TIERED BEHAVIOUR ARCHITECTURE FOR VIRTUAL CHARACTERS USING CYCLIC SCHEDULING AND BEHAVIOUR CAPTURE

by

Richard Zhao

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science
University of Alberta

Abstract

A story-based video game contains many characters. A few are controlled by the player and the majority are virtual characters, controlled by artificial intelligence. In recent years, video game artificial intelligence has developed slower than other aspects of video games, such as graphics, mainly due to the cost of scripting complex and believable virtual characters. To tackle this bottleneck in content creation, several behaviour architectures have been proposed over the past decades. This dissertation proposes metrics for evaluating behaviour architectures and their associated toolsets: expressiveness, performance, quality, and usability. A good mechanism for evaluating architectures / toolsets is essential to replace the current trial and error approach that is in practice today. This dissertation also proposes a new Tiered Behaviour Architecture model and its associated toolset for controlling the behaviours of virtual characters, and evaluates it by applying the metrics. The objective (top) level of the architecture determines the general schedules of the virtual characters composed of objectives, and the roles that will accomplish these objectives. Cyclic scheduling is a technique that allows for the automatic generation of schedules based on partially specified constraints. The role (bottom) level of the architecture uses Behaviour Capture with Hidden Markov Models, a self-contained technique, to generate actual fine-grained behaviours. This dissertation presents studies that show behaviours generated by this architecture / toolset have high scores for all four metrics. It also shows that Behaviour Capture with Hidden Markov Models generates higher quality (more believable) behaviours for virtual characters than the state of the art in commercial games.

Preface

This thesis is an original work by Richard Zhao. The research project, of which this thesis is a part, received research ethics approval from the University of Alberta Research Ethics Board with the following projects: Project Name “Advantages of Behaviour Capture vs. Manual Scripting”, No. Pro00020584, January 26, 2011; Project Name “Advantages of Behaviour Capture vs. Manual Scripting (with gender study)”, No. Pro00032743, July 3, 2012; Project Name “Comparison of Video Game Behaviour Generation Methods”, No. Pro00040342, May 24, 2013; Project Name “Comparison of Behaviour Scripting Methods”, No. Pro00042563, August 28, 2013.

Acknowledgments

I am deeply grateful to my supervisor, Dr. Duane Szafron, for patiently helping me through this long journey. This dissertation would not have been possible without the countless hours Duane spent providing guidance to my research. I appreciate the insightful feedback provided by my supervisory committee, Dr. Martin Müller and Dr. Michael Bowling, and my other examining committee members, Dr. Sean Gouglas and Dr. Mark Riedl. Thanks also to Dr. Vadim Bulitko, Dr. Eleni Stroulia, and Dr. Michael Carbonaro for their valuable questions and suggestions.

Thanks to members of the BELIEVE ScriptEase research group for their help and company throughout the years: Elyse Hill, Zak Turchansky, Neesha Desai, Jessica Yuen, Kevin Schenk, Eric Graves, Matthew Church, Robin Miller, Adel Lari, Jason Duncan, Kirsten Svidal, Delia Cormier, Alex Czeto, Wei Li, Jamie Schmitt, as well as my user study participants for contributing to my research.

Thanks to Dr. Levi Lelis, AmirAli Sharifi, Cailu Zhao, Chenlei Zhang, Jing Zhang and other awesome fellows that I have worked with throughout the years. Thanks to the teaching teams of CMPUT 250 and CMPUT 201 who certainly made my job a lot more enjoyable. I offer my gratitude and blessings to everyone else at the University of Alberta who helped me in any respect during the completion of this thesis.

Last but certainly not least, thanks to my parents, Min Guo and Peter Zhao, for their many years of support. Thanks to Qiushan Li for always being there for me, in life and work, and for providing an interesting outsider's perspective on my research.

I would like to acknowledge the funding support provided by the GRAND Network of Centres of Excellence, NSERC, as well as grants from the University of Alberta Graduate Students' Association, the Faculty of Graduate Studies and Research at the University of Alberta, and the Society for the Advancement of the Science of Digital Games.

Table of Contents

List of Tables

List of Figures

1. Introduction

Video or computer games continue to dominate the entertainment market. They produced over 14 billion dollars in sales each year since 2009. Many modern games have included deeper and more involved stories. One genre of games that especially emphasizes its story component is role-playing games (RPGs), where the player assumes the identity of a fictional character immersed in a virtual world. These games are generally called story-based games in this dissertation.

Story-based games contain many characters. Most of them are non-player characters (NPCs), often referred to as virtual characters in this dissertation, controlled by artificial intelligence (AI) techniques. These virtual characters interact with the player character (PC) or player avatar, with each other, and with the virtual environment.

Over the years, while other areas of gaming technology, such as computer graphics, have had large improvements, behaviours of virtual characters have improved relatively slowly. This is due to a few factors. Creating natural-looking behaviours for virtual characters is not inexpensive. Typically, in a virtual environment, all objects except the player character are controlled by individual pieces of programming code called scripts, and these scripts are interpreted by the game engine to determine how the character will behave in a particular context. In most commercial story-based games, there are hundreds or thousands of virtual characters. Games such as the relatively recent The Elders Scroll V: Skyrim [7] deploy randomly generated virtual characters, and their numbers are only limited by the time a player spends in the game. Therefore, manually writing scripts for each virtual character requires extensive resources. To avoid this, companies re-use the same scripts across groups of similar virtual characters, resulting in simple and repetitive behaviours for most virtual characters who are not involved directly in the

main plot. Manual scripting has become a bottleneck of content creation for virtual character behaviours.

1.1 Behaviours in Games

Video games have greatly evolved since their conception. The advances in computer graphics are perhaps the most notable. As graphics researchers have noted, "the quest for photorealistic renderings has long been the holy grail of computer graphics." [26] With modern hardware and better algorithms, games have been able to render virtual characters and environments in vivid detail.

However, the simplicity of virtual character behaviours stands in staggering contrast with the realism conveyed by advanced graphics. Increasingly, virtual characters are falling into the "uncanny valley". The uncanny valley [42], originally used by Mori in robotics design theory, refers to the situation where humans respond very negatively to a robot whose appearance is very close to being human, but not quite perfect. The term has been adopted in video games to refer to virtual characters as well [56]. The valley exists because cartoon-like animated characters are sufficiently different than humans that observers do not expect human characteristics. Once graphical fidelity improves enough, an observer shifts expectations to look for human characteristics and deviations are identified as strange. Although further improvements in graphical fidelity could be expected to close the uncanny valley for character appearance, observers' detection of un-humanlike behaviours (not appearance) might prevent the uncanny valley from disappearing. A virtual character that looks perfect but acts unnaturally is also uncanny.

Poor virtual character behaviours distract players from the immersion of the gaming experience. Rather than standing or wandering aimlessly, virtual characters should converse with other virtual characters and interact with game objects in believable

ways. They should also be able to react to unexpected events such as the destruction or realignment of buildings and other important game objects.

To understand how virtual characters behave in games, it is essential to understand how modern game engines work. A modern game engine is comprised of several layers. At the core, there is a software framework created by computer programmers which typically includes basic game functions, memory management, sound, animation, a rendering engine for graphics, and a physics engine to simulate collisions, gravity and other physics interactions to produce a realistic game world. The core engine is usually compiled into machine code for fast execution by a computer. On top of the core level, there is usually a scripting engine, which reads "instructions" and executes the instructions to generate events in the game and to make objects respond to these events. These instructions are commonly structured in the form of "scripts", which are written in a computer programming language. Scripts specify the interactions in a game world and direct the flow of a story. For example, a script can specify that when the player character approaches a bartender, the bartender should ask the player character to order a drink. Since compiling the core engine to machine code takes a considerable amount of time, the separation of the scripting engine from the core engine allows scripts to be modified and tested quickly. Thus the story can be modified, without having to recompile the underlying core engine code. This also enables the use of the same core engine with many different scripted stories or games.

The Aurora engine of Neverwinter Nights [10] by BioWare Corp.[1] is a typical game engine. It contains a scripting engine using its own NWScript scripting language. It has an associated Aurora Toolset that can be used by writers and artists and by technical designers who are computer programmers that write NWScript code. According to Cutumisu [22], the main campaign of Neverwinter Night contains 7,857 script files, totalling 141,267 lines of NWScript code. Clearly, writing these

[1] www.bioware.com

scripts was no small effort. Research tools such as ScriptEase [23] and its successor ScriptEase II [55] attempt to address the scripting issue by offering game designers graphical interfaces and reusable common patterns in human-readable English language text. ScriptEase II also provides a drag-and-drop interface and features a game-independent model, where common design patterns can be translated into programming codes for multiple game engines, as opposed to exclusively for the Aurora engine. As illustrated in Figure 1, ScriptEase operates on top of the scripting engine and automatically generates NWScript code from designer selected and adapted patterns. Recently, several game companies have made attempts to replace manual scripting by high-level scripting tools or interfaces such as Kismet for Unreal and authoring interfaces for Frostbite 3. The idea of replacing manual scripting by high level tools is used extensively in this dissertation. In fact, this dissertation proposes a structured model to solve a particular authoring problem, the authoring of the behaviours for virtual characters in a large game world. Just as importantly, this dissertation proposes metrics to evaluate solutions to this problem.

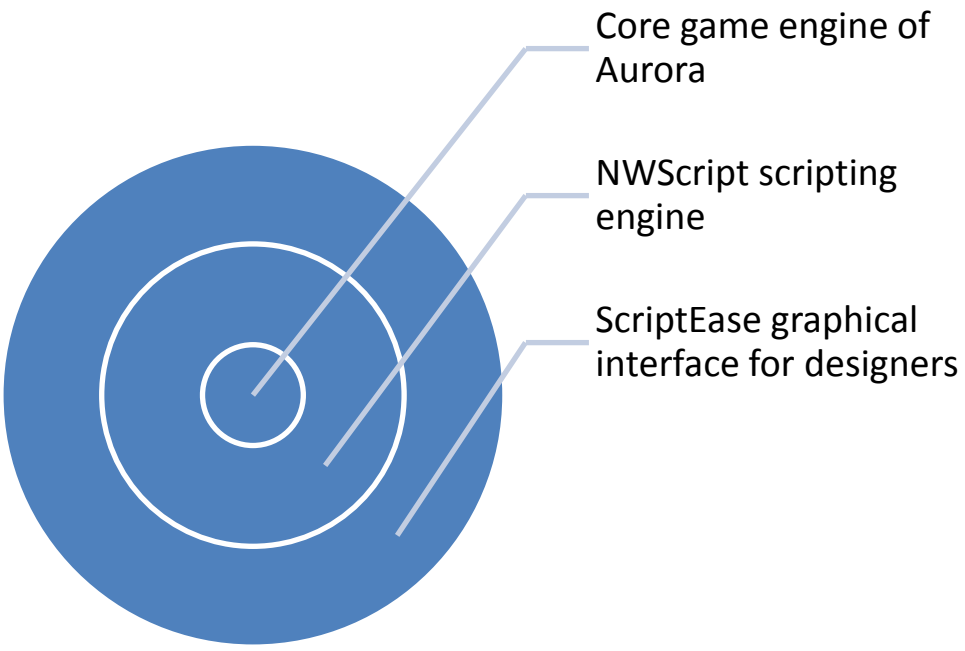


Figure 1: The relationship between ScriptEase and the Aurora engine. A similar relationship exists for Kismet and the Unreal engine and other scripting interfaces

1.2 Research Contributions

The ultimate goal of this research is to alleviate the effort of writing behaviour scripts manually, by providing a behaviour architecture and associated behaviour design tools. This dissertation presents an evaluation process for behaviour architectures and associated tools and an evaluated Tiered Behaviour Architecture and toolset. Based on an analysis of conducted studies, this work advances the state of the art in creating believable behaviours for virtual characters in large game worlds. The research contributions of this dissertation are the following:

1. Metrics for evaluating an architecture and toolset for behaviours of virtual characters. Four metrics are defined: Expressiveness, Performance, Quality, and Usability. These metrics provide tangible standards to measure the usefulness of any proposed behaviour architecture and toolset.

2. A Tiered Behaviour Architecture model and toolset, which can be used to represent the behaviours of virtual characters. This model divides the behaviour control of a character into tiers: circumstances, schedules, objectives and roles. Evaluations based on the aforementioned metrics show that the architecture and toolset are expressive and have high performance, and that the behaviours generated by the architecture are more believable (have higher quality) than the current state of the art.

3. A Cyclic Scheduling technique for the automatic generation of daily schedules based on partially specified constraints. Cyclic Scheduling conforms to the Tiered Behaviour Architecture and automates the behaviour authoring process for game designers. A usability study shows that the

technique enables non-programmer designers to produce behaviours, and that it is more efficient than manual scripting.

4. A technique using Behaviour Capture with Hidden Markov Models to produce fine-grain behaviours for characters in local roles. This technique is self-contained and can be used with the Tiered Behaviour Architecture at the role level. The resulting behaviours produce a noticeable improvement in quality over manually scripted characters in current commercial games, as shown by user studies.

Parts of these research contributions have appeared in the following peer-reviewed publications:

- "Using Cyclic Scheduling to Generate Believable Behavior in Games". In Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, October, 2014 [70].
- "Virtual Character Behavior Architecture using Cyclic Scheduling". In Proceedings of the 9th International Conference on the Foundations of Digital Games, April, 2014 [71].
- "Generating Believable Virtual Characters Using Behavior Capture and Hidden Markov Models". In Advances in Computer Games 13 Conference, Lecture Notes in Computer Science Volume 7168, 2012 [68].

1.3 Organization

This dissertation is organized as follows. Chapter 2 elaborates the background for the behaviours of virtual characters in games and discusses the current efforts at creating virtual character behaviours, both in the academic literature and in industry. Chapter 3 presents metrics for evaluating an architecture and toolset for behaviours,

presents the model for the new Tiered Behaviour Architecture and toolset, and evaluates the model / toolset based on three of the discussed metrics: expressiveness, efficiency and quality. Results based on the gender of user study participants are also discussed. Chapter 4 describes Cyclic Scheduling, a technique for automatically generating daily schedules for virtual characters that conforms with the Tiered Behaviour Architecture, and evaluates the fourth metric (usability) of the technique. Chapter 5 describes Behaviour Capture with Hidden Markov Models and how it can be used alone or as part of the Tiered Behaviour Architecture to generate local behaviours in scenes. It validates this approach by presenting the results of user studies and discusses similarities and differences based on the gender of user study participants. Finally, Chapter 6 provides a summary of the work presented in this dissertation and some directions for future research. The appendices provide the detailed statistics from the user studies.

2. Background and Related Works

The setting is the world of Dragon Age: Origins [9]. You are a Grey Warden who is tasked with the mission to slay the Archdemon and save the world from certain destruction. You travel to the village of Lothering to gather news and supplies. You enter the only tavern in town, Dane's Refuge. Upon clearing the soldiers who were conveniently waiting there to arrest you (as the game designers intended), you look around. There are about a dozen people in the tavern. Some of them appear to be in conversations with one another, others just standing. After observing them for a while, the scene is getting strange. No one is moving around, not the bartender whose name is Danal, not the tavern patrons, not the bards, no one (Figure 2).

This world setting is shared by Dragon Age: Inquisition [8]. As an inquisitor wielding a unique power to save a world plunged into chaos, you travel to different places in the world to gather support for your inquisition. While in Herald's Rest, a tavern in Skyhold, you see a very similar environment. There is a bartender named Cabot, a bard singing, and a few patrons sitting around. The patrons appear to engage in conversations, but as you watch them, you notice the same strange scenario: no one ever moves around (Figure 3).

You now enter the world of The Elder Scrolls V: Skyrim [7]. You are a Dragonborn who is destined to save the world by defeating the dragon god Alduin. You travel to the city of Solitude to make some purchases. This time, people do move around in the city, and the city seems lively. You meet Greta, who is a nice woman living with her husband Addvar and her child Svari. She asks you to help her find her deceased brother's amulet. After getting to know her, though, you notice something a little bit strange. You see her leaving her house at exactly 3pm, every day. Well, people have routines so perhaps this is not strange. However, she follows her routine exactly to the minute, every single day without variation.

Figure 2. A tavern, Dane's Refuge, in Dragon Age: Origins, showing tavern patrons, a bartender on the right, and two bards on the second floor.

Figure 3. A tavern, Herald's Rest, in Dragon Age: Inquisition, showing tavern patrons in the background, a bartender on the left, and a bard.

Perhaps you are an experienced gamer and you see these scenarios all the time in current video games, and you get used to them, thinking "it's just a game". However, it does not have to stay that way. These games are examples of interactive narrative systems and one of the goals of an interactive narrative system is to provide an immersive experience for its audience. Providing an immersive experience has been part of the interactive narrative research for about twenty years [52]. Riedl and Bulitko summarized the three dimensions of interactive narrative systems: authorial intent, player modelling, and virtual character autonomy [52]. These dimensions represent distinct but interconnected approaches to creating interactive narrative experiences. Artificial intelligence has been used at different levels to support interactive narratives, such as to fill the role of an artificial playmate (AI as Actor), to automatically generate or adapt game content (AI as Designer), or to examine multiple player interactions both in an in-game and out-of-game community (AI as Producer) [53].

An important part of an immersive experience incorporates believable characters. Current virtual characters behave in ways that do not conform to players' expectations, and that can break the immersion of the virtual world. Controlling the behaviours of virtual characters in a virtual game world is a sub-domain of artificial intelligence. These virtual characters need only exhibit behaviours appropriate to the domain of the game world and the intended story, which is quite limited compared to the full spectrum of human behaviour. Nevertheless, creating behaviours that are believable to human observers/players, even in a limited domain, is a difficult task. Many different approaches have been attempted.

2.1 Scripting Languages

In the earliest games to feature non-player characters (NPCs), such as the original Ultima game [51], the NPCs simply moved randomly with no complex intelligence, if they moved at all. Sharifi [58] has a good short summary of NPC behaviour

progress. The first NPC behaviours were defined by small sections of game engine programming code written by programmers. As games got larger and more complex, many games began to support scripting capabilities that allow objects and events in the game to be controlled and customized via scripting codes (scripts) instead of game engine code. Scripts are manually written by computer programmers and can be used to directly specify the behaviours of each individual virtual character. These scripts are usually interpreted by an interpreter that communicates with the game engine. Despite the existence of these scripting languages, most games such as Neverwinter Nights provide only minimal behaviours, allowing virtual characters to perform meaningless actions (such as wander randomly) until the player character approaches them and the plot is advanced according to the intent of the game designer. Unfortunately, as games have become more visually realistic, these plot-serving virtual characters have not added much to the believability of the world. To appear believable, these virtual characters should have their own lives (or at least appear to), instead of existing to solely serve the player character's interests. Believable virtual characters contribute to the overall interactive narrative experiences for players.

One reason for the simplicity of the behaviours in Neverwinter Nights and other games in this genre is the efforts required to create the scripts that control the behaviours. This is partly due to the fact that the scripting languages are usually at the same abstraction level as the game engine code. Therefore, instead of general purpose scripting languages, specialized behaviour scripting languages at a higher abstraction level have been proposed. One of them is ABL, used for the interactive game Façade [40] [41]. ABL is based on a previous goal-orientated language called Hap [3]. With ABL/Hap, behaviours are used to accomplish goals, and are specified as sequences of steps, such as *wait*, *act*, and *subgoal*. A *subgoal* step creates a new goal that must be accomplished in order to complete this step. ABL extends Hap by providing support for concurrent behaviours (by executing behaviours and goals in parallel) and support for synchronizing multiple actors (by having joint goals). Furthermore, Gomes and Jhala [32] have demonstrated the use of ABL in the

believability of NPC social conflict resolution. However, even with such better scripting languages the production costs are high. The previous ABL example uses extensive hand-written ABL scripts to control only two virtual characters in one scene.

2.2 Generative Design Patterns

Graphical drag-and-drop interfaces and a pattern-based approach to scripting have been explored. With pattern-based scripting, commonly occurring interactions and events in a game are generalized into libraries of generative design patterns that are parameterized and reusable [44]. Each pattern can then be used in multiple situations by customizations. In the academic environment, CMU's Alice [49], MIT's Scratch [50], and University of Alberta's ScriptEase [55] are three examples of pattern-based scripting. These tools are all aimed at story designers or those who are learning to program.

ScriptEase II is a game-independent scripting tool developed at the University of Alberta. Built with game platform independence in mind, ScriptEase II is able to read the objects in a game scene file and then allow a designer to drag-and-drop from a library of causes and effects to form a desired story, utilizing objects in the game (Figure 4). ScriptEase II automatically generates the scripting code for the game, based on the story created by the designer in the ScriptEase II tool.

Some commercial game engines have employed visual interfaces as well, such as the Blueprints visual scripting system of the Unreal Engine [2], formerly known as Kismet. While these tools are able to provide event-based scripting capabilities, they are general purpose tools, which are not specialized in generating believable

[2] www.unrealengine.com

behaviours for virtual characters. Since the tools are pattern-based, behaviours could be generated from behaviour patterns.

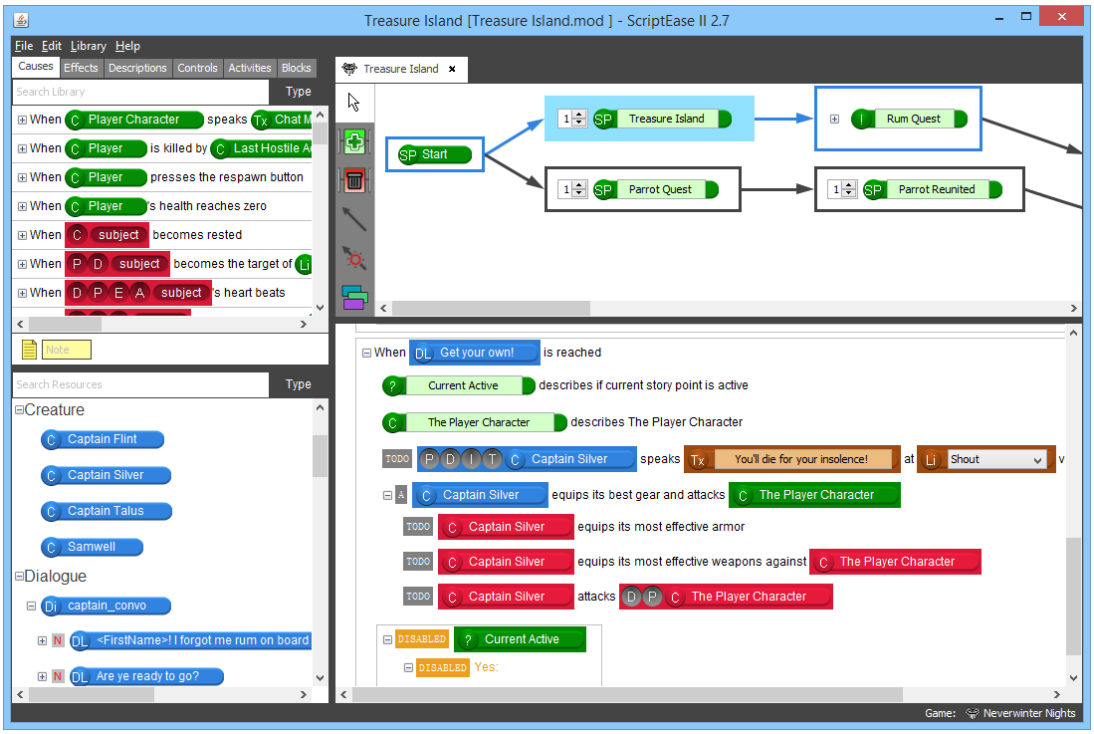


Figure 4. Screenshot of ScriptEase II.

2.3 Multi-Queue Behaviour Architecture

While ScriptEase is a general-purpose scripting tool, Cutumisu [22] introduced a library of reusable behaviour patterns for ScriptEase. Behaviours of virtual characters are categorized according to the behaviour ontology shown in Figure 5. Behaviours are divided into independent and collaborative behaviours. Independent behaviours are performed by one virtual character, while collaborative behaviours are performed with a partner. Behaviours can also be classified as proactive (acting spontaneously), latent (triggered by an event in the game external to the virtual character), or reactive (reacting to a partner's initiative in a collaboration, where the initiating behaviour can proactive or latent).

Cutumisu summarizes the requirements for a virtual character behaviour architecture using the following nine words: adaptability, clarity, effectiveness, variety, autonomy, alertness, interactivity, reusability, and scalability. Based on these requirements, Cutumisu introduced a multi-queue behaviour architecture [24] that allows for the interruption and resumption of the behaviours.

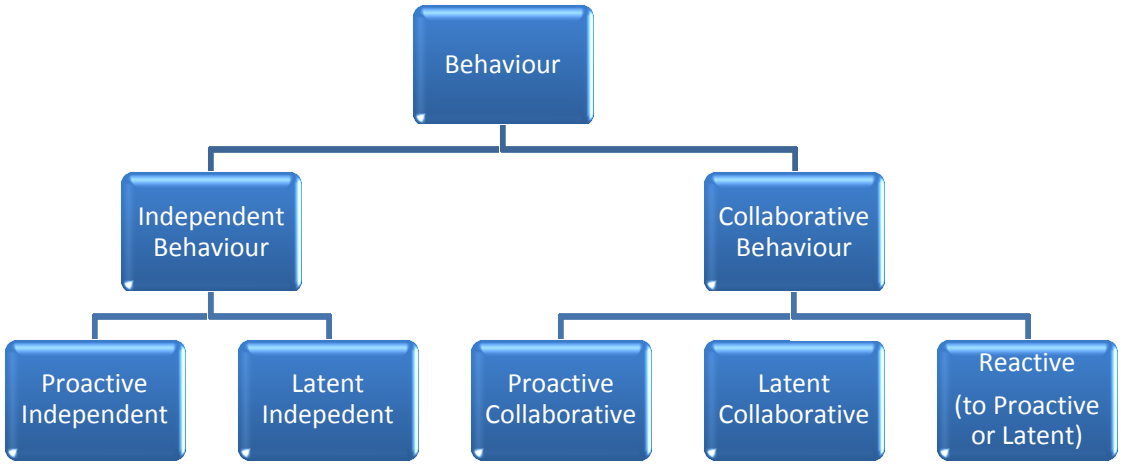


Figure 5. Behaviour classification, adapted from Cutumisu [22].

With the multi-queue behaviour architecture, a virtual character deploys multiple kinds of action queues to organize their behaviours. A queue can be proactive independent, proactive collaborative, or latent. Proactive independent queues store proactive independent behaviours; proactive collaborative queues store proactive collaborative behaviours and reactive behaviours, and latent queues store latent independent and latent collaborative behaviours. Each behaviour has a priority, which determines whether one behaviour can interrupt another. For example, as a tavern patron, "order drink from bartender" may be a proactive collaborative behaviour, with bartender as the partner of the collaboration. This behaviour can have priority 1. If a higher priority latent behaviour is triggered by an event, such as when the player character approaches the tavern patron (with priority 2), then the "order drink" behaviour can be interrupted to allow for a latent "shout out a one-liner to the player character" behaviour. This "shout out" behaviour is often

employed as a game mechanic to give the player some hints about their current quest or to suggest starting points for other quests.

The multi-queue behaviour architecture also allows for the resumption of behaviours. Interrupted behaviours remain on their respective queues and can be resumed when higher priority behaviours are finished. The previously mentioned tavern patron will return to the "order drink" behaviour after giving the player the hint. Queues also have time-outs to prevent dead-lock. Although ScriptEase includes an implementation of this Behaviour architecture and the architecture works well, the behaviour patterns have not been as popular with ScriptEase users as non-behaviour patterns. This could be due to the complexity of the ScriptEase text-based interface and the graphical user-interface of ScriptEase II may encourage more use of behaviour patterns.

2.4 Finite State Machines and Behaviour Trees

When creating behaviours for virtual characters, other alternatives to manual scripting exist. Finite State Machines (FSMs) have been used in commercial games for many years. FSMs are one of the most frequently used methods in first-person shooters [35]. FSMs contain a finite number of states and transitions between states. The machine can only be in one state at a time step, meaning only one state can be the current state. Time is divided into discrete steps. In the context of virtual character behaviours, each state represents an action sequence that the character takes if this state is the current state, and each state transition represents a change in condition in the game. While FSMs are easy to program and easy to understand in simple situations, with increasingly complex game environments, FSMs do not scale well and become harder to maintain [57]. Hierarchical FSMs improve this by grouping related states into sets of states, allowing states in each group to reuse and share the same transitions. This reduces transition logic and brings about a simpler structure to understand [16]. However, grouping states and reusing transition logic is

not trivial and may take a lot of effort to achieve, and Hierarchical FSMs have the same expressive power as their non-hierarchical counterparts.

Behaviour Trees were introduced with the first-person shooter game Halo [35] and since used in other games, such as Spore [34]. Behaviour Trees are static hierarchical directed acyclic graphs with behaviour sub-trees as nodes. A node can have multiple parents, which allow for reuse. Further improvements include Data-Oriented Behaviour Trees and Event-Driven Behaviour Trees which reduce memory consumption and reduce nodes traversed in the tree [17]. More recently, learning Behaviour Trees have been proposed, which adapt behaviour trees with known player traces. The resulting virtual characters produce behaviours that resemble human players. This is useful in a massive-multiplayer online game environment since player traces are readily available [64].

2.5 Reinforcement Learning

Research has been done on improving the behaviours of virtual characters in specific situations, such as virtual characters in an arena-style combat [25] [69], and companions who learn to help the player character or learn not to help if it is too disadvantageous for the companion [59]. In both cases reinforcement learning techniques have been shown to improve the resulting behaviours. Reinforcement learning (RL) [61] is an on-line learning technique based on the notion that an agent (a virtual character) should take actions to maximize future cumulative reward. The agent does so by observing the environment, which provides positive or negative reward to the agent for the actions the agent takes in the environment. Time is divided into discrete steps and the agent takes only one action at any one time step. With RL, virtual characters can dynamically change their behaviours to adapt to changes in the environment, including actions of the player. This approach can provide players with a gaming experience that is more interesting.

In this RL-based research, traditional RL techniques have been augmented with adaptive learning rates, action-dependent learning rates and adaptive exploration rates. These approaches are consistent with the WoLF learning policy of "learn quickly while losing, slowly while winning." [14] Other improvements include a double-reward system, softmax-greedy action selection policy, and agent modelling that allows a virtual character to build a model of another character (e.g. the player character) and act accordingly. These improvements have been shown to be effective in a few well-defined sub-domains such as combat or companion behaviours. While small-scale combat is an interesting topic and has been explored by researchers [48] [60], it has specific properties that do not apply to general behaviours, such as a well-defined goal of "winning" a combat.

2.6 AI Director

AI has also been introduced into games to play the role of an overarching "director". In these cases, AI is not used on individual characters to produce their behaviours, but rather as a story and drama-management mechanism. In this view, virtual characters exist to serve the intended story of the designer. A famous example is in the commercial first-person shooter game Left 4 Dead [65]. Left 4 Dead deploys an adaptive drama pacing mechanism using an AI Director [12]. Enemies and items are not spawned in fixed locations. Instead, the AI Director places enemies and items in places it determines during game-time, based on the player character's status and location. For example, if the player is killing enemies fast, then more enemies are spawned sooner and faster. The system also understands pauses in action that can lead to greater suspense later. This system aims to create a dramatic experience tailored to each player and it has received positive reviews. The sequel, Left 4 Dead 2, features an improved AI Director that is also able to alter the level layout to provide a more dynamic environment. However, the Left 4 Dead series features simple first-person shooter games with minimal story-line, and the AI Director does nothing to alter the game plot-wise or character-wise.

The PaSSAGE (Player-Specific Stories via Automatically Generated Events) system [63] is another example of dynamic content control. PaSSAGE is able to present the player with a dynamically selected subset of all content created by game designers in an interactive story-based game. The system models the player via the choices the player made previously in the game, and matches player models with the content most preferred by that player. As a simple example, if the player prefers to solve a dispute by fighting, then game content involving fighting will be more likely to appear in the story. This system provides dynamism in the story-telling of the game. PaSSAGE also has some limited dynamism in the behaviours of the virtual characters. When a quest is dynamically selected for a player, the engine dynamically selects virtual characters to fill quest roles based on location. However, this selection is used to serve the quest for the player character and do not depend on the behaviours of the virtual characters filling the quest.

2.7 Planning

Planning techniques have been deployed in some games. In the popular life simulation game series The Sims [27], virtual characters have basic motives, such as "hunger" and "social", which drive their choice of actions. An advertisement is attached to a game object to define how interaction with the object can satisfy these motives. For example, the "hunger" motive drives a virtual character to obtain food. This can be seen as a one-tier planning system, with a search algorithm of depth one. In contrast to story-based games with complex levels of interactions between virtual characters and the player character, the main objective of the virtual characters in The Sims is not to interact with the player, but to live out their lives in the world without specifically telling a pre-designed story. The Sims Stories is a set of expansions for the Sims, which adds a story mode, but the story mode consists of mainly hand-scripted sequences of events. Applying the behaviour system in The Sims to a story-based game would therefore be difficult, as the amount of scripting required would increase substantially.

While the technique deployed in The Sims is relatively simple, games such as *F.E.A.R.* and *S.T.A.L.K.E.R.: Shadow of Chernobyl* use goal-oriented action planning (GOAP) for their virtual character behaviours [46]. These commercial first-person shooter games use planning systems to direct the actions of their enemy characters. In these games, the actions of the enemy characters are usually very limited, such as to run, to hide, or to shoot. Since these virtual characters in first-person shooters all have a relatively narrow combat-related role, the planning system only needs to work in a very specific situation. Even so, complex behaviours such as team co-operations and flanking are not produced by the planner and have to be manually designed based on the map terrain. Kearney [36] describes a planning system using GOAP that applies to an environment a little more diverse, where a virtual character wants to achieve one of five goals (KillEnemy, MoreGold, MoreLand, MoreProduct, MoreSeeds) with 12 actions. No evaluations were described for this system.

Planning approaches have also been used in a Real-Time Strategy (RTS) game [18] and an RPG scenario that uses an RTS engine [21]. Since imperfect information, stochasticity, and simultaneous moves are inherent to these games, these are the some of the difficulties a planning system must overcome. One of these approaches is to combine planning with Monte Carlo tree search (MCTS) [18]. The planning system was applied at a strategic high level, instead of at the individual action level. The authors claim that the impact of individual actions requires a very deep search to see the consequences of the actions and thus it is not worth the effort or time, while searching at a high level allows the program to envision the consequences of actions much further into the future. This makes sense in an RTS game where the actions of individual units are not the focus of player attention. A similar idea of using high level “macro-operators” and its enhancements to speed up search is presented in the classical planning domain [13]. With virtual characters in a story-based game, the search space is relatively smaller since individual actions need not be as detailed as RTS units.

The story-oriented game The Elder Scrolls IV: Oblivion [6] claims to improve virtual character behaviour with its Radiant AI system. In Oblivion's pre-release interviews, designers claimed that the game's virtual characters are given goals that they must accomplish in a given day, and they must use their knowledge of the game world to find ways to accomplish these goals (e.g., to get food, virtual characters can buy, hunt, or steal). However, beta testers found hilarious situations resulting from virtual character behaviours, mostly involving virtual characters killing each other for food or other necessities. The final release of the game features a much more restricted version of the behaviour system. The behaviour of virtual characters is said to have improved with the next game in the series, The Elder Scrolls V: Skyrim, with virtual characters having a larger set of plausible actions to perform [5]. However, most characters still follow a fixed schedule and goals are missing.

The Elder Scrolls series is not the first game series to deploy schedules for its virtual characters. The Ultima series, starting with Ultima V: Warriors of Destiny [45], have virtual characters that follow daily routine schedules. However, these were hard-coded rudimentary schedules. Characters were hard-coded to follow one set of destinations and they would follow it throughout the entire game without ever changing. The Elder Scrolls V: Skyrim would allow a design to specify priorities and conditions for a character’s destinations, thus allowing for multiple schedules in a single character.

Kelly et al. [37] have taken The Elder Scrolls IV: Oblivion and implemented a custom offline planning system for the behaviours of its characters. The authors stated that real-time planning is computationally expensive and that plans have to be available in real time, while the CPU and memory resources available to game AI modules at runtime are limited. From its experimental results, generating plans for 40 virtual characters for 4 in-game hours require 12.32 seconds. This is likely not acceptable in real time, as in an actual game only a tiny fraction of the CPU resources is given to the AI system, as opposed to the full CPU resources utilized by

the experiments. The authors also indicated that the generated scripts require human effort to "compose and debug."

When examining the daily behaviours of a character in an open world story-based game such as the latest titles in The Elder Scrolls series, at a high level, behaviour specification can be viewed as a planning exercise since the designer should be able to generate behaviours from a set of general constraints. Kelly et al.'s goal of shifting the workload offline is shared in this dissertation, but this dissertation proposes a broader architecture that embeds offline scheduling and supports dynamic online assignment of low-level behaviours (roles) to the root nodes (objectives) of the scheduler.

With these vastly different approaches to virtual character behaviours, it is clear that comparing the different approaches for their advantages and disadvantages is no trivial task. The next chapter describes some standard metrics for evaluating behaviour architectures, and proposes a new tiered behaviour architecture to address some of the issues mentioned in this chapter. Chapter 4 describes an implementation of the proposed architecture that provides a graphical user interface with different constraint types that user studies have shown to be efficient and reliable.

3. Behaviour Architecture for Virtual Characters

Architecture is the foundation of a software system and its design is necessary to ensure the success of the final product. As American software engineer and Distinguished Professor Barry Boehm puts it, "Marry your architecture in haste and you can repent in leisure." [11] Without a carefully designed architecture, no amount of clever implementation can cover up the deficiencies in the quality of a system. The term software architecture is defined below by Bass et al. [2]

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. [2]*

Designing a good architecture is not an easy task, and it is even harder to quantitatively measure how "good" an architecture is. In this dissertation, I look at behaviour architectures, which are a specific type of software architecture aimed at providing a structured model for believable virtual characters. Since behaviour architectures have very specific applications, some qualities of general architectures apply while others less so. The next section aims to provide a set of standard measurements to an architecture specific to behaviours of virtual characters in story-based games and simulations.

3.1 Metrics for Evaluating a Behaviour Architecture

Chapter 2 discussed many different approaches to providing behaviours for virtual characters. The diversity in these methods suggests that it would be very helpful to have a common well-defined evaluation system. Researchers have examined the problem of evaluating a general software architecture. Ten attributes are defined in an Architecture Tradeoff Analysis Method (ATAM) [20] which measure qualities

ranging from the security of the system to the portability of the system to multiple computing environments. They are named as performance, reliability, availability, security, modifiability, portability, functionality, variability, subsetability, and conceptual integrity.

Metrics are broadly divided into two categories: directly-measured metrics, and indirectly-measured metrics. Directly-measured metrics are recorded using objective measurements, such as time spent, while indirectly-measured metrics are those more subjective, such as performance as gathered from a questionnaire. A behaviour architecture is confined within the structure of a game or virtual world, thus extraneous factors such as security need not apply. Based on the characteristics of a behaviour architecture, this thesis proposes four main metrics for evaluating a behaviour architecture, namely: architecture expressiveness, architecture performance, architecture quality, and architecture usability.

- *Architecture Expressiveness*: The expressiveness of an architecture measures the expressive power, the different types of virtual character behaviours the architecture can represent. Expressiveness can be measured as a percentage value from a pre-determined set of virtual character behaviours.

- *Architecture Performance*: Performance refers to the consumption of resources by the architecture during game time. Performance can be measured through the impact of the architecture on frame rates. Frame rate is the frequency at which the computer screen produces unique images called frames. In a video game, it is especially crucial for a game to be able to produce an adequate number of frames per second to produce a smooth graphical gaming experience.

- *Architecture Quality*: Quality applies to the resulting behaviours of the virtual characters employing the architecture during game time. Quality of an architecture can be a relative measure in comparison to other architectures or

it can be an absolute measure of believability. This is an indirectly-measured metric that is usually gathered from user feedback.

- *Architecture Usability*: Usability measures how reliable an implementation of the behaviour architecture, such as a tool or toolset is and how efficient a user can be with the implementation of the architecture. A usability study can collect data on the completeness and correctness rates of behaviours generated by users with the architecture toolset, as well as time required to generate behaviours using the architecture toolset. Efficiency has been defined by Alberta and Tullis as completeness rate over time [1]. In addition, they report that, "in almost every situation, the faster a participant can complete a task, the better the experience." Therefore, the efficiency measure also provides an indirect measure of designer experience.

3.2 Tiered Behaviour Architecture

You return to the world of The Elder Scrolls V: Skyrim. You are back at the city of Solitude. As you watch the citizens of Solitude go about in their daily lives, you wonder if the characters could exhibit a little more variety in their schedules. Perhaps Greta does not have to leave her house at exactly 3pm every single day; perhaps Beirand does not have to work at the smithy day after day without a break; perhaps Addvar could have a second job importing the fish he sells every day at the market.

Before we dive into the details of how these behaviours can be implemented, we should take a look at the bigger picture. What kind of functionalities should an architecture for behaviours support? How can a behaviour architecture best represent these functionalities? This section answers these questions by introducing a new virtual character behaviour architecture that generates behaviours for the daily lives of virtual characters.

Most story-oriented game worlds are comprised of hundreds of smaller areas (or scenes). Using a medieval setting as an example, these areas may be houses, markets, taverns, city alleys, city gates, blacksmith shops, etc. At a grand scale, the day-to-day routine of a virtual character usually consists of accomplishing several objectives: sleeping, eating, working, and social activities. The virtual character determines the scene (home, tavern, etc.) and the role (sleeper or eater at home, patron or server at tavern etc.), where these objectives will be satisfied. These decisions are affected little by the specific actions that the virtual character has to perform in a given role in a given scene. Therefore, it is natural to divide a behaviour model for these characters into two levels, as shown in Figure 6.

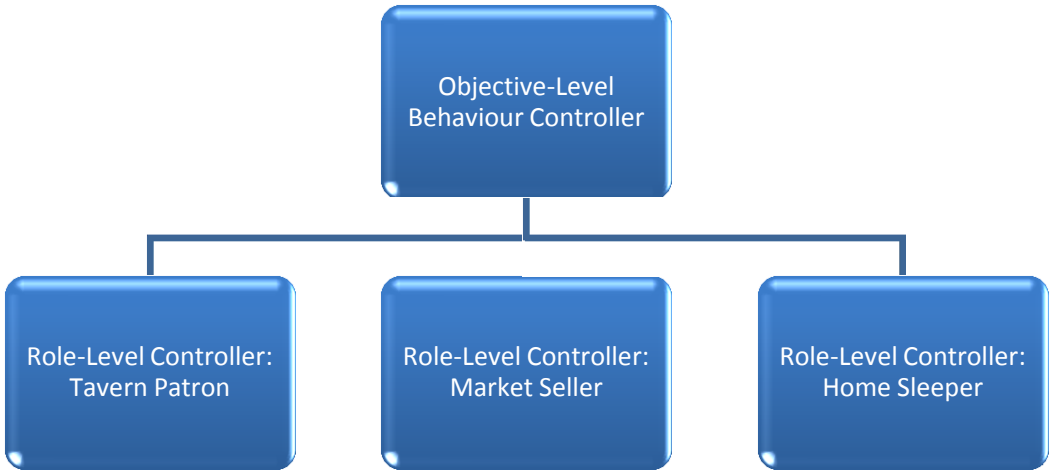


Figure 6. The two parts of the behaviour architecture for a virtual character.

The model divides a behaviour controller into two parts, objective-level and role-level. Role-level controllers are specific to each role, such as a tavern patron, or a market seller. These role-level controllers are modular and reusable in that the same objective-level model can be used with many different role-level models. Self-contained techniques such as finite state machines, behaviour trees, or reinforcement learning can be used to produce role-level behaviours for specific roles as discussed in the previous chapter. Chapter 5 will introduce a new technique called Behaviour

Capture which is shown to produce believable behaviours at this role level and is accessible to general game designers.

To explain what the objective-level controller should be capable of, let us go back to our example. Our virtual character wants to accomplish several objectives on a daily basis: sleep, eat, work, and engage in social activities. The objective-level controller should determine the daily schedules consisting of these objectives and takes the virtual character to different locations to assume different roles that will satisfy these objectives.

For each objective, there can be multiple roles that satisfy the objective. For example, for the Eat objective, there can be a number of ways to fulfill this objective: Eat at home, Eat at a friend's place, Eat at a tavern, etc. We can view an objective as a set of roles and can use a *selector* to map the set of roles that satisfy an objective to a particular role that the virtual character will take in a particular scenario.

A list of objectives forms a schedule. When we view a schedule as a list of objectives, time serves as a natural ordering mechanism. A simple schedule is a daily 24-hour schedule, with one objective at each hour. A different selector can be used to map a schedule (list of objectives) to one objective in the schedule at a particular time.

Multiple schedules are grouped into a circumstance. Multiple schedules can be useful in instances where a game designer would like a virtual character to maintain multiple different daily routines, such as a weekday routine and a weekend routine. Another selector is used to map a circumstance into a particular schedule in the circumstance, such as selecting a weekday schedule or weekend schedule.

Circumstances usually represent important life segments of a virtual character. Changing circumstances are usually the result of life-changing plot events, such as the marriage of a character, or the game story moving into the next act. If a character

has multiple circumstances, then another selector picks a particular circumstance based on game context. Finally, at the highest level, there is a virtual world with a set of virtual characters being controlled by the behaviour architecture, where the selector picks the character of interest. The fully-expanded behaviour architecture is expressed using the hierarchy shown in Figure 7. This model is called the *Tiered Behaviour Architecture* model.

In this view, while the role-level behaviours are still generated by low-level behaviour controllers, the high-Level behaviour controller is expanded in a hierarchy of Virtual Characters, Circumstances, Schedules, Objectives, and Roles. Roles are atomic units in the architecture. Once a role is selected, generation of behaviours is passed onto low-level behaviour controllers.

The hierarchy is made up of alternating data layers and selectors. Each data layer is a collection (set or list) of items, where each item is composed of items from the next lowest layer. Formally, if we use L_0 to L_5 to denote the six layers, then

L_0: $\{Character_1, Character_2, \ldots, Character_n\}$
L_1: $Character_i = \{Circumstance_{i1}, Circumstance_{i2}, \ldots, C_{io}\}$
L_2: $Circumstance_{ij} = \{Schedule_{ij1}, Schedule_{ij2}, \ldots, S_{ijp}\}$
L_3: $Schedule_{ijk} = [Objective_{ijk1}, Objective_{ijk2}, \ldots, O_{ijkq}]$
L_4: $Objective_{ijkl} = \{Role_{ijkl1}, Role_{ijkl2}, \ldots, R_{ijklr}\}$
L_5: $Role_{ijklm}$ = { basic role-level behaviours }

Each layer L_i is a set except L_3, which is a list. A schedule is a list of objectives. Since a schedule is based on time, time serves as a natural ordering mechanism for the objectives. We use a selector to choose one item from each layer at any given time. The *selector* is a mapping σ_s from L_s to L_{s+1}.

Selector: $\sigma_s (L_s) \rightarrow L_{s+1}$ for $0 \leq s \leq 4$

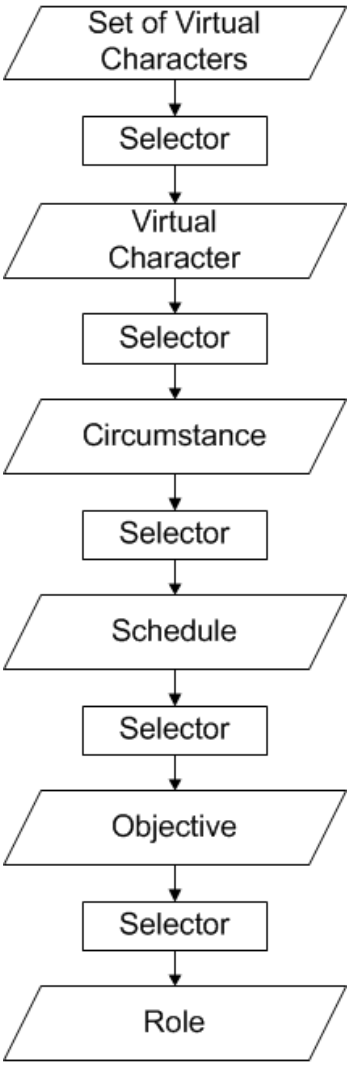


Figure 7. The Tiered Behaviour Architecture model, with High-Level Controller fully expanded.

A selector can be any mapping that maps a collection to a single item. Examples of selectors include:

- *Time selector* picks an item from a list based on a particular in-game time.
- *Event selector* picks an item based on an event that happened in the game world.

- *Probability selector* picks an item based on pre-set probabilities for each item.
- *Character selector* picks one virtual character from the set of virtual characters being controlled.

In addition, each of these simple selectors can have a filter attached to it that filters the layer as part of the selection process, to signify the availability of the items in that layer at a particular instance in the game.

In the domain of interactive storytelling, decision time can either be offline (before system deployment) or online (after system is deployed – authorship still affects story decisions) [62]. To address the issue of the long running time of an online algorithm, the Tiered Behaviour Architecture contains two components, an offline generation component, and an online selection component.

In the Tiered Behaviour Architecture, the items in each data layer are generated offline statically, while the selectors are used at game time to dynamically pick the items. Each static data layer is generated according to some requirements set by the game designer. This process can take a considerable amount of time depending on the complexity of the requirements, and this may not be feasible while the game is being played. The static component of this mechanism can save considerable game time when the generation of objectives for each schedule is complex. Chapter 4 will describe the algorithm used to generate the data layers. In the next section, an evaluation of the architecture is presented.

3.3 Evaluation Results

To explore the plausibility of the tiered behaviour architecture, this section presents the results of the evaluations of the Tiered Behaviour Architecture according to the metrics proposed in Section 3.1. In order to generate measurable statistics, the

architecture model needs to be implemented for an actual game engine. An implementation in a commercially-available game engine would be preferable since the goal is to provide evidence that the architecture can be used in commercial games. As previously described, The Elder Scrolls V: Skyrim is a relatively recent commercial game. It is powered by Bethesda's own Creation Engine and Bethesda released a Creation Kit that can modify various aspects of the game world, including the default behaviours of the virtual characters. This is ideal for evaluation purposes so the Tiered Behaviour Architecture is evaluated in Skyrim.

3.3.1 Expressiveness

Can the Tiered Behaviour Architecture express the behaviours used by state-of-the-art virtual characters in recent commercial games? While many story-based commercial games do not have virtual characters that go beyond walking between a set of waypoints, the recent games in the Elder Scrolls series provide good examples as they have virtual characters that follow daily schedules. The latest game in the series, Skyrim, has arguably the most mature daily schedules for its virtual characters with the updated Radiant AI system [43].

Using the Skyrim Creation Kit, the virtual characters of Skyrim were examined, specifically at how the virtual characters behave on a daily schedule. A measurement of expressiveness is the percentage of characters from Skyrim that can be represented by the Tiered Behaviour Architecture. There are an unlimited number of virtual characters in the game as some virtual characters are dynamically generated, therefore a subset must be chosen. Only named characters that persisted in the game world were included in the study.

The Unofficial Elders Scrolls Pages (UESP) is a wiki site where players and fans of the game have come together and describe the various parts of the game in details. The wiki lists all named characters in Skyrim together with detailed descriptions of their behaviours [66]. An inspection of these named characters in Skyrim reveals

that their daily schedules can be generated by the Tiered Behaviour Architecture, based on the descriptions given. However, to be sure, the actual scripting code should be inspected.

For this expressiveness evaluation, a sample large city, Solitude, was chosen as a fair representation of all areas. Solitude contains eighty-five named characters, including food vendors (merchants), bards, farmers, blacksmiths, soldiers of different kinds, and other citizens. These characters represent many different professions in the game.

Inspecting the behaviour code using the Skyrim Creation Kit for each of these characters confirms that the Tiered Behaviour Architecture is able to reasonably express the behaviours of all the Solitude virtual characters, giving us an expressiveness rate of 100% of the Solitude virtual characters. In this measurement, specifically-designed in-game cut scenes in which some of the characters appear in are excluded as they are not part of the daily lives of these characters.

Greta is an example of a character with one of the most complex behaviours in Solitude. Her behaviours are comparable to the most complex behaviours elsewhere in Skyrim. We met her briefly in Chapter 2, and here is a detailed account of her life in Skyrim: if her husband Addvar is dead, she will go to the market at 6am and stay for 14 hours selling goods before going back home for the night (let us call this Schedule 1). Otherwise, if the player completed the quest "Return to Grace", she will go to a temple at 6am and stay for 9 hours. At 3pm, she will go wander around near a well for 3 hours before going back home (Schedule 2). If the above quest is not completed (and Addvar is alive), she will sleep until 8am, do some housework until 3pm, and then go wander around the well as before (Schedule 3). These behaviours can be expressed with three different schedules in our Tiered Behaviour Architecture, managed by a default circumstance and two additional circumstances, "Addvar Dead" and "Addvar Alive and 'Return to Grace' Quest Completed", as shown in Figure 8.

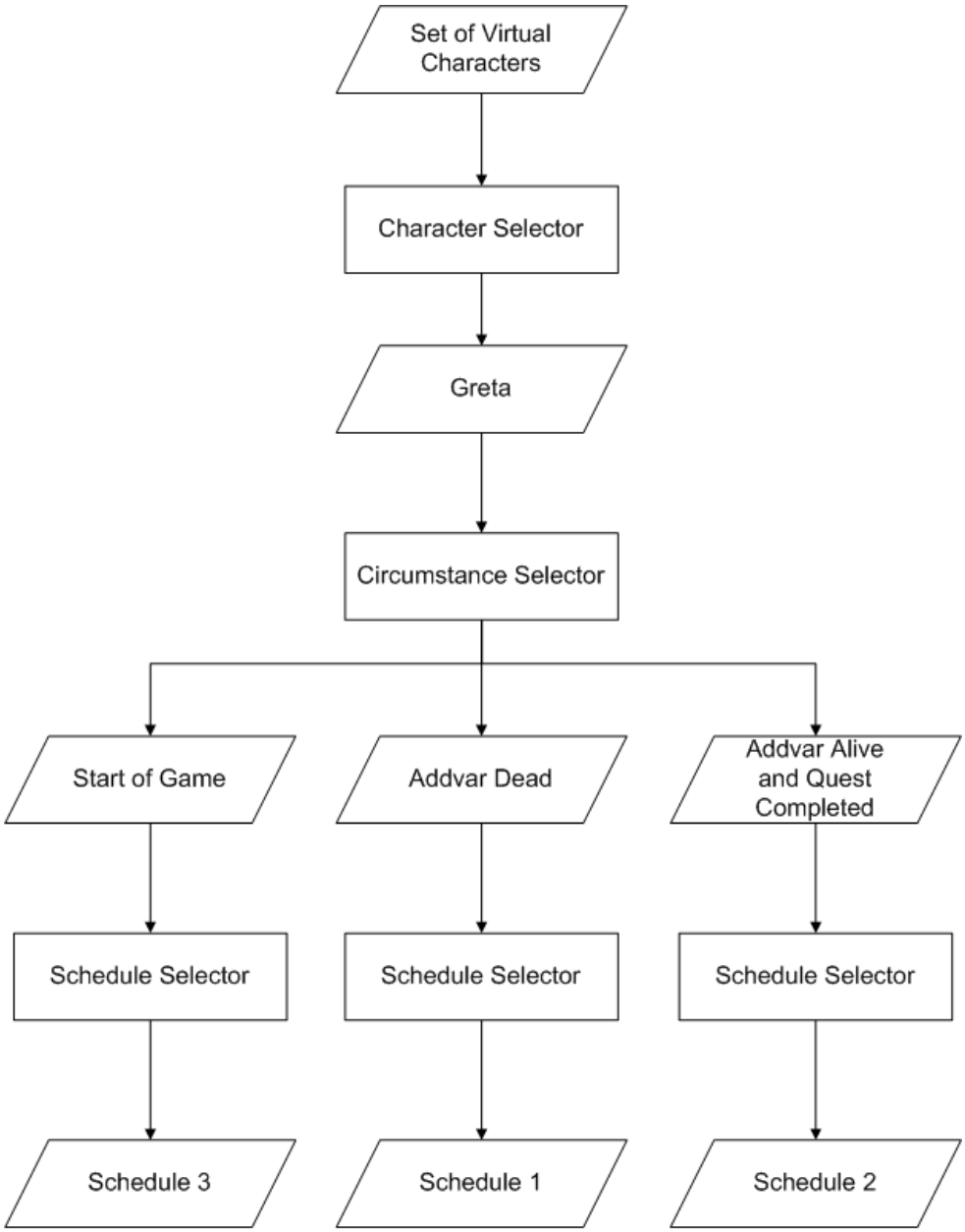


Figure 8. The Tiered Behaviour Architecture model representing Greta.

3.3.2 Performance

The performance of the architecture is measured through frame rates. Researchers have shown that a suitable frame rate in first-person shooter games allows game players to play noticeably better than non-suitable frame rates [19], where a frame

rate of 60 fps provides a 7-fold increase in the score the player received over a frame rate of 3 fps. The same study has also shown that users perceive higher frame rates as better picture quality. Therefore, frame rate is used as an indication of game performance.

To determine whether the overhead of dynamic scheduling and the new behaviours would perceptibly reduce frame rates, we followed the Greta character throughout the city of Solitude and measured frame-rates with and without the code for the behaviours from the architecture.

Performance was measured with two machine settings, a high-end gaming computer and a low-end laptop computer. The high-end computer has an Intel Core i7-3930 processor at 3.20GHz, 16 GBytes of memory and an NVidia GeForce GTX 680 graphics card with 2 GBytes of video memory, while the more modest computer has an Intel Core i7-2670M processor at 2.20GHz, 8 GBytes of memory and an NVidia GeForce GT 525M graphics card with 1GByte of video memory.

On the high-end gaming computer the frame rates during normal activities varied from 59 to 60 FPS whether our Tiered Behaviour Architecture was used or not. Other factors affected the frame rate more than the architecture. For example, whether the architecture was enabled or not, the frame rate dropped to 52 FPS when children were playing nearby. On the more modest computer the frame rates varied from 9 to 15 FPS whether the architecture was used or not.

On the high-end computer an ENB[3] was used to measure frame rate, and on the low-end computer FRAPS[4] was used. The results showed that the Tiered Behaviour Architecture did not affect the performance in a measurable way on either computer.

[3] http://enbdev.com/

[4] http://www.fraps.com

3.3.3 Quality of Behaviours

Are the behaviours created by the proposed Tiered Behaviour Architecture a viable alternative to typical commercial game virtual character behaviours? To answer this question, a user study was designed to compare the behaviours created by the architecture and the default behaviours in Skyrim.

To measure the quality of the behaviours generated by the architecture, a single city populated by virtual characters was presented to users (Solitude in Skyrim). As the observed character walks into local scenes (for example, a tavern building), a fade-out/fade-in effect was used to show only the transitions to and from the local scene. Activities inside local scenes were not presented to focus participants on the daily schedule aspect of the architecture. Evaluating behaviours in local scenes will be discussed in detail in Chapter 5 of this dissertation.

In this evaluation of behaviour quality, study participants were asked to watch six sets of behaviours that implemented the daily lives of a virtual character, each set of behaviours generated using one method. The six behaviour sets were presented as in-game videos. Each set of videos focused on the daily lives of one observed character, Greta, over three days. All six Gretas look identical (Figure 9 and Figure 10), and the six sets of behaviours for her were performed in identical world settings (the city of Solitude in Skyrim).

These are the hypotheses that were created and tested:

- Daily behaviour deploying stochastic schedules would be more believable than behaviours deploying only fixed schedules.
- Daily behaviours deploying multiple schedules would be more believable than behaviours deploying a single schedule.
- Daily behaviours deploying multiple roles to satisfy objectives in a schedule would be more believable than behaviours deploying only a single role per objective.

- Daily behaviours deploying dynamic roles would be more believable than fixed roles.

The six behaviour variations are listed in Table 1. The difference between a fixed schedule and a stochastic schedule is that a stochastic schedule supports a maximum plus or minus one hour duration for each objective. Dynamic roles imply that the roles are constantly checked for validity and dynamically switched to a different role that satisfies the same objective if one role becomes unsatisfiable. The behaviours SS and MS are default Skyrim behaviours. The MSSMDR behaviour showcases the most complex capabilities of the Tiered Behaviour Architecture model.

Figure 9. Greta, the main character in each set of videos in the user study, is leaving her house in this screenshot.

Figure 10. Greta is seen working at a stall in the market.

Behaviour	Details
SS	Fixed Single Schedule, with Single Roles
SSS	Stochastic Single Schedule, with Single Roles
MS	Fixed Multiple Schedules, with Single Roles
MSS	Stochastic Multiple Schedules, with Single Roles
MSSMR	Stochastic Multiple Schedules, with Fixed Multiple Roles
MSSMDR	Stochastic Multiple Schedules, with Multiple Dynamic Roles

Table 1. The six behaviour variations.

Here are the descriptions of the actual behaviours:

SS – Greta goes from her house to her market stall at 6am. She goes to the "Angeline's Aromatics" tavern at 3pm, then goes home at 6pm. The schedule is the same for three days.

SSS – This schedule is the same as SS except that the times of transition are stochastic, meaning that each time she goes to a place, she can leave any time (up to one hour) earlier than specified in the schedule.

MS – Here Greta has the same schedule as SS on the first two days and has a different schedule on day 3, where she goes from her house to church at 6am. She goes to the same tavern at 12 noon, then goes home at 6pm.

MSS – This schedule is the same as MS except that the times of transition are stochastic by one hour.

MSSMR – This schedule extends the MSS schedule with multiple roles for each objective. Instead of going to only the "Angeline's Aromatics" tavern, Greta chooses between this tavern and a "Bits and Pieces" tavern. Instead of working only at the market, she chooses between the market job and a bard job.

MSSMDR – This schedule extends the MSSMR schedule with dynamic roles, so that Greta is able to dynamically switch roles to go to a friend's house for the night upon seeing that the road to her own house is rendered inaccessible by fallen trees.

Note that Greta's default behaviour in Skyrim can be represented as an MS behaviour, except that instead of changing schedules according to the day of the week she changes schedules after some game events, such as when her husband is dead. Most Skyrim characters have SS behaviours, but some have MS behaviours that depend on game events.

After watching the main characters, participants were asked to rank and rate the characters according to believability of behaviours. Some demographic information was also gathered. The user study had 80 participants, who were undergraduate students taking a first-year psychology class. There were 50 females and 30 males.

Of these, 9 of the females were gamers and 41 were non-gamers, while 18 of the males were gamers and 12 were non-gamers. A gamer in this context is defined as someone who plays story-based video games at least once a week.

The resulting averages of ranking and rating scores are presented in Table 2. Ranking scores are from 1 to 6: for each participant response, the highest ranked behaviour received a score of 6, the second highest ranked received a score of 5, etc. and participants were not able to repeat a ranking for a different behaviour (no ties). Rating scores range from 1 to 4: 1-very unbelievable, 2-unbelievable, 3-believable, 4-very believable. The trends of rankings and ratings are consistent with each other, with MSSMDR as the best, indicating that stochasticity, multiple schedules, and multiple dynamic roles together make the best behaviours.

Ranked data was analyzed with a Friedman statistical test and rating data was analyzed with ANOVA. The results show that there are statistically significant differences in the results at 95% statistical confidence (p-value < 0.05). Paired T-tests at a confidence of 95% indicate that MSSMDR is better than each of the other alternatives.

Behaviour	**Average Ranking Score**	**Average Rating Score**
SS	2.54 (1.65)	2.01 (0.92)
SSS	2.64 (1.51)	2.05 (0.94)
MS	3.26 (1.20)	2.43 (0.78)
MSS	3.40 (1.32)	2.54 (0.86)
MSSMR	3.90 (1.51)	2.55 (0.83)
MSSMDR	5.26 (1.48)	3.35 (0.83)

Table 2. Average ratings and rankings of the behaviours. Standard deviations are shown in parentheses.

Figure 11 is a graphical illustration of the results of paired T-tests comparing the six behaviour variations, pair-wise. Starting from SS, adding a multiple-schedule to get to MS is significantly better. Adding stochasticity to either SS or MS is better, but not significantly. One potential reason for some people not perceiving stochastic schedules as more realistic could be the belief that individuals usually go to work at the same time every single day. This is consistent with some feedback I have collected at the end of the user study, in which there were comments such as "if the character does the same thing at the same time, I rather consider it more natural than otherwise." Finally, adding both the multiple roles and dynamic roles produce significantly better results. The raw p-values of the t-tests are shown in Appendix A.

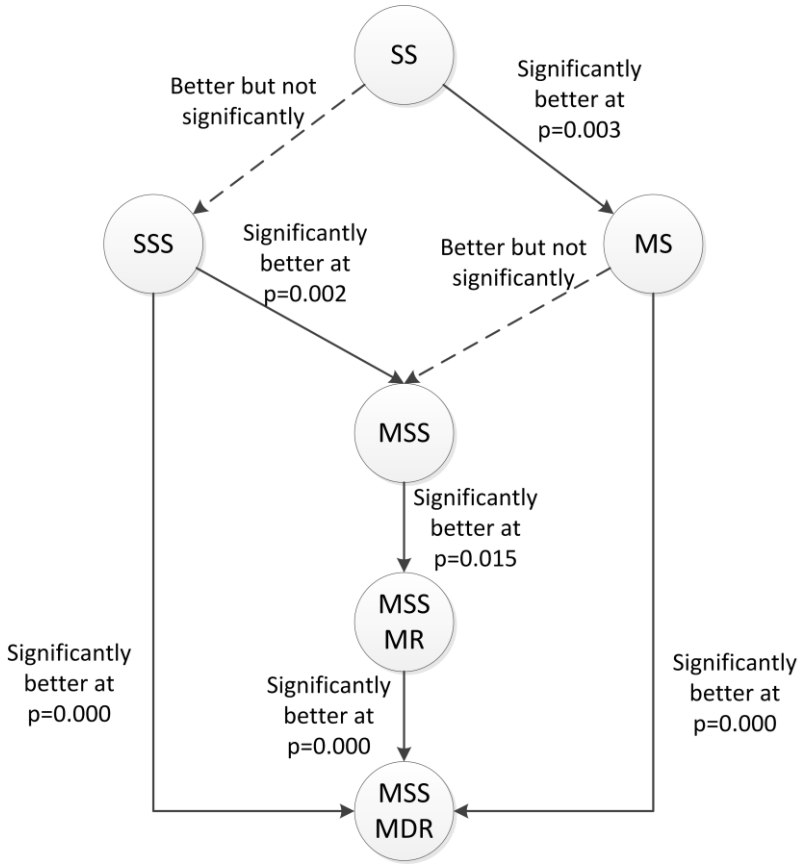


Figure 11. Statistical significance diagram comparing the rankings of the six behaviours with 95% confidence.

3.3.4 Gender Analysis

Is there a difference in ways males and females perceive behaviours? I would like to compare the responses of male and female participants to determine if it is true that one group perceived daily behaviours of virtual characters differently than the other group. Since the user study gathered participant gender information, it is relatively straightforward to segment the results according to gender and analyze the results. Please note that in the context of this dissertation, the word "gender" is used exclusively to mean biological gender, or biological sex.

Table 3 and Table 4 show that results of ranking and rating scores grouped by gender. For males, the results matched closely with the overall results. Those relationships with statistical significance in the overall results remained, and no additional relationships became statistically significant. For females, however, there was one minor difference, as it was no longer statistically significant for MSSMR to be ranked higher than MSS (with a t-test at 95% confidence). This suggests that for a female audience, multiple roles may not be enough to significantly improve believability – dynamic roles are necessary. More discussions on gender differences will continue in Chapter 5.

Behaviour	**Average Ranking Score**	**Average Rating Score**
SS	2.50 (1.68)	1.87 (0.90)
SSS	2.37 (1.33)	1.93 (0.87)
MS	3.30 (1.12)	2.40 (0.77)
MSS	3.40 (1.28)	2.47 (0.90)
MSSMR	4.10 (1.65)	2.57 (0.82)
MSSMDR	5.33 (1.30)	3.33 (0.80)

Table 3. Average ratings and rankings of the behaviours, for male participants. Standard deviations are shown in parentheses.

Behaviour	Average Ranking Score	Average Rating Score
SS	2.56 (1.64)	2.10 (0.93)
SSS	2.80 (1.60)	2.12 (0.98)
MS	3.24 (1.25)	2.44 (0.79)
MSS	3.40 (1.36)	2.58 (0.84)
MSSMR	3.78 (1.43)	2.54 (0.84)
MSSMDR	5.22 (1.59)	3.36 (0.85)

Table 4. Average ratings and rankings of the behaviours, for female participants. Standard deviations are shown in parentheses.

There are limitations in the user study results. The results discussed above are based on a sample of 80 university students, where 50, or 62.5%, are self-identified as female. The average age of the participants is 20.0. Therefore one must be cautious when applying the results to a different age group or a group with a different gender composition. As a reference, according to a study by the Entertainment Software Association of Canada, in 2014 the average age of a gamer (defined as someone who has played a video game in the past 4 weeks) in Canada is 33 years old, with 52% of them male [15].

4. Automated Cyclic Scheduling

The last chapter presented the results of applying three (expressiveness, performance and quality) of the four metrics to the Tiered Behaviour Architecture. The fourth metric (usability) can only be measured by having designers use specific tools that support the architecture. This chapter describes the most essential component of the Tiered Behaviour Architecture, Cyclic Scheduling, and a tool that supports it. It then applies the usability metric to this tool.

The Tiered Behaviour Architecture introduced in the last chapter, can be used to fill the world of The Elder Scrolls V: Skyrim with virtual characters who exhibit behaviours that are perceived as more believable than their default behaviours. Greta, the main example from the last chapter, can have one schedule for weekdays and one schedule for weekends. She can pick between different taverns to socialize and pick between different jobs to work at. In addition, the times in her day-to-day schedules are not fixed to the minute.

All these behaviours can be manually scripted by game designers, if given enough time and resources. However, with the large number of characters in Skyrim and other story-based open-world games, how can the effort to create these more believable behaviours be reduced so that they are feasible on a large scale? The short answer: with the help of AI techniques (planning, scheduling, learning), as described in this chapter and the next chapter.

One of the most tedious steps in creating the varying behaviours is the generation of daily schedules. Instead of a designer specifying every single detail of a daily schedule, there should be an algorithm at the high level that produces the objectives of the virtual characters and the roles that will satisfy these objectives, based on some minimal but essential requirements specified by the designer. Such an algorithm must be designed and implemented in a way that balances the amount of work game designers need to do with the level of control they desire over the virtual

characters. The amount of in-game computation must be at acceptable levels in a commercial game.

Chapter 2 described several AI techniques used to produce behaviours for virtual characters in different scenarios. In the next section, two approaches are examined in detail, planning and Monte Carlo tree search (MCTS). Examples of the shortcomings of these approaches are also described. These problems led directly to the creation of the Cyclic Schedule model that is presented in this dissertation.

4.1 Alternate Approaches that Led to Cyclic Scheduling

Since planning has been used in many other scenarios, perhaps an obvious approach to be considered uses a planning system that automatically fills in the daily schedule of a virtual character. With a planning system, a game designer needs to specify the initial world state, the available roles and their consequences, in terms of pre-conditions and post-conditions. The planner determines the daily life of the virtual character by producing an ordered list of roles at each time period. At each step of the list, the planner takes the virtual character into a specific role in a specific scene, where a low-level behaviour controller corresponding to the situation takes over. The control is returned to the high-level controller once the specific situation is either completed or interrupted.

In the following working example, a planning system is described and implemented. The world state is expressed as a motivation vector for each virtual character and pre-conditions and post-conditions are expressed in terms of motivation values. In this model, in order for the planner to generate roles, a game designer needs to specify motivations for a virtual character. A motivation is a scalar value describing some aspects of the well-being of the virtual character. Depending on how complex a game designer wants to build the virtual character, the game designer can specify a number of custom motivations. The popular life simulation game The Sims calls

them motives. The Sims has the following nine motives for their virtual characters: energy, comfort, hunger, hygiene, bladder, room, social, fun, mood [30].

In this working example, there is a virtual character, Adam, who works as a city guard at the gate of a medieval city. During the day, Adam goes to the city gate and assumes the role of a guard. When night falls, Adam is off duty and assumes other roles such as a tavern patron or a market customer. The high-level planner determines which role Adam will assume next. The motivations for Adam are: Money, Energy, Satiation, Social, and Resources, represented by a motivation vector [M, E, Sa, So, R]. Each motivation value is normalized to a number between 0 and 100 (since this scale is easier for non-technical designers than a real number between 0 and 1). "Resources" refers to non-monetary resources such as food.

The motivation vector can start with random values, or values determined by a designer, for example, [10, 10, 10, 10, 30]. A game designer can choose a goal for Adam (thus the goal of the planner for this virtual character) to be maximizing the motivation value in a fixed amount of time (which translates to the number of steps in the plan generated by the planner), or alternatively, to maintain a threshold for the motivation values, or to return to the starting motivation value by the end of the 24 steps if the goal is to define a cyclic behaviour. A step count of 24 is chosen so that they correspond to 24 hours in a day.

For the planner, the available roles for Adam might be: work at City Gate (as guard) (with precondition "job = city gate"), sleep at home (as sleeper), socialize at market (as buyer) (with precondition "M>10"), socialize in tavern (as patron), work in the woods (as hunter), etc. Pre-conditions are specified either with a requirement on a motivation value, or with additional variables such as "job". Post-conditions will change the motivation values or change additional variables. A plan might look like:

1. Sleep at home (as sleeper),
2. Work at City Gate (as guard),

3. Socialize in tavern (as patron),
4. Socialize at market (as buyer),
5. Sleep at home (as sleeper).

Once a plan has been produced, Adam assumes the roles according to the plan. During game-time, when Adam arrives at the specific scene, e.g. a tavern, the low-level controller that corresponds to this situation takes over Adam's behaviours, e.g. a controller designed for a patron in a tavern scene (see Chapter 5 for details). Each specific scene, when completed, modifies the motivations and additional variables of the virtual character in a stochastic way. For example, a tavern patron might have the motivation change [-5, -1, -1, 7, 0] (spend 5 money, lose 1 energy, get hungrier by 1, gain 7 points of social interaction and gain no resources). In computer programming terminology, these effects on motivations are analogous to post-conditions of functions.

One might ask: how are these effects on motivations determined? One way is to let the low-level controller determine these effects in an automatic way. Each low-level controller directly changes the motivation values through the roles performed. Using the tavern patron as an example, if Adam converses with another tavern patron, it will likely increase the "social" motivation and decrease the "energy" motivation for Adam. Such effects can be specified by a game designer. In calculating the overall effects of a scene, the planner can use the expected value of the sum of the motivation changes by the role in the scene.

This method, however, will require the specification of motivation change by each individual role in a scene, which may not be desirable. A simpler approach is to let the designer specify the motivation change for each scene at the high level, thus making it independent from the actions (such as to sit in a chair) performed at the low-level scene.

Each low-level controller can incorporate stochasticity in its behaviour generation. Therefore, each time Adam is in a particular scene, the scene plays out differently. There could be incidents where Adam's role as a tavern patron is interrupted, causing the scene to end prematurely. Virtual characters should be able to react to unexpected events. If the player causes an explosion in the tavern, tavern patrons should immediately stop what they were doing and run out of the building. This behaviour can be implemented using the interruptible and resumable behaviour architecture described by Cutumisu and Szafron [24]. If a scene is interrupted, then the high-level controller takes over, and the planner will re-plan for the next high-level role, taking into account the current motivation values. The system can also be set-up so that in addition to these interruptions, the planner could be forced to re-plan at regular intervals (for example, each 1 or 2 hours of game time). In this case, if the planner puts Adam in the tavern for an hour with the goal of increasing the social score and if stochasticity resulted in a lower than expected increase, after a re-plan, the planner could decide that Adam should spend another hour in the tavern. All these happen during game-time.

Note that roles have pre-conditions depending on variables of the virtual characters. For example, the role of working at the city gate is only possible if "job = city gate" where "job" is an additional variable.

In a test implementation, the following 11 roles were used:

- Work at City Gate (as guard)
- Eat at home (as diner)
- Sleep at home (as sleeper)
- Socialize at market (as buyer)
- Work at market (as seller)
- Work at market (as thief)
- Work at farm (as farmer)
- Socialize in tavern (as patron)
- Work at tavern (as bard)

- Work at tavern (as server)
- Work in the woods (as hunter)

The words in brackets are the roles to assume when in the scene, since a character can be in different roles in a single scene. Motivation changes corresponding to each role above were, respectively:

- 3, -2, -2, 1, 0
- 0, 0, 24, 0, -24
- 0, 4, -1, -3, 0
- -5,-1, -1, 2, 5
- 2, -2, -2, 4, -2
- 2, -2, -2, 0, 2
- 2, -2, -2, 0, 2
- -5, -1, -1, 7, 0
- 2, -2, -1, 1, 0
- 2, -2, -1, 1, 0
- 0, -2, -1, -2, 5

The motivation changes were chosen so that after any change, the sum of the motivation values remains the same. There is an implicit pre-condition on each role: after choosing the role, the motivation values cannot fall outside the [0, 100] range. Two different kinds of goals were tried, one kind was to maintain a certain threshold for the motivations and the other kind was to maximize certain motivations.

A problem with this particular planning approach is running time. For this approach to be effective, planning needs to be done on-line during game play since motivations are dynamically changing. However, the running time can be prohibitive. For example, running the implemented planner using exhaustive search to find an optimal plan produced a 6-step plan in a 10-second limit (running on an Intel Core-i7-2670M processor at 2.20GHz). Producing 24 steps as intended would be infeasible in a real game setting.

However, in a story-based game, the daily routines do not have to be "optimal" according to some behind-the-scene criteria. The goal is to produce believable virtual characters in real-time, and believability is more important than optimality. Therefore, a sub-optimal algorithm can be used instead to produce solutions in a shorter amount of time.

Many classic games, such as the board game Go, have computer players that utilize Monte-Carlo tree search (MCTS) and the UCT method [28] [29]. UCT [39] (Upper Confidence bounds applied to Trees) is a well-established Monte-Carlo search algorithm that builds a sparse search tree over the state-space at every time-step, estimating the value function at each state by Monte-Carlo rollouts. After each simulation, node values along the trajectory of the search tree are updated and the simulation policy is updated according to the new values.

With UCT, each node in the search tree stores:

- n(*s*), number of times *s* visited in previous rollouts
- n(*s,a*), number of times *a* has been explored in *s* in previous rollouts
- Q(*s,a*), Action value estimate for *a*

UCT chooses an action according to:

- If actions (in current state) have yet to be explored, select one randomly
- Otherwise, choose the action that maximizes

$$Q\ (s,a) + c\ \times \sqrt{\frac{log\ n(s)}{n(s,a)}}$$

Node values are updated according to:

$$n(s,a) \leftarrow n(s,a) + 1$$

$$n(s) \leftarrow n(s) + 1$$

$$Q(s,a) \leftarrow Q(s,a) + \frac{1}{n(s,a)}(R - Q(s,a))$$

In the formula above, R is the reward calculated from the objective function.

A UCT approach was implemented for the previous example character, Adam. The algorithm builds a sparse tree over the state space of roles, carries out a set number of Monte Carlo rollouts to the end of the search (24 steps), biases the rollout trajectories towards the most promising roles, and picks the most promising role sequence after the rollouts.

Given a 10-second limit, 10,000 rollouts were used in the experiments. Here is a sample solution found by the UCT implementation:

Hour 1. Sleep at home (as sleeper)
Hour 2. Socialize in tavern (as patron)
Hour 3. Socialize at market (as buyer)
Hour 4. Sleep at home (as sleeper)
Hour 5. Eat at home (as diner)
Hour 6. Work at market (as seller)
Hour 7. Work at market (as seller)
Hour 8. Work at farm (as farmer)
Hour 9. Sleep at home (as sleeper)
Hour 10. Sleep at home (as sleeper)
Hour 11. Work in the woods (as hunter)
Hour 12. Sleep at home (as sleeper)
Hour 13. Socialize in tavern (as patron)
Hour 14. Work at City Gate (as guard)
Hour 15. Work at farm (as farmer)
Hour 16. Work in the woods (as hunter)
Hour 17. Work at City Gate (as guard)
Hour 18. Work in the woods (as hunter)
Hour 19. Socialize at market (as buyer)
Hour 20. Sleep at home (as sleeper)

Hour 21. Work at market (as thief)
Hour 22. Work at market (as thief)
Hour 23. Work at market (as seller)
Hour 24. Sleep at home (as sleeper)

This schedule is not desirable as Adam keeps moving from location to location without regard to what he was doing at the last time step. To partially alleviate this problem, a cost could be added to switching locations, though that cost must be fine-tuned by a designer. Even with additional pre-conditions that restrict the type of work Adam is allowed to do, it is inherently difficult to create an intuitively good schedule with this system. The designer must fine-tune each motivation value, pre-conditions and post-conditions of each role in order to generate reasonable behaviours.

One can see that perhaps a better way to specify behaviours is to allow roles to be grouped by time step, preventing undesired changing of roles. It is expected that Adam would be working continuously for at least five hours, and sleeping continuously for seven hours.

In addition, to satisfy our goal of providing the designer with control, a timeline should be available to the designer so that the designer has the ability to force a virtual character to choose a role at a specific hour. For example a designer may want to require that Adam sleeps at home from hours 0 to 6. These are examples of domain-specific knowledge that can greatly help a system produce more believable behaviours. A good daily schedule is usually cyclic, and a system that utilizes UCT has no direct way of specifying a cyclic schedule. We want to give designers a system that allows them to directly specify portions of the schedule that they deem important.

4.2 Cyclic Scheduling

With the shortcomings of the planning and MCTS models, an alternate scheduling method is required. This research aims to provide game designers with a system that allows them to control the components of the schedule they deem important. The intuition behind this system is that it should allow game designers to directly specify the important aspects of a good daily schedule and for the AI scheduling / planning system to provide suggestions for the unimportant aspects.

In the scheduling domain, the interval scheduling maximization problem is one of the oldest problems – the goal is to schedule the largest set of non-overlapping intervals. A greedy polynomial time algorithm exists as the solution [38], by repeatedly choosing the interval with the earliest finishing time. The scheduling problem presented here does not aim to find an optimal solution. It determines whether a solution is possible that fits all requirements, and if possible, it finds a family of solutions. Instead of intervals of fixed start and end times, it has blocks of fixed lengths with variable start and end times to better match requirements for scheduling believable cyclic behaviours of virtual game characters. With this scheduling method, game designers specify the duration of objectives, any specific important assignments of objectives to specific hours and they include a set of specific roles for each objective. Again, since each role is an atomic element in the architecture, once a role is selected, a role-level controller takes over and produces behaviours accordingly.

A scheduling problem can be formulated as a Constraint Satisfaction Problem (CSP) [54]. A CSP is defined as a set of variables, each with a non-empty domain of values, and a set of constraints on the values. A consistent assignment is an assignment of values to variables such that no constraints are violated. Solutions to CSPs require a consistent assignment to each variable. In some problems, the goal is to find all solutions, and some problem formulations also require an optimal solution that maximizes an objective function.

The cyclic scheduling problem presented here is a CSP. However, it is not explicitly represented as a CSP, since this is a very specialized instance where there is a natural representation of the problem in the form of a timeline. Special properties on the timeline, such as the requirement for hourly objectives to form consecutive blocks of objectives, can be exploited in specialized algorithms. An optimal solution is also not required as no objective functions are defined. In terms of the user interface, a timeline representation is also more intuitive to a game designer than a CSP formulation.

With this cyclic scheduling problem, the scheduler creates a 24-hour schedule (or however many hours a game world sets in a day) consisting of objectives. This is the most computationally expensive part of behaviour scheduling due to the complexity and the range of tightness-looseness of the constraints that could be provided by the designer. The mapping of a designer's schedule concept to a particular list of objectives generally has many open variables so there are many ways it can be satisfied. If the designer wants to express some constraints such as consecutive blocks, but leave some choices open, then it becomes a planning problem that can take considerable time to run. The planning is also iterative so the designer can change constraints based on seeing generated plans. Therefore, the scheduler is run offline so that it is not bound by time constraints online. It is also possible that during this process, the designer could specify constraints that are unsatisfiable. In this case, the architecture can inform the designer offline. At game time, the selection of a role to an objective in the schedule involves only a filter of a small collection using the current game state (a few condition checks), a probability spin and then a selection from the filtered collection. This makes the in-game process fast.

Specifically, the designer uses a 24-hour timeline. At each hour, instead of choosing roles, a game designer only chooses common objectives. In this research, there are four objectives, Eat, Sleep, Work, or Other. The "Other" objective can include social and other roles. These objectives were chosen based on common daily schedules of

virtual characters. There is nothing that prevents different objectives from being defined in a different virtual world.

The timeline lets the designer specify constraints at specific hours as desired. This gives the designer total control over the behaviours of a virtual character. If the designer fills in all 24 hours, then the virtual character will do exactly as the designer specifies, with no emergent behaviour in objectives (although a probability selector can still provide probabilistic roles for each objective). If the designer wants even more control, only a single role need be provided for each objective.

The power of the scheduling system is that it provides emergent behaviours for virtual characters when the designer wants to use them. The system allows a designer to specify durations for each objective instead of specific time slots. For example, a virtual character might be required to sleep for 8 hours, eat for 3 hours, work for 8 hours and do other activities for the remaining 5 hours. The designer should be able to specify these durations and if desired, add specific assignments for specific hours, such as be sleeping at 5am and be working at 9am.

A designer considers the principal mission of a particular virtual character, and how the character can execute this mission. For example, a farmer usually does typical farming tasks, along with sleeping, eating, and socializing. All virtual characters fitting the farmer mission can use similar daily routines.

The designer should be able to specify how a selector produces roles at run time. Each objective can contain multiple roles that can satisfy the objective. For example, for the Eat objective, there can be a number of ways to fulfill this objective: one can eat at home, eat at a friend's place, eat at a tavern, etc. according to the designer's wishes, and the availability of executable roles during game time.

Due to the dynamism of story-based games, not all roles are available during the course of the game-play. If the virtual character has no friends available at a

particular time, then “Eat at a friend’s place” would not be a viable option. Similarly, if the only tavern in town burns down, then “Eat at a tavern” would not be a viable option after that incident. Even if an option is available, the designer may want to control how often the virtual character chooses it. The system architecture must support stochasticity during game time.

The system needs a way to give the designer direct control in how a probability selector chooses a role, given the available roles at the current game time, depending on what has happened in the game so far. In essence, the designer needs to have the ability to specify the percentage chance of choosing each role relative to every subset of available roles.

For example, there are three roles under the Eat objective. We call these R1, R2, and R3 for short. There needs to be a way for the designer to specify all of these probabilities:

- P (R1 given exactly R1, R2, R3 are available)
- P (R2 given exactly R1, R2, R3 are available)
- P (R3 given exactly R1, R2, R3 are available)
- P (R1 given exactly R1, R2 are available)
- P (R2 given exactly R1, R2 are available)
- P (R1 given exactly R1, R3 are available)
- P (R3 given exactly R1, R3 are available)
- P (R2 given exactly R2, R3 are available)
- P (R3 given exactly R2, R3 are available)

If only one role is available, then the percentage chance of choosing this role is 100%. For the percentages listed, the system should also provide default values to avoid unnecessary work by the designer. One way of inferring default probabilities is by using the average values of specified supersets.

4.2.1 Cyclic Scheduling Tool

Given these requirements, screenshots of an exemplar tool that implements the cyclic scheduler system is shown in Figure 12 through Figure 16 and Figure 18 through Figure 20. Figure 12 shows an example timeline. The designer has the ability to choose one of four objectives at each time step from the drop down menu (so far each hour is blank).

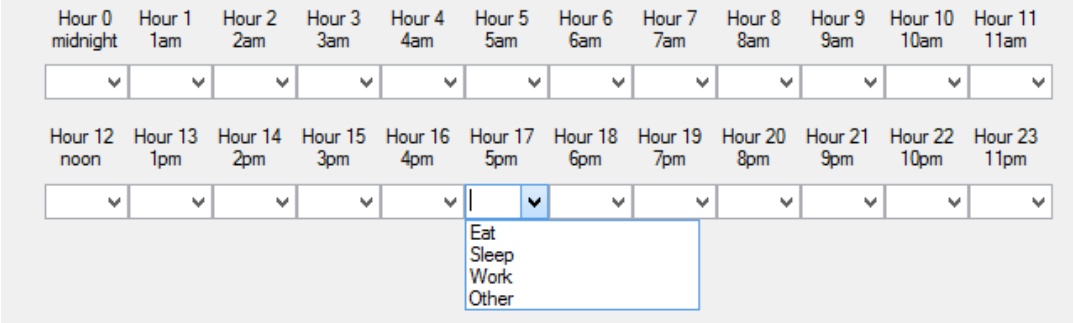


Figure 12. An example timeline with a dropdown menu shown.

The timeline allows key constraints at specific hours to be specified. For example, the designer can ensure that the character is asleep at 1am and 6am (Figure 13), using the timeline. The scheduler may add more sleep times at unconstrained times on the timeline, but the times on the timelines are honoured. One-hour increments are created in our timeline but this can be easily generalized to arbitrary increment sizes.

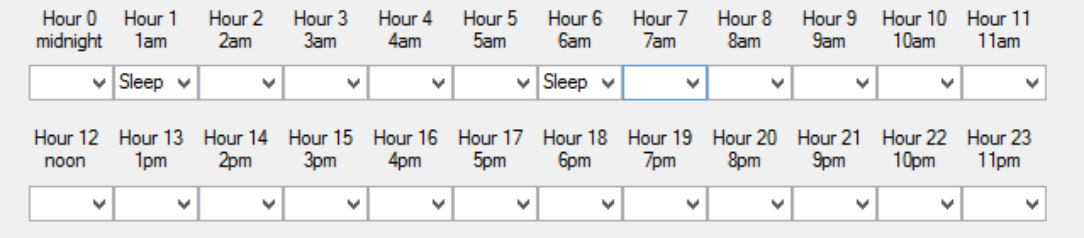


Figure 13. An example timeline with two slots filled by a designer.

The cyclic scheduler tool provides designers with the option to set total hours for each objective (Figure 14), as well as options to group the objective hours into consecutive or non-consecutive blocks. For example, the designer may require a total of 9 consecutive sleep hours. Along with a timeline constraint of sleeping at 1am and 6am, this would require the character to sleep for any 9-hour consecutive block that includes the hours 1am to 6am.

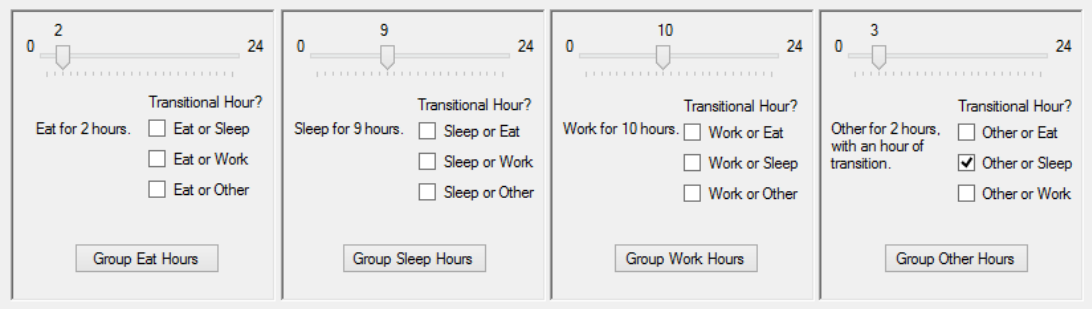


Figure 14. The designer can specify hours for each objective.

Each of the sliders represents the number of hours for this objective in a 24-hour period. For example, in the screenshot, objective Eat happens 2 hours a day, Sleep 9 hours a day, work 10 hours and do other activities for 3 hours. Roles in an objective do not have to happen consecutively. The designer is allowed to specify that the 9 hours of Sleep happen in two blocks, one 5-hour consecutive block, and one 4-hour consecutive block. This is done in a separate Group Hours window.

In the Group Hours window (Figure 15), the designer can specify how the objective hours should be grouped into consecutive blocks. In this example, the designer selects a single checkbox to split the 9 hours of Sleep into two blocks as desired. Note that the designer can further require the scheduler to insert a non-sleeping hour between the two Sleep blocks, by selecting the option as shown. Otherwise, the scheduler is free to insert other objectives between the consecutive blocks or not. The objective can be further split into smaller blocks by checking more boxes between hours, and the designer may or may not require non-sleeping blocks to be inserted between Sleeping blocks by checking or un-checking that option.

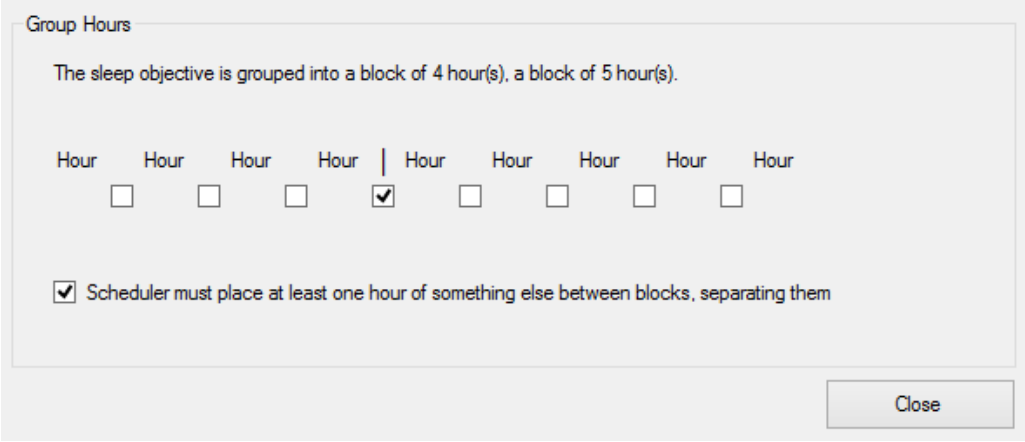


Figure 15. A Group Hours window.

The ability to specify the exact location and role of a character at a particular time allows designers to coordinate the schedules of multiple characters without over-specifying the behaviours of these characters at other times. For example, the designer may arrange for a thief and a fence to be in the roles buyer and seller in a specific alley at mid-night and design the rest of their behaviours independently (or let the scheduler decide).

A designer can also specify transitional hours (Figure 14), where the character has a probability of using the next objective an hour earlier, an hour later, or anytime in between. This provides stochasticity in the schedules so that they do not look identical from day to day. The concept of stochasticity can be expressed in multiple ways. A transitional hour is just one implementation choice.

Checkboxes beside each Objective control, allow a designer to specify one or more additional Objectives for a Transitional Hour. For example, if the “Sleep or Work” checkbox beside the “Sleep” objective is checked, it means the scheduler can choose either “Sleep” or “Work” for the transitional hour. If “Sleep or Eat” is also checked, then either “Sleep”, “Work”, or “Eat” can be chosen by the scheduler.

After the scheduling constraints are specified by the designer, the cyclic scheduler then attempts to produce a schedule. The algorithm is shown in Figure 17. The scheduler informs designers if the specified constraints are satisfiable or unsatisfiable. The greedy depth-first search algorithm employed in this case is complete: if a solution exists, the solution must contain every specified objective block in some location. The greedy algorithm iterates through every location for each block until a satisfiable location is found. Therefore the algorithm will find a solution if one exists.

Since an objective consists of one or more roles, the tool must allow a designer to specify one or more roles that can be used to satisfy each objective, and the selector that is to be used to pick the roles online.

With the role picker, the designer can choose the roles that belong to each objective (Figure 16). By default a role belongs to one of the four objectives, but the tool allows the designer to change its objectives at will.

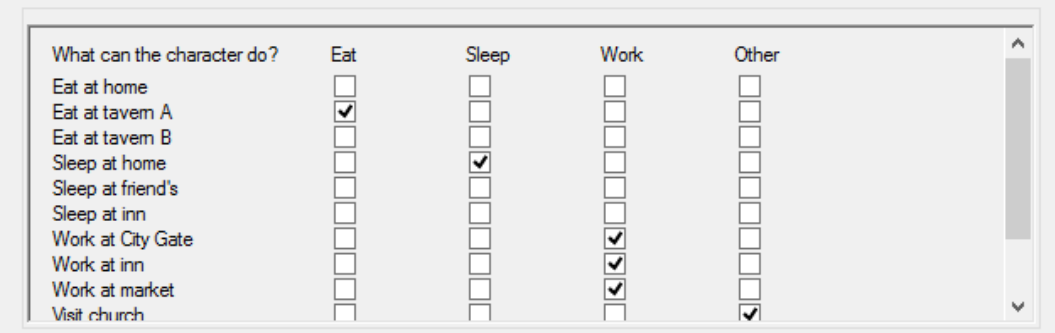


Figure 16. The role picker that allows a designer to specify the roles in each objective.

For this research, the list contains 13 available roles.

- Eat at home
- Eat at Tavern A
- Eat at Tavern B

- Sleep at home
- Sleep at friend's
- Sleep at inn
- Work at City Gate (as guard)
- Work at inn (as server)
- Work at market (as seller)
- Visit church
- Socialize in Tavern A (as patron)
- Socialize in Tavern B (as patron)
- Socialize at market (as buyer)

Note that “Eat at home”, “Sleep at home” and “Sleep at friend’s” are the common names for categories of similar roles. The exact “home” or “friend” differs depending on the actual character that this role is applied to.

In this research, a Probability Selector is chosen to pick roles from an objective at game time. The tool has an interface for the designer to specify the probabilities (weights), as shown in Figure 18. In the figure, the designer has chosen three roles to be included in the Work objective, and they have equal probabilities. These numbers are automatically normalized.

By default, each role will be chosen with equal probability (in this case, 33%). The designer is free to change the probability numbers for each specific virtual character behaviour.

Since not all roles will be available during game play, the designer must also be allowed to specify probabilities for subset of roles, subject to dynamic availability of each role at game time.

```
schedule = []
itemList = sort(itemList) // longest objective block first
CyclicSchedule(itemList)

bool CyclicSchedule(itemList)
        if itemList is empty, return true
        item = pop(itemList)
        for each location L in schedule
                if satisfiable(L, item)
                        schedule[L] = item
                        returnValue = CyclicSchedule(itemList)
                        if (returnValue == true)
                                return true
                        else
                                schedule[L] = ""
        return false

bool satisfiable(location L, obj block Item)
        if (length of obj block > available obj blocks at L)
                return false
        else if (non-consecutive obj block requirement is violated)
                return false
        else if (currently assigned obj hours > specified max obj hours)
                return false
        else
                return true
```

Figure 17. The Cyclic Scheduling Algorithm

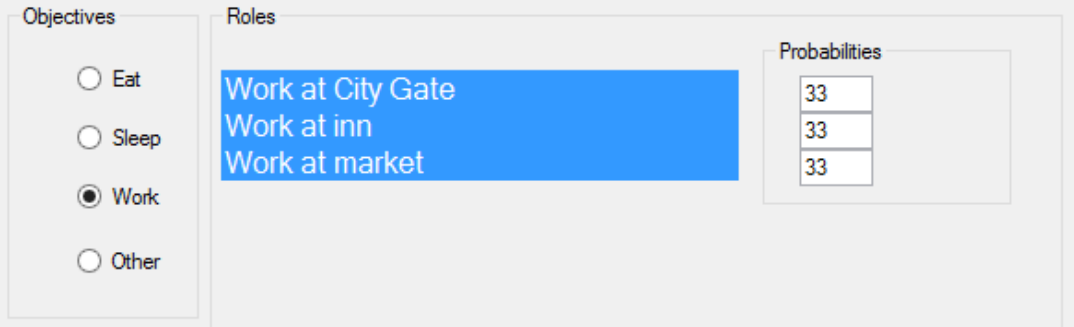


Figure 18. The designer can specify probabilities (weights) for each role in each subset of roles. In this figure all three roles are selected in the subset.

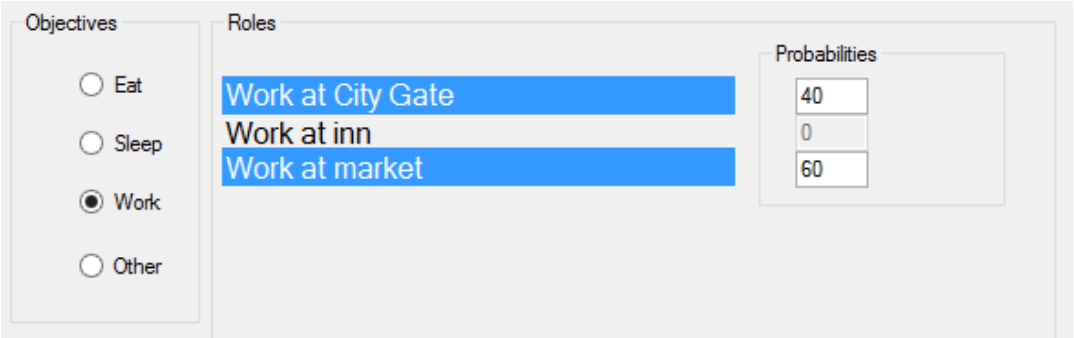


Figure 19. The designer can specify probabilities (weights) for each role in each subset of roles. In this figure only "Work at City Gate" and "Work at market" are selected in the subset.

The designer can select any subset of these roles (in Figure 19 "Work at inn" is unselected), and specify the probabilities of the selected roles, in the event that only the selected roles are available at a particular game time. For example, if the inn burned down or otherwise became inaccessible at some point, the second role would be unavailable for the objective, even if the innkeeper role was the preferred role for a particular virtual character. In this case, the system would use the custom probabilities assigned by the designer for the remaining set of two roles. With the probabilities specified in Figure 19, the character would prefer to then work at the

market more (60% to 40%) than the City Gate. This approach allows a virtual character to react to changes during the game and act accordingly.

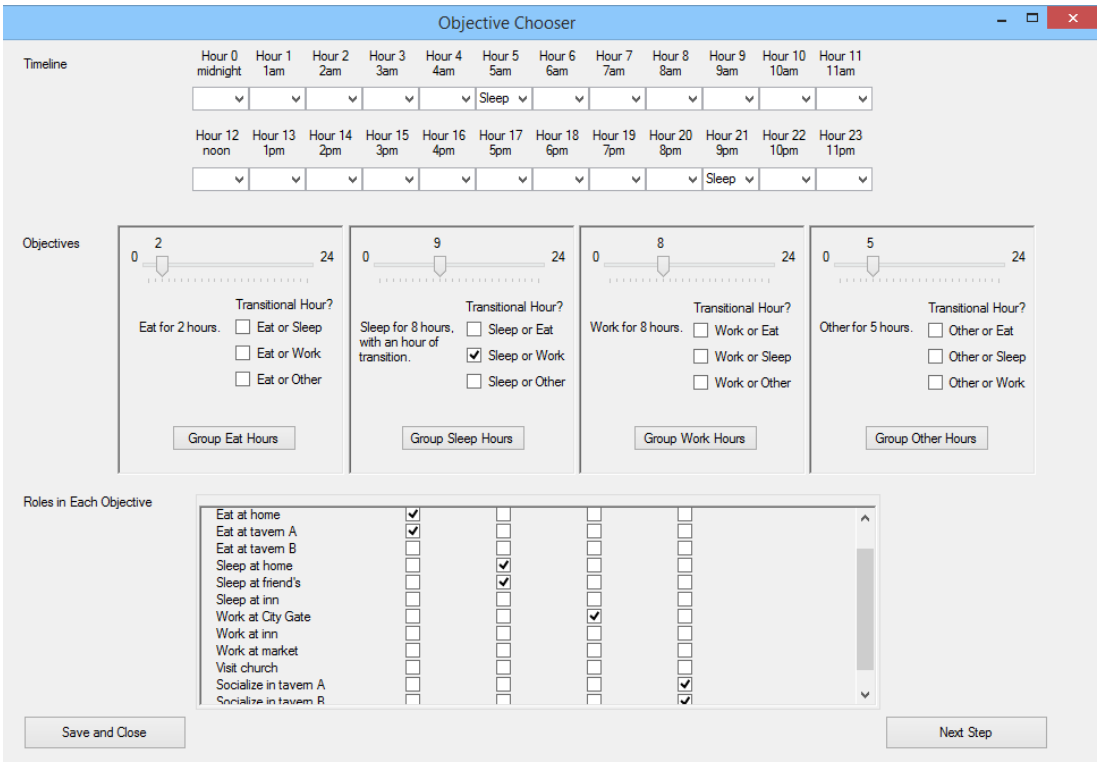


Figure 20. The main Objective Chooser interface.

The main interface window of the tool is shown in Figure 20, which contains a typical example as specified by a designer. The virtual character Adam has to "Sleep" at 5am and 9pm. Adam has to "Eat" for 2 non-consecutive hours, "Sleep" for 8 consecutive hours plus one transitional hour of either "Sleep" or "Work", and "Work" for another 8 consecutive hours. The "Eat" objective can be satisfied by eating at home or at Tavern A; the "Sleep" objective can be satisfied by sleeping at home or at the place of Adam's friend, Bob; the only role allowed in "Work" is to work at the City Gate.

The scheduling system then generates a working schedule for Adam. Figure 21 shows an example result. One schedule and two consecutive days of roles are shown. Note that the roles shown are samples only, since roles are chosen by the

Probability Selector as the game is played. The result fits exactly what the designer has specified. The generation process took less than one second.

Timeline of Objectives:	Sample Roles in a day:	Sample Roles in another day:
0 : Sleep	Sleep at home	Sleep at home
1 : Sleep	Sleep at home	Sleep at home
2 : Sleep	Sleep at home	Sleep at home
3 : Sleep	Sleep at home	Sleep at home
4 : Sleep	Sleep at home	Sleep at home
5 : Sleep / Work	Sleep at home / Work at City Gate	Sleep at home / Work at City Gate
6 : Work	Work at City Gate	Work at City Gate
7 : Work	Work at City Gate	Work at City Gate
8 : Work	Work at City Gate	Work at City Gate
9 : Work	Work at City Gate	Work at City Gate
10 : Work	Work at City Gate	Work at City Gate
11 : Work	Work at City Gate	Work at City Gate
12 : Work	Work at City Gate	Work at City Gate
13 : Work	Work at City Gate	Work at City Gate
14 : Eat	Eat at tavern A	Eat at home
15 : Other	Socialize in tavern B	Socialize in tavern B
16 : Other	Socialize in tavern B	Socialize in tavern B
17 : Other	Socialize in tavern B	Socialize in tavern B
18 : Other	Socialize in tavern B	Socialize in tavern B
19 : Other	Socialize in tavern B	Socialize in tavern B
20 : Eat	Eat at home	Eat at home
21 : Sleep	Sleep at home	Sleep at friend's
22 : Sleep	Sleep at home	Sleep at friend's
23 : Sleep	Sleep at home	Sleep at friend's

Figure 21. The generated schedule of objectives and two consecutive days of roles.

It is worthy of note that if the designer is not happy with the resulting schedule, the cyclic scheduler can easily produce another schedule based on the same constraints, which requires no extra efforts by the designer. Figure 22 shows another schedule for Adam. Note that Adam now eats at 6am instead of the 2pm in the previous schedule.

Schedules are generated off-line before the start of the game, and selectors dynamically assign roles to objectives as the game is played, allowing dynamic adaptation. As noted by Wright and Marshall [67], game AI must be fast. With only a fraction of the processor time allocated to AI each frame, the game-time component of the AI must be computed between frame displays.

Timeline of Objectives:	Sample Roles in a day:	Sample Roles in another day:
0 : Sleep	Sleep at friend's	Sleep at home
1 : Sleep	Sleep at friend's	Sleep at home
2 : Sleep	Sleep at friend's	Sleep at home
3 : Sleep	Sleep at friend's	Sleep at home
4 : Sleep	Sleep at friend's	Sleep at home
5 : Sleep / Work	Sleep at friend's / Work at City Gat	Sleep at home / Work at City Gate
6 : Eat	Eat at home	Eat at tavern A
7 : Work	Work at City Gate	Work at City Gate
8 : Work	Work at City Gate	Work at City Gate
9 : Work	Work at City Gate	Work at City Gate
10 : Work	Work at City Gate	Work at City Gate
11 : Work	Work at City Gate	Work at City Gate
12 : Work	Work at City Gate	Work at City Gate
13 : Work	Work at City Gate	Work at City Gate
14 : Work	Work at City Gate	Work at City Gate
15 : Other	Socialize in tavern A	Socialize in tavern A
16 : Other	Socialize in tavern A	Socialize in tavern A
17 : Other	Socialize in tavern A	Socialize in tavern A
18 : Other	Socialize in tavern A	Socialize in tavern A
19 : Other	Socialize in tavern A	Socialize in tavern A
20 : Eat	Eat at home	Eat at home
21 : Sleep	Sleep at home	Sleep at friend's
22 : Sleep	Sleep at home	Sleep at friend's
23 : Sleep	Sleep at home	Sleep at friend's

Figure 22. The different schedule of objectives and two consecutive days of roles.

4.3 Usability Evaluation Results

Would a designer find the Tiered Behaviour Architecture easy to use and powerful enough to specify the kinds of behaviours needed, compared to existing methods for creating behaviours of virtual characters? Chapter 3 showed positive results for the expressiveness, performance and quality metrics. The only metric that has not been presented is architecture usability.

In Chapter 3, the user study showed that a cyclic architecture is able to produce behaviours that are more believable than the default behaviours of the commercial game, The Elder Scrolls V: Skyrim. However, unless the architecture can generate reliable behaviours quickly, the scripting bottleneck will remain. This section

presents a user study that measures the usability aspects of the architecture by comparing designers who used the Tiered Behaviour Architecture tool to designers who manually scripted the behaviours. The hypotheses are:

- Tool users would produce more complete daily behaviours than manual scripters in a given period of time.
- Tool users would produce more reliable daily behaviours than manual scripters in a given period of time.
- To produce the same daily behaviours, tool users would require less time than manual scripters.

Skyrim was used to evaluate behaviour expressiveness, performance and quality (believability). However, it was more convenient to evaluate designer efficiency and behaviour reliability using Neverwinter Nights (NWN), based on the availability of a pool of suitable study participants with previous NWScript experience who could write manual scripts. Therefore, the Skyrim-based implementation was modified to work with the NWN engine and to generate NWN scripting code, but the same GUI was used.

As discussed in Chapter 2, there are visual scripting tools that allow a designer to create complex game scenarios without manual scripting. One such tool is ScriptEase II [55], which works with the NWN game engine. A preliminary experiment was conducted to determine if a visual scripting tool such as ScriptEase II could serve as an alternative to manual scripting at the specific task of creating cyclic daily behaviours. This experiment was conducted in a controlled environment identical to the main user study described below, with three participants who are experienced programmers. The results showed no notable differences between manual scripting and using a visual scripting tool in producing cyclic daily behaviours following a set of identical requirements, in terms of the usability measures described in detail below. Therefore, manual scripting was chosen as the comparison with the cyclic architecture, since it is the more general approach that is available for most games.

In the main study, there were two groups of participants: one group used a tool that implemented the cyclic architecture. The other group used the manual scripting method for NWN. To ensure a fair comparison, all participants were required to have played the NWN game so that they could test their behaviours quickly using familiar settings and controls. The Scripting Group participants were also required to be programmers who had prior experience with NWN scripting. Participants in the Tool Group were not required to have NWN scripting experience. No participants had seen the cyclic architecture tool before the study.

At the start of the study, both groups were given an instruction manual for their respective methods (Appendix C). The Tool Group was given a detailed manual on how to use the architecture tool, while the Scripting Group was given access to the NWN Lexicon[5], a familiar online manual for the scripting language. Both groups were also given a game file, which contained an identical pre-made town area, populated by four characters (Figure 23). Participants used this game file, and creating the behaviours was their only task.

Lastly, the participants were each given a sample behaviour for a sample character. The sample behaviour was identical for both groups, but was implemented in the respective method for each group. Participants were asked to first examine the sample behaviour to see how it was implemented (either with the tool or with scripting code as appropriate to their group), and were told they could use the sample as a starting point. The sample character behaved as follows: He sleeps at home from midnight until hour 6. He starts to work at the Market at 7, for 10 hours, then eats at Tavern A at hour 17 for 2 hours, and then sleeps at home from hour 19 until the next day. He repeats the same behaviours for three days.

[5] http://www.nwnlexicon.com/

Figure 23. This pre-made town was presented to participants in the user study.

The actual behaviours that the participants were asked to create were identical for both groups. They were asked to follow instructions to create the behaviours for four characters in order, Adam, Bob, Cathy, and Donna, each for three consecutive days in-game. Each subsequent character had increasingly complex behaviours as shown in Table 5. A character with multiple schedules has a different schedule on the last day; stochastic schedules refer to schedules where a character can leave anytime within a range of an hour from the scheduled time; a character with multiple roles can choose two different roles to satisfy each objective; blocks of roles refers to the ability to separate a role into multiple blocks of time periods (such as work in the morning and afternoon, separated by lunch); and dynamic roles refer to the ability to switch roles on the fly when one becomes unavailable during gameplay.

Participants were allocated a maximum of three hours to complete all behaviours, after which the study was stopped regardless of completion. Participants were also asked to record the time that they spent on each character.

A total of 25 participants were recruited in the user study. The Tool Group had 15 participants, where 5 were programmers and 10 were not. A programmer was defined as someone who self-reported having written many computer programs (in any language). The Scripting Group had 10 participants, all experienced NWN script programmers.

Behaviour	**Adam**	**Bob**	**Cathy**	**Donna**
Multiple Schedules	Yes	Yes	Yes	Yes
Stochastic Schedules	Yes	Yes	Yes	Yes
Multiple Roles		Yes	Yes	Yes
Blocks of Roles			Yes	Yes
Dynamic Roles				Yes

Table 5. The aspects of the behaviours.

4.3.1 Completeness

Figure 24 shows the percentage of participants who completed at least 80% of the requirements of all behaviours for each character. 100-160 requirement correctness points were assigned for each day of each character. The same scoring rubric was used for tool users and scripters. For example, on Day 1, character Adam had the following points assigned: 10 points (Adam starts at home), 20 points (Adam goes to the city gate at hour 7), 20 points (Adam goes to Tavern B at hour 17), 20 points (Adam goes home at hour 18 or 19), 20 points (previous time is random), 10 points

(Adam stays home for rest of day). For each of the 20 point destination-time requirements, 10 points were awarded for the correct time, and 10 points for the correct location. 80% completeness was used to avoid penalizing a participant who misinterpreted a requirement. For example, they may pick the wrong location. A participant who made small misinterpretation errors and proceeded to the next character would not be penalized for completeness, only correctness.

In addition, the NWN scripting environment notifies a user if a script does not compile. In this study, all scripts of all scripting participants compiled successfully and ran without crashing the game. Therefore a scripter was not penalized for completeness due to undetected scripting errors.

All 15 participants of the Tool Group completed at least 80% of the requirements of each character. From the Scripting Group, all 10 participants completed at least 80% of the requirements of Adam and Bob, 7 participants completed at least 80% of the requirements of the third character, Cathy, and 6 participants completed at least 80% of the requirements of the fourth character, Donna. Due to the small sample sizes, there are no statistically significant differences between the two groups for the percentage of participants who completed 80% of each character.

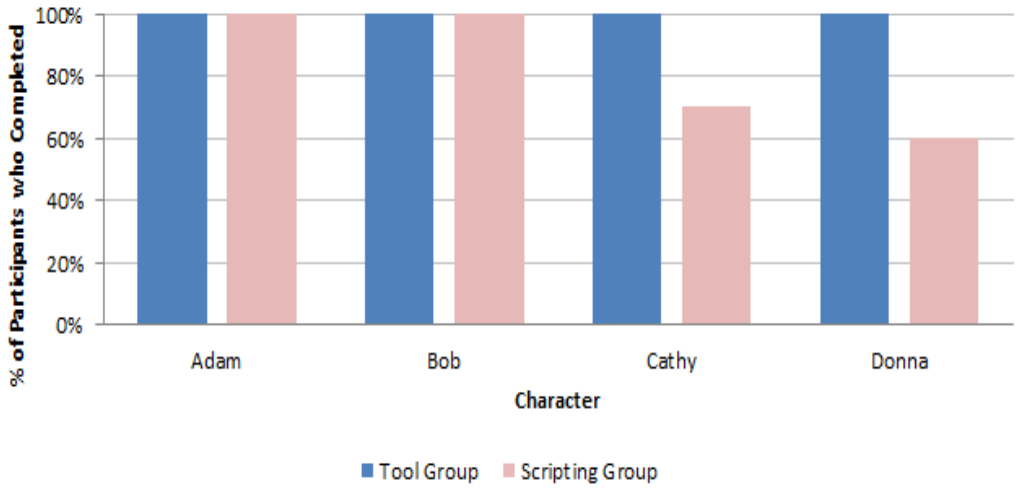


Figure 24. Completeness at 80% or higher: Tool Group vs. Scripting Group.

Participants were asked to finish one character before starting the next one, and when they believe they have finished a character, record the time that they finished it. This setup also tells us how many characters the participants believed they had finished. All 15 participants of the Tool Group believed that they had finished all characters. From the Scripting Group, all 10 participants believed that they had finished Adam and Bob, 9 participants believed they had finished the third character, Cathy (only 7 had completed, according to the objective measurement in Figure 24), and 6 participants believed that they had completed the fourth character, Donna (agrees with the objective measurement in Figure 24).

4.3.2 Correctness

Three different rules were used to measure correctness: one using all characters created by participants, the second one using only the characters completed at the 80% level, and the third one using only the characters which the participants themselves believed they had finished. Both the second and third measures allow minor requirement misinterpretation errors to not adversely affect the interpretation of correctness as a measure of reliability, since a blunder in interpretation made by participants is not necessarily a reliability issue. Inherently both the second and the third measures exclude characters that were not attempted by a participant so it only measures correctness of the work that was finished (objectively or by belief respectively).

For each of the characters, the correctness number represents the percentage of the requirements of the behaviours that were correctly created (as measured by correctness points), averaged over the participants. Figure 25, Figure 26, and Figure 27 show the results. The bottom of each bar indicates the minimum correctness value across all participants, and the top of each bar indicates the maximum correctness. The solid line in between represents the average correctness scores.

For all four characters, correctness was higher with the Tool Group than the Scripting Group. Looking at the overall results (rightmost two bars), the Tool Group had an average correctness rate of 97.01%, while the Scripting Group had a correctness rate of 79.72% counting all characters (Figure 25), 91.30% counting only at least 80% completed characters (Figure 26), or 90.51% counting only characters the participants self-reported to have finished (Figure 27). A one-tailed unequal variance T-test indicates this difference is significant at the 95% level, regardless of which measure is used (Table 6).

Looking at each individual character, there is a significant difference between the Tool Group and the Scripting Group for the most complex character Donna as well as Bob (Table 6). For Cathy the difference is significant when counting all characters or those self-reported as finished.

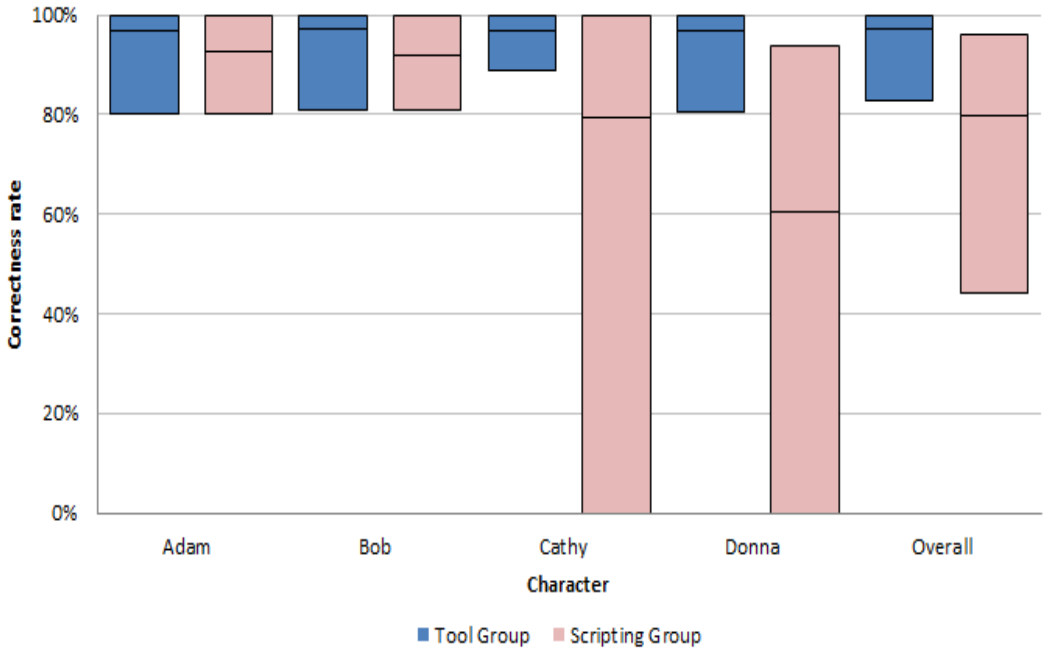


Figure 25. Correctness: Tool Group vs. Scripting Group, counting all participants.

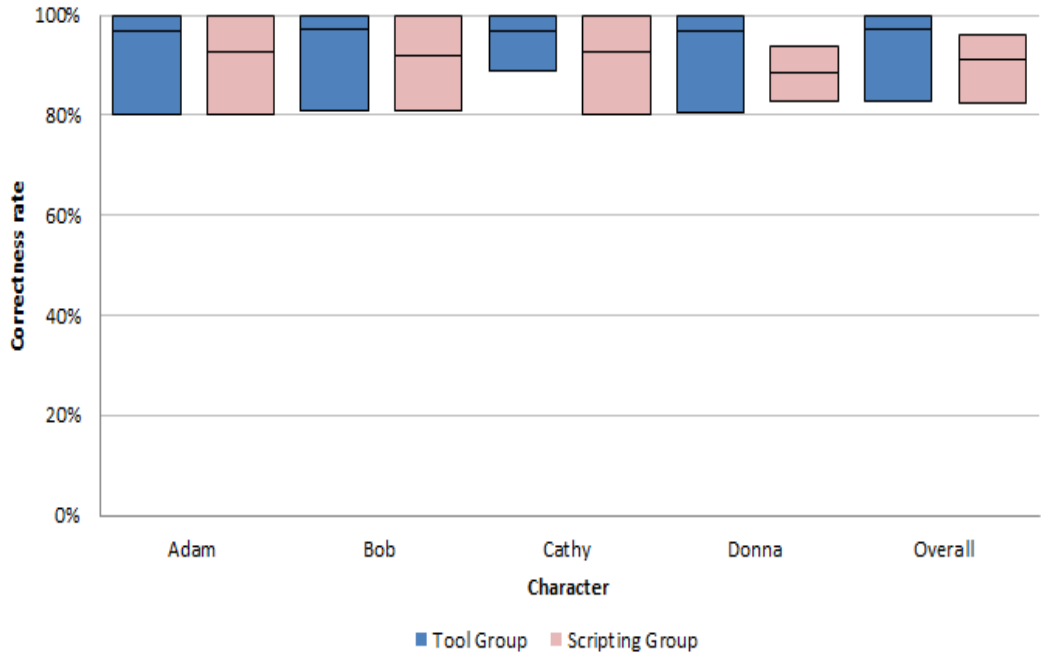


Figure 26. Correctness: Tool Group vs. Scripting Group, counting only characters completed at the 80% level.

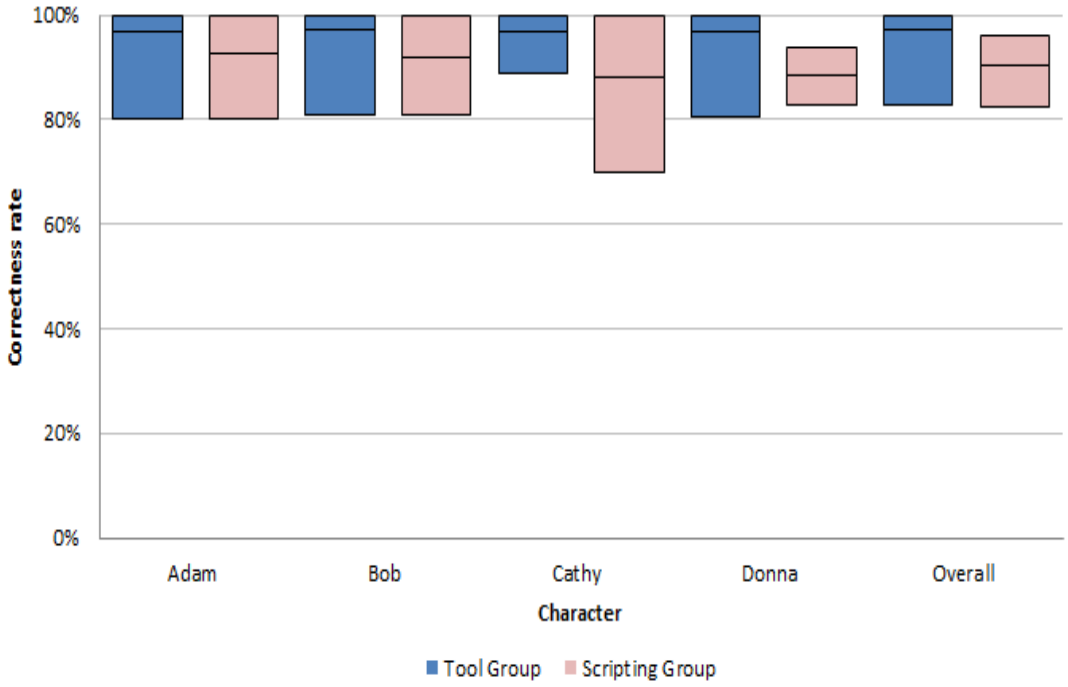


Figure 27. Correctness: Tool Group vs. Scripting Group, counting only characters for which the participants themselves believed to have finished.

Character	Tool Group vs. Scripting Group (all)	Tool Group vs. Scripting Group (80% completed)	Tool Group vs. Scripting Group (self-reported as finished)
Adam	0.057	0.057	0.057
Bob	**0.039**	**0.039**	**0.039**
Cathy	**0.045**	0.094	**0.023**
Donna	**0.011**	**0.001**	**0.001**
Overall	**0.006**	**0.007**	**0.009**

Table 6. P-values of T-tests comparing the two groups. P-values less than 0.05 (in bold) indicate significance at 95% level.

4.3.3 Completion Time

A fair measurement is needed to compare the completion times of the two groups. Since three participants did not complete Cathy, and four did not complete Donna (all from the Scripting Group), there are no readily-available completion times for these characters.

This section presents two different methods to create a fair comparison. The first method uses estimation for the missing completion time data. For the participants who did not finish a character, the time (T) they would have taken to complete each of these characters was estimated using their own time on the previous character, together with the average of other participants' ratio of times between the uncompleted character and the last completed character, using the formulas:

$$E(T_{Cathy}) = T_{Bob} \times (T^*_{Cathy} / T^*_{Bob})$$

$$E(T_{Donna}) = T_{Cathy} \times (T^*_{Donna} / T^*_{Cathy})$$

E() represents the estimated time. T^* represents the average time spent on the character denoted in subscript by all participants who completed that character.

Once the completion times for the incomplete characters (three Cathys and four Donnas) were estimated, the average time for each character was calculated. Figure 28 shows the average time needed to implement each character by the two groups. Again, each bar indicates the minimum, average, and maximum time values across participants. The Scripting Group took a longer time to complete than the Tool Group for each character. The results are significant at the 95% level for all characters, and for the total completion time (Table 7).

It is perhaps not surprising to observe a decrease in completion time from Adam to Bob (statistically significant for both groups). Although Bob has more complex behaviours, the time to implement Adam includes a steep learning curve for both groups, as participants were getting familiar with the tasks and solution environment. With Adam done, participants were able to use what they learned creating Adam's behaviour to help them with Bob. Note that these completion times do not include the time participants were asked to examine a sample character before they started working on Adam.

Character	Tool Group	Scripting Group	P-value of T-test
Adam	0:34	0:47	**0.032**
Bob	0:19	0:35	**0.005**
Cathy*	0:27	0:57	**0.001**
Donna*	0:28	0:52	**0.002**
Total	1:49	3:12	**0.001**

Table 7. The average time (hh:mm) for each character, in hours and minutes, with the p-value of T-tests comparing the times. Starred characters include estimates.

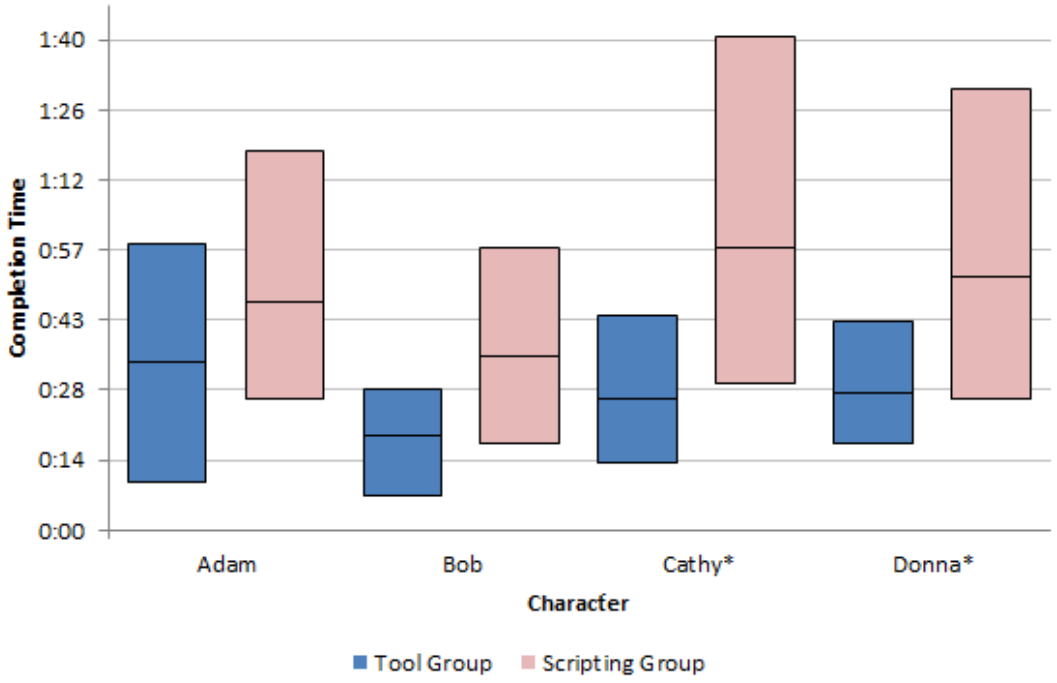


Figure 28. Completion time: Tool Group vs. Scripting Group for all characters. The starred characters include estimates.

The second method to provide a fair comparison between the two groups does not involve estimations. Instead, for the Scripting Group, only those who have finished are counted. Since it is likely that those who have finished were the fastest participants in the Scripting Group, for each group, only the fastest participants in were counted, to eliminate biases in favour of the Scripting Group.

In the Scripting Group, 100% of participants completed Adam and Bob, so 100% of participants were counted in each group. Only 7 out of 10 (70%) participants in the Scripting Group completed Cathy. Therefore only the fastest 70% (11 out of 15) participants in each group were counted. Similarly, only 6 out of 10 (60%) participants in the Scripting Group completed Donna, so only the fastest 60% (9 out of 15) participants in each group were counted.

Figure 29 shows the completion time comparing the two groups. Results for Adam and Bob are the same as in Figure 28. For Cathy and Donna, the average and

maximum times were reduced for both groups from the estimated completion time in Figure 28, since only the fastest participants were included from each group. The conclusions drawn previously still hold true: the Tool Group completed each character in a shorter amount of time than the Scripting Group. The results are significant at the 95% level for all characters (Table 8). Overall time is not used with this method of counting, since not all counted participants finished everything.

Character	Tool Group	Scripting Group	P-value of T-test
Adam	0:34	0:47	**0.032**
Bob	0:19	0:35	**0.005**
Cathy	0:23	0:52	**0.001**
Donna	0:24	0:40	**0.013**

Table 8. The average time (hh:mm) for each character, in hours and minutes, with the p-value of T-tests comparing the times. Only the fastest participants are counted from each group, respectively 100%, 100%, 70%, and 60%.

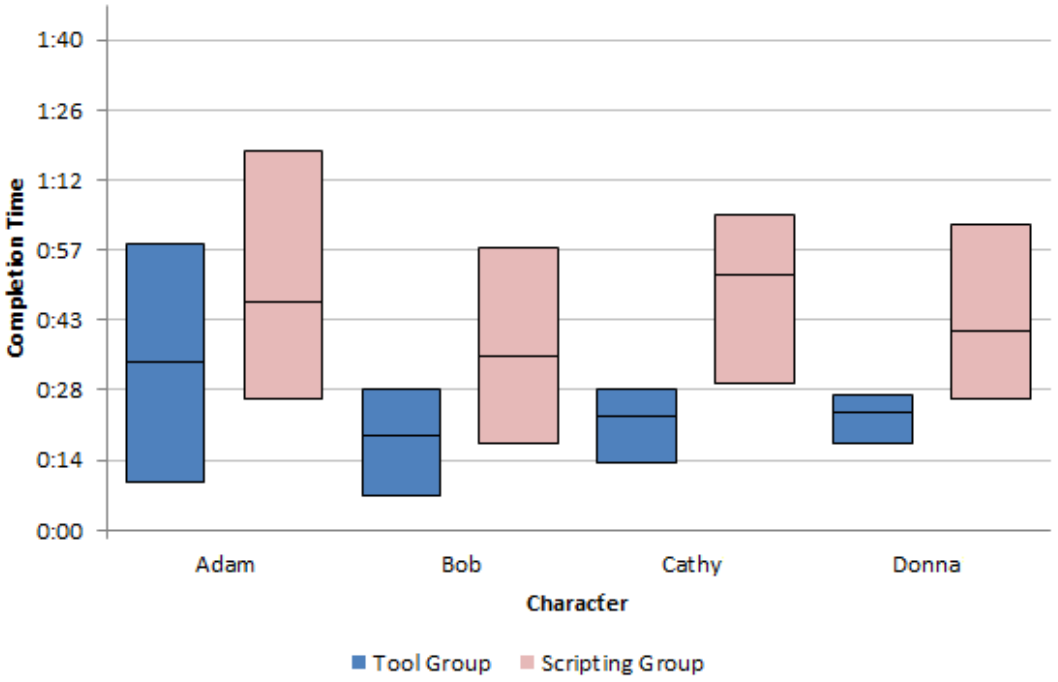


Figure 29. Completion time: Tool Group vs. Scripting Group for all characters. Only the fastest participants are counted from each group, respectively 100%, 100%, 70%, and 60%.

4.3.4 Efficiency

Albert and Tullis [1] define efficiency in the context of usability testing as "the ratio of the task completion rate to the mean time per task." A larger ratio implies that more participants were able to successfully complete a task per unit time. As expected, the Tool Group has a significantly higher efficiency than the Scripting Group (Table 9). There are two rows in the table. The first row excludes time spent by participants examining the sample character before they started to create new behaviours. The second row includes the time participants spent examining the sample character. Since total time is required, time is calculated using the former of the two methods presented in the previous section (the method using estimation).

Each entry in Table 9 represents the percentage of all character behaviours for all four characters that were successfully completed per minute, averaged over all participants in a group. For example, on average, to complete all behaviours for all four characters someone using the tool would require (100% behaviours) / (0.91% behaviours/minute) = 109.89 minutes, excluding time spent examining the sample character.

Efficiency	**Tool Group**	**Scripting Group**	**P-value of T-test**
Excluding sample time	0.91	0.46	0.000
Including sample time	0.78	0.43	0.000

Table 9. The percent efficiency (completion/time).

4.3.5 Discussion

While it was necessary to implement a specific scheduling tool to conduct a user study, the architecture is general. A scheduling tool may need to be tailored to the game being designed. For example, in a stealth game, the scheduling of guards/targets needs to be more fine-grained. A scheduling tool could be constructed so that designers specify the duration and granularity of a schedule before being presented with a scheduling template graphical interface. Instead of presenting the objectives Eat, Sleep, Work, and Social, stealth game objectives could be used. For example, patrol an area, guard a set of portals (doors), check the security (lock state) of portals, rest, eat, etc. Each of these objectives can be satisfied by multiple roles. For example, patrolling an area can be done using random waypoints, a fixed path or patrolling subareas with frequencies based on current threat levels. One example of dynamic roles is a blocked patrol path where the guard may switch to random waypoints. Another example is when a designer only wishes to switch to threat-level patrols when the threats in some subareas go above a threshold.

The four metrics: expressiveness, performance, quality and usability have now been applied to the objective level of the Tiered Behaviour Architecture, with positive results. However, how do we provide quality behaviours at the role level? Chapter 5 provides one answer to this question.

5. Behaviour Capture for Local Behaviours

The Tiered Behaviour Architecture presented in Chapter 3 divides behaviour control into objective level and role level, where role-level controllers are responsible for generating the fine-tune actions in local scenes. However, up until this point, no details have been given on how the role-level controllers work.

As noted previously, the role-level controllers are meant to be self-contained. These controllers are modular in the sense that while they are operating, they do not make decisions based on factors extraneous to the controllers. These controllers, however, can be initialized with certain inputs when they start, and can produce certain outputs when they finish. These inputs and outputs are the means of communication with other parts of a behaviour architecture. As such, these controllers are reusable in the sense that they can be utilized in different parts of an architecture to provide behaviours for different individuals or the same individual in different circumstances.

Chapter 2 discussed techniques for generating behaviours in specific scenarios. Methods such as FSMs and Behaviour Trees have been utilized in first-person shooters where the scenario dictates that your main goal is to shoot and kill your enemies [35]. Reinforcement learning techniques have been shown to produce superior results in medieval fantasy combat scenarios and to create adaptive behaviours for virtual characters who are companions to the player character [59]. These techniques are created with specific goals in mind, and they are promising for the scenarios they work in. Surviving a combat is exciting and important, but unfortunately the average daily life of a person is not always full of exciting moments. As this dissertation aims to look beyond specific scenarios and to provide a behaviour architecture that is able to express believable behaviours for characters in large open worlds with daily lives, other techniques will be explored.

With the Tiered Behaviour Architecture described in previous chapters, we have some ideas of the types of roles a character must fill in a medieval setting: eating in a tavern, working as a city guard, socializing at a market, etc. These scenarios generally have some common properties: they are non-hostile; they involve some repetitive actions; they involve actions in some reasonable sequences; and they involve actions specific to the role. For this role level, in the context of local scenes, this dissertation proposes a data-driven technique for the creation and generation of behaviours for virtual characters. This technique is called Behaviour Capture.

The term Behaviour Capture had been used in commercial software to describe the LiveAI tool introduced by AiLive[6] (development since stopped), and the Artificial Contender software by TruSoft[7]. Unfortunately, there are no public descriptions of the techniques used to generalize behaviours and no discussions on the level of behaviours that can be learned.

Our role level Behaviour Capture system is inspired by the idea of motion capture. Motion capture [31] records the actions of a human actor and uses the information to animate digital characters in computer animation. With motion capture, sensors are attached to the bodies of actors, and as the actors move their bodies, the spatial locations of their body parts are recorded. The data is used to animate virtual characters to move in the same way. Behaviour Capture is based on a similar idea of using captured traces to guide virtual character behaviours. Instead of a programmer specifying how each virtual character should move, speak, and interact with the environment, a designer takes control of the virtual character and performs the actions that the designer would like to see this character perform. This is done in a pre-published game session called training mode. The system remembers what each character did and uses this data to generate new behaviours during game play. With this technique, there is no need to write programming scripts. Although a motion

[6] http://www.youtube.com/watch?v=u8oNTLzCFNU

[7] http://www.trusoft.com/principles.html

capture analogy inspired the concept, the behaviour capture traces are not motion paths, they are higher-level designer intents, such as "play animation" or "speak to" as elaborated in Section 5.1 and Section 5.3.

The training examples are collected and used to train Hidden Markov-Models (HMMs). Each character deploys their own HMM to guide their behaviours in the game. The training traces are generalized to help speed up the behaviour designing process (detailed in Section 5.2). The idea of training virtual characters from user-generated traces is also similar to learning by demonstration. It had been used in The Restaurant Game by Orkin and Roy [47], though in that scenario behaviour patterns were extracted and combined through thousands of records of different human gameplays, where the players were given only vague goals and were free to play the game however they wanted. In our case the Behaviour Capture system combined with HMM is used by an expert designer who gets to decide what behaviours each character should exhibit.

To evaluate this idea, a prototype tool implementing the Behaviour Capture system was produced using the game engine of BioWare's Neverwinter Nights (NWN). An on-screen user interface was created utilizing the existing quick-slot bars available in the game engine. With the on-screen interface, a designer is able to capture all the types of behaviours in Cutumisu's behaviour ontology [22]. Proactive behaviours are used to train an HMM for each character. However, HMMs are not used to trigger reactive and latent behaviours since they only happen due to initiations external to the virtual character, either by another virtual character or by an event. Instead, the behaviour traces are used without HMMs, to directly generate the scripting code for the reactive and latent behaviours performed by the trainer.

5.1 Training Behaviours

A prototype of the Behaviour Capture system was constructed using the game engine of NWN. Since NWN is a medieval fantasy game, tavern scenes are

common. A tavern typically has patrons, servers, and sometimes entertaining bards. A typical tavern scene, called The Tavern, was created and used as a test-bed for my Behaviour Capture system. The Tavern (shown in Figure 30) consists of a stage for bards (marked by a stage spotlight), a counter for a tavern server, and numerous tables and chairs. Characters were placed in the tavern and designated as tavern patrons, a tavern server, and two bards. To start, they had no behaviours except the default idle animations of the NWN game engine.

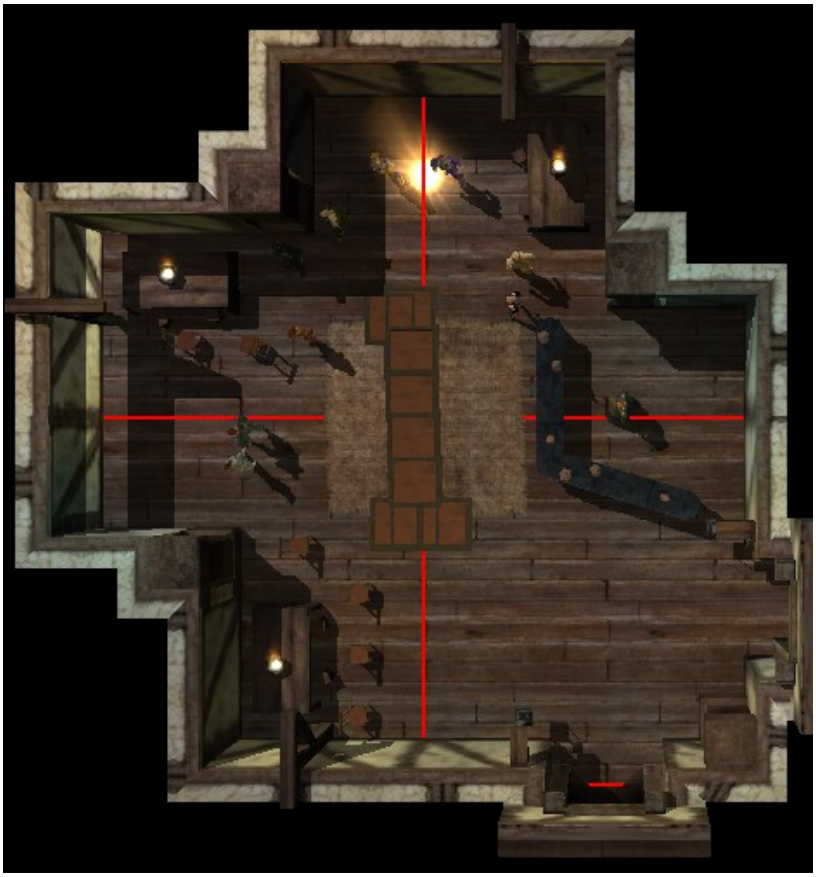


Figure 30. A top-down view of The Tavern.

To create behaviours for a tavern character, a designer starts the tavern "game" in training mode. This training mode starts with all the characters placed stationary in the scene. A designer takes control of a character, and this character becomes a trainee character. Figure 31 shows a trainee at the centre of the screen. At the bottom of the screen, there are sets of buttons representing the actions a trainee can perform

(pressing modifier keys display other sets of actions). If the designer clicks on the *Face* button and then clicks on another character, the trainee will turn and face the clicked character, and the *Face* action will be recorded as an action in a sequence of actions this trainee should perform. Some buttons are containers of actions, such as *Play animation*, which will further reveal different animation choices for the designer to choose from. The designer can switch trainees at any time, by clicking on the *Become* button and clicking on another character. If the trainer makes an error, the unwanted trace can be deleted from the training record.

Figure 31. Training a character. A portion of the action bar is enlarged in the figure for clarity.

Among the buttons there are also options to train a collaborative behaviour, where a designer would take control of one character, train one side of the behaviour, and then take control of the partner character, and train the other side of the behaviour. The Behaviour Capture system synchronizes the two sides of the behaviours when generating them, using behaviour multi-queues as proposed by Cutumisu et al. [24].

Training in the actual 3D virtual world lets the designer visualize the behaviours in the context that they will be performed by the characters being trained.

Event-triggered latent behaviours can be trained as well. The Behaviour Capture system lists allowable events in the game, such as stepping on a floor trigger, and by using the *Start Latent* button, the design can trigger an event and have it recorded as the event to trigger a sequence of latent behaviours.

This training interface was designed within the limitations of this particular game engine. The NWN game engine provides quick-slot bars at the bottom of the screen that can be customized to show different text and trigger different scripted actions. Using the quick-slot bars is one of the straight-forward ways to implement the Behaviour Capture system in NWN without the need to modify the source code of the game engine. If a different game engine allows the design of a fresh user interface, the allowable actions need not all appear at the bottom of the screen in a toolbar. They can be organized in different formats, such as pop-up dialog boxes, contextual menus, or groups of icons instead of texts. They can also be better organized to tailor to a gamepad controller instead of keyboard and mouse.

5.2 Generating Behaviours

After training data is gathered using the training mode, the Behaviour Capture system can produce behaviours for the virtual characters. However, a simple system that trains one virtual character at a time to interact with specific game objects would be too labour intensive and too deterministic to produce interesting variations in behaviour. Therefore, the Behaviour Capture system defines / supports three types of generalizations. First, with many virtual characters the designer may want the training of one virtual character to apply to multiple virtual characters – character generalization. Second, the designer may want a trained virtual character interaction with a specific object to generalize to an interaction with any one of a group of

objects – object generalization. Third, the designer may not want the virtual character to perform the training actions strictly in the training sequence order during game play and then repeat them in the same order over and over, so sequence generalization is performed using HMMs.

5.2.1 Character and Object Generalization

To support character and object generalization, the Behaviour Capture system uses categories of objects and characters. For example, if a designer trains a character to sit on a chair, the action does not have to be interpreted as "sit on that specific chair", but to sit on any chair in a category. During game play, the character would sit on any chair (or other object) in that category. Of course, to force a character to sit on a specific chair, the designer could give this chair its own category.

Object categorization mechanisms are game-specific. For example, in NWN, the designer can use two different categorization mechanisms – tags and blueprints. The designer assigns a tag string to each game object. The same tag can be assigned to different kinds of objects. During training, any interaction with an object can be generalized to an interaction with a random object with the same tag. For example, the trainer can train a virtual character to converse with any tavern patron with a specific tag, by just conversing with one of them. Alternatively, in NWN, the trainer can choose to generalize by blueprint – the template used to create an object. In this case, sitting on a chair would train an NPC to sit on any object created using the chair blueprint, regardless of tags. To support character generalization, when a designer trains a virtual character for one blueprint, all virtual characters with the same blueprint receive this training. It is easy to make custom blueprints from existing blueprints so creating categories that correspond to groups with common behaviours is straightforward. In this NWN implementation, the designer can easily toggle between using tags or blueprints to categorize objects. In Figure 31, the bottom left of the screen shows two buttons, By Blueprint, and By Tag, where the

designer can choose how to generalize the captured actions. Object generalization can also be turned off for specific actions or characters if desired.

5.2.2 Sequence Generalization

A Behaviour Capture system needs an algorithm to order behaviours based on the training traces. A simple approach, *no sequence generalization,* generates a sequence of actions that exactly matches the recorded sequence and then repeats this sequence. However, a player may regard this repetition as unnatural. Alternative approaches could select actions from the set of training actions in a non-deterministic manner. For example, the system could sample uniformly from the set of all trained actions using *random action sequence generalization*. However, in many situations the order is important, such as to approach a bartender before attempting to order a drink. To provide some designer control over the non-determinism, for each virtual character, the training actions are divided into a set of traces of actions. A designer starts a trace, performs a plausible sequence of actions and ends the trace. The designer usually performs many traces, each of which contains a short sequence of actions that form a cohesive sequence.

The *random trace sequence generalization* technique uniformly selects traces instead of actions. It tries to maintain the plausibility of action sequences created by the designer. However, over longer periods of time (many traces), this technique can still produce behaviours that players view as repetitious. Therefore, a third sequence generalization is created. It maintains traces to some extent, while producing emergent sequences that will reduce repeatability. This is the *HMM sequence generalization* that uses a Hidden Markov Model (HMM) to produce actions that have a bias towards selecting actions in the order specified by the designer.

A Markov Model is a statistical model with states, transitions and outputs. One state is a special state called the start state. Each state is connected to a set of other states by probabilistic transitions. In addition, each state has a set of output probabilities of

producing a set of outputs. An HMM is a Markov Model whose states are unobserved (hidden). In this application, the output is one of the behaviour actions that the designer used in a trace. Hidden states are hidden to a game designer. The number of hidden states is a parameter of the HMM sequence generalization technique. An example is shown in Figure 32.

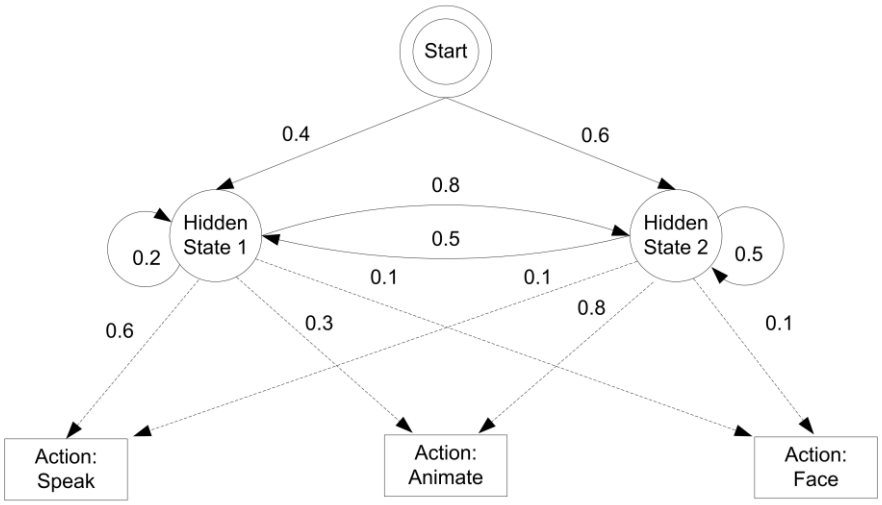


Figure 32. An example Hidden Markov Model with two hidden states, three outputs (actions), with transition and output probabilities.

Each character uses one HMM to generate the proactive behaviours. The Baum-Welch Algorithm [4], a generalized expectation-maximization algorithm, uses the training traces to teach the HMM. The HMM adjusts its transition and output probabilities to fit the trace sequences. If the trainer trains the virtual character to converse three times as often as ordering a drink, the HMM will generate a similar 3 to 1 ratio. The behaviours generated by the HMM are stochastic, but are somewhat consistent with the training traces. The number of hidden states parameter controls the consistency, with higher consistency achieved by more hidden states. The Baum-Welch algorithm can start with random initial conditions so prior knowledge is not necessary. The algorithm finds a local maximum for the HMM parameters that maximize the probability of the trace sequences. While the algorithm is not guaranteed to converge at the global maximum, in practice it has been found to yield good results. If the HMM produces strange behaviour sequences, the designer can

easily re-train the HMM with the same training data, or capture more sequences of data.

5.3 Quality Evaluation

To evaluate the quality of the behaviours produced by Behaviour Capture, a user study was designed with The Tavern. Inside The Tavern, there are 12 tavern patrons, 2 bards, and 1 tavern server. For the duration of the scene, these characters stay in The Tavern and do not ever leave.

To provide a fair comparison of the Behaviour Capture system plus HMM with other techniques, six variants of The Tavern were created. Each scene variation takes place in an identical setting and with an identical set of characters. The only difference was the technique used to control the behaviours of the virtual characters in each scene (Table 10). The six scene variations were constructed using the techniques listed in Table 1. In an attempt to reduce bias, a random order of the scene variations was generated that resulted in Technique T1 - Scene 4, Technique T2 - Scene 6, Technique T3 - Scene 5, Technique T4 - Scene 1, Technique T5 - Scene 3, Technique T6 - Scene 2.

Technique T1 is a baseline, with all characters exhibiting only stock idling animations provided by the NWN game engine, such as stretching their arms once in a while. They do not move around.

Technique T2 is hand-scripted to mimic tavern characters in a representative commercial role-playing game, Dragon Age: Origins. This scene variation was scripted to combine the behaviours in two taverns of Dragon Age: Origins, one in Lothering (with two bards entertaining) and one in Redcliffe (a server who walks around), forming a tavern with tavern patrons, a moving tavern server, and bards. Only using one of the taverns would have resulted in even simpler behaviours and even worse in comparison. In Dragon Age: Origins, tavern patrons engage in

conversations but do not respond to bards and do not move around. An inspection of a newer game in the same series, Dragon Age: Inquisition [8], reveals that the behaviours in the tavern scenes have not changed perceptibly since the previous two games were released, even though a more modern game engine is being used that has significantly improved the graphical fidelity, as noted in the abstract and introduction of this dissertation.

Technique	Behaviour Generation Method
T1	No behaviours added (idle animation only)
T2	Behaviours hand-scripted by a programmer to mimic commercial game Dragon Age: Origins
T3	Behaviour capture with no sequence generalization
T4	Behaviour capture with random action sequence generalization
T5	Behaviour capture with random trace sequence generalization
T6	Behaviour capture with HMM sequence generalization (using 8 hidden states)

Table 10. Behaviour generation techniques.

The other four techniques used Behaviour Capture to train the characters. Ten traces were captured in training mode. In these scene variations, the three types of characters behave as follows:

1. Tavern patrons: they exhibit independent behaviours (walking around, saying one-liners to themselves, finding a table to stay at, animating their hands or facing an object); collaborative behaviours (talking to one another on several topics and talking to tavern server to order a drink); and latent behaviours (responding to bards). The independent and collaborative behaviours are illustrated in Figure 33 and Figure 34. Figure 35 shows the latent behaviour where the tavern patrons turn and cheer for the two bards as

they finish their performance. The cheer behaviour is triggered by the event of the bards leaving their performance spotlight. The patrons resume their interrupted behaviour once they finish cheering.

2. A tavern server: he exhibits independent (walking around) and collaborative behaviours (talking to tavern patron to fill a drink order).

3. Bards: exhibit independent behaviours (performing on stage under spotlight).

The differences between these four scene variations lie in the behaviour sequence generation methods described in Section 5.2.2. One uses no sequence generalization, one uses random action sequences, one uses random trace sequences, and the final one uses HMM sequence generalization.

Figure 33. A tavern patron displaying an independent behaviour of saying "I'm tired" to himself.

Figure 34. A tavern patron and the server displaying a collaborative behaviour to fulfill a drink order.

Figure 35. Tavern patrons displaying a latent behaviour of cheering in response to the performances of the bards.

The goal here was to show that Behaviour Capture is a viable alternative to typical commercial game virtual character behaviours created by manual scripting. There are two hypotheses:

- First, the behaviours trained by Behaviour Capture would be considered better than the *no behaviour* and *manually scripted behaviour* techniques.
- Second, sequence generalization would be a positive factor in the perception of believable characters. Specifically, that the order of Behaviour Capture rankings would be: HMM, random traces/random actions, and no sequence generalization.

Study participants were recruited from a first year university psychology class. They did not necessarily have experience with role-playing video games. Participants were asked to watch the six scene variations, and to rank them according to these criteria: active characters, unpredictable characters, plausible sequences of actions, diverse actions and overall believability. I also asked the participants to rate the overall believability of each variation. Ranking scores are from 1 to 6: for each participant response, the highest ranked behaviour received a score of 6, the second highest ranked received a score of 5, etc. Rating scores range from 1 to 4: 1-very unbelievable, 2-unbelievable, 3-believable, 4-very believable. A neutral option was not included to force a choice.

5.3.1 Preliminary User Study

A preliminary user study was conducted to evaluate the effects of the number of training traces. I wanted to determine how the number of traces would influence user perceptions. The goal was to determine if users could distinguish between a small number of training traces (4) and a larger number (9). The user study was constructed similarly to the main user study that is described in the previous section, except that scene variations with 4 training traces were compared to scene variations

with 9 training traces. The results show that with 95% confidence the variations with higher numbers of traces produced more believable scenes. This is an expected trade-off between quality and workload. Based on this result, I have set the number of training traces in the main user study closer to the higher number (ten traces were selected). A higher number of traces could be used, but significantly more traces might make the technique impractical in the commercial domain, since it could require an unacceptably long training time and that would defeat the entire purpose of this method.

5.3.2 Results and Analysis

In the main user study, 27 participants were recruited from a first-year undergraduate psychology class. Unfortunately, a few participants did not answer carefully enough for their responses to be considered valid, with some participants not answering some questions and one participant providing what seemed to be random answers (e.g. ranking 1,2,3,4,5,6 for all criteria). Therefore, the results of each questionnaire were validated for self-consistency. One question asked the respondent to rank the six variations according to overall believability, while another question asked the respondent to rate the six variations individually on a scale of 1 to 4 according to overall believability. To ensure that the participants answered the questions carefully, a questionnaire was removed if the rankings and ratings scores contained more than one inconsistency between the rating and ranking questions. After this consistency check, a total of 21 valid questionnaires remained.

Table 11 shows the average ranking scores for the 6 techniques. Each number represents the average ranking score for the particular technique for the particular criteria over the 21 responses. For example, technique T6 is ranked 4.29 on average (where 6 is the highest ranking score) among the 6 techniques, according to the criteria *active characters*. The results show that in general, technique T6 is ranked highly for all the criteria except *unpredictable characters*. Note that high rankings are better than low ones for all criteria except possibly for *unpredictable characters,*

since believability and unpredictability are probably not conducive in this particular scene (at least that was what we expected).

Table 12 shows the average ranking scores and the average rating scores of the six scene variations according to *overall believability*. Again, T6 received higher scores than the rest of the scene variations in both measures. Overall believability is not an average of the other criteria. It was a separate question on the survey.

Behaviour Technique	Active characters	Unpredictable characters	Plausible sequences	Diverse actions
T1	1.19 (0.60)	3.00 (2.24)	2.00 (1.45)	1.24 (0.54)
T2	3.38 (1.72)	3.29 (1.65)	2.38 (1.63)	3.29 (1.68)
T3	3.90 (1.67)	4.48 (1.60)	3.57 (1.57)	4.10 (1.67)
T4	4.29 (1.23)	3.52 (1.50)	4.38 (1.40)	3.86 (0.96)
T5	3.95 (1.53)	3.62 (1.60)	4.29 (1.35)	3.81 (1.66)
T6	4.29 (1.06)	3.10 (1.34)	4.38 (1.20)	4.71 (1.10)

Table 11. Average Technique Ranking Score (6 is Highest, 1 is Lowest). Higher numbers are better in all criteria except *unpredictable characters*. Standard deviations are shown in parentheses.

Behaviour Technique	Average Ranking Score	Average Rating Score
T1	1.33 (0.80)	1.20 (0.41)
T2	3.05 (1.69)	2.05 (1.16)
T3	3.67 (1.28)	2.19 (0.87)
T4	4.00 (1.26)	2.71 (0.90)
T5	4.10 (1.61)	2.57 (0.87)
T6	4.86 (1.15)	2.85 (0.59)

Table 12. Average Overall Believability Ranking and Rating Score. Standard deviations are shown in parentheses.

Criteria	Average Importance
active	1.76
unpredictable	0.71
plausible	2.19
diverse	1.52

Table 13. The average importance of the four criteria. A positive number means important in contributing positively to overall believability. A negative number means important in contributing negatively.

A Friedman statistical test was used to compare each column of Table 11 and Table 12 with ranking scores to avoid the alpha-inflation effect. It indicated that there are significant differences in the average scores of the six techniques for each criterion at the 95% confidence level. For rating scores, ANOVA was used. Based on the positive result of the Friedman or ANOVA tests, T-tests were used to compare pairs of scores.

The most obvious result is that T6 was ranked significantly higher than T1 and T2 in all aspects except unpredictability. This study indicates that hand-scripted characters mimicking Dragon Age: Origins are perceived as less diverse, less plausible and have less active characters than character whose behaviours were generated using Behaviour Capture with HMM sequence generalization. The study also shows that T6 ranked significantly higher than all other tested techniques for overall believability. The p-values of the T-tests are presented in Appendix D.

To evaluate the importance of our criteria, I asked participants to rate the importance of each of the four criteria contributing to a natural-looking scene, positively or negatively. Participants rated each criterion on a scale of -3 to 3 (-3: very negatively, -2: negatively, -1: mildly negatively, 0: neutral (not important), 1: mildly positively, 2: positively, 3: very positively). The larger the absolute value the more important it is (in either direction). Table 13 show the average importance computed from the

responses of the study participants. As expected, unpredictability is the least important in the eyes of the participants, with many participants responding that it contributes negatively to a natural-looking scene.

5.3.3 Gender Analysis

One of my subsidiary research goals is to explore the difference between genders in perceiving the behaviours of virtual characters in video games. Section 3.3.4 presents results that suggested potential differences in male and female participants in perceiving cyclic daily behaviours. To test the hypothesis that there would also be differences in how male and female participants perceive role-level behaviours, additional user studies were conducted to examine what differences gender might play on the different levels of sophistication of the role-level behaviours.

The gender study focuses on three of the behaviour variations, one for each scene: the baseline idle behaviour, the behaviour hand-scripted to mimic Dragon Age: Origins, and the behaviour produced using Behaviour Capture plus HMM. Participants were asked to examine each scene and rate them according to the overall believability of the behaviours of virtual characters, on a scale of 1 to 4 (very unbelievable, unbelievable, believable, very believable). In addition, the participants were asked to complete a questionnaire on their gender and how often they played video games.

A consistency check similar to the main study was used to validate the participant responses. 79 valid participant responses were included in this gender study. These participants were undergraduate students taking a first-year psychology class. They were between the ages of 17 and 58 (mean of 20.6), with 51 females and 28 males. Of these, 12 of the females were gamers and 39 were non-gamers, while 12 of the males were gamers and 16 were non-gamers. For this study, a gamer is defined as someone who plays story-based video games at least once a week.

This study examined whether gender and gaming experience affect perception of virtual character behaviour. In this study, for males, gaming experience does not affect the perception of behaviour quality - the values in the last two columns of Table 14 are similar on a row-by-row basis. However for females, gaming experience increases their ability to discriminate between differences in behaviour. There is a significant difference between the 1.42 and 2.26 entries in Table 14 at a 95% confidence level ($p = 0.000$), indicating that female gamers are less impressed than female non-gamers by the level of behaviours in the commercial game Dragon Age: Origins (technique T2). This may indicate that as females gain more gaming experience they will be more appreciative of better behaviours. In other words, female non-gamers may be just as satisfied with existing commercial game behaviour quality as male gamers and male non-gamers, but as females become more experienced gamers, this study indicates, they may become more appreciative of the improved behaviours. More research should be conducted in this area to verify the results, as this study included a relatively small number of female and male gamers.

Behaviour Technique	Everyone	Female	Male	Female Gamer	Female Non-Gamer	Male Gamer	Male Non-Gamer
T1	1.23	1.20	1.29	1.17	1.21	1.50	1.13
T2	2.15	2.06	2.32	1.42	2.26	2.25	2.38
T6	2.97	3.02	2.89	3.00	3.03	2.92	2.88

Table 14. Average Technique Rating Score (4 is Highest, 1 is Lowest) for overall believability, divided by participant gender and gaming experience.

5.4 Usability Evaluation

It is worth noting that the implemented tool of the Behaviour Capture system vastly outperforms traditional manual scripting in the aspect of usability. The Tavern, a scene containing tavern patrons, bards, and a server, is described in the previous section. In one experiment, the entire set of behaviours for every character in the scene was produced both with the Behaviour Capture tool, and with manual scripting in the NWScript language. For an experienced user with both the tool and manual scripting in NWScript, the difference in time investment between the two methods was extensive. Re-creating the scene with the tool required about 20 minutes. Technique T6 with HMM sequence generalization using 8 hidden states was deployed by the Behaviour Capture tool, generating the most believable scene out of all six variations. The less believable variation, Technique T5 with random trace sequences, was re-created using manual scripting. Manually re-creating The Tavern with NWScript took about nine and a half hours. To fully re-create the scene to the extent of Technique T6 would take even longer with manual scripting.

This experiment was repeated with three other participants, who are experienced programmers but had no previous experience with the Behaviour Capture tool. The three participants were given a written tutorial describing the Behaviour Capture tool, and the NWScript language reference (NWN Lexicon). They were asked to create the behaviours for all characters in The Tavern, using one method, and then the other.

With the Behaviour Capture tool, the three participants spent an average of 14 minutes on the written tutorial of the tool, followed by using the tool to create the behaviours. On average, each participant spent 37 minutes using the tool, and the resulting behaviours were complete and correct.

With manual scripting though, none of the three participants were able to complete the required behaviours within a three-hour limit. The resulting behaviours ranged

from less-than-half completed to a quarter completed, according to the stated requirements. The participants estimated that another four to twelve hours of work was necessary to complete the scene. From this experiment it is clear that the Behaviour Capture tool outperforms manual scripting in terms of usability.

5.5 Discussions

The quality user study showed that Behaviour Capture with HMMs produces behaviours that are perceived as significantly superior with regards to overall believability compared to the scripted behaviours and Behaviour Capture without HMMs in the commercial story-based game NWN. It is worthy to note that in the user study, the tavern patrons required only 10 traces of 3 actions each to train successfully. With each tavern patron, 8 hidden states were used with 14 unique actions, resulting in 64 internal transition probabilities and 112 output probabilities.

While the Behaviour Capture system works in a specific situation, extending the system to a world with multiple scenes can be problematic. Training a virtual character replaces the need to manually script a virtual character. As the number of situations increase, the length of the training traces needs to increase, the number of unique actions increases, and the size of the HMM increases with the length of the training traces and the number of unique actions. This is one of the reasons why the Behaviour Capture system has been used only in the context of a local role, and the behaviours beyond a role are handled by the other layers of the Tiered Behaviour Architecture.

The Behaviour Capture system is powerful at tracking a sequence of plausible actions, such as first picking up a drink, then delivering the drink to a person, but the nature of an HMM implies that it does not implicitly take into account the passing of time. This is also a limitation of the technique if we were to apply it to a higher level of the Tiered Behaviour Architecture. When determining a daily schedule, for

example, a virtual character needs to be aware of the time of day and factor the time into their decisions. Moreover, producing a daily cyclic schedule requires the character to repeat certain actions at regular (even with stochasticity) times each day. The fundamentally different problem of decision-making at the objective level is one instance where the Cyclic Scheduling method introduced in the previous chapter is naturally preferred over the Behaviour Capture system. However, when producing local behaviours where the actions are not set according to a timed schedule (such as ordering drinks in a tavern or having a business dealing in a market), Behaviour Capture shows its advantage in quickly producing actions with relatively short design time.

6. Conclusions

The video game industry continues to grow and game designers are becoming more specialized in their own areas. Recent story-based video games have started providing tools so that non-professionals can design their own stories within the game engine. However, in order to successfully use these tools, a designer needs to have background in computer programming, since the majority of the virtual characters are controlled by programmed artificial intelligence. In recent years, the believability of the behaviours of these virtual character have developed more slowly than other aspects of video games, mainly due to the cost of scripting complex and believable virtual characters. To tackle this bottleneck in content creation, this dissertation provides the following important contributions.

This dissertation proposes a set of standardized metrics for evaluating an architecture for behaviours of virtual characters. Four metrics are defined: Expressiveness, Efficiency, Quality, and Usability. With these metrics, this dissertation proposes a Tiered Behaviour Architecture model approach to virtual character behaviours. Behaviour control is divided into objective level and role level, and each level of control is modular and reusable. Behaviours as produced by this model have been shown through studies to be more believable than the current state of the art. The model represents a general behaviour architecture with powerful scheduling capabilities across a wide range of story-based games.

At the objective level, this dissertation proposes a hierarchical Cyclic Scheduling technique for the automatic generation of daily schedules for virtual characters. Designers are able to specify partial constraints and details they deem important, and the Scheduler fills in the rest. A usability study shows that the tool implementing Cyclic Scheduling enables designers to produce behaviours more efficiently than traditional manual scripting methods.

At the role level, this dissertation presents a technique called Behaviour Capture, together with the machine learning of Hidden Markov Models to produce fine-grain behaviours for characters assuming individual roles in local scenes. User studies have shown that the resulting behaviours are a noticeable improvement over manually scripted characters in commercial games, and that behaviours generated by trained Hidden Markov Models are more believable than using simpler combinations of behaviour traces. Together, these new game designer models and tools allow them to create behaviours for virtual characters, without having to learn complicated programming.

There is a strong interest in the AI and games community to explore the use of AI to provide a more enjoyable playing experience. The architecture and its implementation described in this dissertation is one step in this direction.

6.1 Future Work

While the Behaviour Capture system has been validated through user studies, it is currently a stand-alone system in a larger behaviour architecture. While the Behaviour Capture system is designed to produce role behaviours in a modular and reusable fashion, its implementation details depend greatly on the underlying game engine. The user studies validating the system presented in this dissertation were all conducted in the NWN game engine, with the Behaviour Capture interface implemented within the limitations of the NWN game UI. The NWN game engine provides a 3D third-person top-down view of the world. However, an unrelated user study found that people sometimes prefer to look at 2D top-down representations of the real world as opposed to 3D representations [33]. With this study, participants were asked to play a game of "Outbreak: Safety First" in three versions: 2D virtual world, 3D virtual world and a real-world mobile phone version where participants

ran in a physical location. The 2D version was rated by participants as "most valuable, useful, comfortable, pleasant, and stimulating" of the three. Would designers prefer a Behaviour Capture system where the training interface is a 2D representation of the game world and detached from the particularities of a game engine, rather than performing training in the same environment as the actual game?

Figure 36 shows a mock-up of a 2D representation of a tavern scene described in Section 5. The change to game-independent Behaviour Capture would make it easier to create a common training interface that is consistent across all game platforms, instead of being specific to a game engine. In this case, the system would need to read the spatial location of objects from the game-dependent scene files to present the objects in a common format.

Figure 36. A mock-up of a 2D representation of a tavern scene.

A game-independent Behaviour Capture interface could be integrated into an existing visual scripting tool. One such example is ScriptEase II developed at the

University of Alberta [55]. ScriptEase II uses libraries of causes and effects, which map directly onto events and action API calls of the targeted game engine. If integrated, the library of causes and effects used by ScriptEase II could also be used by the Behaviour Capture system. Figure 37 shows a mock-up of an integrated interface. The right-side panel is an interactive 2D representation of a game scene, where a designer can take control of a virtual character in a scene and train the virtual character with the desired behaviours. However, instead of relying on the user interface of a particular game engine to provide access to different actions (such as buttons on the quickbar), the effects panel on the left side can be used for action selection. With the existing drag-and-drop interface, a desired effect (an action) could be dragged (or selected) for a character, signifying that the action is to be performed by the virtual character as a part of the behaviour being trained.

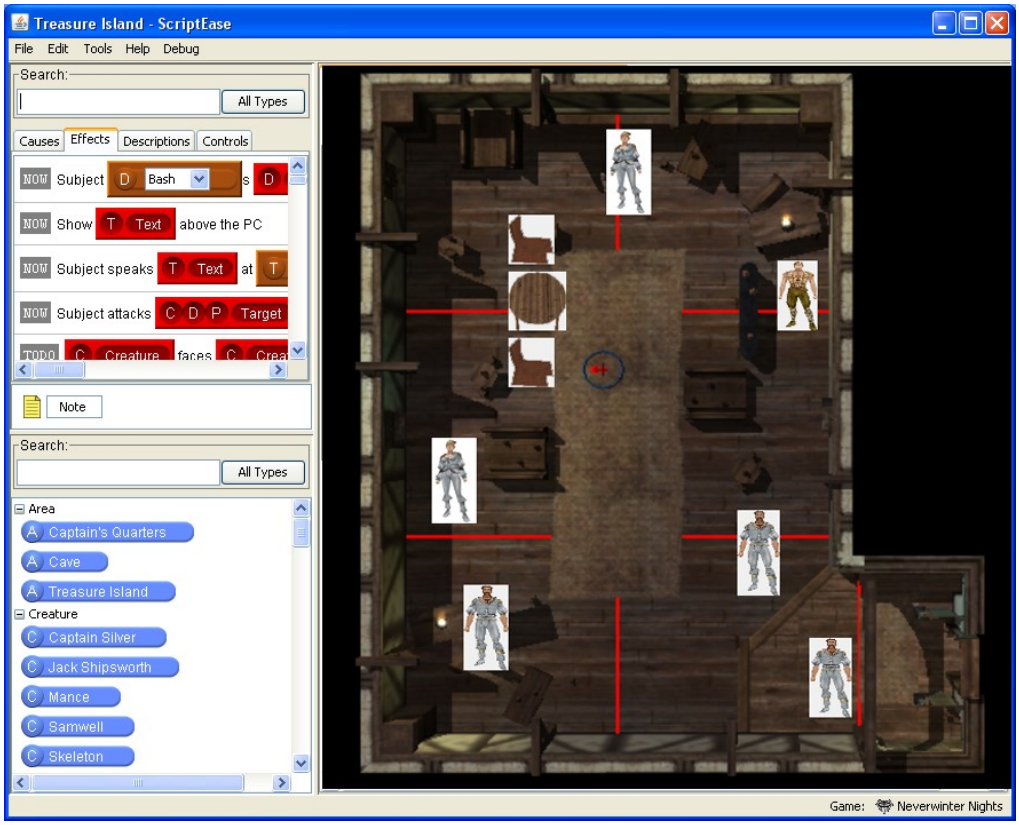


Figure 37. A mock-up of a visual scripting tool (ScriptEase II) integrated with the Behaviour Capture system.

Latent behaviours, which are triggered by events (or causes, as they are called in ScriptEase II), could also be dragged-and-dropped (or selected) from the causes panel on the top-left onto a virtual character in the Behaviour Capture interface on the right side.

A future evaluation could be conducted to find out whether such a game-independent representation is preferred over an in-game view in the context of Behaviour Capture. The evaluation could be done through a usability study which compares the game-independent Behaviour Capture system with the in-game Behaviour Capture system that was described previously. The user study could ask participants to use each of the systems to create a pre-determined set of behaviours for virtual characters. The scenes that are scripted by the participants could also be analyzed to evaluate the completeness, correctness, completion time, and efficiency.

A challenge to making a game engine-independent tool is the integration of the tool with a particular underlying game engine. A common interface must be developed so that the tool can communicate with a game engine. ScriptEase II utilizes "translators" that are custom-written for each game engine, which transform ScriptEase commands into the code that the game engine can understand. Such a step is necessary for any tool that is game engine-independent.

Bibliography

[1] William Albert and Thomas Tullis, *Measuring the User Experience*, 2, Ed. Waltham, MA: Morgan Kaufmann, 2013.

[2] Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice*. Boston, MA, USA: Addison-Wesley, 1998.

[3] Joseph Bates, "Virtual reality, art, and entertainment," *Presence: Teleoperators and Virtual Environments*, vol. 1, no. 1, pp. 133 - 138 , Winter 1992.

[4] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164-171, 1970.

[5] Matt Bertz. (2011, January) The Technology Behind The Elder Scrolls V: Skyrim. [Online]. http://www.gameinformer.com/games/the_elder_scrolls_v_skyrim/b/xbox360/archive/2011/01/17/the-technology-behind-elder-scrolls-v-skyrim.aspx

[6] Bethesda Game Studios. (2006, March) The Elder Scrolls IV: Oblivion.

[7] Bethesda Game Studios. (2011, November) The Elder Scrolls V: Skyrim.

[8] BioWare. (2014, November) Dragon Age: Inquisition.

[9] BioWare. (2009, November) Dragon Age: Origins.

[10] BioWare. (2002, June) Neverwinter Nights.

[11] Barry Boehm. (2000, September) And Very Few Lead Bullets, Either. Keynote Address.

[12] Michael Booth. (2009) The AI Systems of Left 4 Dead. [Online]. http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf

[13] Adi Botea, Martin Mueller, and Jonathan Schaeffer, "Fast Planning with

Iterative Macros," in *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, Hyderabad, 2007, pp. 1828-1833.

[14] Michael Bowling and Manuela Veloso, "Rational and Convergent Learning in Stochastic Games," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI)*, 2001, pp. 1021-1026.

[15] Entertainment Software Association of Canada. (2014) Essential Facts about the Canadian Video Game Industry. [Online]. http://theesa.ca/wp-content/uploads/2014/11/ESAC-Essential-Facts-2014.pdf

[16] Alex J. Champandard. (2007, September) AIGameDev. [Online]. http://aigamedev.com/open/article/hfsm-gist/

[17] Alex J. Champandard. (2012, February) Understanding the Second-Generation of Behavior Trees. [Online]. http://aigamedev.com/insider/tutorial/second-generation-bt/

[18] Michael Chung, Michael Buro, and Jonathan Schaeffer, "Monte Carlo Planning in RTS games," in *Proceeding of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2005, pp. 117-124.

[19] Mark Claypool, Kajal Claypool, and Feissal Damaa, "The effects of frame rate and resolution on users playing First Person Shooter games," in *Proceedings of SPIE 6071, Multimedia Computing and Networking*, 2006.

[20] Paul Clements, Rick Kazman, and Mark Klein, *Evaluating Software Architectures: Methods and Case Studies*.: Addison-Wesley, 2002.

[21] Alexandra Coman and Hector Munoz-Avila, "Plan-Based Character Diversity," in *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-12)*, Stanford, 2012, pp. 118-123.

[22] Maria Cutumisu, *Using behaviour patterns to generate scripts for computer role-playing games*. Canada: University of Alberta, 2009.

[23] Maria Cutumisu et al., "ScriptEase: A Generative/Adaptive Programming

Paradigm for Game Scripting ," *Science of Computer Programming*, vol. 67, no. 1, pp. 32-55, June 2007.

[24] Maria Cutumisu and Duane Szafron, "An Architecture for Game Behavior AI: Behavior Multi-Queues ," in *Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-09)*, Stanford, 2009, pp. 20-27.

[25] Maria Cutumisu, Duane Szafron, Michael Bowling, and Richard S. Sutton, "Agent Learning using Action-Dependent Learning Rates in Computer Role-Playing Games," in *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-08)*, Stanford, 2008, pp. 22-29.

[26] Paul Debevec and Leonard McMillan, "Guest Editors' Introduction: Imaged-Based Modeling, Rendering, and Lighting," *IEEE Computer Graphics and Applications*, vol. 22, no. 2, pp. 24-25, 2002.

[27] EA Maxis. (2000, February) The Sims.

[28] Markus Enzenberger, Martin Mueller, Broderick Arneson, and Richard Segal, "Fuego - An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 259-270, 2010.

[29] Sumudu Fernando and Martin Mueller, "Analyzing Simulations in Monte Carlo Tree Search for the Game of Go," in *Computers and Games*, 2013.

[30] Kenneth D. Forbus and Will Wright, "Some notes on programming objects in The Sims," Northwestern University, 2011.

[31] Michael Gleicher, "Animation From Observation: Motion Capture and Motion Editing," *ACM SIGGRAPH Computer Graphics*, vol. 33, no. 4, 2000.

[32] Paulo Gomes and Arnav Jhala, "AI Authoring for Virtual Characters in Conflict," in *Proceedings of the Ninth Annual AAAI Conference on Artificial*

Intelligence and Interactive Digital Entertainment (AIIDE-13), Boston, 2013, pp. 135-141.

[33] Lucio Gutierrez, Eleni Stroulia, and Ioanis Nikolaidis, "fAARS: A Platform for Location-Aware Trans-reality Games," *Lecture Notes in Computer Science*, vol. 7522, pp. 185-192, 2012.

[34] Chris Hecker. (2014, July) My liner notes for spore. [Online]. http://chrishecker.com/My_Liner_Notes_for_Spore

[35] Damian Isla, "Handling Complexity in the Halo 2 AI," in *Proceedings of Game Developers Conference*, 2005.

[36] Cian Kearney, *Character Based Interactive Storytelling for Role Playing Games*. Ireland: University of Dublin, Trinity College, 2012.

[37] John-Paul Kelly, Adi Botea, and Sven Koenig, "Offline Planning with Hierarchical Task Networks in Video Games," in *Proceedings of the Fourth Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE-08)*, Stanford, 2008, pp. 60-65.

[38] Jon Kleinberg and Éva Tardos, *Algorithm Design*.: Addison-Wesley, 2005.

[39] Levente Kocsis and Csaba Szepesvári, "Bandit based Monte-Carlo planning," *European Conference on Machine Learning (ECML)*, pp. 282–293, 2006.

[40] Michael Mateas and Andrew Stern, "A Behavior Language for Story-Based Believable Agents," *IEEE Intelligent Systems*, vol. 17, no. 4, pp. 39-47 , July 2002.

[41] Michael Mateas and Andrew Stern, "Façade: An Experiment in Building a Fully-Realized Interactive Drama," in *Game Developers Conference*, 2003.

[42] Masahiro Mori, "The Uncanny Valley," *Energy*, vol. 7, no. 4, pp. 33-35, 1970.

[43] Robert Mullon. (2012) What's new in Skyrim: New Radiant AI and Radiant Story. [Online]. http://elder-scrolls.alteredgamer.com/tes-5-skyrim/116302-whats-new-in-skyrim-new-radiant-ai-and-radiant-story/

[44] Curtis Onuczko et al., "A Pattern Catalog for Computer Role Playing Games," in *Proceedings of GameOn North America*, Montreal, 2005, pp. 33-38.

[45] Origin Systems. (1988, March) Ultima V: Warriors of Destiny.

[46] Jeff Orkin, "3 States and a Plan: The AI of F.E.A.R.," in *Proceedings of the Game Developers Conference*, 2006.

[47] Jeff Orkin and Deb Roy, "The Restaurant Game: Learning Social Behavior and Language from Thousands of Players Online," *Journal of Game Development*, vol. 3, no. 1, pp. 39-60, December 2007.

[48] Ushma Kesha Patel, Purvag Patel, Henry Hexmoor, and Norman Carver, "Improving Behavior of Computer Game Bots Using Fictitious Play," *International Journal of Automation and Computing*, vol. 9, no. 2, pp. 122-134, 2012.

[49] Randy Pausch et al., "Alice: Rapid Prototyping System for Virtual Reality," *IEEE Computer Graphics and Applications*, May 1995.

[50] Mitchel Resnick et al., "Scratch: Programming for All," *Communications of the ACM*, vol. 52, no. 11, pp. 60-67, November 2009.

[51] Richard Garriott; Origin Systems. (1981, June) Ultima I: The First Age of Darkness.

[52] Mark O. Riedl and Vadim Bulitko, "Interactive Narrative: An Intelligent Systems Approach," *AI Magazine*, vol. 34, no. 1, pp. 67–77, 2013.

[53] Mark O. Riedl and Alexander Zook, "AI for Game Production," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, Niagra Falls, 2013.

[54] Stuart Russell and Peter Norvig, "Constraint Satisfaction Problems," in *Artificial Intelligence: A Modern Approach*.: Prentice Hall, 2002, ch. 5, pp. 137-160.

[55] Kevin Schenk et al., "ScriptEase II: Platform Independent Story Creation Using High-Level Patterns," in *Proceedings of the Ninth AAAI Conference*

on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-13), Boston, 2013, pp. 170-176.

[56] Edward Schneider, Yifan Wang, and Shanshan Yang, "Exploring the Uncanny Valley with Japanese Video Game Characters," in *DiGRA*, 2007, pp. 546-549.

[57] Brian Schwab, *AI Game Engine Programming*. Boston: Course Technology, 2009.

[58] AmirAli Sharifi, *Generating adaptive companion behaviors using reinforcement learning in games*. Canada: University of Alberta, 2010.

[59] AmirAli Sharifi, Richard Zhao, and Duane Szafron, "Learning Companion Behaviors Using Reinforcement Learning in Games," in *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-10)*, Stanford, 2010, pp. 69-75.

[60] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma, "Adaptive Game AI with Dynamic Scripting," *Machine*, vol. 63, no. 3, pp. 217-248, 2006.

[61] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[62] David Thue, Vadim Bulitko, and Marcia Spetch, "Making Stories Player-Specific: Delayed Authoring in Interactive Storytelling," in *Interactive Storytelling*.: Springer, 2008, pp. 230-241.

[63] David Thue, Vadim Bulitko, Marcia Spetch, and Eric Wasylishen, "Interactive Storytelling: A Player Modelling Approach," in *Proceedings of the Third Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-07)*, Stanford, 2007, pp. 43-48.

[64] Emmett Tomai and Roberto Flores, "Adapting in-game agent behavior by observation of players using learning behavior trees," in *Proceedings of the 9th International Conference on the Foundations of Digital Games*, Fort Lauderdale, 2014.

[65] Turtle Rock Studios; Valve Corporation. (2008, October) Left 4 Dead.

[66] The UESPWiki. (2014) Skyrim:People. [Online]. http://uesp.net/wiki/Skyrim:People

[67] Ian Wright and James Marshall. (2000, June) More AI in Less Processor Time: 'Egocentric' AI. [Online]. http://www.gamasutra.com/view/feature/131567/more_ai_in_less_processor_time_.php

[68] Richard Zhao and Duane Szafron, "Generating Believable Virtual Characters Using Behavior Capture and Hidden Markov Models," *Advances in Computer Games: Lecture Notes in Computer Science*, vol. 7168, pp. 342-353, 2012.

[69] Richard Zhao and Duane Szafron, "Learning Character Behaviors using Agent Modeling in Games," in *Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-09)*, Stanford, 2009, pp. 179-185.

[70] Richard Zhao and Duane Szafron, "Using Cyclic Scheduling to Generate Believable Behavior in Games," in *Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-14)*, Raleigh, 2014.

[71] Richard Zhao and Duane Szafron, "Virtual Character Behavior Architecture using Cyclic Scheduling," in *Proceedings of the 9th International Conference on the Foundations of Digital Games*, Fort Lauderdale, 2014.

Appendices

Appendix A

Additional data on the quality study of the Tiered Behaviour Architecture are shown below. The tables below show the p-values of T-tests, preformed pair-wise between two behaviour variations. Tables 15 and 16 are for the overall results combining both male and female participants. Tables 17 and 18 are for female participants only, and Tables 19 and 20 are for male participants.

	MSSMDR	MSSMR	MSS	MS	SSS
SS	**0.000**	**0.000**	**0.002**	**0.003**	0.314
SSS	**0.000**	**0.000**	**0.002**	**0.006**	
MS	**0.000**	**0.005**	0.233		
MSS	**0.000**	**0.015**			
MSSMR	**0.000**				

Table 15. p-values from T-tests of ranking scores overall. Bold entries are significant at the 95% level.

	MSSMDR	MSSMR	MSS	MS	SSS
SS	**0.000**	**0.000**	**0.000**	**0.000**	0.347
SSS	**0.000**	**0.000**	**0.000**	**0.000**	
MS	**0.000**	0.092	0.053		
MSS	**0.000**	0.448			
MSSMR	**0.000**				

Table 16. p-values from T-tests of rating scores overall. Bold entries are significant at the 95% level.

	MSSMDR	MSSMR	MSS	MS	SSS
SS	**0.000**	**0.001**	**0.001**	**0.023**	0.159
SSS	**0.000**	**0.006**	**0.046**	0.087	
MS	**0.000**	**0.029**	0.266		
MSS	**0.000**	0.089			
MSSMR	**0.000**				

Table 17. p-values from T-tests of ranking scores for female participants. Bold entries are significant at the 95% level.

	MSSMDR	MSSMR	MSS	MS	SSS
SS	**0.000**	**0.003**	**0.000**	**0.008**	0.434
SSS	**0.000**	**0.006**	**0.003**	**0.012**	
MS	**0.000**	0.195	**0.035**		
MSS	**0.000**	0.368			
MSSMR	**0.000**				

Table 18. p-values from T-tests of rating scores for female participants. Bold entries are significant at the 95% level.

	MSSMDR	MSSMR	MSS	MS	SSS
SS	**0.000**	**0.002**	**0.034**	**0.024**	0.364
SSS	**0.000**	**0.000**	**0.002**	**0.010**	
MS	**0.000**	**0.041**	0.358		
MSS	**0.000**	**0.039**			
MSSMR	**0.000**				

Table 19. p-values from T-tests of ranking scores for male participants. Bold entries are significant at the 95% level.

	MSSMDR	MSSMR	MSS	MS	SSS
SS	**0.000**	**0.001**	**0.003**	**0.001**	0.339
SSS	**0.000**	**0.001**	**0.001**	**0.005**	
MS	**0.000**	0.153	0.313		
MSS	**0.000**	0.270			
MSSMR	**0.000**				

Table 20. p-values from T-tests of rating scores for male participants. Bold entries are significant at the 95% level.

Appendix B

This is the questionnaire used to measure quality of behaviours in Chapter 3.

Questionnaire

You will watch six (6) different videos from a story-based computer game in a medieval fantasy setting. Please compare all six scenes according to the following criteria.

Comparing All Six Videos

Overall most natural/believable character:
Rank the six videos according to the overall naturalness and believability of the behaviours of the virtual character in a fantasy world. Rank 6 means most natural-looking/believable, Rank 1 means least natural-looking/believable. Each number can be used **EXACTLY** once.

Overall most natural/believable behaviours	**Rank 1**	**Rank 2**	**Rank 3**	**Rank 4**	**Rank 5**	**Rank 6**
Scene						

Rate the six videos according to the overall naturalness and believability of the behaviours of the virtual character in a fantasy world.

Please circle one (1) number for each row below. You may circle the same number for multiple rows.

	1: very unbelievable / unnatural	**2: unbelievable / unnatural**	**3: natural/believable**	**4: very natural/believable**
Scene 1	1	2	3	4
Scene 2	1	2	3	4
Scene 3	1	2	3	4
Scene 4	1	2	3	4
Scene 5	1	2	3	4
Scene 6	1	2	3	4

Why did you rank or rate the videos the way that you did? What criteria (if any) did you use in comparing the behaviours for naturalness and believability?

Appendix C

Instructions for the Tool Group

Your participation involves creating behaviours for four virtual characters in the Neverwinter Nights game. You will be using the Behaviour Architecture tool.

A pre-made city is provided to you. The city is populated by four characters, Adam, Bob, Cathy and Donna. The map of the city is as follows. Names of places or Names of house owners are annotated. The entrance to each place will be used to represent each location (since the entrances are not actual area transitions and there are no indoor areas).

You will start by observing the sample behaviours of Adam.

Adam sleeps at home from midnight until hour 6. He starts to work at the Market at 7, for 10 hours, then eats at Tavern A at hour 17 for 2 hours, and then sleeps at home from hour 19 until the next day. He does the same thing for three days.

Double-click the icon on your desktop to start up the game.

Start the game and observe Adam. Adam will appear as a ghostly image. This signifies that he is inside his house. The ghostly image outside the door tells you which building he is currently in. This is for your convenience.

In a real game, of course, you wouldn't see Adam once he is inside a house.

1 hour of game time is 2 minutes in real life. You can press the F1 key to skip time ahead by 1 hour, each time you press it. Skip to hour 7, and Adam should start to move. Once Adam starts to move to his next location, he will no longer appear ghostly. This signifies that he is actually walking outside of the building. Continue

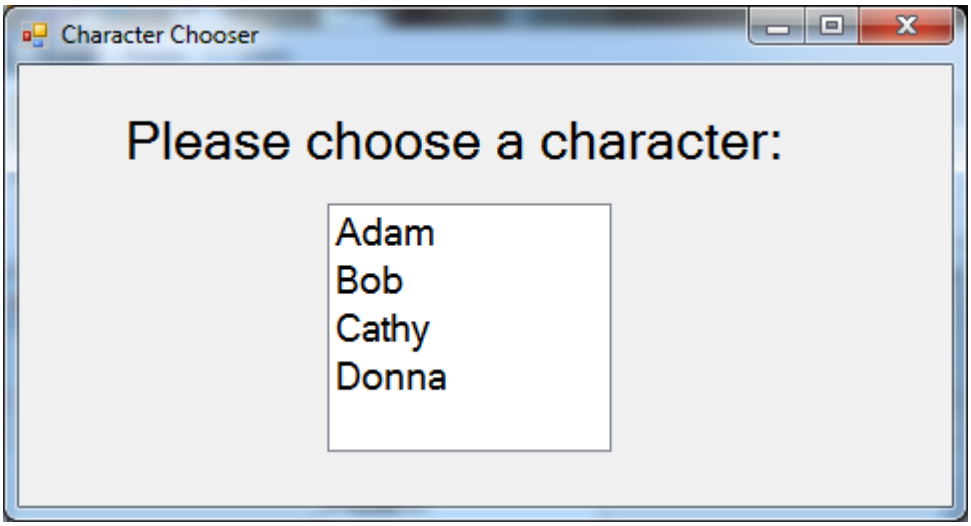
to press F1 and see his entire day. Use your mouse and the arrow keys on the keyboard to move the camera to follow Adam.

Close the game by clicking the x in the upper-right corner.

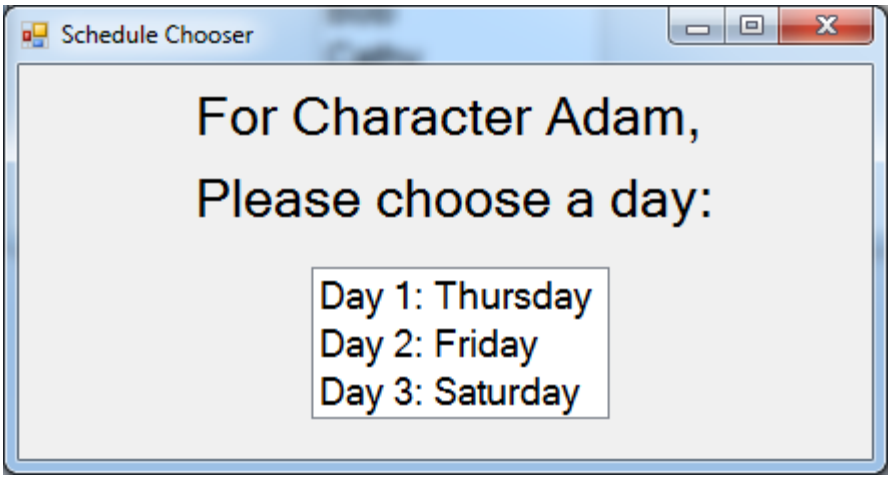
Now, please read the manual to find out how to use the behaviour tool, **but don't start with the tool yet**.

Behaviour Tool Manual

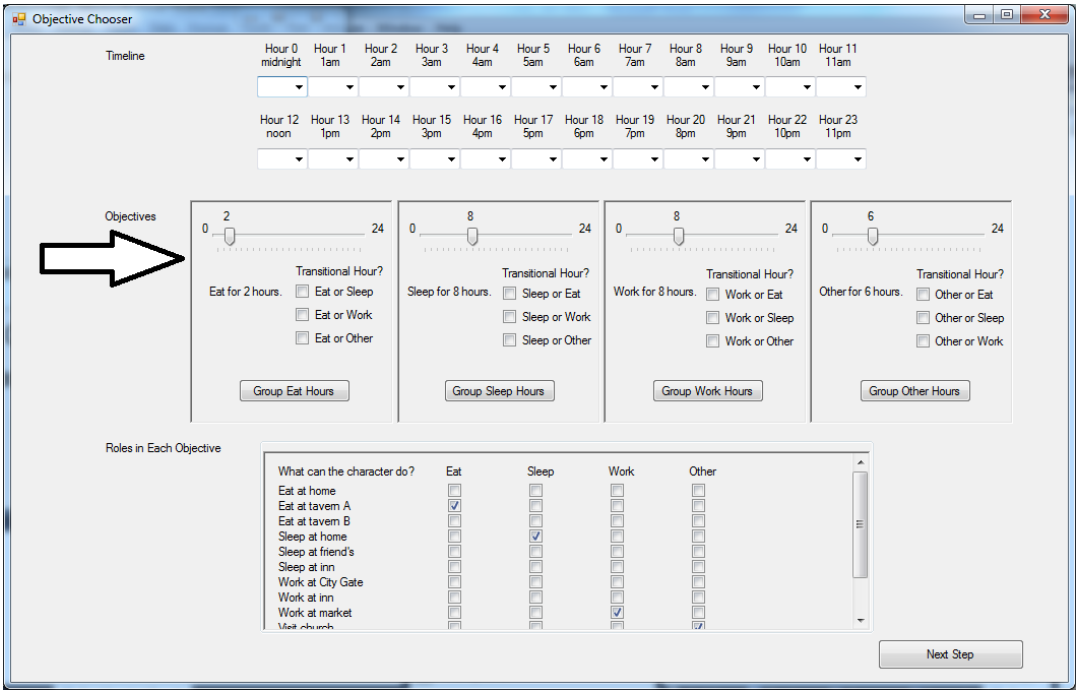
When you start up the Behaviour Tool, the first window is the Character Chooser.



Choose a character, for example, Adam. You will need to finish creating one character before moving on to the next one.



The next window is the Schedule Chooser. You need to choose what day you want to create a schedule. Since this is a prototype, only three days are available to you. You will usually do so in order, starting with Thursday (Day 1).

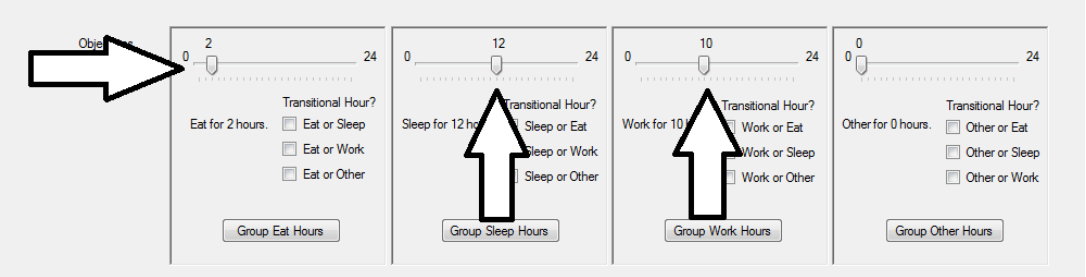


The next window is the Objective Chooser. Most of the work will be done here. We first look at the Objectives in the middle, and determine how many hours we should assign to each Objective in a day. For example, with sample Adam:

> Adam sleeps at home from midnight until 6. He starts to work at the Market at 7, for 10 hours, then eats at Tavern A at hour 17 for 2 hours, and then

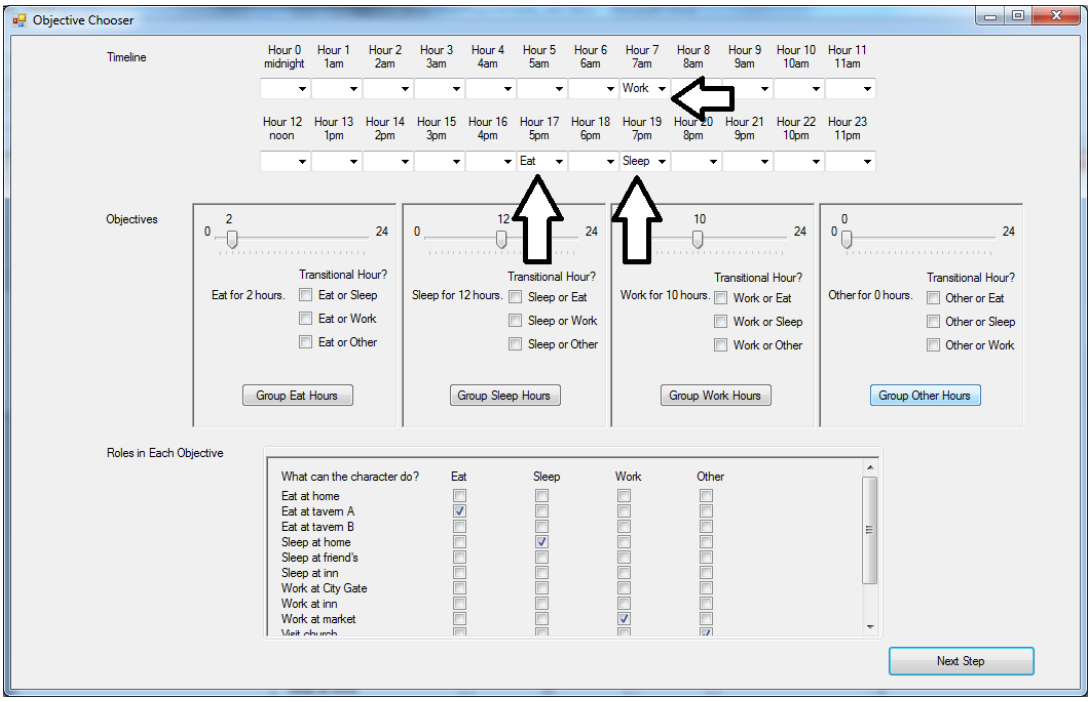
sleeps at home from hour 19 until the next day. He does the same thing for three days.

We see that Adam eats for a total of 2 hours in a day, works for a total of 10 hours, and spend the rest (12 hours) sleeping. We can assign the hours accordingly. You will notice that the hours must add up to 24.

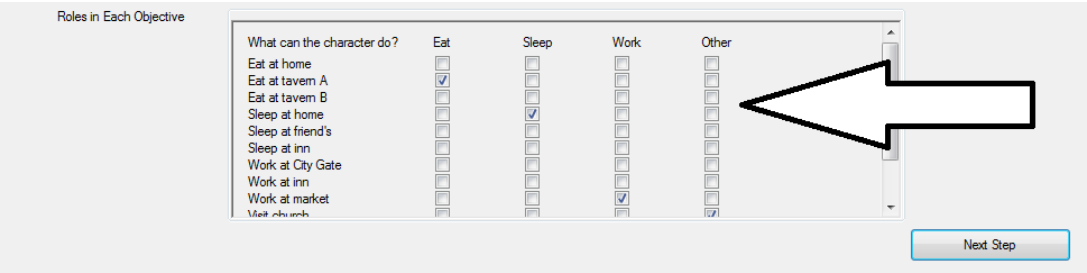


The next step is to find out what needs to be filled on the timeline. The timeline will be automatically filled according to the hours you just assigned, if you do not specify anything. However, we don't want Adam to sleep for any random 12 hours. In fact, we know that Adam has to sleep at hour 19, eat at hour 17, and work at hour 7, so we can fill in these.

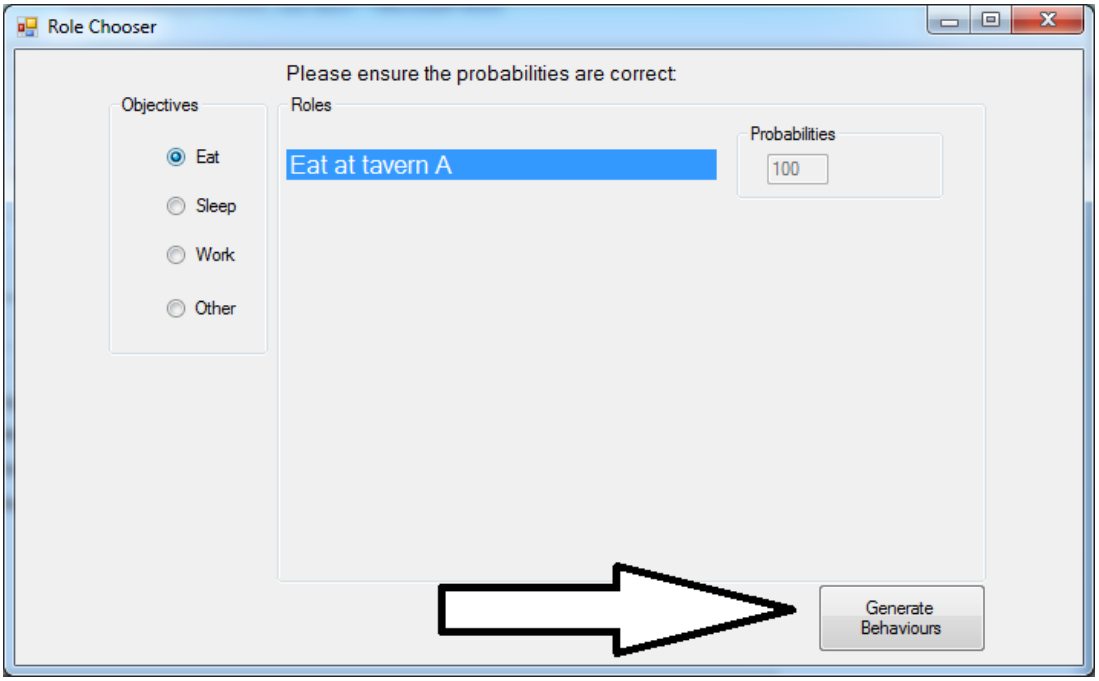
- Usually you can leave the rest blank. However, if the computer does not produce the schedule you want, you can fill them it on the timeline to force the computer to schedule it the way you want.



The next thing to do is to make sure the roles are set correctly. What can the character do to eat? To sleep? To work? We check “Eat at tavern A” for Eat, etc. according to the sample Adam requirements.



When you believe you have your schedule specified, click Next Step. **Everything you specified is automatically saved by the computer, and will automatically show up the next time you open this schedule.**

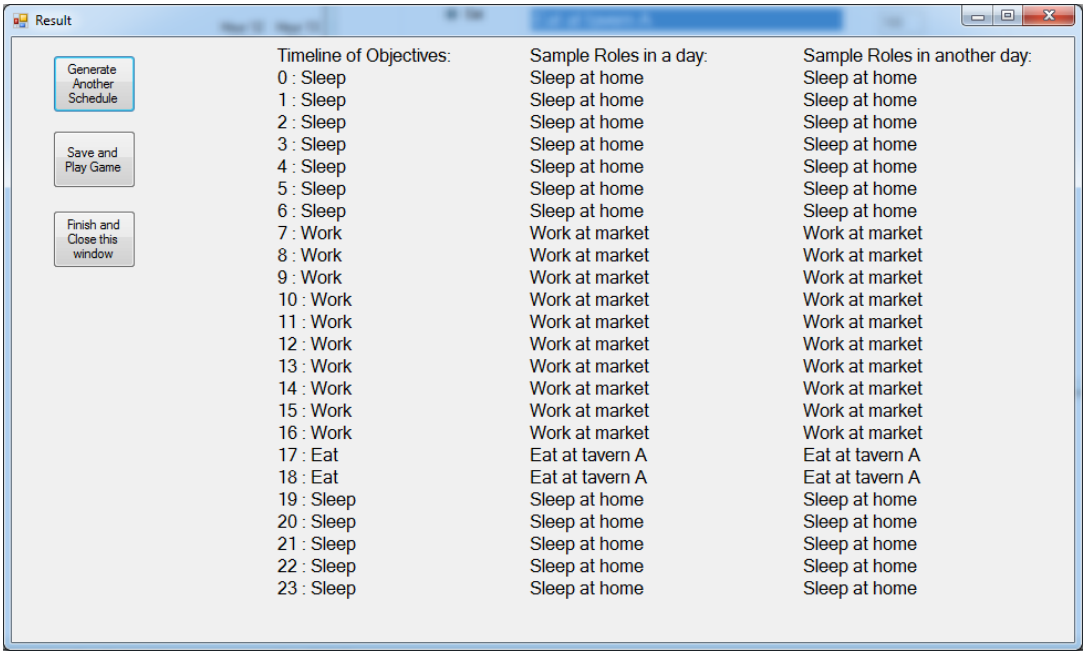


The next window allows us to select the probabilities of each role in each objective. For example, if you can eat at Tavern A and Tavern B, then you can specify that with 40% chance you'll go to Tavern A, and 60% chance you'll go to Tavern B. If you have only chosen one role per objective, then it has 100% probability and you cannot change it.

If you specify that one option has probability 0, while another option has probability 100, then the option with probability 0 will only be chosen if the other option is not possible to do.

(If you want to go back, just close this window.)

Click Generate Behaviours.



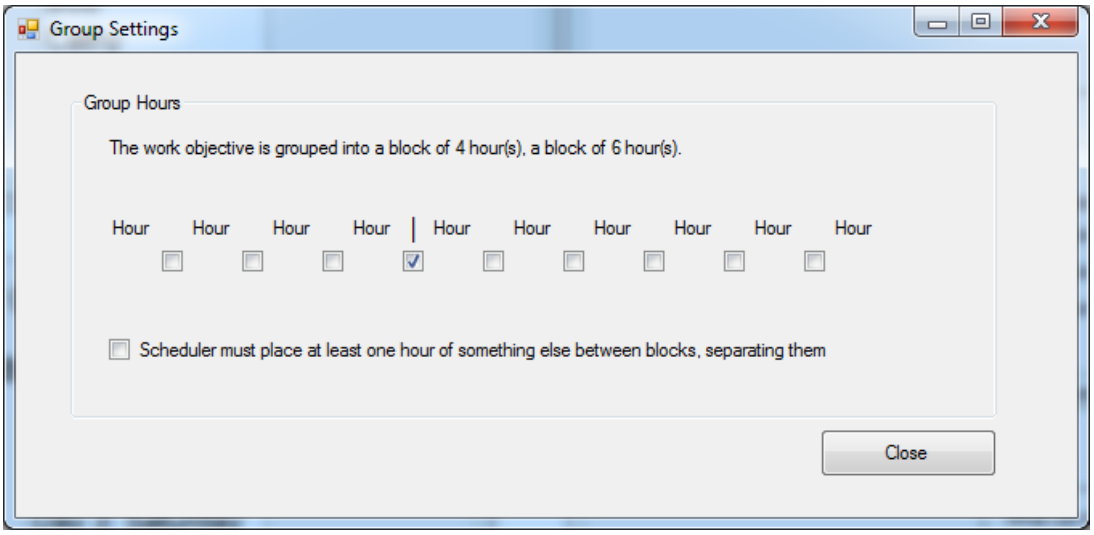
Now we have the results! The Timeline is automatically filled with what you wanted.

If this result is not what you want, you can click “Finish and Close this window” to go back to the timeline and change it.

If you get an error saying that no results are possible, please go back to check your timeline and your objectives to see if you asked for something impossible, for example, eat for a total of 2 hours, but on the timeline you want to eat at hours 5, 6, and 7.

If you are happy with it, you are done. If you want, you can click “Save and Play Game” to watch the character in action!

Click “Finish and Close this window” to go back and select the next character or the next day.

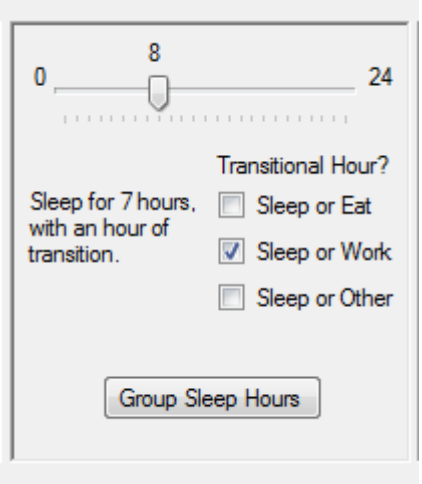


The other functions that you should be aware of include the Group Hours, which allows you to specify groups of hours into blocks. This is very useful when, for example, you need have lunch between working in the morning and working in the afternoon. You can separate the work hours into two blocks by putting a separator checkbox between the hours.

The checkmark *"Scheduler must place at least one hour…"* forces the blocks to be non-adjacent to each other.

This window can be brought up by clicking the button "Group Work Hours" from the Objective Chooser.

The computer will by default group all hours together into one block, unless you tell it otherwise.



The last function is the Transitional Hour, which allows you to specify an hour of transition. In the transition hour, the character will randomly choose one of the specified objectives. In the screenshot above, the character will sleep for 7 hours, and for the 8th hour, randomly choose to sleep or work.

Now is the time to start trying out the tool for yourself! You are asked to first create a different behaviour for Adam. Start up the Behaviour Tool by double-clicking the "Start Study" icon.

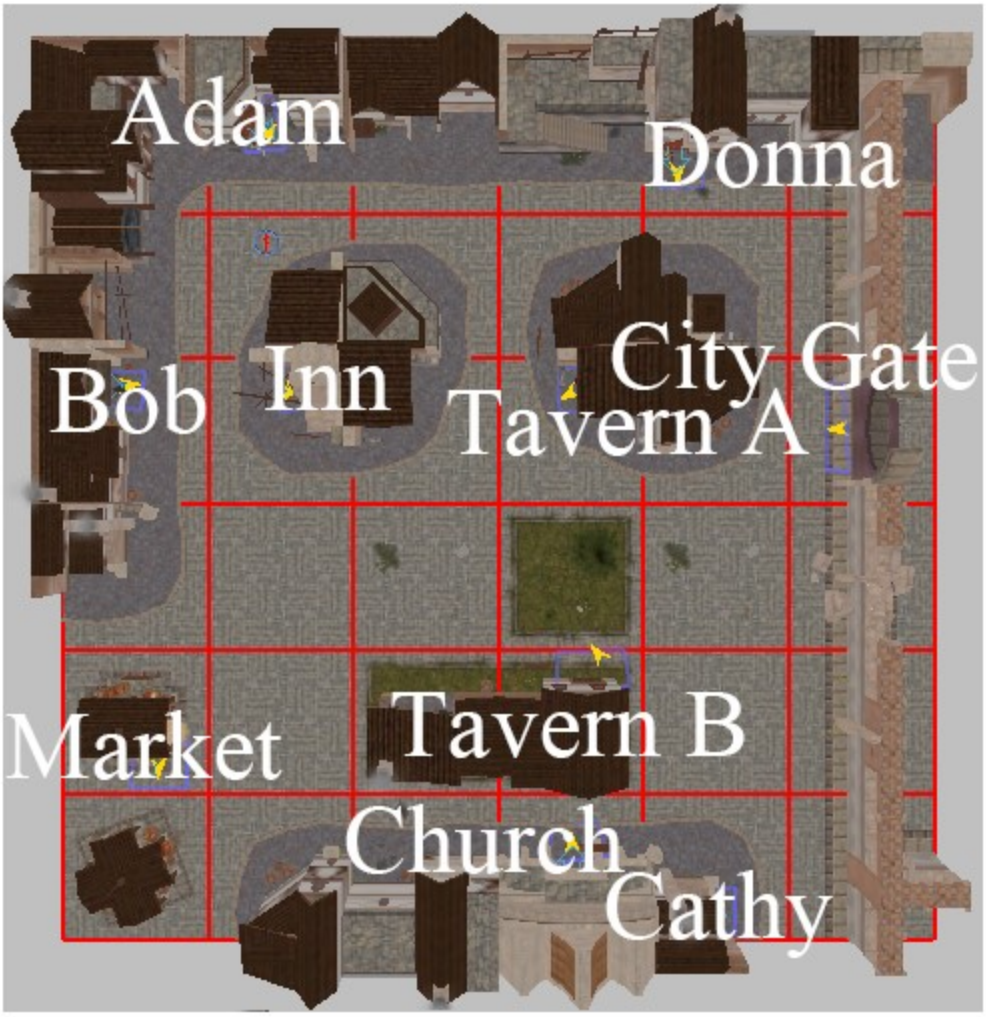
Instructions for the Scripting Group

Your participation involves creating behaviours for four virtual characters in the Neverwinter Nights Aurora Engine. You will be using the Aurora Toolset and the NWScript language. *You may use any references available with you or found on the web, including NWN Lexicon: www.nwnlexicon.com*

The module provided to you has a pre-made city. The city is populated by four characters, Adam, Bob, Cathy and Donna.

When you open the city module, only Adam will have some behaviours. Bob, Cathy and Donna will have no behaviours and they will stand outside their own houses. Adam's behaviours are provided for you as an example and a starting point.

The map of the city is as follows. Names of places or Names of house owners are annotated. The waypoint marks the entrance to each place, and will be used to represent each location (since the entrances are not actual area transitions and there are no indoor areas).



You will start by observing the sample behaviours of Adam.

Adam sleeps at home from midnight until hour 6. He starts to work at the Market at 7, for 10 hours, then eats at Tavern A at hour 17 for 2 hours, and then sleeps at home from hour 19 until the next day. He does the same thing for three days.

Double-click the icon on your desktop to start up the game.

Start the game and observe Adam. Adam will appear as a ghostly image. This signifies that he is inside his house. The ghostly image outside the door tells you which building he is currently in. This is for your convenience.

In a real game, of course, you wouldn't see Adam once he is inside a house.

1 hour of game time is 2 minutes in real life. You can press the F1 key to skip time ahead by 1 hour, each time you press it. Skip to hour 7, and Adam should start to move. Once Adam starts to move to his next location, he will no longer appear ghostly. This signifies that he is actually walking outside of the building. Continue to press F1 and see his entire day.

Close the game by clicking the x in the upper-right corner.

Open the module **City_Script**.mod with the Aurora Toolset. The sample Adam behaviour is scripted in the script file schedule_adam , which is on Adam's heart-beat. This is a starting point to look at. Similarly, schedule_bob, schedule_cathy, and schedule_donna will be the scripts for you to fill.

Appendix D

Additional data on the quality study of the Behaviour Capture technique are shown below. The tables below show the p-values of T-tests, preformed pair-wise between two scene variations.

Criteria	T1 vs. T6	T2 vs. T6	T3 vs. T6	T4 vs. T6	T5 vs. T6
Active	**0.000**	0.049	0.216	0.500	0.195
Unpredictable	0.443	0.370	**0.005**	0.093	0.140
Plausible	**0.000**	**0.000**	**0.044**	0.500	0.409
Diverse	**0.000**	**0.004**	0.135	**0.003**	**0.023**

Table 21. p-values from T-tests on Technique T6 versus each other technique for each criterion. Bold entries are significant at the 95% level.

Overall Believability	T1 vs. T6	T2 vs. T6	T3 vs. T6	T4 vs. T6	T5 vs. T6
Average Ranking Score	**0.000**	**0.001**	**0.004**	**0.029**	**0.027**
Average Rating Score	**0.000**	**0.005**	**0.005**	0.340	0.065

Table 22. p-values from T-tests on Technique T6 versus each other technique for overall believability. Bold entries are significant at the 95% level.

Appendix E

This is the questionnaire used to measure quality of behaviours in Chapter 5.

Questionnaire

You will watch six (6) different tavern scenes from a story-based computer game in a medieval fantasy setting. Please rank all six scenes according to the following criteria. For each criterion, each number can be used **EXACTLY** once.
One example is shown below:

EXAMPLE	**Rank 1**	**Rank 2**	**Rank 3**	**Rank 4**	**Rank 5**	**Rank 6**
Scene	*Scene 3*	*2*	*4*	*1*	*6*	*5*

LEAST **MOST**

This is an example only.

Criterion 1

Most active characters:

Rank the six scenes according to the amount of activity exhibited by the characters in the scene. Rank 6 means most active, Rank 1 means least active. Each number can be used EXACTLY once.

Most active characters	**Rank 1**	**Rank 2**	**Rank 3**	**Rank 4**	**Rank 5**	**Rank 6**
Scene						

LEAST ACTIVE **MOST ACTIVE**

Criterion 2

Most unpredictable characters:

Rank the six scenes according to how hard it was for you to predict the next action that a character would take in the scene. Rank 6 means most unpredictable, Rank 1 means least unpredictable. Each number can be used EXACTLY once.

Most unpredictable characters	**Rank 1**	**Rank 2**	**Rank 3**	**Rank 4**	**Rank 5**	**Rank 6**
Scene						

LEAST UNPREDICTABLE **MOST UNPREDICTABLE**

Criterion 3

Most plausible (reasonable) action sequences:

Rank the six scenes according to the overall plausibility of the sequence of actions for individual characters. Rank 6 means most plausible, Rank 1 means least plausible. Each number can be used EXACTLY once.

Most plausible action sequences	**Rank 1**	**Rank 2**	**Rank 3**	**Rank 4**	**Rank 5**	**Rank 6**
Scene						

LEAST PLAUSIBLE **MOST PLAUSIBLE**

Criterion 4

Most diverse selection of character actions:

Rank the six scenes according to how diverse the actions were for each character. Rank 6 means the highest variety of actions, Rank 1 means the least variety of actions. Each number can be used EXACTLY once.

Most diverse selection of character actions	**Rank 1**	**Rank 2**	**Rank 3**	**Rank 4**	**Rank 5**	**Rank 6**
Scene						

LEAST DIVERSE **MOST DIVERSE**

Criterion 5

Overall most natural /believable scene:

Rank the six scenes according to the overall naturalness and believability of the character behaviours in a tavern in a fantasy world. Rank 6 means most natural-looking/believable, Rank 1 means least natural-looking/believable. Each number can be used **EXACTLY** once.

Overall most natural/believable behaviours	**Rank 1**	**Rank 2**	**Rank 3**	**Rank 4**	**Rank 5**	**Rank 6**
Scene						

WORST **BEST**

Why did you pick the Rank 6 scene as overall most natural/believable?

Why did you pick the Rank 5 scene as next most natural/believable?

Why did you pick the Rank 4 scene as next most natural/believable?

Why did you pick the Rank 3 scene as next most natural/believable?

Why did you pick the Rank 2 scene as next most natural/believable?

What criteria (if any) did you use, besides active characters, unpredictable characters, diverse characters, and plausible action sequences in ranking the overall most natural/believable scenes?

Rate the six scenes according to the overall naturalness and believability of the character behaviours in a tavern in a fantasy world.

NOTE: Please ignore the quality of the graphics and animation in each scene, and rate on character behaviours only.

Please circle one (1) number for each row below. You may circle the same number for multiple rows.

	1: very unbelievable / unnatural	**2: unbelievable / unnatural**	**3: natural/ believable**	**4: very natural/ believable**
Scene 1	1	2	3	4
Scene 2	1	2	3	4
Scene 3	1	2	3	4
Scene 4	1	2	3	4
Scene 5	1	2	3	4
Scene 6	1	2	3	4

Given the following scene characteristics, evaluate how you think each of them contributes positively or negatively to a natural-looking scene in a game environment and the relative importance of each.

Please circle one (1) number for each row below. You may circle the same number for multiple rows.

	-3: very negatively	-2: negatively	-1: mildly negatively	0: neutral (not important)	1: mildly positively	2: positively	3: very positively
active characters	-3	-2	-1	0	1	2	3
unpredictable characters	-3	-2	-1	0	1	2	3
plausible action sequences	-3	-2	-1	0	1	2	3
diverse character actions	-3	-2	-1	0	1	2	3