the same time. The large attack is considered the core event, while the smaller frogs are a possible sub-event, as discussed in Section 3.4.

When the event is triggered, the manager populates the shared event graph (another Memory Graph) with the specific memories that form the event. The messaging system is used to tell each agent within the village that the event has occurred, the UID of which maps to the event stored within the shared event graph. A series of line-of-sight tests then occur between the agents and the frogs, with them creating local memories about what they personally observe. If an agent cannot see any of the smaller frogs, then they will only be aware of the larger event, not the sub-event.

Once the event is over, a player can then interact with the agents and quiz them about what has happened. As with the above, a more general system would be designed to handle multiple events in a real game, but for the prototype this is simplified and a

high-level 'Discuss Event' goal is triggered. This goal then queries the agent's memory for their personal interpretation of the event and verbalises this to the player in the form of generated text. The response can vary greatly depending on where the agent was at the time of the event. It is rare for an agent to be fully aware of everything, although most will know the main details, while a few will only spot one or two frogs, and usually at least one agent will have been indoors so will only be able to tell the player that "the ground was shaking". In fact, trying to get a player to piece together the full facts of the event from the differing eye-witness accounts was an interesting case of emergent game-play that was not considered during the initial design.

4.9 Implementation Architecture

Figure 4.7 presents an agent's architecture, showing the main interactions between components. Following these links demonstrates how, either directly or indirectly, the memory model impacts every facet of an agent's behaviour.

This impact is probably best shown through the view controller, which is a component which can in real-time adapt pre-set animations through inverse kinematics to have an agent look at a particular point in space. The view controller is provided with a target point and a effect strength, which controls how much the agent will turn towards the target point. Various constraints were also set up, so that the animations did not distort in unrealistic ways. Typically this is used to have a character look towards another character, making them seem more alive. However, when the controller is assigned a new target character it first queries the memory graph to see what the agent thinks of that character. If they have a positive opinion then the animation is as expected and the agent looks towards the character, however if they have a negative opinion then the animation is actually reversed, causing the agent to look away in a scornful manner.

Similarly, when two agent's meet they usually greet each other. However before this behaviour is triggered, a similar opinion lookup is performed. Two unfriendly agents will ignore each other, while two especially friendly ages will stop to have a conversation.

The above behaviours are just a small example of how the connections shown in the architecture diagram use the memory model to drive a much greater sense of believability

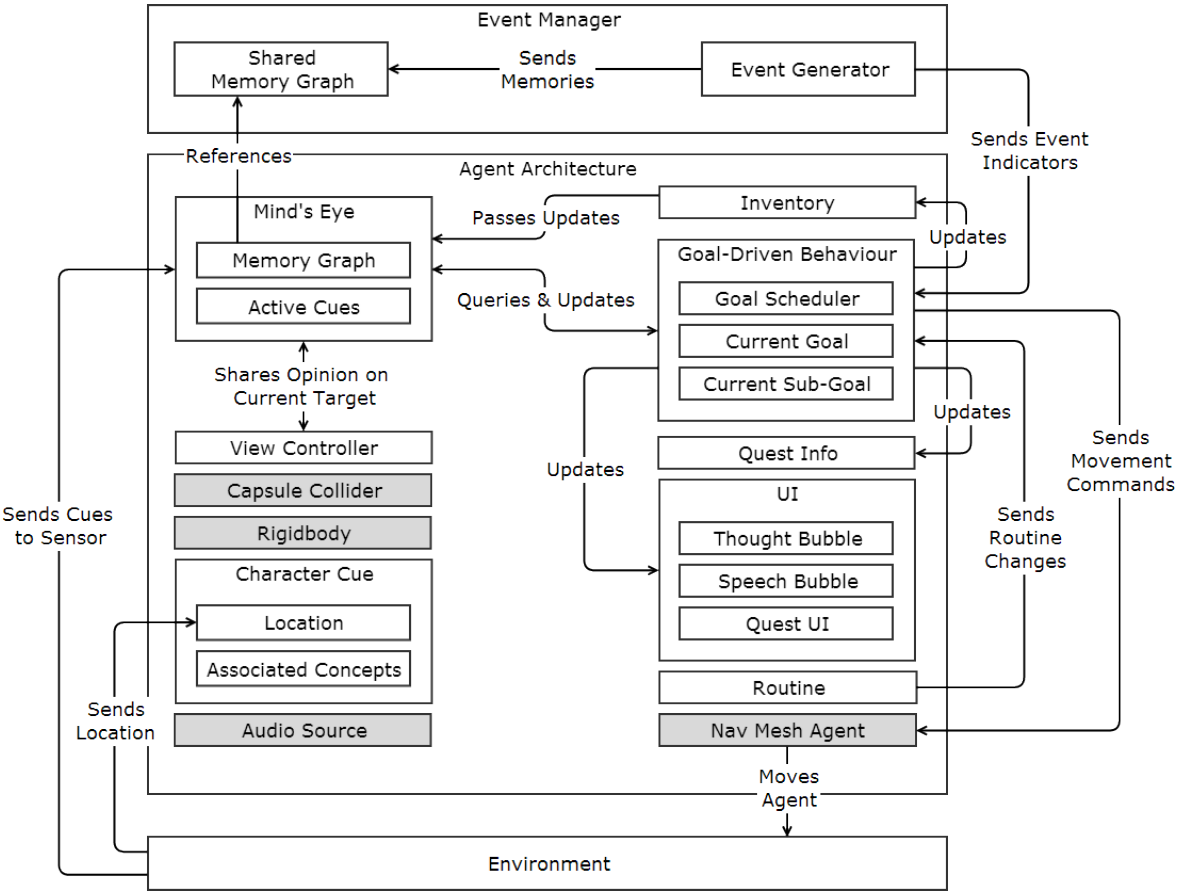

Figure 4.7: The agent's architecture in the completed prototype. Boxes denote individual components, which may contain sub-components. Only core component interactions are shown. Grey components = Unity components used for additional functionally. Arrows denote the flow of information, i.e., Nodes in the Agent's Memory Graph 'reference' nodes in the Shared Memory Graph. Two way arrows represent data being passed both ways, i.e., Goal-Driven Behaviour ↔ Mind's Eye represents a behaviour querying the Memory Graph, getting an response, and then potentially passing an updated memory back to the graph.

than would be possible with simple scripted behaviours.

Finally, as per the design specifications in Section 3.6, the implementation was created with tunability in mind. It was decided to focus on the parameters that would have the strongest impact on the model, as they allow memory capabilities to be set on a per character and system-wide basis. Character specific changes can even be made in real-time, to temporarily simulate human effects that impact memory, i.e., forgetfulness when drunk. The core tunable values are:

- *Influence Rate*: How strongly another agent influences Opinion Strength when sharing memories.
- *Retention Rate*: How quickly inactive memories will decay.
- *Update Frequency*: How frequently the graph is updated.
- *Decay Factor*: Represents the steepness of the forgetting curve.
- *Fuzzy Threshold*: Separates vivid and fuzzy memories.
- *Forgetting Threshold*: Point at which memories are too faded to be remembered.

Chapter 5

Evaluation

This chapter presents an evaluation of the entire project, starting with an assessment of how successfully the prototype achieved the objectives outlined in Section 1.5. The behaviour fidelity of the background characters is measured to see how it compares to existing approaches, the model's ability to procedurally generate behaviours is considered, and the memory and computational requirements of the prototype are profiled to determine how efficient the implementation is. Following this, methods for tuning the model are discussed, and some suitable values are suggested. Finally, the major impediments are discussed, along with a consideration of their impact on the project.

5.1 Improvement over Existing Approaches

The system's ability to adapt current tasks based on the memory graph provides a sense of purpose that is noticeably lacking from current approaches. The decision to allow generic information to be stored in the graph also creates opportunity for the self resolution of problems. Agents store information that they might not need immediately, but which can assist when they run into trouble or when they share it with others, and this behaviour would not be easily possible with scripted behaviours.

To provide a more direct comparison, behaviour logs for characters within the prototype can be compared to those of similar characters within Skyrim. One such example has been chosen here, but similar observations can be made for all characters in the prototype. Table 5.1 covers Flynn's morning routine, which begins with him waking

Table 5.1: Flynn's Morning Routine

Time	High-Level Goal	Current Goal	Current Sub-Goal
6:00am	Sleeping	-	-
8:00am	Blacksmith	Collecting Hammer	-
8:05am	Blacksmith	Collecting Hammer	Looking for Chester
8:10am	Blacksmith	Looking for Chester	Searching Gardens
8:30am	Blacksmith	Looking for Chester	Searching Town Square
8:45am	Blacksmith	Collecting Hammer	Asking Chester
9:15am	Blacksmith	Collecting Hammer	Searching Shop
9:30am	Blacksmith	Collecting Hammer	Purchasing Hammer
9:30am	Blacksmith	Walking to Smithy	-
10:00am	Blacksmith	Working	-

at 8am. The scheduler provides him with his first top-level goal, be a Blacksmith. He requires a hammer to do this, so the first sub-goal is to collect one. Flynn has no memory of where a hammer is located, so decides to ask his friend, Chester. This new sub-goal recalls that Chester is most likely in the gardens, so Flynn heads there. Chester is not found, so the goal then checks the second most likely place, the Town Square. Flynn finds Chester there, so they enter a conversation and discuss hammers. This shares related memories from Chester's memory graph, which tells Flynn that a hammer is in the shop. Flynn goes there and purchases one, so with the goal now completed the final requirement is to head to work.

In the afternoon, shown in Table 5.2, Flynn needs to eat and recalls seeing food in the shop. While on the way there, Asbel approaches him and asks him if he is aware

Table 5.2: Flynn's Afternoon Routine

Time	High-Level Goal	Current Goal	Current Sub-Goal
12:00pm	Eating	Collecting Food	-
12:05pm	Eating	Collecting Food	Walking to Shop
12:20pm	Eating	Having a Conversation	Talking about Hammers
12:45pm	Eating	Collecting Food	Purchasing Food
12:50pm	Eating	-	-
1:00pm	Blacksmith	Collecting Hammer	Equip Hammer
1:00pm	Blacksmith	Walking to Smith	-
1:30pm	Blacksmith	Working	-

of any hammers. Like with Chester, Flynn then shares relevant information from his memory graph. Flynn then continues to the shop, purchases food, and then consumes it. He then resumes his high-level blacksmith goal. This time he recalls that he has a hammer in his inventory, so he can simply equip it in his hand.

Finally in the evening, shown in Table 5.3, Flynn finishes work and must eat again. He then wants to relax with some fishing, and so decides to ask Leon if he knows where to find a rod. As it happens, Leon currently has a fishing rod he is not using, and so gives it to Flynn. The goal is satisfied, and Flynn heads towards the fishing spot. He spends the evening relaxing there, before heading home to bed.

Compare this behaviour to a character within Skyrim, who's routine is shown in Table 5.4. Alvor is a blacksmith within the small village of Riverwood, an area similar in size and complexity to the environment within the prototype. In general, Alvor does not not display the same level of fidelity as Flynn. He never searches for a tool, or food, as they simply appear in his hands when required. Similarly, he will never actively seek other characters, or have dynamic conversations. Alvor's logs frequently display him as idling, although in Skyrim's case this does not represent him standing still. Skyrim features an activity manager which drives where characters go, but not

Table 5.3: Flynn's Evening Routine

Time	High-Level Goal	Current Goal	Current Sub-Goal
5:00pm	Eating	Collecting Food	Equip Food
5:00pm	Eating	-	-
7:00pm	Fishing	Collecting Fishing Rod	-
7:00pm	Fishing	Collecting Fishing Rod	Searching for Leon
7:00pm	Fishing	Searching for Leon	Searching Town Square
7:20pm	Fishing	Searching for Leon	Asking Leon
7:20pm	Fishing	Asking Leon	Borrowing Fishing Rod
7:20pm	Fishing	Walking to Fishing Spot	-
7:35pm	Fishing	Relaxing	-
8:00pm	Sleeping	Walking to Bed	-
8:15pm	Sleeping	-	-

specifically what they do. For example, at 8am Alvor's activity becomes working and so he walks to the Blacksmiths. Invisible 'interaction markers' are attached to the objects within the area which tell characters what animations to use so that it looks correct. While 'idling' as a blacksmith, Alvor randomly moves between each of the blacksmith interaction markers, but in reality he has no genuine concept of work, and there is no purpose to his actions. Although he will be frequently noticed as making tools, they never have any physical presence and simply vanish once the animation is completed. Similarly, while idling at home he will be sitting in his chair, or perhaps reading a book, and when he goes to the Inn his idling can cause him to drink, or work at an alchemy station. However, again, he will never get drunk (or even interact with the barman to refill his drink), and although he could use the alchemy station for hours, he will not ever comment on it, or actually create anything using it. This

Table 5.4: Alvor's Routine

Time	Activity
6:00am	Idle (At Home)
8:00am	Walking to Work
8:15am	Idle (At Blacksmiths)
8:00pm	Walking to Home
8:15pm	Idle (At Home)
9:00pm	Walking to Inn
9:25pm	Idle (At Inn)
11:00pm	Walking Home
11:25pm	Idle (At Home)
12:00am	Sleeping

approach is particularly well suited to creating the *illusion* of believable characters, but any scrutiny of their behaviour can easily cause this illusion to shatter.

Both systems share one general flaw in the strictness of their actions. Observing the logs, it can be noted that behaviour changes always happen at specific intervals. This raises the interesting question of how the characters are aware of the time, considering both environments have no clocks. Discussion in Section 2.5 suggested that people find 'fuzzy' timeframes, such as 'morning' or 'evening' more believable than specific times, so both systems could be improved if this was incorporated into the timing methods. For example, if Alvor and Flynn were to finish work 'in the evening', which was a slightly different time each day, it would seem more believable than the current approach.

In summary, while both systems can be further improved, the the logs for the memory model convey a character driven by specific purpose, while the logs for Skyrim display a character driven only by scripting.

Finally, it is worth considering the risks of both systems. The simplistic and predefined nature of the approach used within Skyrim means that it is unlikely to break, and so it could be considered more robust than the memory model, which is more complex

so contains more risk of errors. However, the graceful fall back of behaviours does mitigate this to a large extent, since there is always at least one thing an agent can do. It could even be possible, although less than ideal, to create a hybrid approach, where the model could fall back to more traditional scripted behaviours if the agent finds itself without the necessary memories to function correctly. However, while the Skyrim approach could be considered more robust, an argument could be made that its repetitive, simplistic behaviours diminish believability in the long-term just as much as broken behaviours would.

Overall, it can be argued that the model does provide a noticeable improvement over existing approaches. More work would need to be done before such a claim could be definitively proved, but within the constraints of the prototype the objective can at least be considered partially satisfied.

5.2 Support for Procedural Generation

Two potential methods for using the model to procedurally generate behaviours were presented in Section 3.7. The manual method was implemented and allows for characters to be quickly quickly pre-populated with memories, which then powers their behaviours without the explicit need to define every action. In addition, the routine system provides a simple way to provide characters with their top-level goals, while leaving them to decide how best to achieve them.

Unfortunately, time constraints prevented the simulated method from being implemented, so it is not possible to save the memory state of agents and then reuse them as initial states later.

It is possible to run the prototype without providing the characters any memories. They initially have no idea of the world but will still attempt to achieve their goals, showing how the model can procedurally generate behaviours even with no input.

As only one of the two methods was implemented during the prototype, it is fair to say this method was only partially satisfied, but some future work could easily achieve this fully.

5.3 Efficiency of Implementation

The model makes memory demands for the storing of graphs, as well as computational demands for accessing and operating on them. An efficient model should have small memory and processing requirements, so these two areas will be evaluated to see how the model performs.

Memory Requirements

Unity runs on the Mono Framework, which offers a managed heap from which memory is allocated. Objects that are allocated within scripts, such as the memory graph, are allocated from this heap. While this does mean that memory management is less of a concern, it does prove to be an issue when analysing the model's memory requirements. Unity's profiling tools provide a comprehensive breakdown of all the game objects within a scene, but it does not list the size of individual objects allocated from the heap.

However, it does provide the total size of allocated heap memory. This means that comparing the difference in size of the heap with and without the model can provide a fairly accurate estimation of how much memory the model is using. To ensure a fair test, the same scene is used for all experiments. Similarly, the average size of the heap is measured for the same number of frames, which in this case is the first 16. Finally, the agents are provided at initialisation with graphs of specific sizes, and forgetting is temporarily disabled so that an exact measure can be taken. Multiple experiments with a variety of graph sizes are preformed, and for each size of graph the number of agents is also varied. This provides insight into the impact of larger graphs and the impact of more agents. These tests were executed on a compiled build, as profiling in the editor adds undesirable noise.

Table 5.5 shows the results of the experiments, and Figure 5.1 plots the results. The experiment shows that demand increases linearly, both as the number of agents and the size of the graph increases. The prototype is a large, complex environment and features just over 100 possible environment cues, so an agent who remembers every possible cue would have a graph of approximately 200 nodes. For ten agents, this is approximately 1.2 MB. However, this is a worst case estimation as the forgetting process makes it highly unlikely an agent will ever be aware of such a number. In fact,

Table 5.5: Model's Memory Requirements

Number of Agents	Nodes in Graph	Allocated Memory
5	100	0.4 MB
5	200	0.6 MB
5	400	1.1 MB
5	1000	2.1 MB
5	10000	16.7 MB
10	100	1.0 MB
10	200	1.2 MB
10	400	2.1 MB
10	1000	4.1 MB
10	10000	33.1 MB

for the current scene the average graph is around 80 nodes, with an allocation size of 0.6 MB for ten agents. This shows that the model could handle a much large number of nodes without any real memory concern.

While information for current generation memory budgets is lacking, it is possible to extrapolate based on data available for previous generations. Millington states that the memory budgeted for A.I. on previous generation machines was typically 8MB, out of 512MB [59]. Current generation hardware boosts 16x the amount of memory in the previous generation, so scaling this suggests an average memory budget of 128MB for current generation A.I., which could comfortably fit over 300 agents with graphs of approximately 1000 nodes. Skyrim has approximately 600 background characters, and so each could have a graph of about 400 nodes within this budget. However, games such as Skyrim typically allocate a larger budget for character A.I., and an argument could be made for allocating the model a higher budget if the increased believability is deemed worthwhile.

In addition, agents only remember cues in their own vicinity. If a game world contains 10,000 cues but an agent only ever exists in a village area with 500 nearby cues, then

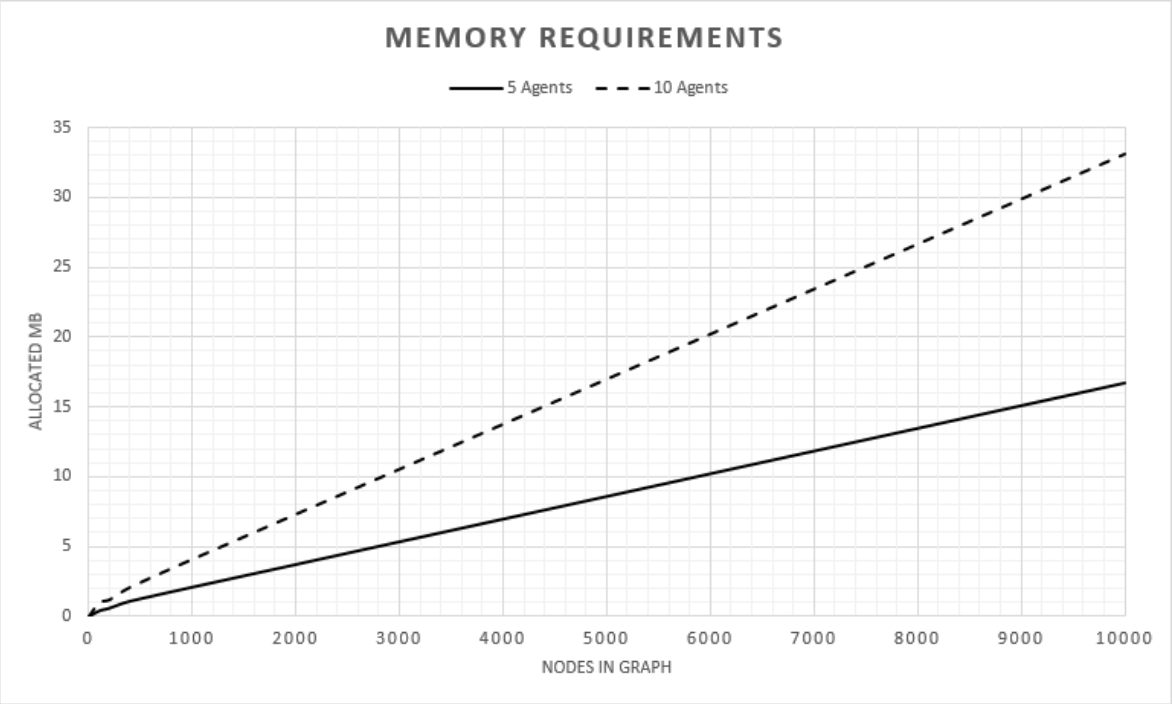

Figure 5.1: A graph of the model's memory requirements, displaying linear complexity.

while 10,000 is a potential worst case scenario, in reality it is almost impossible for it to occur. The realistic worst case is actually the total number of cues within the agent's active area. If a game did require agents to frequently traverse the world, then the forgetting process would still likely prevent the worst case from ever occurring. Controlling the rate at which they forget memories makes it possible for them to have forgotten the last area by the time they reach the new one, meaning the realistic worst case is still only the size of the area they visit with the most cues.

CPU Requirements

Unity's profiler has a much easier time analysing a scene's computational requirements, but the time spent accessing the graph is difficult to capture because agents access it so infrequently. The actual computational requirement of the model per frame is so small that the Unity profiler has trouble identifying it.

Agents update their memory graph, to perform the decay process, much more consistently than they access it, so this can be more easily measured. Ensuring a fair

Table 5.6: Model's CPU Requirements

Number of Agents	Nodes in Graph	Time Per Update	Percentage of Frame Time
6	200	0.07 ms	0.6%
12	200	0.18 ms	1.0%
24	200	0.42 ms	1.7%
48	200	0.81 ms	2.9%
128	200	2.21 ms	4.1%
6	200	0.07 ms	0.6%
6	400	0.12 ms	1.2%
6	1000	0.29 ms	2.5%
6	10000	2.35 ms	13.7%

test in the same way as with the memory requirements, Table 5.6 lists the average time required to update all graphs for a variety of agent and graph sizes, with the results being plotted in Figure 5.2 and Figure 5.3. The frequency of which an agent updates their graph can be tuned, in this case every half a second, so this presents an opportunity for graph updates to be staggered across multiple frames. The frame time percentage grows much slower than the milliseconds per update, as the overheads associated with adding more characters impacts a scene alongside the actual cost of updating their graphs. This shows the graph update itself is not necessarily a limiting factor on the amount of characters possible in a scene, as they would be limited by other factors regardless.

An exception to these measurements occurs in the morning of each simulation, as the computation spikes when the agents all simultaneously wake up and starting accessing the graph. This spike takes an average of 31.27 ms, or 46.6% of the entire frame time. Analysing the spike reveals that a large amount of time is spent within the agent behaviours considering the graph, and not accessing the graph itself. The spike occurs because multiple agent's are simultaneously going through multiple fall-back behaviours, accessing their graphs multiple times, and then searching the results to determine the best course of action. When a single, or even a small number, of agents

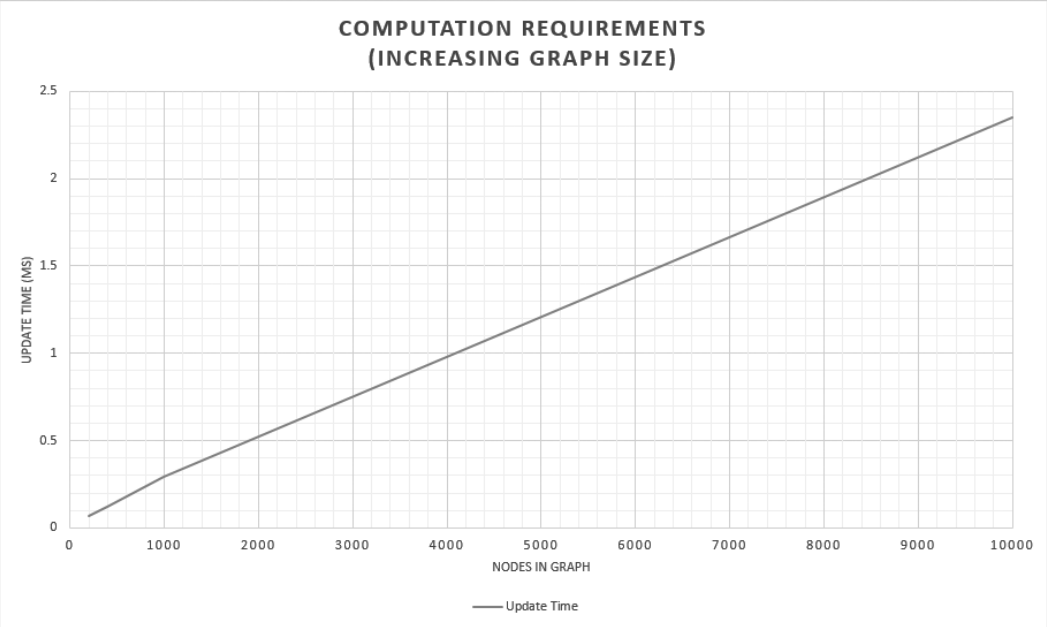

Figure 5.2: A graph showing how the computational requirements for updating the model increases linearly as memory graph size increases.

does this at other points in the day there is no noticeable spike, which suggests that a simple solution would be to manage the number of agents that can 'think' in a single frame. Agent planning could be staggered across multiple frames to keep performance stable. A player is not going to notice if an agent takes a few extra milliseconds to respond, and so this would not impact believability. Of course, more simply, the issue can be mitigated against by having the agents wake up at different times. Not only does this space out initial calculations, but it would actually make the system appear more believable if agents had varying sleep cycles.

An open world game will likely contain dozens, if not hundreds, of agents and so initially the time needed to update 128, or even 48, agents seems potentially concerning. However, while the prototype is simulating the entire world and all agents within it, a real game would only simulate the area and agents around the player. As such, the time for updating 12 or 24 agents is much more likely to occur in a real game. The agents and areas that are not nearby are only partially simulated, so could be updated less frequently with a simple method. As a player is not going to see these agents for a while, a simpler update is not likely to impact believability.

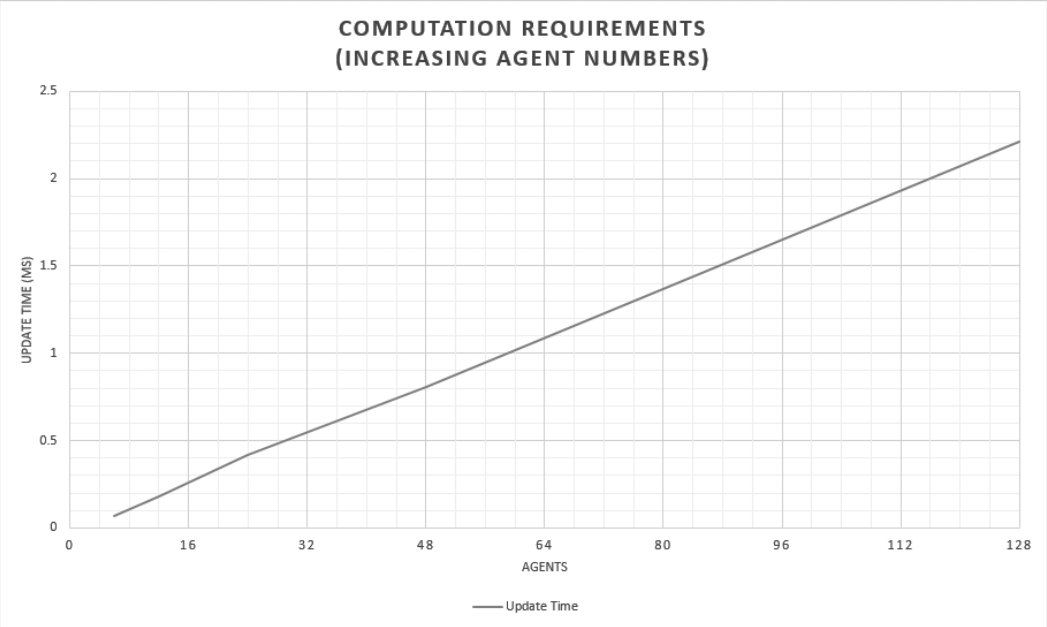

Figure 5.3: A graph showing how the computational requirements for updating the model increases linearly as the number of agents increases.

Considering these results, it can be said that for most scenarios the memory and CPU requirements are entirely reasonably, and so the objective has been achieved. However, increasing the number of agents in a scene or nodes within a graph increases computational demand faster than it does memory. This suggests that CPUs are the factor most likely to limit the model's capabilities.

5.4 Parameter Tuning

As discussed in Section 3.6 and Section 4.9, the project incorporated tunable parameters that could personalise the model's performance either on a per-character or system wide basis. Revisiting these parameters, it is possible to evaluate the range of values that are appropriate for each of them.

- *Influence Rate*: Small non-negative values (< 0.5) work best here, although negative values could create an interesting personality type as it would drive an agent to take the opposite view of any of his peers.
- *Retention Rate*: This is a multiplier which controls the rate of decay, and is best set at a rate which matches the speed of time within the world. If agents move

above more often, then a higher rate works well, but if they spend longer in one place then a lower rate prevents memories from decaying while they idle. A value of 0.05 was used for the prototype, meaning that it would take 200 seconds to completely forget a memory if the decay was linear. It takes much longer due to the forgetting curve, but does ensure memories get 'fuzzy' quickly. Timing the decay shows that memories last approximately 24 minutes (36 hours for the characters) if not reinforced, which suited demonstrations.

- *Update Frequency*: Updating the graph every half second worked perfectly fine for the prototype, although as the computational requirements were not particularly significant a quicker frequency would allow a finer decay curve.
- *Decay Factor*: 1.0 was used here, meaning the forgotten curve was a normal exponential. This meant that memories would become fuzzy quickly but would not be forgotten for quite a while. This created interesting behaviour, but if fuzzy memories are not as necessary then a flatter curve (i.e., higher value) would help avoid this.
- *Fuzzy Threshold*: A value of 5 (out of 10) was used here, but lower values can create more confident agents. Higher values create very scatterbrained agents, which could be desirable for certain personalities.
- *Forgetting Threshold*: A value of 0.007 was used in the prototype, ensuring memories stayed around until they were almost entirely faded. If large graphs are being used then a higher forgetting threshold would keep memory requirements down, as memories will be purged sooner.

The Forgetting Threshold used in the prototype was too extreme, as a value of 0.1, or even slightly higher, would still be perfectly acceptable. A player is not going to notice if an agent has forgotten a memory that has less than 0.1 strength. Considering the forgetting curve decays memories in this range very slowly, keeping them around does not provide any significant benefit.

Retention Rate can seriously impact believability, as too fast of a rate causes the model to collapse as agents forget memories they have just learned. Unfortunately, it is not possible to identify a value which will trigger this behaviour in all cases, as it depends on how fast the game environment moves, but any value above 0.5 should be considered a risk. As a general guideline, ensuring agents remember for at least two days of game

time (or, more specifically, two cycles of their behaviour) produces good results. This ensures they remember things they require daily, and also allows any small details to be retained in the short term. Ideally, however, tuning retention rate to enable memories to last at least a week would allow for a wider variety of daily behaviours (i.e., weekdays and weekends) to be represented without the agents always starting over each morning.

While it was hoped that more concrete values could be provided, experimentation shows that the suitable range is quite implementation specific, and it is the combination of all values that truly impacts how it behaves.

5.5 Impediments

The project initially built upon John Fallon's research, and so it was first proposed that the model be implemented within The Elder Scrolls V: Skyrim. Time was spent investigating, designing, and even implementing this version before it was discovered that it would be impossible to achieve within Skyrim's predefined A.I. framework. This discovery led to the project being swapped to Unity, and as the design of the model was generic it did not requite any major changes. However it does mean that time investigating and implementing within Skyrim was wasted and, in hindsight, if the issues had been uncovered earlier then more time could have been spent on the Unity prototype.

As development progressed the distinction between model and prototype started to blur. When designing goals that access the memory it became difficult to define if an idea would further the model or simply create an interesting implementation of it. This meant that scoping the prototype, which was primarily concerned with creating and showcasing the model, was occasionally difficult. However, this blurring does show how the model manages to impact a great variety of systems, and so is not necessarily a negative from the perspective of the model's potential.

The memory graph itself proved to be a constant challenge through development, as its generic nature and overall high complexity created numerous issues. In particular, the desire to simplify the interface, by automatically adding or deleting nodes when an agent needed it, required meticulous designing and testing to ensure it was robust. These issues primarily surfaced as memory leaks, as nodes with no associa-

tions remained within the graph when the interface should have detected and deleted them. Other issues caused the edge associations, represented by Neighbours, Memory Strength, and Opinion Strength, to go out of sync. Some of these issues could be detected at run-time when an agent's behaviour did not match what was being displayed in their thought bubble, while others had to be manually tracked through the printing of agent graphs or by tracing the programme's execution. The ability to manually generate memories was also very useful for testing, as it allowed specific combinations to be checked without needing to wait for agents to recreate their steps.

Chapter 6

Conclusion

This final chapter provides a conclusion to the dissertation by presenting the main contributions that this project achieved, highlighting some further work that could be undertaken to continue or improve the model, and ending with some final remarks.

6.1 Main Contributions

Although more work may be undertaken in the future, it is possible to identify the main contributions the project has made to date.

Within the area of believability, the project, as shown in Figure 6.1, contributed a practical model that has been tested within a working prototype. The residual memory system provides agents with a new sense of purpose when determining how best to achieve their goals, which fuels more interesting and more believable behaviours. The creation of a prototype also allowed for the true potential of the model to be conveyed, as a user only has to watch the interaction between agents to get a sense of its opportunities. In addition, the prototype highlights the lack of truly believable behaviour in the approaches used for background characters in current games.

Characters have a clear purpose when choosing their actions, which provides a notable advantage over existing approaches. The prototype is fully capable of demonstrating the model and so can be considered functionally complete, but some features could not be implemented within the time-frame.

Figure 6.1: An image taken from the prototype, showing two agents meeting for a conversation.

Although not implemented as completely as was first hoped, the model also supports the procedural generation of behaviours. This allows a variety of character types to be created automatically by simple varying the memories they start with. The behaviours are then driven by these initial memories, and so the 'history' of characters can be created relatively quickly.

Additionally, the project contributes an efficient model which could be used today. The prototype is a practical implementation that tested the design to see if it could be genuinely applied to current hardware, and the results were certainly promising. While the memory demands are more expensive than existing approaches, Section 5.3 showed that they still remain reasonable, and it could be argued that the increase in believability is worth some extra expense in this area. Similarly, Section 5.3 shows that the computational demand remains low for a variety of graph sizes.

The project presents a body of work that collects together a wide variety of sources that were used as inspiration for the model. This forms a powerful foundation for anyone seeking to further the work undertaken here, and it is hoped that the dissertation can help guide anyone looking to create a similar model.

6.2 Future Work

As mentioned in Section 5.2, one aspect of the procedural generation design could not be finished in time, and so completing this would be a top priority for any future work. When considering the project as a whole there are some additional areas that could be improved or extended in the future.

6.2.1 Improved Memory Decay

The current model decays memories from 10 to 0 in a rough emulation of the Ebbinghaus Forgetting Curve, but it became apparent late in development that this design could be improved. Tuning parameters, such as Decay Rate and Retention Rate, change both the steepness of the curve and how quickly the value decays along it. When combined with the Forgetting Threshold, which will be larger than 0, these values can make it difficult to know precisely how memories will decay. Since the player never sees these systems it was not considered an issue, but does add complexity to the design that could be avoided if the method was improved.

An improvement could be made if the value stored in memory strength was actually a lookup value that is transformed into the value of the memory strength when needed. Essentially, if the curve was to be plotted the stored value would represent the distance along the x-axis, i.e., length of decay. A lookup could then transform this into an accurate strength based on the function that describes the curve. This would provide a much more intuitive representation of the memory, and allow for the memory parameters to have a clearer impact on how memories perform.

6.2.2 Long Term Capabilities

There is a lack of understanding with regards to how the prototype would perform over a longer time period. The simulation has been left running for nearly an hour without any issues, but role playing games can be played for hundreds of hours, so in the future it would be interesting to see how it performs over more significant periods of time.

6.2.3 Integration

The routine system employed by the agents is very simple, and quite rigid in design. They are provided new high-level goals at set periods, which they will then try and achieve. While the model gives them increased purpose with how they achieve these

goals, the routine system does not provide the characters with any real ability to decide their own high-level goals. To this end, integration with other systems, such as Niall Mullally's emotional model [60], would provide an interesting method for characters to make decisions for themselves, which would likely further improve their believability.

6.2.4 Extensions & Optimisations

More generally, interesting results could be attained by creating further prototypes which target other popular game genres, such as Stealth or Action, to see how the memory model could impact the behaviour of those characters. In addition, there are likely further optimisations to be uncovered by spending more time analysing the code.

6.3 Perspective

In 2011, game designer Cliff Harris posted his thoughts on the background characters within Skyrim. He highlighted that the game did not appear to improve character interaction beyond that of its predecessor, and then claimed that background characters in most games are "staggeringly stupid". He then suggested a potential alternative system in which every character could "store some data about their attitude to you, plus a list of recent events, plus reaction to your appearance" [34].

It is interesting that his description captures the capabilities of this memory model quite closely, and suggests that the industry is all too aware of the need for improved A.I. models. In addition, Harris claims that voice acting is holding back improvements in character believability. When all lines must be pre-written, then an A.I. model loses out on the ability to generate dialogue for specific situations, making interactions less relevant and believable.

The model generates dialogue for character greetings, as well as more detailed dialogue when characters are explaining their opinions on events. It would be impossible to pre-record voices for all the lines that could be generated, and so the model would be inevitably constrained in such a situation. It could still drive their behaviour and improve upon the pre-existing approach, but ultimately a character would not be able to convey any generated detail about their actions.

However, this is not an impossible task to overcome. Certain games, such as Tomodachi

Life [61] and Metal Gear Solid: Peace Walker [47], make use of voice synthesis to allow a user to customise their experience. Peace Walker allows a user to generate short phrases of robotic speech, while Tomodachi Life uses extensive synthesis for its dynamic interactive dialogues. Although still quite unnatural sounding in those games, it is possible to imagine games in the future having improved synthesis that allows for voice dialogue to be more dynamic.

Alternatively, the model could be incorporated within a smaller budget game, which is not as concerned with voiced dialogue. Solatorobo [30] suggests one solution, as it features thousands of lines of text-based dialogue with no voiced segments. Instead, a selection of pre-recorded 'sound bites', such as "Hi!", "What!?", "It can't be!", and so on, are used to capture the intention of the text and provide it with extra flavour. Such functionality could certainly be added to the model, allowing it to select an appropriate sound bite to match the generated dialogue.

From an academic standpoint, the model presents a practical implementation which attempts to create something that might be used today, and could be used as a springboard for further research into this area, considering other genres, or even approaches that might only become feasible in the near future as hardware continues to improve. In addition, the ability to observe and interact with the agents in the environment suggests that the model could be potentially re-purposed as an educational tool to highlight the concepts of A.I. behaviours.

Appendix A

Reference Materials

A disc with all the files (e.g., the Unity project, plugins, models, scripts, etc.) used in the creation of this project is attached to the back of this dissertation.

Bibliography

[1] Philip E. Agre and David Chapman. What Are Plans for? *Robot. Auton. Syst.*, 6(1-2):17–34, June 1990.

[2] J.R. Anderson, D. Bothell, M.D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004.

[3] R. Arrabales, A. Ledezma, and A. Sanchis. Towards conscious-like behavior in computer game characters. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 217–224, Sept 2009.

[4] R.A. Bartle. *Designing Virtual Worlds*. New Riders Games Series. New Riders, 2004.

[5] Joseph Bates. The Nature of Characters in Interactive Worlds and The Oz Project. Technical report, 1992.

[6] Joseph Bates. The Role of Emotion in Believable Agents. *Commun. ACM*, 37(7):122–125, July 1994.

[7] Nate Blaylock and James F. Allen. Fast Hierarchical Goal Schema Recognition. In *AAAI*, pages 796–801. AAAI Press, 2006.

[8] David Braben and Ian Bell. Elite. [Cassette, Floppy Disk, Cartridge], 1984.

[9] Cyril Brom, Ondřej Burkert, and Rudolf Kadlec. Timing in Episodic Memory for Virtual Characters, 2010.

[10] Cyril Brom, Klára Pesková, and Jirí Lukavský. What Does Your Actor Remember? Towards Characters with a Full Episodic Memory. In Marc Cavazza and Stéphane

Donikian, editors, *International Conference on Virtual Storytelling*, volume 4871 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2007.

[11] D. Brooks. *The Social Animal: The Hidden Sources of Love, Character, and Achievement*. Random House Publishing Group, 2011.

[12] M. Buckland. *Programming Game AI by Example*. Wordware game developer's library. Wordware Pub., 2005.

[13] R. Burke, D. Isla, M. Downie, Y. Ivanov, and B. Blumberg. *Creature Smarts: The Art and Architecture of a Virtual Brain*. Proceedings of the Game Developers Conference, 2001.

[14] Super Flash Cards. Memory Management System. `http://about.superflashcard.com/tour`, 2013. [Accessed 7th August 2014].

[15] Edward Castronova. *Synthetic Worlds: The Business and Culture of Online Games*. University Of Chicago Press, 2005.

[16] Alex J. Champandard. AiGameDev. 10 Reasons the Age of Finite State Machines is Over. `http://aigamedev.com/open/article/fsm-age-is-over/`, December 2007. [Accessed 3rd August 2014].

[17] Alex J. Champandard. AiGameDev. Planning in Games: An Overview and Lessons Learned. `http://aigamedev.com/open/review/planning-in-games/`, March 2013. [Accessed 1st August 2014].

[18] Nicolas Charciarek. Le Gameplay Emergent [In French]. `http://www.jeuxvideo.com/dossiers/00006203/le-gameplay-emergent.htm`, 2006. [Accessed 29th July 2014].

[19] Maria Cutumisu and Duane Szafron. An Architecture for Game Behavior AI: Behavior Multi-Queues. In Christian Darken and G. Michael Youngblood, editors, *AIIDE*. The AAAI Press, 2009.

[20] Kerstin Dautenhahn. *Embodiment and interaction in socially intelligent life-like agents*, pages 102–142. Springer, 1999.

[21] Luke Dicken. AltDevBlog. The Spector of Game AI. `http://www.`

altdevblogaday.com/2012/08/27/the-spector-of-game-ai/, August 2012. [Accessed 20th May 2014].

[22] Hubert L. Dreyfus. A Merleau-Pontyian Critique of Husserl's and Searle's Representationalist Accounts of Action. *Proceedings of the Aristotelian Society*, 100:pp. 287–302, 2000.

[23] H. Ebbinghaus, H.A. Ruger, and C.E. Bussenius. *Memory: A Contribution to Experimental Psychology*. Columbia University. Teachers College. Educational reprints. Teachers College, Columbia University, 1913.

[24] Schneider Edward, Wang Yifan, and Yang Shanshan. Exploring the Uncanny Valley with Japanese Video Game Characters. In *Situated Play*. The University of Tokyo, September 2007.

[25] L. Egri. *The Art of Dramatic Writing: Its Basis in the Creative Interpretation of Human Motives*. Touchstone, 1972.

[26] Clark Davidson Elliott. *The Affective Reasoner: A Process Model of Emotions in a Multi-agent System*. PhD thesis, Evanston, IL, USA, 1992. UMI Order No. GAX92-299Th01.

[27] John Fallon. Believable Behaviour of Background Characters in Open World Games. M.Sc. Thesis, University of Dublin, Trinity College, 2013.

[28] B. Furht. *Handbook of Multimedia for Digital Entertainment and Arts*. Springer, 2010.

[29] D.A. Gallo. *Associative Illusions of Memory: False Memory Research in Drm and Related Tasks*. Essays in Cognitive Psychology Series. Psychology Press, 2006.

[30] Bandai Namco Games. Solatorobo: Red the Hunter. [Nintendo DS Game Card], 2010.

[31] Rockstars Games. Grand Theft Auto 3. [CD, DVD, Digital Download], 2001.

[32] E. Gilboa and A. Ortony. The Structure of Emotion Response Tendencies. Work in progress, 1991 [Described by [26]].

[33] Anshul Goyal. Real-Time Planning Using HTN in Stealth Game. http://www.

anshulgo.com/download/AnshulGoyal_AIProjectPaper_April2010.pdf, april 2010. [Accessed 7th August 2014].

[34] Cliff Harris. A Game Dev's Thoughts on Skyrim NPCs. http://positech.co.uk/cliffsblog/2011/12/27/a-game-devs-thoughts-on-skyrim-npcs/, 2011. [Accessed 31st August 2014].

[35] John Haugeland. Understanding Natural Language. *The Journal of Philosophy*, 76(11):pp. 619–632, 1979.

[36] P. Hingston. *Believable Bots: Can Computers Play Like People?* Springer Berlin Heidelberg, 2012.

[37] Mark Hoogendoorn and Jeremy Soumokil. Evaluation of Virtual Agents Utilizing Theory of Mind in a Real Time Action Game. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 59–66, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[38] International Game Developers Association (IGDA). Working Group on Rule-based Systems. http://archives.igda.org/ai/report-2005/rbs.html, 2005. [Accessed 7th August 2014].

[39] D. Isla. Handling Complexity in the Halo 2 AI. In *Proceedings of the 2005 Game Developers Conference (2005)*, 2005.

[40] D. Isla. Halo 3 Objective Trees: A Declarative Approach to Multi-agent Coordination, 2008. Invited talk, Artificial Intelligence and Interactive Digital Entertainment (AIIDE) 2008.

[41] C. Jones. *Chuck Amuck: The Life and Times of an Animated Cartoonist*. Farrar, Straus and Giroux, 1999.

[42] Rudolf Kadlec and Cyril Brom. DyBaNeM: Bayesian Episodic Memory Framework for Intelligent Virtual Agents. In Ruth Aylett, Brigitte Krenn, Catherine Pelachaud, and Hiroshi Shimodaira, editors, *IVA*, volume 8108 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 2013.

[43] Yilin Kang and Ah-Hwee Tan. Self-organizing Cognitive Models for Virtual

Agents. In Ruth Aylett, Brigitte Krenn, Catherine Pelachaud, and Hiroshi Shimodaira, editors, *IVA*, volume 8108 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2013.

[44] Zerrin Kasap and Nadia Magnenat-Thalmann. Intelligent virtual humans with autonomy and personality: State-of-the-art. *Intelligent Decision Technologies*, 1(1-2):3–15, 2007.

[45] Troy D. Kelley. Symbolic and Sub-symbolic Representations in Computational Models of Human Cognition: What Can be Learned from Biology?, 2003.

[46] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[47] Konami. Metal Gear Solid: Peace Walker. [UMD, Blu-ray Disc, DVD, Digital Download], 2010.

[48] M. Lebowitz. Creating Characters. In *Poetics. Vol. 13. No. 3. December 1984.*, 1984.

[49] M. Lebowitz. Story-Telling as Planning and Learning. In *Poetics. Vol. 14. No. 6. December 1985.*, 1984.

[50] Weizi Philip Li, Tim Balint, and Jan M. Allbeck. Using a Parameterized Memory Model to Modulate NPC AI. In Ruth Aylett, Brigitte Krenn, Catherine Pelachaud, and Hiroshi Shimodaira, editors, *IVA*, volume 8108 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2013.

[51] Aaron Bryan Loyall. *Believable Agents: Building Interactive Personalities*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1997. AAI9813841.

[52] Michael Mateas. An Oz-Centric Review of Interactive Drama and Believable Agents. Technical report, 1997.

[53] Michael Mateas. Interactive Drama, Art and Artificial Intelligence. Technical report, 2002.

[54] Michael Mateas and Andrew Stern. Architecture, Authorial Idioms and Early Observations of the Interactive Drama Faade. Technical report, 2002.

[55] R. R. McCrae and O. P. John. An introduction to the five factor model and its applications. Special issues: The five-factor model: Issues and Applications. *Personality*, 60:175–215, 1992.

[56] James R. Meehan. TALE-SPIN, an Interactive Program That Writes Stories. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'77, pages 91–98, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[57] James Richard Meehan. *The Metanovel: Writing Stories by Computer.* PhD thesis, New Haven, CT, USA, 1976. AAI7713224.

[58] M. Merleau-Ponty. *The Structure of Behaviour.* Methuen, 1965.

[59] I. Millington and J. Funge. *Artificial Intelligence for Games.* Morgan Kaufmann. Taylor & Francis, 2009.

[60] Niall Mullally. An Emotional Model for Background Characters in Open World Games. M.Sc. Thesis, University of Dublin, Trinity College, 2014.

[61] Nintendo. Tomodachi Life. [Nintendo 3DS Game Card, Digital Download], 2014.

[62] Bob Nystrom. Game Programming Patterns - State. `http://gameprogrammingpatterns.com/state.html`, 2009. [Accessed 9th August 2014].

[63] A. Ortony, G. Clore, and A. Collins. *Cognitive Structure of Emotions.* Cambridge University Press, 1988.

[64] A. Ortony, G.L. Clore, and A. Collins. *The Cognitive Structure of Emotions.* Cambridge University Press, 1990.

[65] Robert Powell. Bargaining and Learning While Fighting. *American Journal of Political Science*, 48(2):344–361, 2004.

[66] David V. Pynadath, Ning Wang, and Stacy C. Marsella. Are You Thinking What I'm Thinking? An Evaluation of a Simplified Theory of Mind. In Ruth Aylett, Brigitte Krenn, Catherine Pelachaud, and Hiroshi Shimodaira, editors, *IVA*, volume 8108 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2013.

[67] W. Scott Reilly, Jaime Carbonell, and Reid Simmons. Believable Social and Emotional Agents, 1996.

[68] Mikel Reparaz. Battle of the GTA Clones. `http://www.gamesradar.com/battle-of-the-gta-clones/`, 2007. [Accessed 29th July 2014].

[69] Tiago Ribeiro, Marco Vala, and Ana Paiva. Censys: A Model for Distributed Embodied Cognition. In *Intelligent Virtual Agents*, volume 8108 of *Lecture Notes in Computer Science*, pages 58–67. Springer Berlin / Heidelberg, September 2013.

[70] K. Romdenh-Romluc. Merleau-Ponty and the Power to Reckon with the Possible. In T. Baldwin, editor, *Reading Merleau-Ponty: On Phenomenology of Perception*. Taylor & Francis, 2007.

[71] R.C. Schank and R.P. Abelson. *Scripts, Plans, Goals, and Understanding: An Inquiry Into Human Knowledge Structures*. The Artificial Intelligence Series. Lawrence Erlbaum Associates, 1977.

[72] Jamie Sefton. The Roots of Open-World Games. `http://www.gamesradar.com/the-roots-of-open-world-games/`, 2008. [Accessed 29th July 2014].

[73] Herbert A. Simon. *The Sciences of the Artificial (3rd Ed.)*. MIT Press, Cambridge, MA, USA, 1996.

[74] Branislav L. Slantchev. The Principle of Convergence in Wartime Negotiations. *American Political Science Review*, null:621–632, 11 2003.

[75] Bethesda Softworks. The Elder Scrolls: Arena. [Floppy Disk, CD, Digital Download], 1994.

[76] Bethesda Softworks. Façade [Digital Download], 2003.

[77] Edge Staff. The Making Of: Elite. *EDGE Magazine*, E88, 2000.

[78] Konstantin Stanislavskij and E.R. Hapgood. *Stanislavski's Legacy: A Collection of Comments on a Variety of Aspects of an Actor's Art and Life*. TAB paperbook. Routledge/Theatre Arts Books, 1968.

[79] Saul Sternberg. High-Speed Scanning in Human Memory. *Science*, 153:652–654, 1966.

[80] Bethesda Game Studios. The Elder Scrolls V: Skyrim. [DVD, Blu-Ray Disc, Digital Download], 2011.

[81] Ah-Hwee Tan, Gail A. Carpenter, and Stephen Grossberg. Intelligence Through Interaction: Towards a Unified Theory for Learning. In Derong Liu, Shumin Fei, Zeng-Guang Hou, Huaguang Zhang, and Changyin Sun, editors, *ISNN (1)*, volume 4491 of *Lecture Notes in Computer Science*, pages 1094–1103. Springer, 2007.

[82] Chris Taylor. Ludonarrative Dissonance in Open World Games. `http://thefreecheese.com/2014/06/23/ludonarrative-dissonance-in-open-world-games/`, 2014. [Accessed 29th July 2014].

[83] Unity Technologies. Unity Game Engine, v4.5. [Digital Download], 2014.

[84] F. Thomas and O. Johnston. *Disney Animation: The Illusion of Life*. Abbeville Press, 1987.

[85] Julian Togelius, Georgios N. Yannakakis, Sergey Karakovskiy, and Noor Shaker. *Assessing Believability*, pages 215–230. Springer Publishing Company, 2012. 2012; 1.

[86] Vinoba Vinayagamoorthy, Andrea Brogni, Marco Gillies, Mel Slater, and Anthony Steed. An Investigation of Presence Response across Variations in Visual Realism. In *In Proceedings of Presence 2004: The 7th Annual International Workshop on Presence. Available online at: http://www.cs.ucl.ac.uk/research/ equator/papers/VisualRealism.pdf*, pages 119–126, 2004.

[87] A. Whiten. *Natural Theories of Mind: Evolution, Development, and Simulation of Everyday Mindreading*. B. Blackwell, 1991.

[88] Rizky Winanda. Gamasutra. A Better Narrative using OCC Emotional Model. `http://www.gamasutra.com/blogs/RizkyWinanda/20140513/217622/A_Better_Narrative_using_OCC_Emotional_Model.php`, May 2014. [Accessed 1st August 2014].

[89] M. Wish, M. Deutsch, and Kaplan S. Perceived Dimensions of Interpersonal

Relations. In *Journal of Personality and Social Psychology. Vol. 33. No. 6. April 1976.*, 1976.