2011

Effective Team Strategies using Dynamic Scripting

Robert Price

University of Windsor

Effective Team Strategies using Dynamic Scripting

by
Robert G. Price

A Dissertation
Submitted to the Faculty of Graduate Studies
through Computer Science
in Partial Fulfillment to the Requirements for
the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada
2011

Effective Team Strategies using Dynamic Scripting

by

Robert G. Price

APPROVED BY:

____
Dr. M. Katchabaw, External Examiner
The University of Western Ontario

____
Dr. M. Hlynka
Department of Mathematics and Statistics

____
Dr. D. Wu
School of Computer Science

____
Dr. Z. Kobti
School of Computer Science

____
Dr. S. Goodwin, Advisor
School of Computer Science

____
Dr. H. Maoh, Chair of Defense
Department of Civil & Environmental Engineering

29 September, 2011

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Forming effective team strategies using heterogeneous agents to accomplish a task can be a challenging problem. The number of combinations of actions to look through can be enormous, and having an agent that is really good at a particular sub-task is no guarantee that agent will perform well on a team with members with different abilities. Dynamic Scripting has been shown to be an effective way of improving behaviours with adaptive game AI. We present an approach that modifies the scripting process to account for the other agents in a game. By analyzing an agent’s allies and opponents we can create better starting scripts for the agents to use. Creating better starting points for the Dynamic Scripting process and will minimize the number of iterations needed to learn effective strategies, creating a better overall gaming experience.

Dedication

This work would not have been possible without my wife, Divina. You helped me get to the starting line, you were with me when I crossed the finish line, and you were with me at all the important points in between.

Acknowledgements

Thanks to Dr. Goodwin for all the help you have provided over the years. Your topics made me want to learn, and your sense of humour helped make learning enjoyable. Your patience and guidance through earlier drafts of this work helped make this better.

Thanks to the rest of my committee: Dr. Hlynka, Dr. Kobti, Dr. Wu. Your suggestions from when we met earlier provided ideas which helped me improve this work.

Thank you also to Dr. Katchabaw, for joining my committee.

Very special thanks go to my friends. I would not have been able to complete this journey without your support and kind words. Thanks to the people who inspired me by showing that some teams just performed better than others, even though the level of talent would have suggested otherwise.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

The importance of games

Games are a good test-bed for artificial intelligence (AI) research, since they can be challenging, but easy to formalize. This makes it possible to measure how well new AI methods are working, and to demonstrate that behaviour that is generally thought to require intelligence is possible without putting human lives or property at risk. Recently, inexpensive yet powerful computer hardware has made it possible to simulate complex physical environments, resulting in an explosion of the video game industry [Mii06]. Computer games provide an environment for continual, steady advancement and a series of increasingly difficult challenges [Lai00]. Not only are video games an important test-bed for AI, but advances in video games help in other areas also. "Video games are particularly important in this regard, because in addition to their very realistic visual images and great sound, they are also highly interactive and increasingly collaborative, and thus a good launch pad for thinking about how people should best interact with all kinds of computer applications as well as with each other in the future" [Fri07]. Although this research can directly benefit the AI in video games, advances in machine learning and other AI techniques can have benefits in other areas as well.

Learning in games

"One of the main challenges for AI is to create intelligent agents that adapt, i.e. change their behaviour based on interactions with the environment, becoming more proficient in their tasks over time, and adapting to new situations as they occur." [Sta06]
The following from [Sta05] states some of the properties video games have that challenge traditional reinforcement learning (RL) techniques.

> 1. Large state/action space. Since games usually have several different types of objects and characters, and many different possible actions, the state/action space that RL must explore is high-dimensional. Not only does this pose the usual problem of encoding a high dimensional space (Sutton and Barto 1998), but in a real-time game there is the additional challenge of checking the value of every possible action on every game tick for every agent in the game.
> 2. Diverse behaviors. Agents learning simultaneously should not all converge to the same behavior because a homogeneous population would make the game boring. Yet since RL techniques are based on convergence guarantees and do not explicitly maintain diversity, such an outcome is likely.
> 3. Consistent individual behaviors. RL depends on occasionally taking a random action in order to explore new behaviors. While this strategy works well in offline learning, players do not want to see an individual agent periodically making inexplicable and idiosyncratic moves relative to its usual behavior.
> 4. Fast adaptation. Players do not want to wait hours for agents to adapt. Yet a complex state/action representation can take a long time to learn. On the other hand, a simple representation would limit the ability to learn sophisticated behaviors. Thus, choosing the right representation is difficult.
> 5. Memory of past states. If agents remember past events, they can react more convincingly to the present situation. However, such memory requires keeping track of more than the current state, ruling out traditional Markovian methods.

Machine learning is important in games, since it can allow them to be more interesting and realistic [Sta05]. Many games would benefit from having characters that adapt to new situations, as it would likely give the illusion that the characters understand what they are doing. Lately many games have improved visually, creating very realistic environments, but the AI controlling the characters have not improved as much. “A game’s top-notch graphics and sound manage to keep up a suspension of disbelief quite well. However, the behaviours of a character in a game are usually of an inferior quality. It is all too clear that the characters are lifeless, mindless drones controlled by a computer with little knowledge.” [Spr05]

At one point, all computer controlled agents were hard-coded, which meant that the game AI was implemented in the source code of the game itself. Therefore, a change regarding the game AI required a re-compilation of the game [Lad08]. Since then the dominant approach to programming game AI has been scripting, since they are easy to implement, interpret, and modify. Scripts are basically text files encoding the behaviour of game agents with if-then rules. Some modern games encourage the players to change and add content through the use of script editors and other tools. [Szi09] lists some disadvantages of scripts: They are labour-intensive, since they have to be reasonably complex to be used in a complex game environment, which causes them to be time-consuming to implement by hand, as they must be tested in many different situations. Because of their complexity, they are likely to contain undetected weaknesses that can be exploited. They are predictable, and human players quickly recognize patterns in behaviours generated by scripts. Finally, since they are static, they are unable to adapt to a human player’s style.

Spronk et al. [Spr06] investigated a novel game AI learning technique called ‘dynamic scripting’ (explained below). They also performed a literature survey and communication with game developers to come up with computational and functional requirements for adaptive game AI to be applicable in practice: Speed, Effectiveness, Robustness, Efficiency, Clarity, Variety, Consistency, and Scalability. Dynamic scripting is a reinforcement learning technique that learns effective game AI scripts. For dynamic scripting to work, the game AI needs to be available in the form of scripts. This is the de-facto standard in commercial computer games [Lad08]. [Lad08] also states dynamic scripting “can only build scripts as good as the underlying rules. A bunch of bad rules won’t magically combine into a good tactic just because they are put together by a learning procedure.”

Dynamic scripting

[Spr06] uses the Neverwinter Nights (NWN, discussed more in Chapter 3) environment to combine scripting with a reinforcement learning process. This dynamic scripting maintains several rule databases, one for each agent type. A new script to control the behaviour of an agent is created from these rule databases whenever an agent is generated. This script has a number of rules from the database, each one having a weight. There is a linear relationship between the probability a rule is selected and its weight. After each fight the weights in the database are updated, with rules that lead to success getting increased. Over time, behaviours emerge from the characters that are more effective.

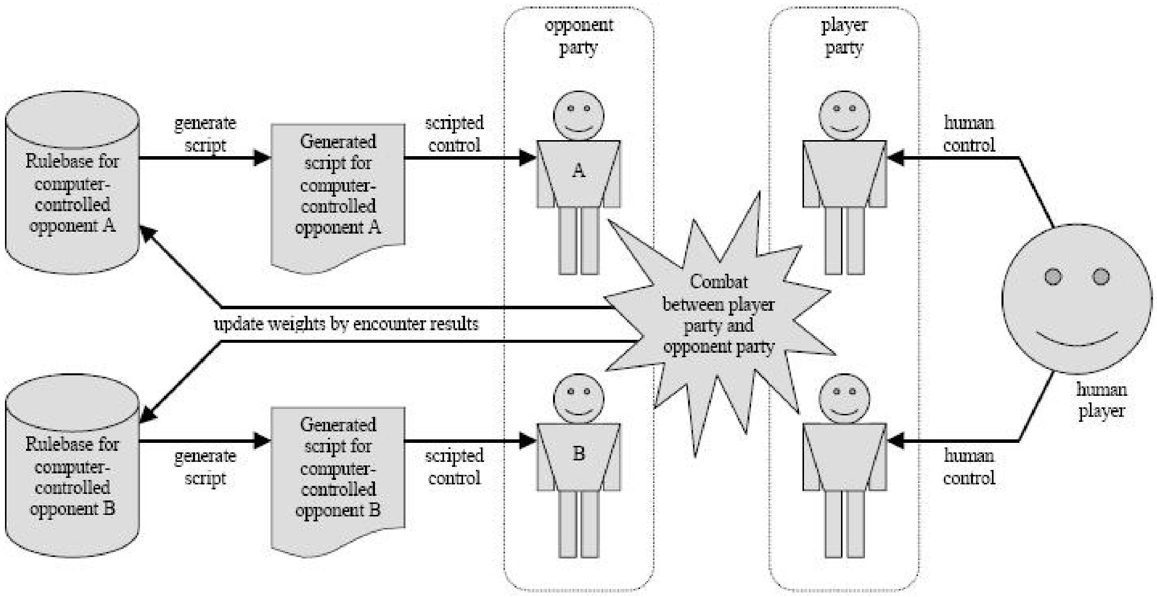

opponent party
player party
Rulebase for computer-controlled opponent A
generate script
Generated script for computer-controlled opponent A
scripted control
A
human control
Combat between player party and opponent party
update weights by encounter results
human player
Rulebase for computer-controlled opponent B
generate script
Generated script for computer-controlled opponent B
scripted control
B
human control

Figure 1: Outline of updating script weights from [Spr03]

[Spr03] states "because of their complexity, AI scripts are likely to contain weaknesses, which can be exploited by human players to easily defeat supposedly tough opponents", and "because they are static, scripts cannot deal with unforeseen tactics employed by the human player and cannot scale the difficulty level exhibited by the game AI to cater to both novice and experienced human players." The updating of weights and having successful characters more likely to be included in future rounds is similar to the updating of weights in neural networks and successful offspring being chosen more often.

One anecdote praising the effectiveness of this type of learning comes from [Spr09]:

> "BioWare has changed the AI in NeverWinter Nights significantly between versions 1.29 and 1.31… What is kind of funny is that the strategy now employed by the opponents is very similar to the strategy that dynamic scripting often learned when pitted against the earlier AI. I find it striking that dynamic scripting managed to discover in half an hour something that BioWare only added to the game after it had been out for more than a year and had been patched over a dozen times."

In Chapter 2, more information is provided about how dynamic scripting works for an individual agent.

Problem topic

Although the characters learning dynamic scripting can learn effective rule orderings to defeat a certain opponent, the strategies the characters learn are less effective against different opponents. [Lad08] states "a script that worked well in one case may perform very poor in another one". Dynamic scripting has been shown to perform well for individual agents against specific opponents, but it does not accommodate changes in opponents or teammates into account. While performing research for [Pri10], it was observed that the rules did not seem to take the opponent into consideration when deciding what action to take. For example, one effective rule is to cast a defensive spell like “Globe of Invulnerability” at the beginning of combat. The “Globe of Invulnerability” spell makes the spellcaster immune to certain spells cast at them by other spellcasters. However, it was also noticed that when put against a team that contained *no* spellcasters, the defensive spell was cast anyways. In this instance, casting this spell was a complete waste since it could offer no benefit and had an opportunity cost as it prevented the spellcaster from casting an offensive spell that might have helped their team in combat. This observation leads to the following question:

Can dynamic scripting use information about teammates and opposing agents to create a better starting point for the script weights which in turn can be used to learn an effective rule ordering more quickly?

Dynamic scripting certainly provides an effective way to learn strategies, but the strategies learned become less effective when the opponents change. Spronck’s work with dynamic scripting was effective, but it does not take the other agents participating in the fight into account when determining the best actions to take. [Spr04] notes “experimental evaluations indicated that, occasionally, the time needed for dynamic scripting to generate effective opponents becomes unacceptably long.”

Creating better starting points should shorten the number of iterations dynamic scripting needs to go through to create an effective strategy. [Pon04] uses different rulebases for different stages in a real-time strategy (RTS) game, and shows that taking context into account can lead to more-effective results. [Lad08] states that different agent classes have their own rulebases since they can have different actions available to them, showing that the effective rule orderings for one class do not imply similar rule orderings for other classes. In this research we will explore having an agent use a different rulebase depending on the other agents in the area, specifically the opponents it will face and the teammates it has. This research supports to the following thesis:

Taking other agents into consideration will allow an individual agent using dynamic scripting to perform better than current scripting implementations.

The optimal behaviour of an agent in a combat situation will differ depending on the teammates and opponents that are participating in that combat. The following chapters in this dissertation will demonstrate how the performance of dynamic scripting can be improved by taking the other agents into account.

Thesis plan

The rest of this thesis will cover the following outline. In Chapter 2 we provide more background and review dynamic scripting and game balance in detail. In Chapter 3 we discuss the experimental outline after giving an explanation of the environment. In Chapter 4 we provide the results of the experiments which indicate that agents perform better with the new starting points, along with some discussion. Finally in Chapter 5 we conclude that the performance of dynamic scripting is enhanced by this process, and discuss how this research can be used in practice, along with future work. By the end of this dissertation we will have demonstrated that using knowledge from teammates and other opponents will significantly improve the performance of dynamic scripting, that using roles to classify the agents can effectively reduce otherwise impractically large search spaces into manageable sizes, and the process of determining weights is robust, and is not as sensitive to errors as hand-picking constants.

Chapter 2: Background

In this chapter we give an overview of select topics to provide the necessary background required to understand this dissertation. It starts with an overview of AI used in computer games, followed by a description of some learning algorithms. A summary of selected multi-agent-systems research highlights relevant topics that emerge when teams of agents, as opposed to just groups of individuals, work together to accomplish a task. The issue of balance in games is discussed, and then the chapter ends with an explanation of dynamic scripting along with a walkthrough of an example to show how dynamic scripting updates weights in a script and change the actions an agent will take.

AI in Computer Games

Computer games have been around for over 50 years, and the field of game artificial intelligence has existed "since the dawn of video games in the 1970s" [Tou02]
Games can be classified into many categories, common ones described in the table below.

Category	Description	Role of AI
Action	Player uses mostly his or her reflexes to win.	Game AI controls individual agents
Adventure	Player follows a path, solving puzzles or quests to win.	None – characters react in pre-defined ways
Puzzle	Players to solve puzzles.	None
Role-Playing	Player assumes the role of an in-game character, and solves quests.	Game AI controls individual agents
Simulation	Players interact with a simulation.	Game AI controls the simulation
Strategy	Player uses tactical skills to guide agents to victory.	Game AI controls lots of agents

Table 1: Game genres

"Complex game AI is encountered mainly in role-playing games and strategy games" [Spr05]. In the past decade there have been game developers and game researchers tended to live in their own communities. Recently that has been improving, and there have been conferences and tracks devoted to Game AI. Many "mainstream" AI techniques have found their way into games. The following sections outline some of these techniques.

Evolutionary Algorithms

Evolutionary algorithms are population-based optimization algorithms that mimic the process of natural evolution. Genetic algorithms or genetic programming are the most popular forms of evolutionary algorithms. Neural networks or artificial neural networks (ANNs) were partially inspired by biological neural networks. These ANNs were first discussed in [McC43]. Although they are modelled after biological neural networks, ANNs do not model all of the complexities of biological neural systems. For example, in an ANN, the individual output of a neuron is a single constant value, whereas in a biological system, the output of a neuron is a complex series of spikes over time.

Evolutionary algorithms can be used to update the weights and structure of neural networks, through the use of a fitness function. Evolutionary algorithms use parents to generate successors through crossovers and mutations. In a crossover, two parents are used and partial information is taken from each parent, and combined to form a new ANN. In a mutation, one or more nodes are changed in a single network to form a new ANN.

Individual agents learning strategies have been the topic in [Sta02], [Sta06], where neural networks were used in a reinforcement learning game. The machine learning game discussed in these papers requires the player to set up scenarios where agents can learn the desired concepts in real time. The NeuroEvolution of Augmenting Topologies (NEAT) is an algorithm for learning that adjusts both the weights and the structure of ANNs, and was introduced in [Sta02]. NERO (Neuro-Evolving Robotic Operatives) is a game that uses an evolutionary algorithm using a fitness function based on the player's input to train agents to perform tasks. The time it takes the agents to learn how to perform the tasks depends on how the player sets out the learning tasks. An artificial neural network trained on a single, isolated very difficult task is unlikely to learn it well [Car97]. The default agents that come with NERO have sensors for where opponents are, but those sensors do not distinguish between different opponent types. This would prevent it from learning to use different tactics based on the opponents it is facing. This problem could be remedied by adding additional sensors, but that would increase the time it takes for the neural network to learn each task.

Bayesian Networks

A Bayesian Network is a data structure that can be used to represent dependencies among variables and to give a concise specification of a joint probability distribution. A Bayesian Network is a directed acyclic graph with the variables represented as nodes in the graph, with an edge between a parent node and a child node if the parent node directly influences the child node. It is usually easy for a domain expert to decide what influences exist in the domain. A Bayesian Network representation can take much less space to specify than a joint probability distribution, and can also be easier to represent. A good introduction to Bayesian networks can be found in [Cha91]. A Dynamic Bayesian Network is a Bayesian Network that represents a temporal probability model. One example of a how a Bayesian Network can be used in a game is explained in Appendix G.

Multi-agent systems

Multi-agent systems focus on how intelligent behaviour can naturally arise from the interaction between multiple agents that may cooperate or compete. The experiments run in [Tim07] focus mainly on the interactions between agents that compete. The experiments described in this paper deal with both types of interactions-agents cooperating with their teammates and competing against the opposing team. Flocking algorithms are an artificial life subcategory where lifelike flocks result from coordinated movements of multiple cooperating AI agents.

[Daw02] states "anytime a group is moving or working together it is expected to do so in an orderly, intelligent fashion". [Hei08] discusses how static formations of agents are not capable of adapting effectively to opponent tactics, and demonstrate how formations of agents can be formed dynamically to deal with changing circumstances. [Fal03] also deals with changing formations in games, and states that systems that dynamically change their strategy are sometimes criticized for being "slow to learn".

There has also been research done in agents forming teams amongst themselves, either by trust and reputation [Huy06], [Bar07], [Smi09], cooperating to accomplish tasks that the agents cannot perform themselves[Kra03], and teams that work together to accomplish goals based on joint intentions [Jen95], [Tam97].

[Jen95] distinguishes between a group of agents which independently have a goal which just happens to be the same and a group of agents which truly share a common goal. This distinction is important because the two relationships imply different consequences with interaction: the former gives rise to competition if resources are scarce, whereas the latter result in cooperation and coordination.

One domain where agents perform together is robot soccer. Bowling et al. [Bow04] noticed that “most robot soccer approaches involve single, static, monolithic team strategies” and “there have been examples of seemingly superior teams being defeated by unexpected opponent play”. Although robot soccer is different than teams fighting in a game, there are many similarities. In both cases there are well-defined goals that need to be achieved in an adversarial setting by teams that have multiple members. The environments are dynamic and unpredictable, and teams that can coordinate plans among several agents and adapt to changes often have an advantage.

Many machine learning techniques involve the use of a fitness function to evaluate performance. A fitness function provides a measure of how well a solution is to achieving its goals. [Lad08] mentions that in many games the fitness functions “comes down to doing as much damage as possible on enemy agents while at the same time trying to avoid as much damage as feasible. Achieving this goal is more difficult than it sounds, but at least makes the design of a fitness functions quite straight forward.”

Balance in Games

The purpose of games is to “provide entertainment” [Ada07] and be enjoyed. “Game AI is all about fun…You have a customer who paid $40 for your game and he or she expects to be entertained.” [Tou02]. Adversarial type games become less enjoyable if one player always wins or loses so for this reason, games try to incorporate balance, so one type of agent does not dominate over all others.

Consider the rock-paper-scissors game. In this game for two players, each player chooses one of three objects, and the result is either a tie (if they both pick the same object) or a win for one of the players otherwise (rock crushes scissors, scissors cut paper, paper covers rock). Similar dynamics exist in many commercial games. In Microsoft’s “Age of Mythology”, infantry do additional damage to cavalry, cavalry do additional damage to archers, and archers do additional damage to infantry.

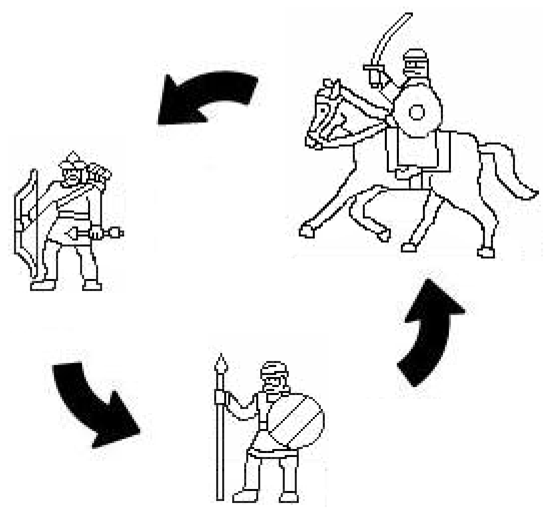


Figure 2: Rock-Paper-Scissors dynamic

If one of the objects (rock, for example) were to win against paper as well as scissors, then a person could guarantee not losing by always picking rock. That would make the game unbalanced, and most people would find such a game to not be fun. It is generally assumed that a balanced game has a higher entertainment value than one that is too easy or too hard [Lan10].

If a game is too easy or too hard people lose interest. Different approaches have been used in the past to allow a player to change the difficulty of a game. Some games have cheat-codes that a player can use to help out if they find the game too difficult, or they have static difficulty levels (i.e.: easy, normal, and hard) that the player can pick from at the beginning of a game. In recent years, research has gone into varying the difficulty level of a game dynamically to keep the game interesting for the player. "Auto-dynamic difficulty refers to the ability of a game to automatically adapt the difficulty level of gameplay to match the skills and tolerances of a player." [Bai05]

The rock-paper-scissors example is an extremely simple one, but serves as a good illustration to explain balance. More discussion about balance and the rock-paper-scissors game can be found in [Ada07], as well as a reason to avoid dominant strategies (a dominant strategy refers to a strategy that reliably produces the best outcome a player may achieve, no matter what the opponent does).

Dynamic scripting in depth

From a theoretical point of view, dynamic scripting belongs to the family of reinforcement learning methods (as it learns from evaluative feedback), but its formalism (assigning weights/credits to individual rules) is also closely connected to learning classifier systems [Szi09]. Dynamic scripting provides computer-controlled agents with the ability to correct mistakes and respond to new situations. It uses rules from knowledge bases to create game scripts, along with a mechanism to order the rules selected for the scripts. The following paragraph from [Tim07] describe how the weights get updated

After an encounter (i.e., a fight) between the human player and an opponent, the opponent's knowledge base adapts by changing the rule-weight values in accordance with the success or failure rate of the rules that were activated during the encounter (i.e., rules that occurred in the opponent's script). The new rule weight value is calculated as $W + \Delta W$, where W is the original rule weight value. The weight adjustment ΔW is expressed by the following formula:

$$\Delta W = \begin{cases} -P_{\max}\dfrac{b-F}{b} \{F < b\} \\ R_{\max}\dfrac{F-b}{1-b}\{F \geq b\} \end{cases}$$

R_{max} and P_{max} are the maximum reward and maximum penalty respectively, $b \in <0, 1>$ is the break-even value, and $F \in [0, 1]$ is the opponent's fitness. The fitness is a relative measure of the opponent's success, which is high for good results (victories) and low for bad results (defeats).

A simple example will help demonstrate how the weights are updated. For this example, two wizards will fight each other, and they will each have 10 spells to pick from. To keep things separate, we will call one White and the other Black. Each wizard will start with 50 hit points, and the fitness will be calculated as a percentage of the remaining hit points. The first step in the example is to pick the spells that will be used.

Algorithm 1: Generating scripts for dynamic scripting

(adapted from [Spr05])

```
rules is an array with the possible actions and associated weights (actions that lead
to successful outcomes have higher weights)
scriptsize is the amount of rules to be placed in the script
maxtries to the maximum amount of tries to add a rule to the script

set sumweights to the sum of the weights of all of the rules
set tries to 0

For every rule that needs to be added to the script
    pick a random number between 0 and sumweights

    go through each rule and add the weights until you get to a rule that
increases the sum to a number greater than or equal to the random one

    if that rule hasn't been added to the script yet then add it

    increase tries by one

    if tries >= maxtries then stop
end for
```

In our example, *scriptsize* is 3, *maxtries* will be 10, and *rules* will be as described in the following table, along with the initial weights.

Spell	Description	Weight	Percentage
Mage Armor	Makes the mage harder to hit physically	10	0-9.99
Magic Missile	Damages opponent 10-25 hp	10	10-19.99
Melf's Acid Arrow	Damages opponent 2-8 hp, plus extra damage subsequent rounds	10	20-29.99
Fireball	Damages opponents 10-60 hp	10	30-39.99
Haste	Doubles amount of physical attacks	10	40-49.99
Stoneskin	Absorbs physical damage	10	50-59.99
Ice Storm	Damages opponents 5-30 hp	10	60-69.99
Chain Lightning	Damages opponent 15-90 hp	10	70-79.99
Spell Mantle	Absorbs damage from enemy spells	10	80-89.99
Horrid Wilting	Damages opponents 15-90 hp	10	90-99.99

Table 2: Example weights for spell picking

Creating White's script for round 1:
Sumweights = 100
Tries = 0
White picks 'random' number 38 for the first spell. Fireball is added to the script.
White picks 'random' number 73 for the second spell. Chain Lightning is added to the script.
White picks 'random' number 30 for the third spell. Fireball is already in the script, so just increment *tries*.
White picks 'random' number 64 for the third spell. Ice Storm is added to the script.
There are 3 spells, so stop.

Creating Black's script for round 1:
Sumweights = 100
Tries = 0
Black picks 'random' number 3 for the first spell. Mage Armor is added to the script.
Black picks 'random' number 41 for the second spell. Haste is added to the script.
Black picks 'random' number 95 for the third spell. Horrid Wilting is added to the script.
There are 3 spells, so stop.

Combat now happens:
In the first round of combat, White casts Fireball. Black is damaged by 13 hit points (37 remain). Black casts Mage Armor, which raises his defense against physical attacks.
In the second round of combat, White casts Chain Lightning. Black is damaged by 42 points (killing him). Black casts Haste, which would have increased the attack speed.

Combat is now over for this trial. The fitness for White's script is 1 (White has all of his initial hit points). The weights for the spells are now adjusted, as described by the algorithm below:

Algorithm 2: Weight Adjustment

(adapted from [Spr05])
adjustment is the amount a spell's weight will be changed based on the last experiment. It can be positive or negative, depending on whether or not the last experiment was a win or a loss. The magnitude of the *adjustment* depends on how decisive the win or loss was. A close win will result in a small *adjustment*, while a resounding win will result in a large *adjustment.*
minweight is the smallest value we want a spell's weight to be set to
maxweight is the largest value we want a spell's weight to be set to

```
set adjustment to an amount based on the fitness results of the last experiment

For every rule in the script
        adjust the weight by the adjustment

        if the weight is below minweight then set it to minweight

        if the weight is above maxweight then set it to maxweight
end for

normalize the weights in the rulebase.
```

The last step keeps the sum of all the weights in a rulebase at 1. If a team won, the spells it used will have their weights increased. After normalization, all the weights would be lowered, resulting in the weights that were increased still having higher weights than before the experiment, and the weights that were not used lowered, so they will have a lower chance of being included in subsequent experiments. The adjustment maximum is 10% for the first spell, which is multiplied by 0.7 for subsequent rounds. It has been found that the earliest actions have a greater effect on whether the team wins or loses. The 0.7 was determined to be effective during initial experiments.

In our example, the maximum fitness was attained, so the maximum adjustments will occur. The weights for White before normalization occur below:

Spell	Description	Weight	Percentage
Mage Armor	Makes the mage harder to hit physically	10	N/A
Magic Missile	Damages opponent 10-25 hp	10	N/A
Melf's Acid Arrow	Damages opponent 2-8 hp, plus extra damage subsequent rounds	10	N/A
Fireball	Damages opponents 10-60 hp	**20**	N/A
Haste	Doubles amount of physical attacks	10	N/A
Stoneskin	Absorbs physical damage	10	N/A
Ice Storm	Damages opponents 5-30 hp	10	N/A
Chain Lightning	Damages opponent 15-90 hp	**17**	N/A
Spell Mantle	Absorbs damage from enemy spells	10	N/A
Horrid Wilting	Damages opponents 15-90 hp	10	N/A

Table 3: White weights before normalization

After normalization, the following weights represent the weights for White's spells.

Spell	Description	Weight	Percentage
Mage Armor	Makes the mage harder to hit physically	8.55	0-8.54
Magic Missile	Damages opponent 10-25 hp	8.55	8.55-17.09
Melf's Acid Arrow	Damages opponent 2-8 hp, plus extra damage subsequent rounds	8.55	17.10-25.64
Fireball	Damages opponents 10-60 hp	17.09	25.65-42.74
Haste	Doubles amount of physical attacks	8.55	42.75-51.28
Stoneskin	Absorbs physical damage	8.55	51.29-59.83
Ice Storm	Damages opponents 5-30 hp	8.55	59.84-68.38
Chain Lightning	Damages opponent 15-90 hp	14.53	68.39-82.91
Spell Mantle	Absorbs damage from enemy spells	8.55	82.92-91.45
Horrid Wilting	Damages opponents 15-90 hp	8.55	91.46-99.99

Table 4: White weights after normalization

The next time spells are picked, the ones that were successful will have a greater chance of being included in future rounds. The original weights for Black were the same for the original ones for White. After the fight, the spells that Black used will lowered, and the pre-normalized weights are shown below (assuming a *minweight* of 3).

Spell	Description	Weight	Percentage
Mage Armor	Makes the mage harder to hit physically	**3**	N/A
Magic Missile	Damages opponent 10-25 hp	10	N/A
Melf's Acid Arrow	Damages opponent 2-8 hp, plus extra damage subsequent rounds	10	N/A
Fireball	Damages opponents 10-60 hp	10	N/A
Haste	Doubles amount of physical attacks	**3**	N/A
Stoneskin	Absorbs physical damage	10	N/A
Ice Storm	Damages opponents 5-30 hp	10	N/A
Chain Lightning	Damages opponent 15-90 hp	10	N/A
Spell Mantle	Absorbs damage from enemy spells	10	N/A
Horrid Wilting	Damages opponents 15-90 hp	10	N/A

Table 5: Black weights before normalization

After normalization, the following weights represent the weights for Black's spells.

Spell	Description	Weight	Percentage
Mage Armor	Makes the mage harder to hit physically	3.49	0-3.48
Magic Missile	Damages opponent 10-25 hp	11.63	3.49-15.17
Melf's Acid Arrow	Damages opponent 2-8 hp, plus extra damage subsequent rounds	11.63	15.18-26.74
Fireball	Damages opponents 10-60 hp	11.63	26.75-38.37
Haste	Doubles amount of physical attacks	3.49	38.38-41.86
Stoneskin	Absorbs physical damage	11.63	41.87-53.49
Ice Storm	Damages opponents 5-30 hp	11.63	53.50-65.12
Chain Lightning	Damages opponent 15-90 hp	11.63	65.13-76.74
Spell Mantle	Absorbs damage from enemy spells	11.63	76.75-88.37
Horrid Wilting	Damages opponents 15-90 hp	11.63	88.38-99.99

Table 6: Black weights after normalization

The next time spells are picked, the ones that were not successful will have a lesser chance of being included in future rounds, and other spells are more likely to be tried.

Dynamic scripting starts exploiting knowledge after a few trials and explores new knowledge continuously. Dynamic scripting updates all state-action rules through a redistribution process, unlike Monte-Carlo learning which updates state-action values only after they have been executed. This may lead to dynamic scripting not converging on a solution. Non-convergence is essential for use in games, since the human player may choose to switch tactics so the learning task continuously changes. It can quickly respond to a variety of behaviours, and the rule orderings dynamic scripting uses in successful strategies can be learned automatically. [Tim07] experiments with three different rule ordering mechanisms for dynamic scripting. While dynamic scripting can learn effective tactics, it has no built-in mechanism to improve diversity, so it tends to converge to static, predictable scripts [Szi09].

One benefit of dynamic scripting worth highlighting is the fact that the scripts it creates do not require game designers to produce complex scripts to try and make agents act in ways that appear intelligent. One example of human coding that produces inadequate results can be found in [Spr05]:

> "According to [Computer Role-Playing Game] tradition, dragons are both physically and mentally powerful creatures. While [the game] does not require the player to fight dragons, the designers realised that most players will attempt to do so anyway. Therefore they created complex game AI that should be able to humiliate any player bold enough to attack a dragon. Soon after the game's release, weaknesses in the game AI were discovered that players could exploit to defeat any dragon in the game, even with a weak team. Furthermore, without exploiting game AI weaknesses, players could still design superior tactics that, while unforeseen by the game developers, allowed weak teams to take on dragons successfully. It is trivial for a dragon to recognise that its current behaviour is inadequate to deal with tactics used by attackers that, according to its domain knowledge, are no match for it. Were the dragons controlled by adaptive game AI instead of static game AI, an answer to the superior and exploiting tactics could have been discovered automatically, keeping up the challenge level of the game."

Further explanation of static scripts producing inferior results occurs in Chapter 4: Results.

Dynamic scripting can learn effective tactics, but those tactics are specific to the situation being learned, which includes the teammates and opponents whose actions affect the effectiveness of the character's actions. Changes to the teammates and opponents can require new effective strategies to be learned. If a game can start using starting weights that are "closer" to the weights of an effective strategy, it will lessen the time needed to learn an effective strategy and there will be less chance of a player encountering a character performing actions that appear unintelligent. Dynamic scripting is only useful in a game where scripts are used to control agents, so Role-Playing, Action, and even

Strategy games could benefit from the following work. [Pol10] demonstrates how dynamic scripting can be used in a First-Person Shooter game, and [Sha10] shows how a companion controlled by a can learn the preferences of a player in a non-combat scenario (disarming traps) in a Role-Playing game.

The topics presented in this chapter will allow the reader to understand the contributions of this dissertation. Some assumptions need to hold true for this research to be useful. Firstly, the game that it is being applied to needs to use scripts. This will not be an issue in most cases, as scripts are used in most modern games. Since this research focuses on team strategies of heterogeneous agents, a game that only has one character type or a single character will not benefit from it. The following chapter introduces the game used in this study. It allows for thousands of combinations of players and millions of combinations of teams, which makes it an ideal tool for this research.

Chapter 3: Experimental Design

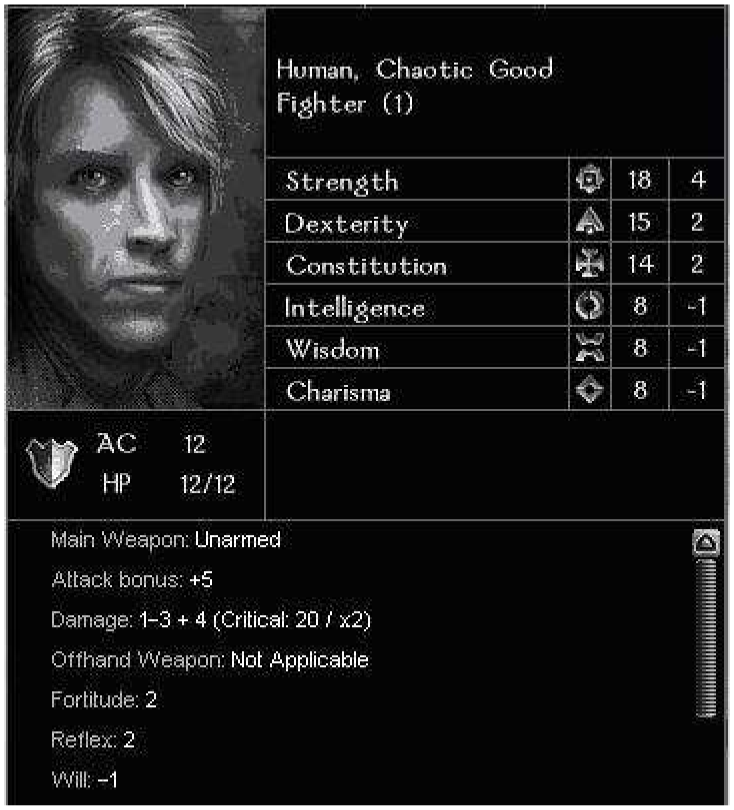
This chapter starts by providing a description of the game that will be used for the experiments that will show that using knowledge about agent's allies and opponents will allow better scripts to be created. It explains the different character types, the large number of possibilities a player has to choose from when creating a character, which leads to an enormous amount of possible combinations for even a small team. It mentions a valid abstraction in these experiments, roles, which can classify the characters based on their abilities. This is a way of abstracting the most relevant details of a team allowing the size of a game’s problem space be reduced to a manageable size. It then describes the experiments that will be performed to support the thesis.

NeverWinter Nights (NWN) overview

NWN is a Role-Playing Game (RPG) created by BioWare Corportation. It is based on rules from the Dungeons and Dragons game and has won multiple awards. NWN ships with the Aurora Toolset, which gives the functionality to create modules for the game. The University of Alberta GAMES group created ScriptEase, which uses pattern templates that allows complex behaviours to be easily programmed. The ScriptEase project has many recent publications ([Cut08], [McN04], and [Zha09] are a few). These reasons make NWN a good tool for this research.

Characters in NWN

The NWN game allows you to make many choices when creating a character. To help narrow down the search space, some analysis can be done that will limit the number of potential characters that should be examined to form part of the team. To help understand, we will start by examining the following two fighters created in NWN.



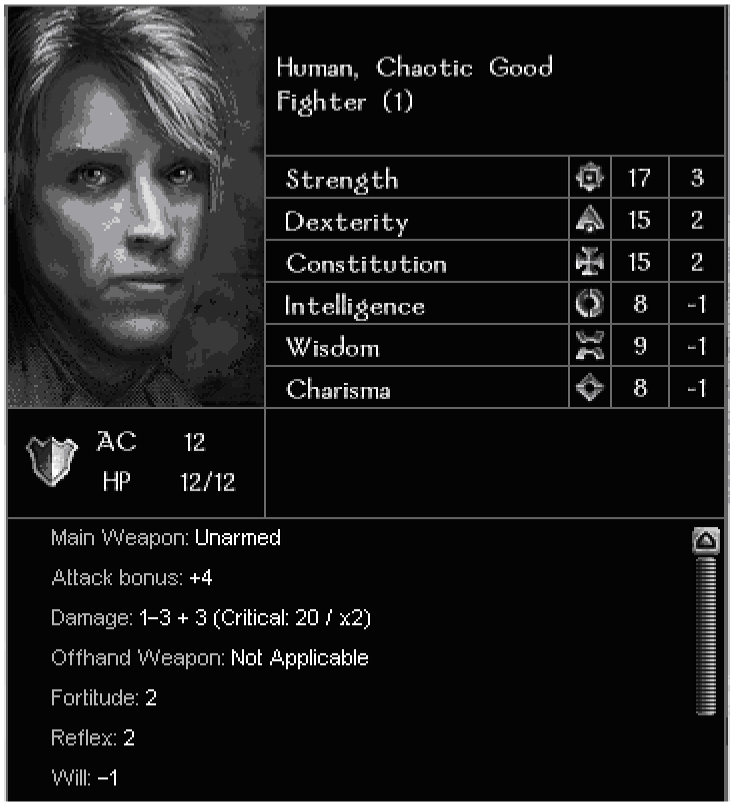
Human, Chaotic Good
Fighter (1)

Strength	18	4
Dexterity	15	2
Constitution	14	2
Intelligence	8	-1
Wisdom	8	-1
Charisma	8	-1

AC 12
HP 12/12

Main Weapon: Unarmed
Attack bonus: +5
Damage: 1–3 + 4 (Critical: 20 / x2)
Offhand Weapon: Not Applicable
Fortitude: 2
Reflex: 2
Will: –1

Figure 3: One possible Fighter in NWN



Human, Chaotic Good
Fighter (1)

Strength	17	3
Dexterity	15	2
Constitution	15	2
Intelligence	8	-1
Wisdom	9	-1
Charisma	8	-1

AC 12
HP 12/12

Main Weapon: Unarmed
Attack bonus: +4
Damage: 1–3 + 3 (Critical: 20 / x2)
Offhand Weapon: Not Applicable
Fortitude: 2
Reflex: 2
Will: –1

Figure 4: A slightly different Fighter in NWN

By examining the differences between the above figures, you will notice the first fighter has a higher Strength score (18, opposed to 17 for the second fighter), which leads to a larger Strength bonus (4, opposed to 3), resulting in larger “Attack bonus” and “Damage” values (+5 and 1-3 +4, opposed to +4 and 1-3 +3). This makes the first fighter more effective when it comes to melee combat. All characters that are created in NWN have the same number of ‘points’ to spend raising the different attributes. In the above example, the second fighter has higher Constitution and Wisdom scores than the first fighter (15 and 9, opposed to 14 and 8), but the bonuses for those scores in the last column are identical, so there is no meaningful advantage for the second fighter. There may be some scenarios where the second fighter might be preferred, but for the purposes of the following experiments that involve fighting in an arena, the second fighter can be ignored.

A graphical comparison of these two fighters appears below. To help with the analysis, the characters are classified based on the following abilities.

- Hit points (HP): These represent how much damage the character can take before falling. Other things being equal, a character with a high number of hit points will outlast a character with a low number of hit points in a fight.
- Melee ability: This is a representation of how well a character fights physically. A character with higher melee ability would be more likely to hit an opponent with a weapon.
- Armor class (AC): This represents how difficult it is to hit and damage the character. A character with a high AC is difficult to damage because he is either nimble and can avoid being hit, or can wear heavy armour.
- Magic resistance: A player with a higher magic resistance is more likely to avoid most or all of the damage from an opponent’s offensive spell.
- Helpful magic (healing): There are lots of spells that are beneficial to the members of a party. To help distinguish, we have divided them into two groups. The first type is ‘healing’, which returns HPs to a character that has been injured. The second type removes a harmful effect that is hindering an ally. For example, a character that has a high healing rating would be able to cast a spell that returns an injured fighter to full health.
- Helpful magic (defensive): This group of spells help a character or team by increasing various other abilities. These can include making a character more resistant to magic, increasing their AC so they are more difficult to hit, adding to their attributes so their other skills and abilities become more effective.
- Offensive magic: A player with a high offensive magic rating can cast spells that damage their opponents. For example, he might cast a spell called ‘Fire Ball’ which causes an explosion around his enemies.
- Special/non-combat: There are many abilities that are not directly related to combat, but instead are useful to the player in other ways. For example, having the ability to move silently (so the character might avoid a fight), pick pockets (so a character might be able to steal a key from a guard instead of needing to fight for it), or barter (so the character can buy equipment cheaper, allowing them to

purchase more or better equipment than a character without that skill) would be useful for some people to have. These skills are important for gameplay, but due to the experiments involving combat, these skills will not play a role in the outcome.

The scale for each of the axes on these radar graphs are relative, with 0 meaning no ability in that area, and 5 meaning the best among the players in that area. The fighters have the best hit points of the characters under consideration, so they have the highest value along that axis. Since the second fighter is not as good as the first fighter with respect to melee ability, it has a lower ranking on that axis. Since it has no extra abilities to compensate, we can ignore the second fighter when trying to form the best team.

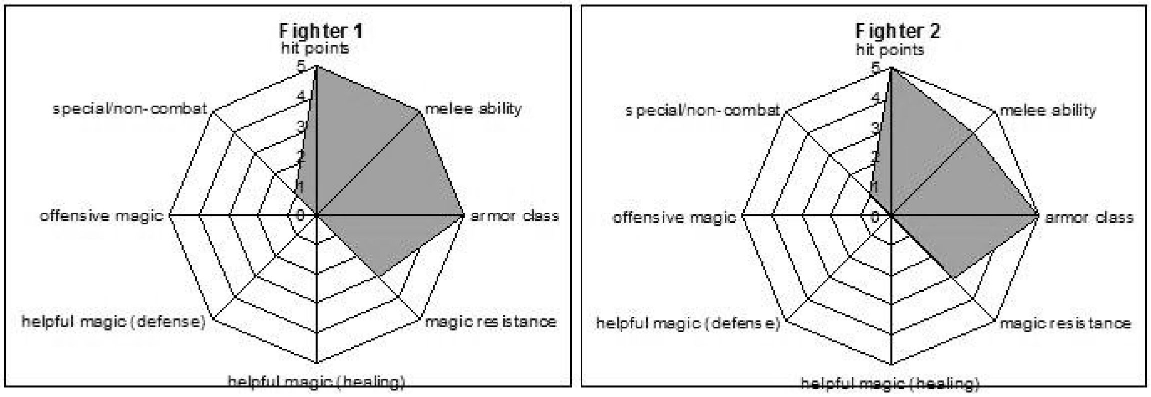


Figure 5: A comparison of the two Fighters

Roles and Abstraction

In the NWN game, there are different races and character classes the player can choose from, along with thousands of ways of assigning the attributes. The number of combinations makes any exhaustive search of all possible teams intractable. There are 7 different races a player can pick from, each with different bonuses or penalties, and 11 different character classes. Even if you only looked at the 11 basic classes and looked at teams of 4 characters, there would still be 11^4, or 14641 combinations. A valid abstraction can be used to remove detail from the representation, making it simpler to find a solution. One useful abstraction is to group similar characters together based on the roles they are best suited for. The application of roles makes it more likely for this paper to be useful in games like NWN with millions of possible team combinations, since a game with only a few possible classes it might be able to determine a 'best' strategy for every situation by exhaustive search. Some games that ship have only a very limited number of playable characters that the player can choose to control. If one of the factors that influenced the decision to ship these games was the need to verify the combinations of playable characters, then this research may be used to make it easier for the game makers to include more playable characters in their games.

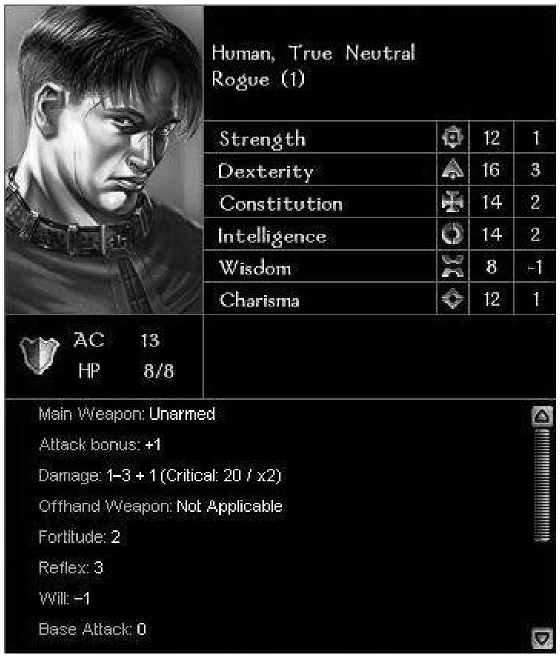
The concept of abstraction in games and other computer science problems has been explored by many others, including [Gil07] who uses abstraction to reduce the complexity of a poker game, and [Cla94] that uses abstraction to create a representation of a circuit with over 10^{1300} states so it could be verified.

The concept of roles is explained in [Lu05], where agents are classified by the capabilities they can offer. To help simplify selection process, roles can be used to guide search; first the roles that appear in the best teams can be determined, and then best character types to fill those roles can be found. Some character types are more effective than others at certain roles, and some can play more roles than others. The ‘Defender’, ‘Striker’, ‘Leader’, and ‘Controller’ roles are described in [Sch09] and refer to characters in the NWN game.

Roles can help simplify the search for the best team. As an example, a hockey coach might not put his best six players on the ice, especially if none of his top six players is a goalie. The concept of a goalie simplifies the search since it limits the criteria for the selection of that player to the abilities that are important for someone who is playing in that position.

Characters are classified in a role based on what they are best at ‘doing’. For example, the ‘Defender’ role should be played by a character who is tough, able to take and dish out a lot of physical damage, so the fighters described above could easily fall into this role. By examining the relative strengths of the team members, it is hoped that rule orderings learned through dynamic scripting can be avoided in situations where they will be less useful. For example, it should not be the case that a wizard would cast a spell that protects him against magic if none of the members on the opposing team can cast offensive magic. Also, some of the beneficial magic spells enhance a character’s other abilities. For example, there is a spell called “Bull’s Strength”, which makes a character stronger. This spell will be most effective if it is used on a teammate that benefits from having a high Strength score, like a fighter (reasons described above), and not as useful if cast on a Wizard.

Below are a few other characters created in NWN, and their corresponding relative abilities.


Human, True Neutral
Rogue (1)

Strength	12	1
Dexterity	16	3
Constitution	14	2
Intelligence	14	2
Wisdom	8	-1
Charisma	12	1

AC 13
HP 8/8

Main Weapon: Unarmed
Attack bonus: +1
Damage: 1–3 + 1 (Critical: 20 / x2)
Offhand Weapon: Not Applicable
Fortitude: 2
Reflex: 3
Will: –1
Base Attack: 0

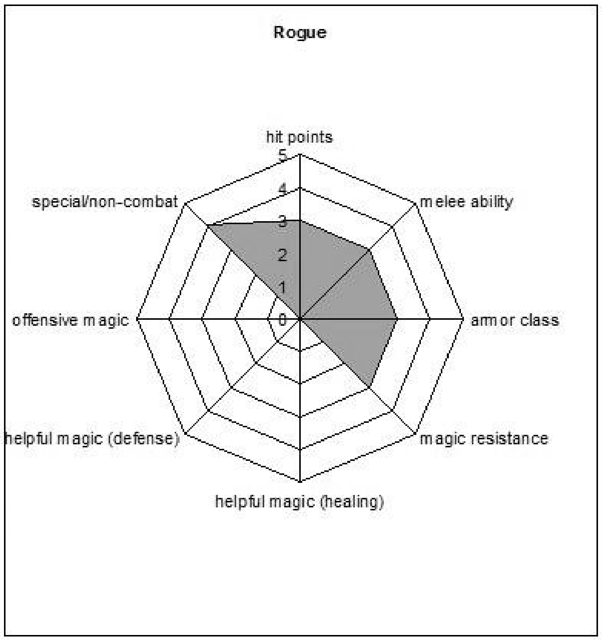
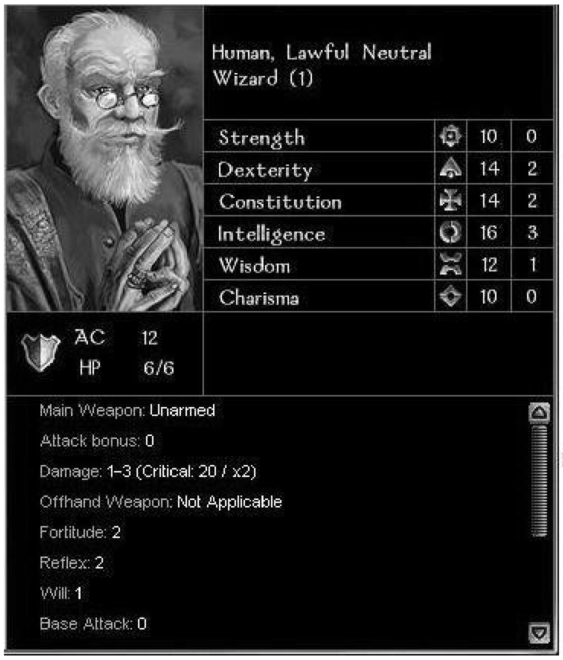


Figure 6: A Rogue in NWN, and relative strengths

A rogue has several abilities that can aid a team outside of combat, like finding traps. Under certain circumstances they can do well in combat, but generally they are less effective than a fighter due to limitations on the armour they can wear and the weapons they can use. They also have fewer HPs than a fighter, and lack the ability to cast magic spells.


Human, Lawful Neutral
Wizard (1)

Strength	10	0
Dexterity	14	2
Constitution	14	2
Intelligence	16	3
Wisdom	12	1
Charisma	10	0

AC 12
HP 6/6

Main Weapon: Unarmed
Attack bonus: 0
Damage: 1–3 (Critical: 20 / x2)
Offhand Weapon: Not Applicable
Fortitude: 2
Reflex: 2
Will: 1
Base Attack: 0

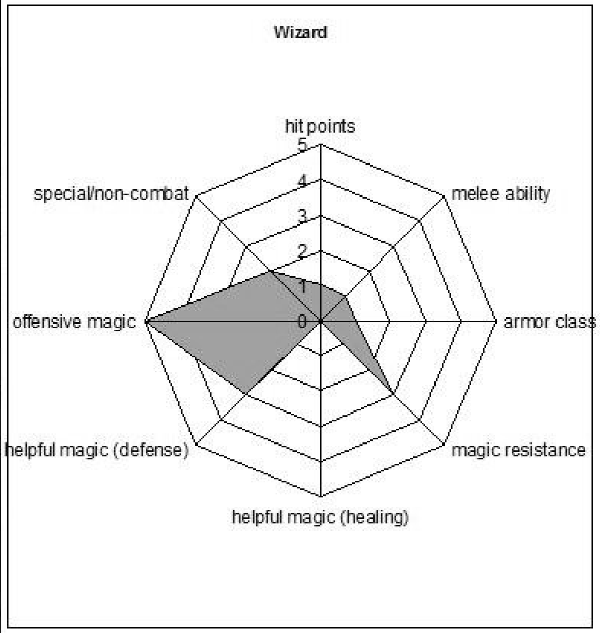
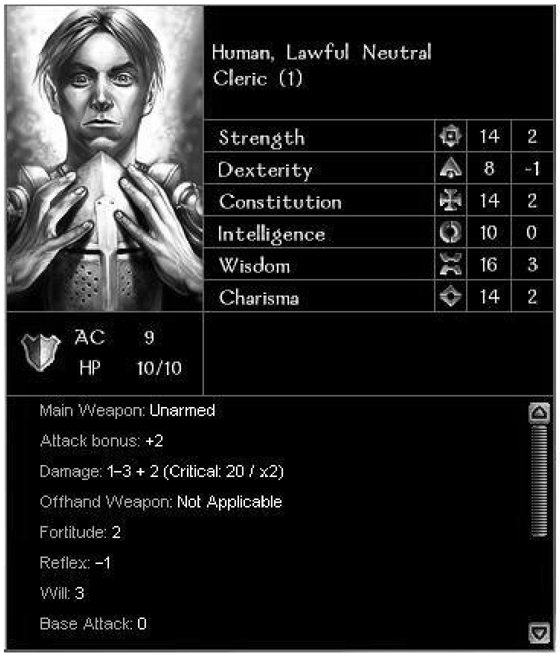


Figure 7: A Wizard in NWN, and relative strengths

A wizard is able to cast magic spells, and has powerful offensive spells. To balance this they have the fewest HPs, cannot wear armour, and are poor in melee combat.


Human, Lawful Neutral
Cleric (1)

Strength	14	2
Dexterity	8	-1
Constitution	14	2
Intelligence	10	0
Wisdom	16	3
Charisma	14	2

AC 9
HP 10/10

Main Weapon: Unarmed
Attack bonus: +2
Damage: 1-3 + 2 (Critical: 20 / x2)
Offhand Weapon: Not Applicable
Fortitude: 2
Reflex: -1
Will: 3
Base Attack: 0

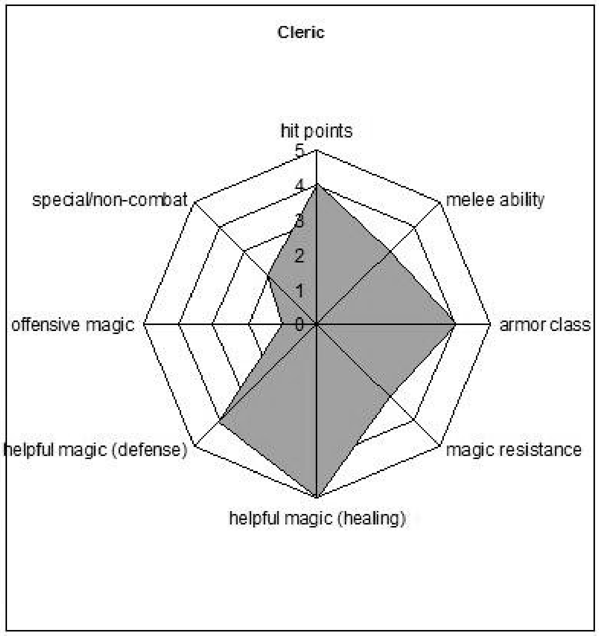


Figure 8: A Cleric in NWN, and relative strengths

A cleric can cast healing spells and other magic that is beneficial to a party. They can also wear good armour, and are fairly good in melee combat.

We will use roles to simplify the number of combinations of team members. Fighters and Rogues have similar actions to choose from in combat. Wizards and Sorcerers can cast the same spells, so if a spell has been found to be effective in a certain situation when a Wizard is on a team, then that spell will also be effective if the team has a Sorcerer instead. There is a large overlap in the spells available to a Cleric and the spells available to a Druid and of the spells that are common to both classes, the effective ones are effective regardless of what character class has cast it.

Experimental Design

The purpose of these experiments is to see if information about the other agents (teammates and opponents) can be used to create better rule orderings faster, and to perform better than current scripting implementations. Since the characters that can cast spells have more options in their rulebases to choose from, this research will focus on their rule orderings (the fighters are effective by moving into combat and then attacking with their sword until combat is finished, the magic users have a large range of spells to choose from, which allows more possible strategies).

We propose expanding the rulebase in dynamic scripting to have different weights depending on the teammates and opponents the agent has, rather than simply have a list of scripts and associated weights. In most games there would be too many combinations of agents to have an exhaustive list of possible teams, so we use roles to keep the complexity down. In our experiments, we create our team from three different character types: fighter (F), cleric (C), and wizard (W). Briefly, a fighter is a tough character that cannot use magic, a cleric's magic is mostly beneficial to his team, and a wizard has offensive magic that directly damages opponents. Should other character types be used, they would be grouped in the role that is closest to their capabilities – i.e., barbarian and monk types could be classified as F since they do not possess magic and have good melee abilities like a fighter. Experimental results will verify that this abstraction is effective in NWN.

At a high level, here are the steps that will be performed:

Step 1: Recreate a simple model of the NWN arena. By avoiding the NWN graphical user interface, results will be obtained faster. A text-based model of the NWN game will be created with the relevant rules. Some assumptions were made when making the text-based version. These assumptions include: Some area-effect spells damage all opponents; Fighters cannot attack on the first turn. More assumptions are listed in the next section. If these assumptions do not hold in the NWN arena, it does not mean that the results are invalid, but only that the discovered strategy in the text-based arena is not a valid strategy in the NWN arena. This step will allow results to be collected much faster than in the actual NWN game, and will provide a complex environment with many character types where the experiments that will support the thesis can be performed. It will also show that the abstraction from an actual commercial game to a text-only version is possible, which will allow this research to be applicable to other games

Step 2: Use dynamic scripting to come up with effective rule orderings for various scenarios (a team of magic users against opponents without magic users, a team with a wizard and many fighters against a team of magic users, a team of magic users against a balanced team). The results from each of these scenarios will be saved and used to create scripts in future steps. This step will also support the claim that a rule ordering that dynamic scripting creates for one character type will not necessarily be effective for that character in a different situation. Since other variables in the arena can be held constant while one factor is changed (for example, the same teammates against a different set of opponents), any changes to the resulting script that dynamic scripting produces can be attributed to the change of agents participating.

Step 3: Analyse the results and find correlations between weights of rules and the agents participating in combat (for example, it should be determined that casting "Haste" on a team with only wizards is less effective than casting it on a fighter, and that casting spells that protect the user against magic are not effective when facing opponents without magic users). The outcome of every battle is stochastic according to the game rules. Even if both teams use fixed tactics, all spell effects and inflicted damage have random factors. The opponents' behaviours will be determined by the scripts. Each time a decision needs to be made, the rules are checked in order and the first applicable rule will be executed. This will help determine which rules are simply effective in most situations and those that are most effective only in certain instances. Determining the situations where certain spells are effective is necessary to support the claim that some spells are effective against certain opponents or when certain teammates are present. The advantage that will arise from this thesis will come from finding the spells that are most effective in certain situations and incorporating them in the scripts for those situations.

Step 4: Create new random teams and opponents. Run experiments with them using rules that should be effective against them based on the results of step 3. Compare the results of the teams with the static strategies delivered with the NWN game to those with strategies created from the results of the dynamic scripting experiments. This step will verify that scripts created with dynamic scripting can be more effective than those hand-created by experts, and taking the agents into consideration can create a better starting point.

Step 5: Verify the results by implementing the script in the NWN arena, and comparing them to the default learning strategies. If the scripts created by the above techniques can find an effective strategy against an opponent more quickly than by the dynamic scripting algorithm starting with the static scripts shipped with the game, we will be able to say that dynamic scripting can create an effective rule ordering more quickly by using information about opposing agents. This step will also show that this research can be applied to improve a commercial game.

The steps are now described in more detail:

Recreating the Arena

As noted in [Tim06], by not using a graphical user interface we can simulate a large number of combats. Similar speed increases were found in these experiments by using a text-based simulation rather than using the actual NWN game. While creating the text-based simulation, some simplifying assumptions were made based on observations gathering information for [Pri10]. First observation: since the arena is small, area-effect spells almost always affected all of the enemies. For the text-based simulation, an area-effect spell will hit every opponent. As there are no "obstacles" (walls, pillars, doors) between the opponents, the rules about partial cover were not implemented. Second observation: when opponents had both melee (close-quarter) and ranged weapons (bows, etc.), the melee weapons were almost exclusively used, and the strategies learned always had the agents using their melee weapons instead of their ranged weapons. For the text-based simulation, the agents will only have melee weapons. Third observation: 'fighter' agents (in this context, fighter refers to any class whose primary function is to damage the opponent by getting into melee combat rather than by casting magic spells) would advance towards the opposing team, and make it to the opposing team in one round (a round is the unit of combat used in NWN). If the opposing team had 'fighters', they would meet near the middle of the arena, and a team's fighters would not be able to advance to the opponent team's magic users until all of the opposing team's fighters were defeated. If there were no fighters on the opposing team, the fighters would be able to make it to the opposing team's magic users at the end of the first round. For the text-based simulation, the fighters would not get an attack on the first round (which would represent the time it took to move into position), and then they would only attack the opposing team's fighters first, and then attack the opposing team's magic users second. In the event that one team had only magic users remaining and they ran out of spells, they would move into melee combat and then attack with their weapons. Some advanced rules were not implemented – occasionally a fighter can knock down an opponent after a hit, some bonuses to hit could occur if an attacker tried to hit someone from behind, an extra attack was allowed if someone tried to perform certain actions near an enemy. This kept the text-only version of the arena a simplified representation of the NWN arena. In a two dimensional world like the NWN arena, fighters could choose different opponents to attack if there were more than one nearby. In the text-based version, all attacks are concentrated on a single enemy, which can easily cause different results.

These assumptions might make the strategies learned unrealistic in some situations. The assumption of area-effect spells hitting all opponents for example would be unrealistic if the fighting was to occur in a large battlefield instead of the arena. For the purposes of these experiments, the assumptions should not affect the results. Verification of the strategies learned will occur in the last step when the fights are reproduced in the arena with the strategies learned.

Use Dynamic Scripting in different situations

Different teams will fight, and effective strategies will be learned. This will be similar to work described in [Spr05] and [Tim07]. It has already been shown in [Spr06] that this method will allow the agents of a team to learn an effective strategy against an opponent. This step is required to collect the data needed for the following step, where the effective strategies that the individual agents develop against different teams will be analysed. Optimal strategies will be learned with dynamic scripting for teams with different amounts of fighters, clerics, and wizards, against teams that also have different amounts of the above agents.

A rulebase will be initially set up and populated by letting certain teams of agents play against similar teams. Dynamic scripting will be used to come up with effective strategies for the cleric and wizard classes. Dynamic scripting will not be used for the fighter class, as that class basically had one effective action to perform: hit an opponent with its sword. The spells for the cleric and wizard classes will be taken from the default spell selection that came with the characters created in the NWN game. One exception will be made; wizard spell “invisibility” will be switched for a different spell of that level “web”, since there was a documented issue with the implementation of “invisibility” in the NWN dynamic scripting module.

To come up with the initial weights for the rules, a well-rounded team (FFCW) will fight a similar team for a thousand rounds five times. After this point, some rules with the consistently lowest weights will be removed. The spells that ended up being removed were the ones that did not have an effect on combat. Appendix C contains a listing of the spells that were removed. [Spr06] states that “it is imperative that the majority of the rules in the rulebase define effective, or at least sensible, agent behavior.” Rules that will not affect combat will be removed since they have no practical value.

Find correlations between effective strategies and participating agents

First, all of the strategies will be examined to see if common elements can be found (for example, while watching fights in arena, it was observed that there are certain situations when a rule is always a good idea, or always a poor one. It is always a poor decision for a character to run past enemy fighters, as that will allow the fighters to get extra attacks against the character, with bonuses from attacking from behind. It is always a poor decision for a wizard to start a combat by moving closer to the enemies and attacking with a melee weapon. A wizard can be much more effective by casting an offensive spell. It is a good decision for a cleric to heal a fighter on the same team that is wounded, regardless of the opponents being faced, as this will potentially allow the fighter to stay on his feet longer, which will help the team win. It is always a good idea to start combat by casting a spell that improves the abilities that a team mate is using, as this will maximize the amount of time the benefit will be in effect.

Since different scripts should be used in different situations, it is necessary to determine what rules are more effective in the different situations. The goal of this research is to see if the information about other agents can be used to create better starting points for dynamic scripting. There are two groups of agents that need to be examined when determining a strategy, an agent's opponents and an agent's teammates.

Opponents

The teams will be categorized by their relative strengths and weaknesses (for example, we may classify a team as having low melee abilities, average AC, …and high offensive magic ability, or having high melee abilities, high AC,… and no offensive magic abilities). An agent can come up with a strategy that is effective against one set of opponents but not effective against another. The scenario of a wizard casting a spell that protects him from magic is effective against opponents that have offensive magic capabilities, but not against opponents that have no offensive magic capabilities.

Teammates

The teammates of an agent will also be classified according to strengths and weaknesses. The most effective strategy a player can take may depend on the abilities of allies. A strategy for increasing the melee abilities of allies on a team will be more effective if there are teammates with high melee abilities to begin with. The scenario of a wizard casting a spell that increases the speed of a teammate is effective when there is a teammate who is a fighter, but much less effective when all of the teammates are also wizards.

Using roles and the three character types above, we can create a partition that groups all teams into one of seven sets depending on whether or not it has fighters (hasF), clerics (hasC), and wizards (hasW). An eighth partition where all 3 functions are false might exist in a game – if the player was facing a monster for instance, but for our purposes we will ignore this case. The abstraction of using roles is necessary to reduce the problem space into a manageable size.

In our experiments our fights will contain teams of four agents verses four agents. A team of four fighters (FFFF) would be the only team to fall in the hasF=Y, hasC=N, hasW=N set, whereas there are three teams in the hasF=Y, hasC=N, hasW=Y set (FFFW, FFWW, FWWW). Similar classifications can be made for the opponent teams: oHasF would be Y if the opponent team had fighters, and if an agent was fighting a team of FFWW, then oHasF and oHasW would be Y and oHasC would be N.

If an action is beneficial in a certain situation regardless of the teammates an agent has (for example, casting a spell that protects it from enemy magic), its weight should be independent of the values of hasF, hasC, and hasW. Similarly, if an action is beneficial in a situation regardless of the opponents (summoning a creature to help fight if there is no fighters on your team), then it should have a high value in all situations of oHasF, oHasC, and oHasW.

Create new teams, and synthesize starting strategies

Teams that have not been used in the above steps will be created, and starting strategies will be created from the results in the above step. The starting strategies will be made up of ‘common’ rules that are effective in all scenarios (healing a teammate who is injured), and of rules that have been found to be effective in similar situations (with similar teammates or against similar opponents). Fights will then occur as usual, and we will be able to determine the effectiveness of the new starting strategies against the default starting strategies. If the teams can consistently start with a dominant strategy using the new starting points than with the default strategies, then this will demonstrate the usefulness of this technique.

Other teams fought each other (FFFW), and then teams of different agents fought each other (FCWW vs. FFWW, FFFW vs. FFCW, FCWW vs. CCWW, etc.). The different teams and their weights were classified based on six functions hasF, hasC, hasW, oHasF, oHasC, and oHasW. These classifications are used to reduce the large number of possible combinations of character classes into a smaller number of groups to help determine the best starting strategy.

The training process was done a total of three times for each set of fights for verification purposes. Although exact matches were not expected for any of the weights, trends were observed, and the top spells in any given scenario were often the same spells in other scenarios, with slightly different weights. The variations in the results between different runs are expected as each round in combat has many random factors (whether or not a fighter hits its opponent with a sword, or the amount of damage a wizard does with an offensive spell). This randomness also affects the dynamic scripting process, as the weights that are updated depend on the fitness of the agents (whether or not the team won or not, and how decisive the win was).

To synthesize the starting weights, the first step is to look for matches from previous training processes, and if no match is found to average the weights from the nearest neighbors. An example will help explain the process. Training was performed with the following test cases. The six letter code after each line corresponds to the hasF, hasC, hasW, oHasF, oHasC, oHasW functions (in that order) with respect to the W in the first group. Since the W in first case has an F and a C as team-mates, but does not have a W, the first letters are YYN. Since the opponent team has a member of F, C, and W, the last letters are YYY.

1: FFCW vs. FFCW	YYNYYY
2: FFFW vs. FFFW	YNNYNY
3: FCWW vs. CCWW	YYYNYY
4: FFWW vs. FFCC	YNYYYN

Now for combat, we want to generate a starting script for the W in the first team in FFCW vs. CCWW. The corresponding code for this W would be YYNNYY. As there is no exact match, we compare this to the other codes and find that it matches 5/6 letters in both test cases 1 and 3. We average the W weights that resulted from those two training runs. A similar process can be used to come up with the starting weights for the other agents. A complete example is shown in Appendix B, and some highlights are below. The first column contains the names of some of the spells available to the Wizard. The second column contains the weights for those spells from the Wizard in the FFCW team when it faced the FFCW team (row 1 above). The third column contains the weights for those spells from a Wizard in the FCWW team when it faced the CCWW team (row 3 above). The fourth column contains the average weight for those two columns. In some cases, an effective spell in all cases (horrid wilting or ice storm for example) continues to have a high weight. In other cases, some spells have a high weight in one instance but not the other, and in those cases the average is used. In these cases, the dynamic scripting process will cause the averaged weight will go up if these spells are effective in the new scenario, or down if not. The spells that have a high weight will likely be effective, and will appear as an intelligent action to take. Any spells that have a low weight in most cases not have a high chance of being picked, since it is likely to not be effective.

team	FFCW	FCWW	FFCW
opponent	FFCW	CCWW	CCWW
character	W	W	W
classification	YYNYYY	YYYNYY	YYNNYY
melf's acid arrow	0.1797	0.0011	0.0904
evard's black tentacles	0.1797	0.0011	0.0904
ice storm	0.1950	0.1988	0.1969
chain lightning	0.1950	0.0011	0.0980
summon creature 6	0.0011	0.0335	0.0173
horrid wilting	0.1950	0.1999	0.1974

Table 7: Synthesis of selected starting weights for W in FFCW v CCWW

Verify strategies in the Arena

Strategies that have been learned in the above 'text-based' simulation will be verified in the NWN Arena. If the teams learn strategies that perform better than the opponent teams faster than by starting using the default starting strategies, then the assumptions made in the text-based simulation are justified, and this approach will have been demonstrated to be effective in a commercial game.

Chapter 4: Results

The following pages contain results from the text-based arena, which was used to quickly simulate thousands of rounds of combat from the NWN game. Results from the actual NWN game start appearing in Table 12.

The step of populating the initial rulebase had the purpose of verifying the implementation of dynamic scripting, as weights that we expected to be effective often had higher weights, and spells we thought would not be effective ended up with lower weights. This step helped catch a bug in our implementation, as one spell that should have been effective ended up having a very low weight. This spell provided significant bonuses to a teammate, and should have had a high weight. After some troubleshooting, it was found that the spell was actually providing the bonuses to an opponent instead of an ally, and thus was detrimental to the caster. This underscores the premise in [Spr04] that scripts can be complex and can contain errors, and that dynamic scripting can account for them.

Below are the results of the text-based experiments, starting with the weights for W spells in FFCW versus FFCW. These numbers in each row correspond to the weights of the wizard spell in the left-hand column after the fight at the top of that column. For example, after 10 fights, the ‘magic missile’ spell had an 8.2% chance of being included in a script, but by fight 100 it had about a 0.1% chance of being included. Some spells have weights of 0. During preliminary testing, those spells were determined to be ineffective regardless of the scenario, so were removed. Further explanation is given in Appendix C.

after fight:	10	50	100	250	500	750	1000
daze	0	0	0	0	0	0	0
light	0	0	0	0	0	0	0
ray of frost	0.0126	0.0020	0.0011	0.0011	0.0011	0.0011	0.0011
resistance	0.0774	0.0310	0.0355	0.0554	0.0667	0.0351	0.0123
mage armor	0.0126	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
summon creature 1	0.0015	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
magic missile	0.0821	0.0161	0.0011	0.0011	0.0011	0.0011	0.0011
Identify	0	0	0	0	0	0	0
sleep	0	0	0	0	0	0	0
ghostly visage	0.0126	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
melf's acid arrow	0.0844	0.1153	0.1474	0.1685	0.1555	0.1744	0.1797
summon creature 2	0.0023	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
Knock	0	0	0	0	0	0	0
web (WAS invisibility)	0.0074	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
fireball	0.0031	0.0019	0.0011	0.0011	0.0123	0.0011	0.0011
clarity	0.0078	0.0047	0.0015	0.0011	0.0011	0.0299	0.0198
summon creature 3	0.0268	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
haste	0.0335	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
dispel magic	0.0063	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
stoneskin	0.0605	0.0050	0.0016	0.0011	0.0011	0.0011	0.0011
elemental shield	0.0836	0.0171	0.0064	0.0255	0.0011	0.0011	0.0011
evard's black tentacles	0.0877	0.1964	0.1964	0.1987	0.1746	0.1967	0.1797
ice storm	0.0885	0.1964	0.1964	0.1346	0.1742	0.1465	0.1950
minor globe of invulnerability	0.0126	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
lesser spell mantle	0.0264	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
summon creature 5	0.0126	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
cloudkill	0.0694	0.0040	0.0013	0.0011	0.0011	0.0011	0.0011
greater shadow conjurations	0	0	0	0	0	0	0
chain lightning	0.0531	0.1964	0.1964	0.1960	0.1880	0.1967	0.1950
greater spell breach	0.0068	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
summon creature 6	0.0055	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
mordenkainen's sword	0.0123	0.0011	0.0011	0.0011	0.0120	0.0011	0.0011
spell mantle	0.0239	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011
horrid wilting	0.0859	0.1964	0.1964	0.1970	0.1941	0.1967	0.1950
melee	0.0009	0.0011	0.0011	0.0011	0.0011	0.0011	0.0011

Table 8: Spell weights for Wizard in FFCW v FFCW

Selected weights from the previous table appear in the figures below. Even though all of the spells started with equal weights initially, after only 10 fights some of the effective spells were already getting higher weights.

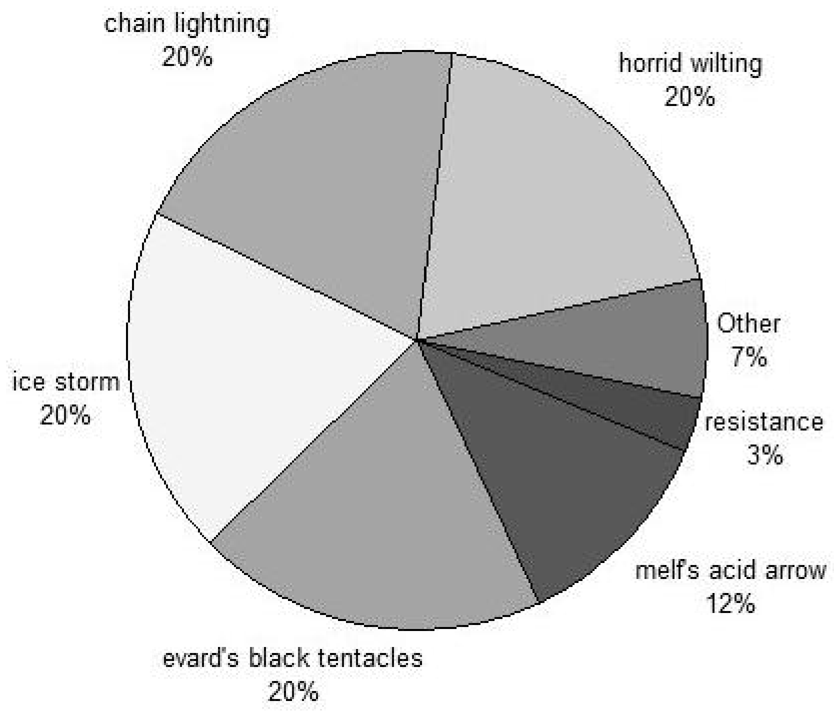


Figure 9: Wizard spell weights after 50 fights in FFCW v FFCW

After 50 fights, the wizard has already learned the most effective spells. From a starting point where the 29 spells had similar weights, dynamic scripting has already raised the weights of some spells and lowered the weights of spells of others.

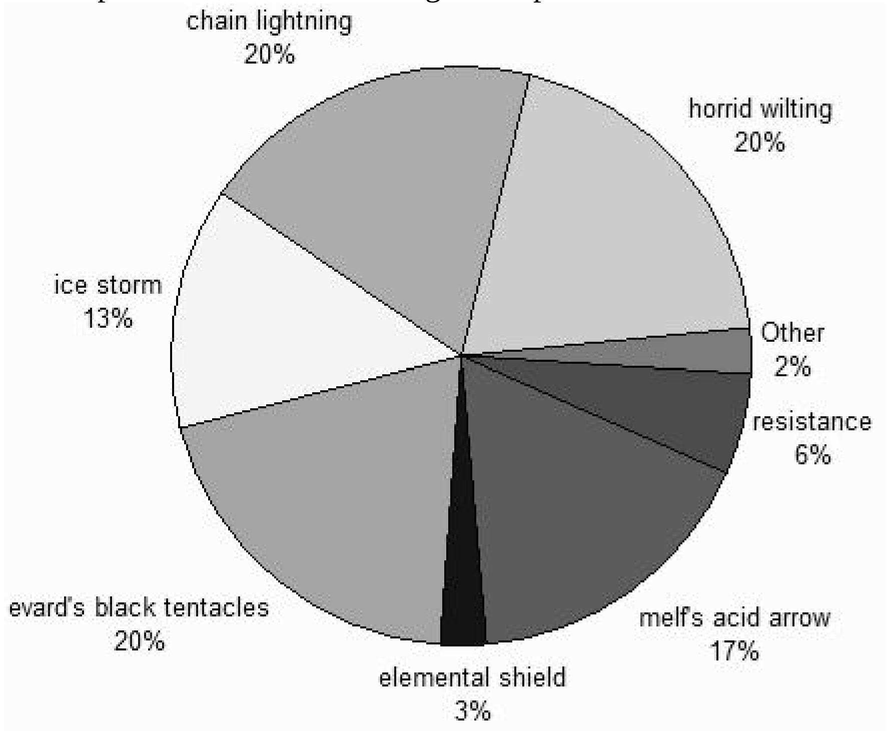


Figure 10: Wizard spell weights after 250 fights in FFCW v FFCW

After 250 fights, the weights have changed, but a script created from the most effective spells still remains fairly similar to the one created after 50 fights.

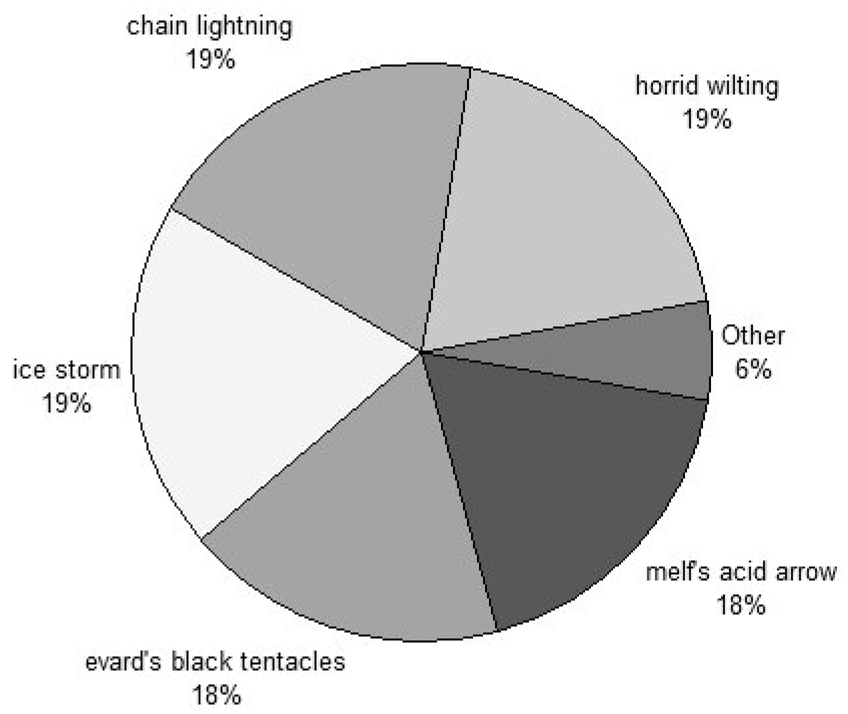


Figure 11: Wizard spell weights after 1000 fights in FFCW v FFCW

In the last three figures the weights have changed. Even though the scripts created after these fights would be different, the top 5 spells would have been present in all of the scripts.

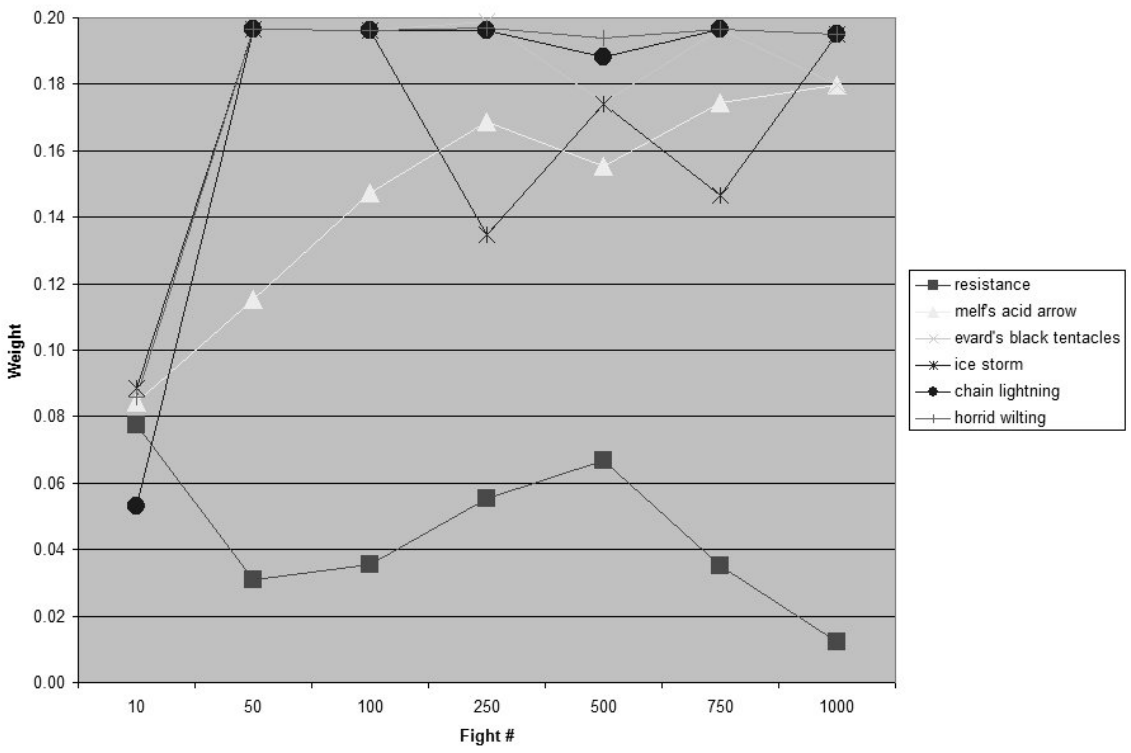


Figure 12: Selected weights for W Spells (FFCW v FFCW)

Figure 12 contains the weights of 6 of the spells after various fights. All of the spells started with an equal weight. Most of the spells were not effective, and their weights fell down to the minimum. From this figure it is easy to see how effective spells quickly can be identified after only a few fights. Some variation in the order of spells in a script occurs as more trials are completed. This type of variation is beneficial in a game, as it allows for different behaviours, and can compensate for changes in a player's behaviour.

The weights for the C spells using the same teams (FFCW v FFCW) appear below:

after fight:	10	50	100	250	500	750	1000
inflict minor wounds	0.0062	0.0021	0.001	0.0011	0.001	0.0011	0.0011
cure light wounds	0.0129	0.001	0.001	0.0011	0.001	0.0011	0.0011
inflict light wounds	0.0359	0.0011	0.001	0.0011	0.001	0.0011	0.0011
bless	0.0071	0.001	0.001	0.0011	0.001	0.0011	0.0011
cure moderate wounds	0.0129	0.001	0.001	0.0011	0.001	0.0011	0.0011
inflict moderate wounds	0.0509	0.0202	0.0024	0.0011	0.001	0.0011	0.0011
aid	0.0402	0.001	0.001	0.0011	0.001	0.0011	0.0071
cure serious wounds	0.0583	0.0836	0.0679	0.1899	0.1826	0.1609	0.1816
inflict serious wounds	0.0009	0.001	0.001	0.0011	0.001	0.0011	0.0011
remove blindness/deafness	0	0	0	0	0	0	0
remove disease	0	0	0	0	0	0	0
dispel magic	0.0327	0.0022	0.001	0.0011	0.001	0.0011	0.0011
animate dead	0.0129	0.001	0.001	0.0011	0.001	0.0011	0.0011
hammer of the gods	0.0468	0.1549	0.1639	0.1795	0.1755	0.1925	0.1955
inflict critical wounds	0.0085	0.001	0.001	0.0011	0.001	0.0011	0.002
freedom of movement	0.0427	0.001	0.001	0.0011	0.001	0.0011	0.0011
cure critical wounds	0.0374	0.001	0.001	0.0011	0.001	0.0011	0.0011
divine power	0.0396	0.001	0.001	0.0351	0.1826	0.1925	0.145
dismissal	0.0494	0.1418	0.1735	0.1899	0.0685	0.0553	0.0011
heal	0.0513	0.001	0.001	0.0011	0.001	0.0011	0.0011
slay living	0.0449	0.001	0.001	0.0347	0.001	0.0011	0.0011
flame strike	0.0536	0.0021	0.001	0.0011	0.001	0.0011	0.0011
cirle of doom	0.0201	0.0022	0.001	0.0011	0.001	0.0011	0.0011
spell resistance	0.0256	0.001	0.001	0.0011	0.001	0.0011	0.0471
harm	0.0244	0.001	0.001	0.0011	0.001	0.0011	0.0011
blade barrier	0.0129	0.001	0.001	0.0011	0.001	0.0011	0.0011
banishment	0.0576	0.1886	0.1891	0.0011	0.001	0.0011	0.0011
control undead	0.017	0.001	0.001	0.0011	0.001	0.0011	0.0011
destruction	0.0731	0.1886	0.1891	0.1563	0.1826	0.1797	0.1955
greater restoration	0	0	0	0	0	0	0
regenerate	0.0009	0.001	0.001	0.0011	0.001	0.0011	0.0011
fire storm	0.051	0.1886	0.1891	0.1882	0.1826	0.1925	0.1955
earthquake	0.0715	0.0053	0.001	0.0011	0.001	0.0011	0.0069
melee	0.0009	0.001	0.001	0.0011	0.001	0.0011	0.0011

Table 9: Spell weights for Cleric in FFCW v FFCW

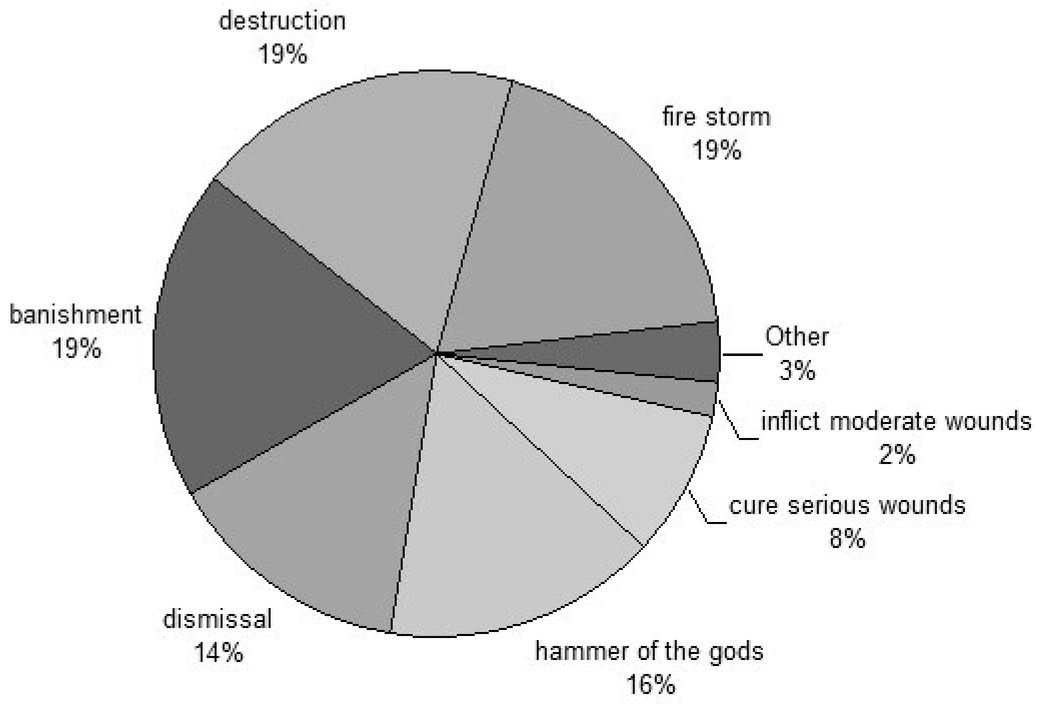


Figure 13: Cleric spell weights after 50 fights in FFCW v FFCW

Similar to the wizard spell weights in Figure 9, the cleric has already learned some effective spells after only 50 fights. Unlike the wizard weights, some spells (banishment) that appeared effective in early rounds will not appear in the scripts created after later rounds.

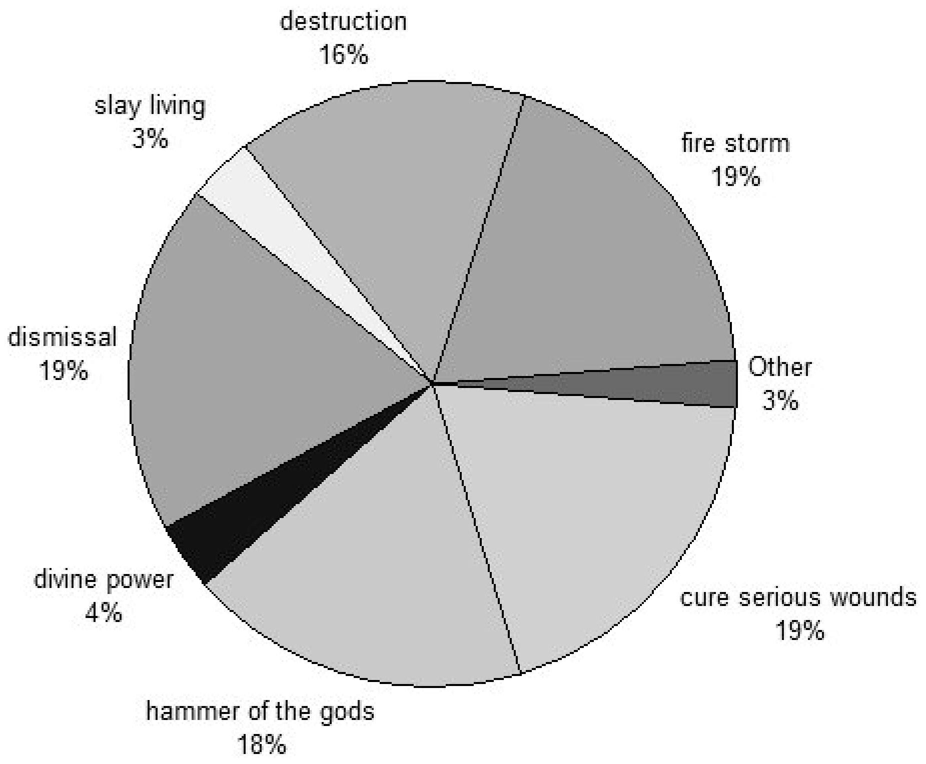


Figure 14: Cleric spell weights after 250 fights in FFCW v FFCW

The cleric scripts created by dynamic scripting varied more between fights than those of the wizard.

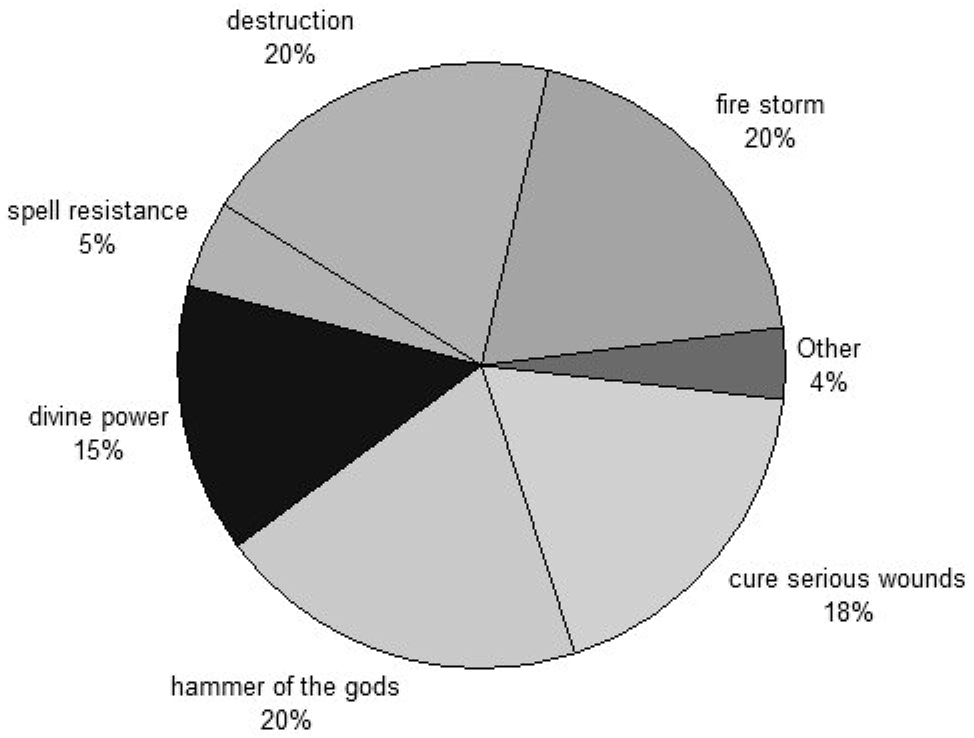


Figure 15: Cleric spell weights after 1000 fights in FFCW v FFCW

After 1000 fights, only three out of five spells that appeared to be most effective after 50 fights remain.

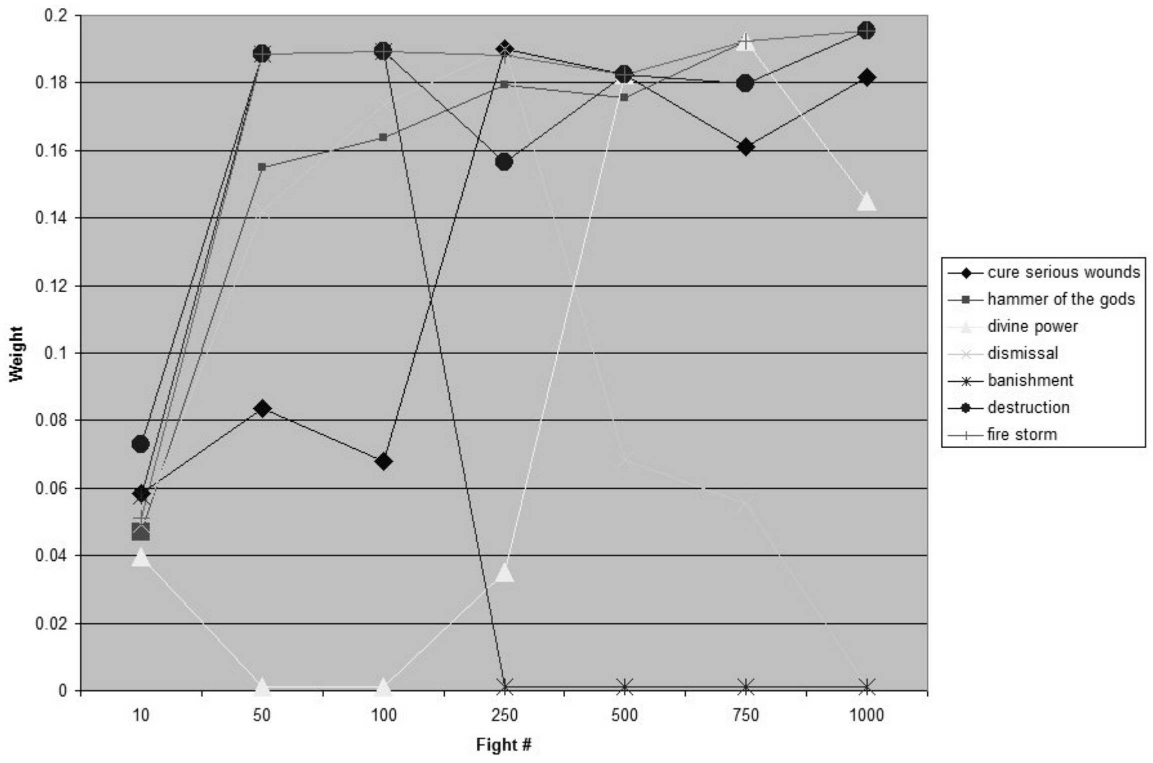


Figure 16: Selected weights for C Spells (FFCW v FFCW)

There is a lot more movement in the weights of the cleric spells which resulted in more variation in the created scripts than the wizards in the same fights. See Figure 12 for a comparison, where there is not as much movement as in Figure 16, which has some spell weights that change from very high to very low or vice versa throughout the experiment.

These results are important since they show that even though no hand-coding of weights were used to signify importance of the spells, effective weights were learned and more-powerful spells ended up with higher weights than their lesser-powerful counterparts. The spell casters were able to determine which spells were effective and learned strategies that were successful on their own, even when a poor selection of spells resulted in a loss. Also of importance is to note that a spell that is most effective in one scenario is not necessarily going to be as effective when there are other teammates or opponents. Those results are demonstrated in the trials below.

It is worth noting that spells like Banishment and Dismissal (both of which instantly get rid of summoned creatures on the opposing team) had high weights during the first half of the fights, since these spells are very effective against summoned creatures. These weights were so effective that the opponent teams weights for the summon creature spells were driven to the minimum weights. After opponents stopped trying to summon creatures, the weights for Banishment and Dismissal dropped (from dynamic scripting reacting to changes in opponents' strategies).

Next we removed the C from each team and replaced it with another F, leading to a situation where there is only 1 spellcaster on each team, and no ability for the characters to heal themselves. Some of the damaging spells are similar, and a defensive spell that helps against physical damage (stoneskin) has a higher weight. The results of these fights appear in the table below:

after fight:	10	50	100	250	500	750	1000
daze	0	0	0	0	0	0	0
light	0	0	0	0	0	0	0
ray of frost	0.0266	0.0201	0.0441	0.0684	0.0987	0.0585	0.001
resistance	0.0292	0.0011	0.0013	0.0011	0.0023	0.0325	0.0058
mage armor	0.0482	0.0011	0.0013	0.0424	0.0364	0.0011	0.001
summon creature 1	0.0096	0.0251	0.0011	0.0011	0.0023	0.0011	0.001
magic missile	0.0284	0.0085	0.0013	0.0011	0.0023	0.0572	0.0388
identify	0	0	0	0	0	0	0
sleep	0	0	0	0	0	0	0
ghostly visage	0.0372	0.0481	0.0779	0.0444	0.0312	0.0023	0.001
melf's acid arrow	0.0152	0.0011	0.0011	0.0346	0.0356	0.0011	0.001
summon creature 2	0.0109	0.0011	0.0013	0.0011	0.0012	0.0011	0.001
knock	0	0	0	0	0	0	0
web (WAS invisibility)	0.0252	0.0049	0.0013	0.0011	0.0214	0.0011	0.001
fireball	0.0284	0.0437	0.0566	0.0458	0.0332	0.0548	0.0649
clarity	0.0254	0.031	0.029	0.0673	0.0143	0.0255	0.0417
summon creature 3	0.0315	0.0014	0.0011	0.0011	0.0023	0.0011	0.001
haste	0.0298	0.0369	0.0698	0.0144	0.0364	0.0606	0.001
dispel magic	0.0576	0.0396	0.0571	0.0925	0.0801	0.0778	0.0666
stoneskin	0.0403	0.0032	0.046	0.0687	0.0367	0.0011	0.0432
elemental shield	0.0434	0.0892	0.0449	0.0011	0.0223	0.0581	0.001
evard's black tentacles	0.0638	0.0766	0.1108	0.0466	0.0346	0.0643	0.0607
ice storm	0.0617	0.0808	0.1219	0.1025	0.1029	0.058	0.1223
minor globe of invulnerability	0.0666	0.006	0.0041	0.0011	0.0663	0.0011	0.001
lesser spell mantle	0.0645	0.0905	0.0429	0.0537	0.046	0.0441	0.001
summon creature 5	0.0699	0.0818	0.0011	0.0011	0.0023	0.0011	0.001
cloudkill	0.05	0.0107	0.0014	0.0452	0.0142	0.0011	0.0579
greater shadow conjurations	0	0	0	0	0	0	0
chain lightning	0.0129	0.0547	0.0465	0.0613	0.0668	0.0977	0.1185
greater spell breach	0.0064	0.0448	0.0407	0.0011	0.0154	0.0011	0.0647
summon creature 6	0.0295	0.054	0.0339	0.0612	0.0653	0.0011	0.001
mordenkainen's sword	0.0039	0.0022	0.001	0.032	0.0325	0.1001	0.1229
spell mantle	0.0401	0.0824	0.0458	0.0011	0.0023	0.0848	0.0578
horrid wilting	0.0428	0.0585	0.1135	0.1063	0.0937	0.109	0.1186
melee	0.001	0.001	0.001	0.001	0.001	0.001	0.001

Table 10: Spell weights for Wizard in FFFW v FFFW

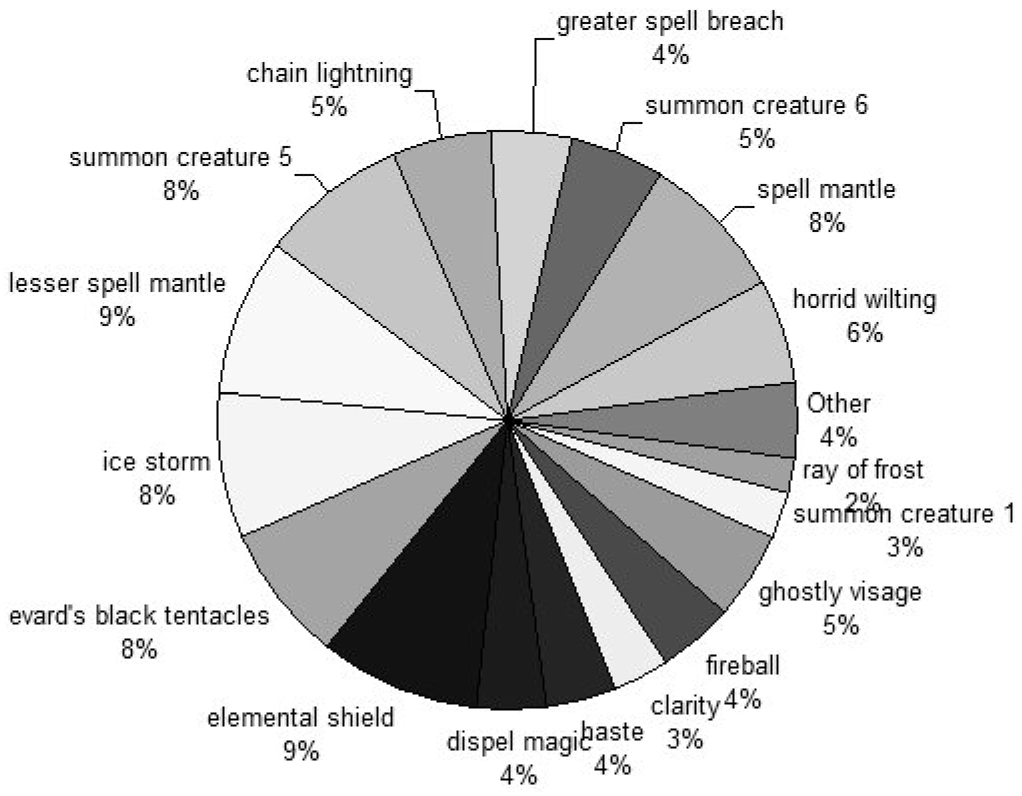


Figure 17: Wizard spell weights after 50 fights in FFFW v FFFW

After 50 fights, there is much more variation in the script created by dynamic scripting for this wizard compared to the one from FFCW v FFCW in Figure 9.

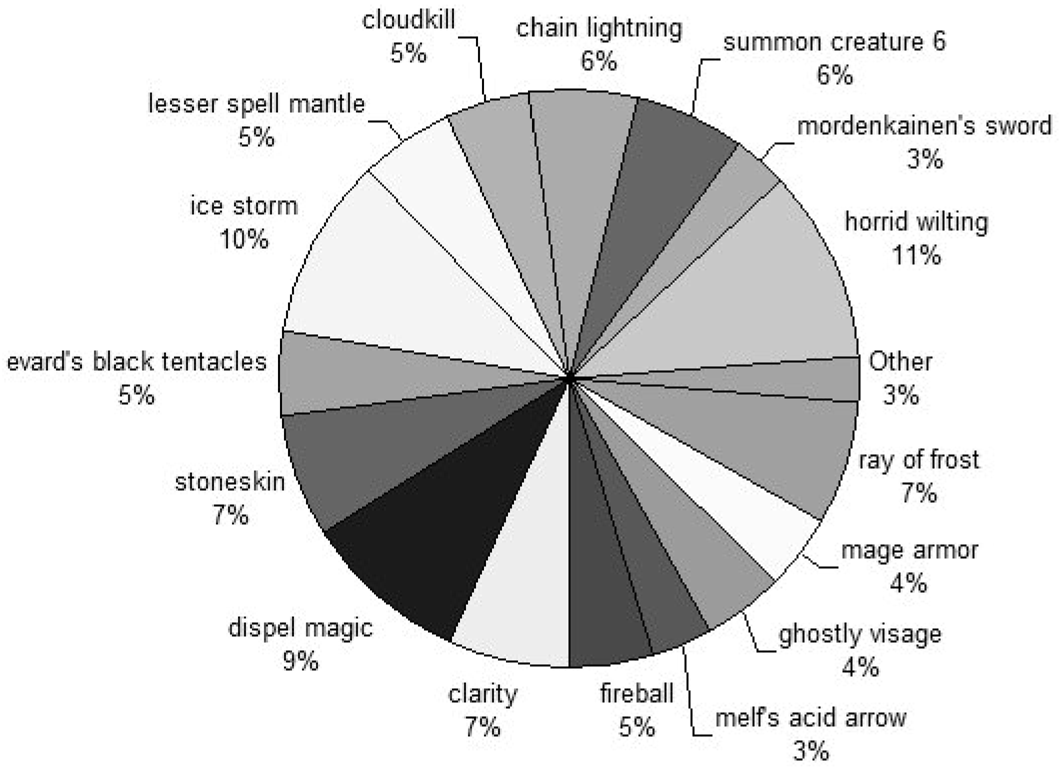


Figure 18: Wizard spell weights after 250 fights in FFFW v FFFW

Some of the spells that were effective in the FFCW v FFCW fights are also effective here, but others (like dispel magic) will appear in these scripts but not others.

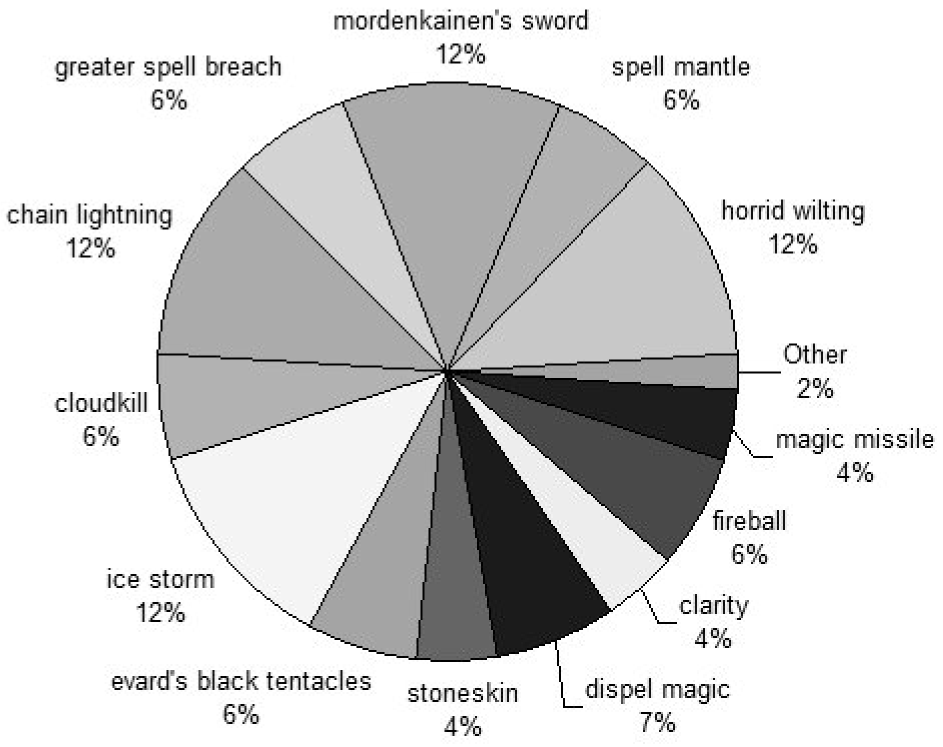


Figure 19: Spell weights after 1000 fights in FFFW v FFFW

Compared to Figure 11, the weights learned and scripts generated by this wizard are significantly different than those learned by the wizard in FFCW v FFCW. Some of the most effective spells are identical and will appear in the wizard's scripts for both fights, but there are also quite a few new spells in these scripts that would appear in FFFW v FFFW but not FFCW v FFCW.

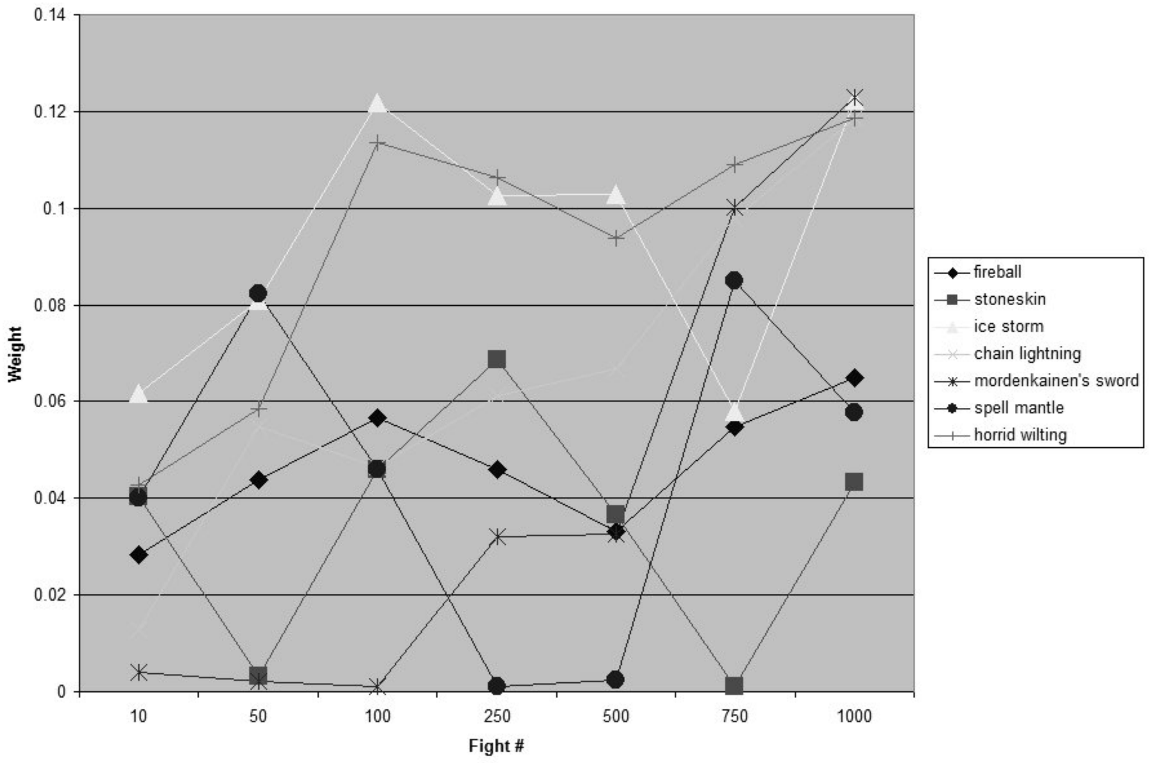


Figure 20: Selected weights for W spells (FFFW v FFFW)

The weights for the spells were much different than those of the wizard from Figure 12, showing that a small change in the composition of the team can have a big difference in the spells that appear in the scripts created by dynamic scripting.

After the most effective scripts were learned in the text-based arena, the scripts were used in the NWN arena to verify their effectiveness.

Spells that received a high weight after one training run had similar weights after similar training runs, and some spells were effective in many situations and had high weights after various training runs. The important concept to highlight is some spells were very effective in some situations, but less effective in others. Table 11 shows the weights of the "Ice Storm" spell. "Ice Storm" is a W spell that damages the opponents on the other team. The "Team" and "Opponent" columns show the agents that were in the combat. This type of variation would be hard to reproduce if the weights were all hand-coded. The people hand-coding the weights would have to enter a value for every spell for different combinations of teams that were fighting, which introduces the chance of a game-designer introducing an error.

Team	**Opponents**	**Weight**
FFCW	FFCW	19.6%
FFFW	FFFW	15.7%
CCWW	FCWW	4.2%

Table 11: Selected resultant weights of the "Ice Storm" spell

In cases where one team lost a significant proportion of the fights, the weights were more varied. This is to be expected, since if a team is significantly inferior, dynamic scripting will continue to attempt to find different strategies to help the team win.

Table 12 contains the results from the FFCW v FFCW fights. The first row contains the results of 100 fights with both teams using the default logic that came with the NWN module. The results do not add up to 100 since there were a few ‘ties’, where the last player on each team died simultaneously. The second row contains the results of having the Wizard on each team used the weights learned from the text-based version. The third row contains the results when the Wizard and Cleric on each team used the weights from the text-based version. The final two rows one team used the default logic and the other used the weights from the text based versions, and for those two rows we can say with a 95% confidence level that the strategy learned from the text-based version performed better than the team using the default logic. The actual weights for the Wizard are in Table 3, and the weights for the Cleric are in Table 4.

	FFCW	FFCW
	White (W) Wins	Black (B) Wins
Default Logic - Both Teams	49	46
Wizard using synthesized starting point - Both Teams	55	41
Wizard and Cleric using synthesized starting point - Both Teams	54	41
White Team using text-based Wizard. Black Team using Default Logic.	67	28
White Team using text-based Wizard and Cleric. Black Team using Default Logic.	64	34

Table 12: Results from FFCW v FFCW

Table 13 shows the results from FFFW v FFFW. When the Wizard used the weights from Table 5, it performed significantly better than by using the default-logic strategy. These results highlight the fact that hand-coded values are not necessarily better (as we can state with a 95% level of confidence that the strategy learned from dynamic scripting significantly outperforms the strategy that shipped with the game), and that following one strategy for all opponents is not always effective, as the default strategy had various levels of effectiveness against different teams.

	FFFW	FFFW
	White (W) Wins	Black (B) Wins
Default Logic - Both Teams	57	41
Wizard using text-based starting point - Both Teams	53	35
White Team using text-based Wizard. Black Team using Default Logic.	97	3

Table 13: Results from FFFW v FFFW

The above two tables show results from balance teams fighting each other. In many situations, one team might be stronger than the other. In such a case, dynamic scripting may continue to try different strategies to come up with one that is effective.
In the following table, the rules for the "Dynamic Scripting Starting Point" were synthesized using the method explained in Table 2. The results below show that the scripts created from the dynamic scripting method performed better than the scripts created by the game designers of NWN.

CCWW	% Wins	FFCW Remaining Hit Points
Default Starting Point	3.4	116.52
Dynamic Scripting Starting Point	7.8	69.64

Table 14: Results from CCWW v FFCW

The complete results are in Appendix D.

To test the validity of using roles as an abstraction in these experiments, trials were run where a character on a team was switched with a different character type that can fulfill the same role, and then ensure the results were still applicable. The Sorcerer class is similar to the Wizard class, as they have the same spells to pick from. A major difference between the two classes is the number of spells they can cast daily, and whether or not they need to prepare them at the beginning of the day. A level 15 Sorcerer can cast 6 spells from each of levels 1-6, and 4 level 7 spells. These spells can be picked right before they are cast from a list of spells the sorcerer knows. A level 15 Wizard can cast 4 spells from each of levels 1-5, 3 level 6 spells, 2 level 7 spells, and 1 level 8 spell. These spells must be picked at the beginning from the spells the wizard knows.

The complete list of spells for the two classes is listed in Appendix F.

The following script was created for the W in FFFW v FFFW.

```
   if (GetHasSpell( SPELL_ICE_STORM ))
   {…
      ActionCastSpellAtLocation( SPELL_ICE_STORM, locTarget );
…}

   if (GetHasSpell( SPELL_CHAIN_LIGHTNING ))
   {…
      ActionCastSpellAtObject( SPELL_CHAIN_LIGHTNING, oEnemy );
…}

   if (GetHasSpell( SPELL_HORRID_WILTING ))
   {…
      ActionCastSpellAtLocation( SPELL_HORRID_WILTING, locTarget );
…}

   if (GetHasSpell( SPELL_MORDENKAINENS_SWORD ))
   {…
      ActionCastSpellAtLocation( SPELL_MORDENKAINENS_SWORD, locTarget );
…}

   if (GetHasSpell( SPELL_EVARDS_BLACK_TENTACLES ))
   {…
      ActionCastSpellAtLocation( SPELL_EVARDS_BLACK_TENTACLES, locTarget
);
…}
```

Which would result in the following spells being cast by the two different characters:
Wizard:
Round 1: Ice Storm
Round 2: Chain Lightning
Round 3: Horrid Wilting
Round 4: Mordenkainen’s Sword
Round 5: Evard’s Black Tentacles

Sorcerer:

Rounds 1-6: Ice Storm	(Sorcerer can cast 6 level 4 spells)
Rounds 7-12: Chain Lightning	(Sorcerer can cast 6 level 6 spells)
Rounds 13-17: Mordenkainen’s Sword	(Sorcerer cannot cast 8th level spell, so skips Horrid Wilting, goes right to Mordenkainen’s Sword. Also will skip Evard’s Black Tentacles, as it is a level 4 spell, and Sorcerer has already cast all level 4 spells in rounds 1-6.)

Even though there is a lot of overlap in the spells available to these two classes, it is apparent that the same script will produce different results for these character classes.

	FFFW	FFFW
	White (W) Wins	Black (B) Wins
Default Logic - Both Teams	57	41
Wizard using text-based starting point - Both Teams	53	35
White Team using text-based Wizard. Black Team using Default Logic.	97	3
	FFFS	FFFS
	White (W) Wins	Black (B) Wins
Default Logic - Both Teams	46	54
Wizard using text-based starting point - Both Teams	51	45
White Team using text-based Wizard. Black Team using Default Logic.	94	6

Table 15: Results from FFFW v FFFW compared to FFFS v FFFS

In both cases, we can state with a 95% level of confidence that the strategy learned from dynamic scripting performed better than the default logic that was shipped with the game. The fact that the script performed well with a Sorcerer instead of a Wizard demonstrates that the abstraction based on roles is a valid one, since the script was created in the text-only arena that did not have the Sorcerer class.

Discussion

It is important to note that this is not a case of “more knowledge leads to better decisions”. The logic in the NWN arena module takes thousands of lines of code.
The scripts produced by this method are under a hundred lines. This method replaces thousands of lines of code written by experts, and replaces it with scripts proven to be effective. By removing the complex scripts generated by hand, the chance of a bug existing in a script is minimized, and allowing adaptation of the scripts through changing the values of the weights further removes the chance of an exploit being used.

The rules that were implemented as scripts in the dynamic scripting version used as little domain-specific knowledge as possible, to ensure the results achieved were the result of the dynamic scripting process instead of domain knowledge. For example: The knowledge that a spell that banished a summoned creature could be cast only if there was a summoned creature on the opposing team was encoded, but the knowledge that the spell “Summon Creature VI” was better than “Summon Creature I” because it summoned a more-powerful creature was not encoded. In games like NWN, this knowledge is encoded, since it is often the case that you would want an agent to cast the most-powerful spell available to it when in a certain situation. In most cases, the higher-level spells are more effective, so they would end up with a higher weight than the lower-level versions after training. There were few instances where a lower-level version of a spell had a slightly higher weight after some training runs– this often was not a significant difference, and was not unexpected given the stochastic nature of dynamic scripting. Adding domain-specific knowledge will not lessen the effectiveness of dynamic scripting finding better strategies, and would lessen the chance of the above happening.

The long, hand-coded logic can contain errors, either due to the complexity, or requirements not completely understood by the programmers. The following bits of code were found in the NWN module. More detailed excerpts are found in Appendix E. The next few lines contain a comment that shows the developers were not confident in some of the constants that were used.

```
// No idea if these distances are correct
const float DISTANCE_COLOSSAL = 15.0;
const float DISTANCE_HUGE = 10.0;
<several more lines>
```

The next few lines of code contain logic to classify creatures. This is similar to the work done in the discussion of roles in chapter 3. This code shows that Clerics and Druids are both classified under the abstraction "Priests".

```
// Returns TRUE if oCreature is a priest.
int IsPriest( object oCreature )
{
   if (HasClass( oCreature, CLASS_TYPE_CLERIC ) ||
      HasClass( oCreature, CLASS_TYPE_DRUID ))
      return TRUE;

   return FALSE;
}
```

The following lines of code contain several parts with complex reasoning to pick the best spell for a certain situation. The code shows many checks being performed before a summon spell is cast, and then various summoning spells ranked in descending order of effectiveness.

```
   if (GetHasSpell( SPELL_SUMMON_CREATURE_IV ))
   {
…
      ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_IV, locTarget );
…
   }

   if (GetHasSpell( SPELL_SUMMON_CREATURE_III ))
   {
…
      ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_III, locTarget );
…
   }

   if (GetHasSpell( SPELL_SUMMON_CREATURE_II ))
   {
…
      ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_II, locTarget );
…
   }

   if (GetHasSpell( SPELL_SUMMON_CREATURE_I ))
```

```
    {
…
        ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_I, locTarget );
…
    }
```

As stated earlier, this knowledge is introduced by a programmer who knows which summon spells are more effective than others. Although it could be determined analytically which of these spells are expected to be more effective, there is a chance that a coding error makes one spell more or less effective than it should be.

There are also lots of places in the code where there is lots of logic built in to take care of special cases are coded which are only used in specific circumstances. This introduces complexity which can make debugging difficult in certain situations. The following code shows how many checks are done before certain spells are cast. The complete section contains comments that mentions some spells, including invisibility, still need to be coded.

```
   // HIGH Priority rules. These rules are fairly specific, but can be used
   // in general at any time during combat. They are executed in specific
   // circumstances or use talents in a specific way, to ensure that
   // the talents a creature has, when they are employed, are employed as
   // effectively as possible.

<several lines removed for clarity>

          case 1:
             if (IsMage( oCreature ) ||
                IsPriest( oCreature ) ||
                HasClass( oCreature, CLASS_TYPE_UNDEAD ) ||
                HasClass( oCreature, CLASS_TYPE_OUTSIDER ) ||
                HasClass( oCreature, CLASS_TYPE_CONSTRUCT ) ||
                HasClass( oCreature, CLASS_TYPE_ELEMENTAL ) ||
                IsGeneralClass( oCreature ))
             {
                if (iJustCheck)
                   return TRUE;
                sRuleText = "CastHighSummon(oNearest)";
                return CastHighSummon( oNearest );
             }
             break;

<several lines removed for clarity>
```

The invisibility example was highlighted as there is a documented error where the two teams will stop fighting each other if members of both teams turn invisible. The following bit of code was found in the NWN module, presumably added to try and correct the error,

```
if (iSomeoneVisible)
   return;

// Since everyone is invisible, Igor will remove the effects from everyone.

 …
```

```
SpeakString( "You are unable to see each other, ladies. Let me correct that." );

 …
while (GetIsObjectValid( oWhite ))
{
   RemoveSpecificEffect( EFFECT_TYPE_INVISIBILITY, oWhite );
 …
}
 …
while (GetIsObjectValid( oBlack ))
{
   RemoveSpecificEffect( EFFECT_TYPE_INVISIBILITY, oBlack );
 …
}
```

The following bits of code show how multiple checks are done to only cast spells that will be effective in certain situations. You can see that before a spell is cast, the opponent is checked to make sure it is not immune, protected from the spell, or even has a certain amount of hit points left.

```
// Tries to cast a cursing spell as high as possible at a single enemy
…
   if (GetHasSpell( SPELL_ENERGY_DRAIN ) &&
      !GetIsImmune( oEnemy, IMMUNITY_TYPE_DEATH ) &&
      !ProtectedAgainstSpells( oEnemy, 9 ))
   {
…
    ActionCastSpellAtObject( SPELL_ENERGY_DRAIN, oEnemy );
…
   }

   if (GetHasSpell( SPELL_POWER_WORD_STUN ) &&
      GetCurrentHitPoints( oEnemy ) <= 150 &&
      !GetIsImmune( oEnemy, IMMUNITY_TYPE_STUN ) &&
      !GetIsImmune( oEnemy, IMMUNITY_TYPE_MIND_SPELLS ) &&
      !ProtectedAgainstSpells( oEnemy, 7 ))
   {
…
      ActionCastSpellAtObject( SPELL_POWER_WORD_STUN, oEnemy );
…
   }

…
```

The amount of checking in the previous sample might be considered 'cheating' by some players – especially if there would be no way for the agent to know the status of the opponent. This can be contrasted to the code snippet below that is effective and does not use any kind of knowledge of the status of the opponents when deciding the spell to cast.

The code samples above contain hundreds of lines of logic manually created by programmers. For contrast, the script below was produced using dynamic scripting for the W character after 1000 fights in FFCW v FFCW(see Figure 11).

```
switch ( GetLastSpellCastClass() )
{
 case 10: //wizard

  if (GetHasSpell( SPELL_HORRID_WILTING ))
  {
…
     ActionCastSpellAtLocation( SPELL_HORRID_WILTING, locTarget );
…
  }

  if (GetHasSpell( SPELL_CHAIN_LIGHTNING ))
  {
…
     ActionCastSpellAtObject( SPELL_CHAIN_LIGHTNING, oEnemy );
…
  }

  if (GetHasSpell( SPELL_ICE_STORM ))
  {
…
     ActionCastSpellAtLocation( SPELL_ICE_STORM, locTarget );
…
  }

  if (GetHasSpell( SPELL_EVARDS_BLACK_TENTACLES ))
  {
…
     ActionCastSpellAtLocation( SPELL_EVARDS_BLACK_TENTACLES, locTarget );
…
  }

  if (GetHasSpell( SPELL_MELFS_ACID_ARROW ))
  {
…
     ActionCastSpellAtObject( SPELL_MELFS_ACID_ARROW, oEnemy );
…
  }


  break;

  }//end switch
```

The results show that the scripts are effective, and straightforward to create. In addition to being effective, the scripts are easy to understand. The ordering of the spells directly relate to the weights as spells with a low weight are considered bad and ones with a high weight are beneficial in the combat situation. Other methods like complex scripts or neural networks do not have that benefit.

Chapter 5: Conclusions, Future Work, and Limitations

In the previous chapters we explained dynamic scripting and one of its shortcomings – how it does not take into account the opponents and teammates when starting off, which can lead it to make decisions that may appear unintelligent while learning an effective strategy. We showed how knowledge of the other agents can be used to create a better starting point, which can be used to learn an effective strategy faster. We showed how these newer starting points can be created even if the exact combinations of teammates or opponents have never been used before. This research resulted in situations where the starting weights were different based on the agents participating, and prevented the situation where a spell that was effective in most cases was cast in a situation where it was not likely to be effective. The case study we used was a game that allows thousands of variations when creating a character and millions of possible teams, and we showed how to reduce this into a manageable size, through abstraction and the use of roles.

We also included a difficult case where a team that used a script that was created using dynamic scripting performed better than a team using logic programmed by the game designers. We also showed specific instances where scripts that were hand-coded by experts were complex and contained errors, and showed how dynamic scripting was able to produce easy-to-understand scripts that allowed for an implementation error to be easily uncovered.

This research can help ensure balance before shipping a game. Any character types that are inadvertently created with too much power that can create imbalances can be determined and fixed. A character that can determine a dominant strategy against all other opponent types could be detected using the procedure followed in this work, and could be modified before the game is released.

Additional work can be done in the area of non-combat roles. The preceding experiments focused on combat situations. In many games, there are also many elements of non-combat situations that could be incorporated into this type of dynamic scripting. For instance, if a player needed to get into a locked room, picking a lock or stealing a key from the guard may be options for the player. Accounting for these actions could be done by including tasks that use these non-combat actions and making appropriate changes to the fitness function. For example, a character could have “pick lock” or “steal key from guard” in addition to “attack” guards, so script weights could include non-combat actions. Ultimately these scenarios would be game-specific, but creating a procedure for handling these types of actions would be a useful extension to this research.

This work is limited to games with teams of heterogeneous agents that are controlled by scripts. Dynamic scripting can adapt to changes in opponent’s behaviour, but it does not keep track of effective strategies. If a player changes a strategy, the dynamic scripting opponent will respond by learning an appropriate response. If the player subsequently reverts to an older strategy, dynamic scripting has to re-learn an effective response, as opposed to “remembering” the strategy it used before. How to handle not forgetting earlier lessons is out of scope of this research.

This research can enable many improvements in commercial games. In addition to replacing the need for manual ‘tweaking’ of parameters, it can create tougher opponents for the player to defeat by creating teams that behave intelligently to challenge the player without the need to resort to cheating. Implementing this research can also improve a player’s experience with a game by providing better teammates. This can be accomplished in two ways, either by providing teammates that complement and work well with the player’s character type, or by customizing the actions of the teammates so they provide better actions based on the player’s character type. In a game where a companion or teammates are not fixed, teammates that can provide actions that best complement the player’s character type can be provided. In the experiments performed for this research, it was discovered that a group that was entirely composed of wizards did not perform well compared to a group of wizards with a fighter. If the player was controlling a wizard, the game could provide a fighter as a companion instead of another wizard. In a game where the teammates are fixed, the scripts that the teammates follow could vary depending on the composition of the team, resulting in more effective rules. We saw from the examples provided earlier that the scripts for the most effective spells a wizard could cast varied depending on the teammates, so customizing the actions of the teammates based on the player’s character would improve the effectiveness of the player’s team. All of these results show that taking other agents into consideration will allow an individual agent using dynamic scripting to perform better than current scripting implementations.

Appendix A: Rulebase

Here are the rulebases used by the characters during the dynamic scripting algorithm to produce the strategies. The spells in the Wizard and Cleric rulebases were the default ones that were selected when those characters were created in NWN.

Fighter:
Move into melee combat
Attack opponent with sword
Do nothing

Wizard:
Move into melee combat
Attack opponent with dagger

(Level 0 spells)
Cast daze (penalizes low-level opponents)
Cast light (lights up the area – removes penalties from darkness)
Cast ray of frost (can damage 1 opponent for 1-3 hit points)
Cast resistance (helps resist opponent spells for 10 rounds)

(Level 1 spells)
Cast mage armor (makes the agent harder to hit)
Cast summon creature 1 (summons a weak ally to help the caster)
Cast magic missile (damages 1 opponent for 10-25 hit points)
Cast identify (identifies a magical item)
Cast sleep (puts low-level opponents to sleep)

(Level 2 spells)
Cast ghostly visage (reduces the amount of damage the caster takes)
Cast melf's acid arrow (can damage 1 opponent for 2-8 hit points, plus damage for several more rounds)
Cast summon creature 2 (summons an ally to help the caster)
Cast knock (unlocks doors)
Cast web (WAS invisibility) (creates webs to hinder opponents)

(Level 3 spells)
Cast fireball (can damage all opponents for 10-60 hit points)
Cast clarity (removes the effects of daze, confusion, stun… spells)
Cast summon creature 3 (summons an ally to help the caster)
Cast haste (lets an ally have twice the amount of attacks for 15 rounds)
Cast dispel magic (removes magical effects)

(Level 4 spells)
Cast stoneskin (absorbs physical damage)

Cast elemental shield (damages opponents when they hit the caster, plus absorbs some damage from offensive spells)
Cast evard's black tentacles (creates allies that grab and hit opponents)
Cast ice storm (damages all opponents for 5-30 damage)
Cast minor globe of invulnerability (gives protection from all spells under 4th level)

(Level 5 spells)
Cast lesser spell mantle (completely absorbs damage from 7-10 spell levels)
Cast summon creature 5 (summons an ally to help the caster)
Cast cloudkill (damages opponents for 1-10 hit points)
Cast greater shadow conjurations (creates semi-illusionary objects)

(Level 6 spells)
Cast chain lightning (can damage one opponent for 15-90 hit points, and damages other opponents)
Cast greater spell breach (removes magical defences from opponents)
Cast summon creature 6 (summons a powerful ally to help the caster)

(Level 7 spells)
Cast mordenkainen's sword (summons a magical sword to attack enemies of the caster)
Cast spell mantle (completely absorbs damage from 9-16 spell levels)

(Level 8 spell)
Cast horrid wilting (damages opponents for 15-90 hit points)

Cleric:
Move into melee combat
Attack opponent with mace

(Level 0 spells)
Cast inflict minor wounds (can damage an opponent for 1 hit point)

(Level 1 spells)
Cast cure light wounds (heals an ally for 6-13 hit points)
Cast inflict light wounds (can damage an opponent for 6-13 hit points)
Cast bless (gives allies bonuses to hit and damage opponents, and to avoid damaging spells)

(Level 2 spells)
Cast cure moderate wounds (heals an ally for 12-26 hit points)
Cast inflict moderate wounds (can damage an opponent for 12-26 hit points)
Cast aid (gives one ally bonuses to hit and damage opponents, and to avoid damaging spells, and gives 1-8 additional temporary hit points)

(Level 3 spells)
Cast cure serious wounds (heals an ally for 18-39 hit points)

Cast inflict serious wounds (can damage an opponent for 18-39 hit points)
Cast remove blindness/deafness (removes blindess and deafness from a blind or deaf ally, which are conditions that give penalties to many tasks they attempt)
Cast remove disease (removes disease and associated penalties from a diseased ally)
Cast dispel magic (removes magical effects)
Cast animate dead (creates an undead ally to attack opponents)

(Level 4 spells)
Cast hammer of the gods (can damage opponents for 5-40 damage, and possibly daze them for 1-6 rounds)
Cast inflict critical wounds (can damage an opponent for 19-47 hit points)
Cast freedom of movement (gets rid of negative effects of daze, being stuck in webs)
Cast cure critical wounds (heals an ally for 19-47 hit points)
Cast divine power (gives the caster an attack bonus, strength bonus, and 15 extra hit points)
Cast dismissal (can instantly get rid of a summoned creature)

(Level 5 spells)
Cast heal (restores all hit points to an ally)
Cast slay living (has a chance to instantly kill an opponent, or do 18-33 hit points of damage)
Cast flame strike (can damage opponents for 15-90 hit points)
Cast circle of doom (has a chance of doing 23-79 hit points of damage to and opponent)
Cast spell resistance (gives complete resistance to some low-level spells – magic missile, fireball, ice storm)

(Level 6 spells)
Cast harm (can remove all BUT 1-4 hit points from an opponent)
Cast blade barrier (can damage opponents in an area for 15-90 hit points)
Cast banishment (can instantly get rid of all summoned creatures)
Cast control undead (can take control of an opponent's summoned undead creature)

(Level 7 spells)
Cast destruction (can instantly kill an opponent, or do 10-60 damage)
Cast greater restoration (restores lost abilities to an ally. Takes a long time to cast)
Cast regenerate (increases healing rate of an ally, healing 16-23 hit points)

(Level 8 spells)
Cast fire storm (can damage all opponents for 15-90 hit points)
Cast earthquake (can damage all opponents for 8-48 hit points)

Appendix B: Synthesis of Starting Rulebases

Once all of the rules were encoded, the initial weights were set to equal amounts, and thousands of fights happened using dynamic scripting between different teams (FWWW v FWWW, FFCW v FFCW, FFFW v FFFW). After this occurred, weights that consistently had low weights in all situations were removed from future experiments. These spells are listed in Appendix C, and are generally spells that have no combat usefulness (spells that created a light source, or identified an item are two examples). After those spells were removed, the weights were reset to be equal, and the tests were run again. Some spells that caused a lot of damage were often found useful against an opponent regardless of who the opponent was or the teammates.

When a new team started a fight, rules from ‘similar’ teams (as described in Chapter 3) were used to create the starting weights. The table below shows the starting weights for the Wizard in FFCW v CCWW, from the weights of the Wizards from FFCW v FFCW and FCWW v CCWW.

team	FFCW	FCWW	FFCW
opponent	FFCW	CCWW	CCWW
character	W	W	W
classification	YYNYYY	YYYNYY	YYNNYY
daze	0.0000	0.0000	0.0000
light	0.0000	0.0000	0.0000
ray of frost	0.0011	0.0011	0.0011
resistance	0.0123	0.0011	0.0067
mage armor	0.0011	0.0011	0.0011
summon creature 1	0.0011	0.0011	0.0011
magic missile	0.0011	0.0011	0.0011
identify	0.0000	0.0000	0.0000
sleep	0.0000	0.0000	0.0000
ghostly visage	0.0011	0.0011	0.0011
melf's acid arrow	0.1797	0.0011	0.0904
summon creature 2	0.0011	0.0011	0.0011
knock	0.0000	0.0000	0.0000
web (WAS invisibility)	0.0011	0.0313	0.0162
fireball	0.0011	0.0011	0.0011
clarity	0.0198	0.0011	0.0105
summon creature 3	0.0011	0.0011	0.0011
haste	0.0011	0.0011	0.0011
dispel magic	0.0011	0.0011	0.0011
stoneskin	0.0011	0.0011	0.0011
elemental shield	0.0011	0.0011	0.0011
evard's black tentacles	0.1797	0.0011	0.0904
ice storm	0.1950	0.1988	0.1969
minor globe of invulnerability	0.0011	0.0011	0.0011
lesser spell mantle	0.0011	0.0011	0.0011
summon creature 5	0.0011	0.0011	0.0011
cloudkill	0.0011	0.0011	0.0011
greater shadow conjurations	0.0000	0.0000	0.0000
chain lightning	0.1950	0.0011	0.0980
greater spell breach	0.0011	0.0101	0.0056
summon creature 6	0.0011	0.0335	0.0173
mordenkainen's sword	0.0011	0.0011	0.0011
spell mantle	0.0011	0.0011	0.0011
horrid wilting	0.1950	0.1999	0.1974
melee	0.0011	0.0011	0.0011

Table 16: Synthesis of starting weights for W in FFCW v CCWW

Appendix C: Results

Text-only version

The results below are selected iterations from the text-only version of the arena. These results are from after an initial round where some spells were removed from consideration, as they were all determined to not be useful for combat in the arena. The spells that were removed were:

- Wizard:
  - Daze (this spell is always ineffective due to the level of the agents)
  - Light (not useful in combat in the arena)
  - Identify (not a combat spell)
  - Sleep (this spell is always ineffective due to the level of the agents)
  - Knock (not a combat spell)
  - Greater Shadow Conjurations
- Cleric:
  - Remove blindness/deafness (not useful in these fights)
  - Remove disease (not useful in these fights)
  - Greater restoration (not a combat spell)

Wizard from FFCW v FFCW

Weights after 10 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0-3)
0, //daze
0, //light
0.0126410614387192, //ray of frost
0.0774395508804581, //resistance
//level 1 (4-8)
0.0126410614387192, //mage armor
0.0015398815240403, //summon creature 1
0.0820835504455086, //6=magic missile
0, //identify
0, //sleep
//level 2 (9-13)
0.0126410614387192, //ghostly visage
0.0844234160732066, //10=melf's acid arrow
0.00231099421337619, //summon creature 2
0, //knock
0.00735843865551269, //web (WAS invisibility)
//level 3 (14-18)
0.00308210690271207, //14=fireball
0.00782235170446284, //clarity
0.0267680180061381, //summon creature 3
0.0335191258796749, //haste
0.00628218067375542, //dispel magic
//level 4 (19-23)
0.0604552316188311, //19=stoneskin
0.0836117138838773, //elemental shield
0.087654558121514, //evard's black tentacles
0.0884660691495351, //22=ice storm
0.0126410614387192, //minor globe of invulnerability
//level 5 (24-27)
0.0263950328877631, //lesser spell mantle
0.0126410614387192, //summon creature 5
0.0693708382053159, //cloudkill
0, //greater shadow conjurations
//level 6 (28-30)
0.0530588354652143, //chain lightning
0.0067917364809922, //greater spell breach
0.00545482388420026, //summon creature 6
//level 7 (31-32)
0.0122554431508828, //mordenkainen's sword
0.0238857012555003, //spell mantle
//level 8 (33)
0.0858796794046119, //horrid wilting
//misc
0.000885414339319928 //melee
};
```

Weights after 50 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0-3)
0, //daze
0, //light
0.00198312470065133, //ray of frost
0.031027030146624, //resistance
//level 1 (4-8)
0.00108025282022827, //mage armor
0.00108025282022827, //summon creature 1
0.0161271809524023, //6=magic missile
0, //identify
0, //sleep
//level 2 (9-13)
0.00108025282022827, //ghostly visage
0.115324045288633, //10=melf's acid arrow
0.00108025282022827, //summon creature 2
0, //knock
0.00108025282022827, //web (WAS invisibility)
//level 3 (14-18)
0.00186933327912724, //14=fireball
0.00473506759160179, //clarity
0.00108025282022827, //summon creature 3
0.00108025282022827, //haste
0.00108025282022827, //dispel magic
//level 4 (19-23)
0.00499686254807553, //19=stoneskin
0.0170583784801067, //elemental shield
0.196409603677867, //evard's black tentacles
0.196409603677867, //22=ice storm
0.00108025282022827, //minor globe of invulnerability
//level 5 (24-27)
0.00108025282022827, //lesser spell mantle
0.00108025282022827, //summon creature 5
0.0039565171776562, //cloudkill
0, //greater shadow conjurations
//level 6 (28-30)
0.196409603677867, //chain lightning
0.00108025282022827, //greater spell breach
0.00108025282022827, //summon creature 6
//level 7 (31-32)
0.00108025282022827, //mordenkainen's sword
0.00108025282022827, //spell mantle
//level 8 (33)
0.196409603677867, //horrid wilting
//misc
0.00108025282022827 //melee
};
```

Weights after 100 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0-3)
0, //daze
0, //light
0.0010962496281037, //ray of frost
0.0354735006048557, //resistance
//level 1 (4-8)
0.0010962496281037, //mage armor
0.0010962496281037, //summon creature 1
0.0010962496281037, //6=magic missile
0, //identify
0, //sleep
//level 2 (9-13)
0.0010962496281037, //ghostly visage
0.147393479601995, //10=melf's acid arrow
0.0010962496281037, //summon creature 2
0, //knock
0.0010962496281037, //web (WAS invisibility)
//level 3 (14-18)
0.0010962496281037, //14=fireball
0.00154336331651076, //clarity
0.0010962496281037, //summon creature 3
0.0010962496281037, //haste
0.0010962496281037, //dispel magic
//level 4 (19-23)
0.00162869361527689, //19=stoneskin
0.00637664717517924, //elemental shield
0.196370546078012, //evard's black tentacles
0.196370546078012, //22=ice storm
0.0010962496281037, //minor globe of invulnerability
//level 5 (24-27)
0.0010962496281037, //lesser spell mantle
0.0010962496281037, //summon creature 5
0.00128960006483745, //cloudkill
0, //greater shadow conjurations
//level 6 (28-30)
0.196370546078012, //chain lightning
0.0010962496281037, //greater spell breach
0.0010962496281037, //summon creature 6
//level 7 (31-32)
0.0010962496281037, //mordenkainen's sword
0.0010962496281037, //spell mantle
//level 8 (33)
0.196370546078012, //horrid wilting
//misc
0.00108003800342907 //melee
};
```

Weights after 250 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0-3)
0, //daze
0, //light
0.00110466263272181, //ray of frost
0.0553673068314011, //resistance
//level 1 (4-8)
0.00110466263272181, //mage armor
0.00110466263272181, //summon creature 1
0.00110466263272181, //6=magic missile
0, //identify
0, //sleep
//level 2 (9-13)
0.00110466263272181, //ghostly visage
0.168529172292573, //10=melf's acid arrow
0.00110466263272181, //summon creature 2
0, //knock
0.00110466263272181, //web (WAS invisibility)
//level 3 (14-18)
0.00110466263272181, //14=fireball
0.00110466263272181, //clarity
0.00110466263272181, //summon creature 3
0.00110466263272181, //haste
0.00110466263272181, //dispel magic
//level 4 (19-23)
0.00110466263272181, //19=stoneskin
0.0254844370284586, //elemental shield
0.198741434621282, //evard's black tentacles
0.134585823414377, //22=ice storm
0.00110466263272181, //minor globe of invulnerability
//level 5 (24-27)
0.00110466263272181, //lesser spell mantle
0.00110466263272181, //summon creature 5
0.00110466263272181, //cloudkill
0, //greater shadow conjurations
//level 6 (28-30)
0.195992504567504, //chain lightning
0.00110466263272181, //greater spell breach
0.00110466263272181, //summon creature 6
//level 7 (31-32)
0.00110466263272181, //mordenkainen's sword
0.00110466263272181, //spell mantle
//level 8 (33)
0.196996743324524, //horrid wilting
//misc
0.00110466263272181 //melee
};
```

Weights after 500 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0-3)
0, //daze
0, //light
0.00108366490376375, //ray of frost
0.0666503001692164, //resistance
//level 1 (4-8)
0.00108366490376375, //mage armor
0.00108366490376375, //summon creature 1
0.00108366490376375, //6=magic missile
0, //identify
0, //sleep
//level 2 (9-13)
0.00108366490376375, //ghostly visage
0.155454664793191, //10=melf's acid arrow
0.00108366490376375, //summon creature 2
0, //knock
0.00108366490376375, //web (WAS invisibility)
//level 3 (14-18)
0.0122993550267137, //14=fireball
0.00108366490376375, //clarity
0.00108366490376375, //summon creature 3
0.00108366490376375, //haste
0.00108366490376375, //dispel magic
//level 4 (19-23)
0.00108366490376375, //19=stoneskin
0.00108366490376375, //elemental shield
0.174574720145967, //evard's black tentacles
0.174224752716077, //22=ice storm
0.00108366490376375, //minor globe of invulnerability
//level 5 (24-27)
0.00108366490376375, //lesser spell mantle
0.00108366490376375, //summon creature 5
0.00108366490376375, //cloudkill
0, //greater shadow conjurations
//level 6 (28-30)
0.188005119543521, //chain lightning
0.00108366490376375, //greater spell breach
0.00108366490376375, //summon creature 6
//level 7 (31-32)
0.011988910096416, //mordenkainen's sword
0.00108366490376375, //spell mantle
//level 8 (33)
0.194061540958352, //horrid wilting
//misc
0.00106733847527094 //melee
};
```

Weights after 750 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0-3)
0, //daze
0, //light
0.00108857463943687, //ray of frost
0.0350852509773747, //resistance
//level 1 (4-8)
0.00108857463943687, //mage armor
0.00108857463943687, //summon creature 1
0.00108857463943687, //6=magic missile
0, //identify
0, //sleep
//level 2 (9-13)
0.00108857463943687, //ghostly visage
0.174353067336769, //10=melf's acid arrow
0.00108857463943687, //summon creature 2
0, //knock
0.00108857463943687, //web (WAS invisibility)
//level 3 (14-18)
0.00108857463943687, //14=fireball
0.0299371505959085, //clarity
0.00108857463943687, //summon creature 3
0.00108857463943687, //haste
0.00108857463943687, //dispel magic
//level 4 (19-23)
0.00108857463943687, //19=stoneskin
0.00108857463943687, //elemental shield
0.196733916081687, //evard's black tentacles
0.146480678878263, //22=ice storm
0.00108857463943687, //minor globe of invulnerability
//level 5 (24-27)
0.00108857463943687, //lesser spell mantle
0.00108857463943687, //summon creature 5
0.00108857463943687, //cloudkill
0, //greater shadow conjurations
//level 6 (28-30)
0.196733916081687, //chain lightning
0.00108857463943687, //greater spell breach
0.00108857463943687, //summon creature 6
//level 7 (31-32)
0.00108857463943687, //mordenkainen's sword
0.00108857463943687, //spell mantle
//level 8 (33)
0.196733916081687, //horrid wilting
//misc
0.00108203653844928 //melee
};
```

Weights after 1000 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0-3)
0, //daze
0, //light
0.00107249207271599, //ray of frost
0.0122590944525473, //resistance
//level 1 (4-8)
0.00107249207271599, //mage armor
0.00107249207271599, //summon creature 1
0.00107249207271599, //6=magic missile
0, //identify
0, //sleep
//level 2 (9-13)
0.00107249207271599, //ghostly visage
0.179673100827668, //10=melf's acid arrow
0.00107249207271599, //summon creature 2
0, //knock
0.00107249207271599, //web (WAS invisibility)
//level 3 (14-18)
0.00107249207271599, //14=fireball
0.0198042022654587, //clarity
0.00107249207271599, //summon creature 3
0.00107249207271599, //haste
0.00107249207271599, //dispel magic
//level 4 (19-23)
0.00107249207271599, //19=stoneskin
0.00107249207271599, //elemental shield
0.179673100827668, //evard's black tentacles
0.194998558675635, //22=ice storm
0.00107249207271599, //minor globe of invulnerability
//level 5 (24-27)
0.00107249207271599, //lesser spell mantle
0.00107249207271599, //summon creature 5
0.00107249207271599, //cloudkill
0, //greater shadow conjurations
//level 6 (28-30)
0.194998558675635, //chain lightning
0.00107249207271599, //greater spell breach
0.00107249207271599, //summon creature 6
//level 7 (31-32)
0.00107249207271599, //mordenkainen's sword
0.00107249207271599, //spell mantle
//level 8 (33)
0.194998558675635, //horrid wilting
//misc
0.00107249207271599 //melee
};
```

Cleric from FFCW v FFCW

Weights after 10 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0)
0.00616913342258713, //inflict minor wounds
//level 1 (1-3)
0.0129107042664664, //cure light wounds
0.0358631129261162, //inflict light wounds
0.00710630246366453, //bless
//level 2 (4-6)
0.0129107042664664, //cure moderate wounds
0.0508912500501716, //inflict moderate wounds
0.0402354229581381, //aid
//level 3 (7-12)
0.058254633067087, //cure serious wounds
0.000861184820420895, //inflict serious wounds
0, //remove blindness/deafness
0, //remove disease
0.032680536631001, //dispel magic
0.0129107042664664, //animate dead
//level 4 (13-18)
0.0468266149065149, //hammer of the gods
0.00851205602528064, //inflict critical wounds
0.0427247076886512, //freedom of movement
0.0374315199169298, //cure critical wounds
0.0395906750212618, //divine power
0.0493897394292991, //dismissal
//level 5 (19-23)
0.0513459495337289, //heal
0.0448880972737445, //slay living
0.0535582654987718, //flame strike
0.0201041771909471, //cirle of doom
0.0256372095279022, //spell resistance
//level 6 (24-27)
0.0243978759218666, //harm
0.0129107042664664, //blade barrier
0.057606024528066, //banishment
0.0170335566098285, //control undead
//level 7 (28-30)
0.0730597286017606, //destruction
0, //greater restoration
0.000861184820420895, //regenerate
//level 8 (31-32)
0.0509882933236066, //fire storm
0.0714787459559455, //earthquake
//misc 33
0.000861184820420895 //melee
};
```

Weights after 50 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0)
0.00213081048795917, //inflict minor wounds
//level 1 (1-3)
0.00103727610988267, //cure light wounds
0.00111606625275106, //inflict light wounds
0.00103727610988267, //bless
//level 2 (4-6)
0.00103727610988267, //cure moderate wounds
0.0201889529933602, //inflict moderate wounds
0.00103727610988267, //aid
//level 3 (7-12)
0.0835928456081811, //cure serious wounds
0.00103727610988267, //inflict serious wounds
0, //remove blindness/deafness
0, //remove disease
0.00222235630778943, //dispel magic
0.00103727610988267, //animate dead
//level 4 (13-18)
0.154873850248782, //hammer of the gods
0.00103727610988267, //inflict critical wounds
0.00103727610988267, //freedom of movement
0.00103727610988267, //cure critical wounds
0.00103727610988267, //divine power
0.141825709878919, //dismissal
//level 5 (19-23)
0.00103727610988267, //heal
0.00103727610988267, //slay living
0.00213105201135827, //flame strike
0.002202128145507l, //cirle of doom
0.00103727610988267, //spell resistance
//level 6 (24-27)
0.00103727610988267, //harm
0.00103727610988267, //blade barrier
0.188595656342303, //banishment
0.00103727610988267, //control undead
//level 7 (28-30)
0.188595656342303, //destruction
0, //greater restoration
0.00103727610988267, //regenerate
//level 8 (31-32)
0.188595656342303, //fire storm
0.00525828906059573, //earthquake
//misc 33
0.00103727610988267 //melee
};
```

Weights after 100 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0)
0.00104023149100506, //inflict minor wounds
//level 1 (1-3)
0.00104023149100506, //cure light wounds
0.00104023149100506, //inflict light wounds
0.00104023149100506, //bless
//level 2 (4-6)
0.00104023149100506, //cure moderate wounds
0.00241110554219066, //inflict moderate wounds
0.00104023149100506, //aid
//level 3 (7-12)
0.0678507257756659, //cure serious wounds
0.00104023149100506, //inflict serious wounds
0, //remove blindness/deafness
0, //remove disease
0.00104023149100506, //dispel magic
0.00104023149100506, //animate dead
//level 4 (13-18)
0.163877158823102, //hammer of the gods
0.00104023149100506, //inflict critical wounds
0.00104023149100506, //freedom of movement
0.00104023149100506, //cure critical wounds
0.00104023149100506, //divine power
0.173496458981248, //dismissal
//level 5 (19-23)
0.00104023149100506, //heal
0.00104023149100506, //slay living
0.00104023149100506, //flame strike
0.00104023149100506, //cirle of doom
0.00104023149100506, //spell resistance
//level 6 (24-27)
0.00104023149100506, //harm
0.00104023149100506, //blade barrier
0.189132998364557, //banishment
0.00104023149100506, //control undead
//level 7 (28-30)
0.189132998364557, //destruction
0, //greater restoration
0.00104023149100506, //regenerate
//level 8 (31-32)
0.189132998364557, //fire storm
0.00104023149100506, //earthquake
//misc 33
0.00104023149100506 //melee
};
```

Weights after 250 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0)
0.00110043630548751, //inflict minor wounds
//level 1 (1-3)
0.00110043630548751, //cure light wounds
0.00110043630548751, //inflict light wounds
0.00110043630548751, //bless
//level 2 (4-6)
0.00110043630548751, //cure moderate wounds
0.00110043630548751, //inflict moderate wounds
0.00110043630548751, //aid
//level 3 (7-12)
0.189917396963941, //cure serious wounds
0.00110043630548751, //inflict serious wounds
0, //remove blindness/deafness
0, //remove disease
0.00110043630548751, //dispel magic
0.00110043630548751, //animate dead
//level 4 (13-18)
0.179465728224759, //hammer of the gods
0.00110043630548751, //inflict critical wounds
0.00110043630548751, //freedom of movement
0.00110043630548751, //cure critical wounds
0.0350691261772944, //divine power
0.189917396963941, //dismissal
//level 5 (19-23)
0.00110043630548751, //heal
0.0347371724327458, //slay living
0.00110043630548751, //flame strike
0.00110043630548751, //cirle of doom
0.00110043630548751, //spell resistance
//level 6 (24-27)
0.00110043630548751, //harm
0.00110043630548751, //blade barrier
0.00110043630548751, //banishment
0.00110043630548751, //control undead
//level 7 (28-30)
0.156303327226422, //destruction
0, //greater restoration
0.00110043630548751, //regenerate
//level 8 (31-32)
0.188179380679196, //fire storm
0.00110043630548751, //earthquake
//misc 33
0.00110043630548751 //melee
};
```

Weights after 500 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0)
0.00102642151516912, //inflict minor wounds
//level 1 (1-3)
0.00102642151516912, //cure light wounds
0.00102642151516912, //inflict light wounds
0.00102642151516912, //bless
//level 2 (4-6)
0.00102642151516912, //cure moderate wounds
0.00102642151516912, //inflict moderate wounds
0.00102642151516912, //aid
//level 3 (7-12)
0.182602769160189, //cure serious wounds
0.00102642151516912, //inflict serious wounds
0, //remove blindness/deafness
0, //remove disease
0.00102642151516912, //dispel magic
0.00102642151516912, //animate dead
//level 4 (13-18)
0.175477335684356, //hammer of the gods
0.00102642151516912, //inflict critical wounds
0.00102642151516912, //freedom of movement
0.00102642151516912, //cure critical wounds
0.182602769160189, //divine power
0.0684731560804457, //dismissal
//level 5 (19-23)
0.00102642151516912, //heal
0.00102642151516912, //slay living
0.00102642151516912, //flame strike
0.00102642151516912, //cirle of doom
0.00102642151516912, //spell resistance
//level 6 (24-27)
0.00102642151516912, //harm
0.00102642151516912, //blade barrier
0.00102642151516912, //banishment
0.00102642151516912, //control undead
//level 7 (28-30)
0.182602769160189, //destruction
0, //greater restoration
0.00102642151516912, //regenerate
//level 8 (31-32)
0.182602769160189, //fire storm
0.00102642151516912, //earthquake
//misc 33
0.00100431523038104 //melee
};
```

Weights after 750 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0)
0.00107227150876675, //inflict minor wounds
//level 1 (1-3)
0.00107227150876675, //cure light wounds
0.00107227150876675, //inflict light wounds
0.00107227150876675, //bless
//level 2 (4-6)
0.00107227150876675, //cure moderate wounds
0.00107227150876675, //inflict moderate wounds
0.00107227150876675, //aid
//level 3 (7-12)
0.160882772669498, //cure serious wounds
0.00107227150876675, //inflict serious wounds
0, //remove blindness/deafness
0, //remove disease
0.00107227150876675, //dispel magic
0.00107227150876675, //animate dead
//level 4 (13-18)
0.192453440916937, //hammer of the gods
0.00107227150876675, //inflict critical wounds
0.00107227150876675, //freedom of movement
0.00107227150876675, //cure critical wounds
0.192453440916937, //divine power
0.055304773554602, //dismissal
//level 5 (19-23)
0.00107227150876675, //heal
0.00107227150876675, //slay living
0.00107227150876675, //flame strike
0.00107227150876675, //cirle of doom
0.00107227150876675, //spell resistance
//level 6 (24-27)
0.00107227150876675, //harm
0.00107227150876675, //blade barrier
0.00107227150876675, //banishment
0.00107227150876675, //control undead
//level 7 (28-30)
0.179659120889645, //destruction
0, //greater restoration
0.00107227150876675, //regenerate
//level 8 (31-32)
0.192453440916937, //fire storm
0.00107227150876675, //earthquake
//misc 33
0.00105849392504315 //melee
};
```

Weights after 1000 fights:

```
double[] weightArray = new double[]{ //weight of spells
//level 0 (0)
0.0010753808131337, //inflict minor wounds
//level 1 (1-3)
0.0010753808131337, //cure light wounds
0.0010753808131337, //inflict light wounds
0.0010753808131337, //bless
//level 2 (4-6)
0.0010753808131337, //cure moderate wounds
0.0010753808131337, //inflict moderate wounds
0.00708927450010023, //aid
//level 3 (7-12)
0.181634336998221, //cure serious wounds
0.0010753808131337, //inflict serious wounds
0, //remove blindness/deafness
0, //remove disease
0.0010753808131337, //dispel magic
0.0010753808131337, //animate dead
//level 4 (13-18)
0.195523784206128, //hammer of the gods
0.00197304550035285, //inflict critical wounds
0.0010753808131337, //freedom of movement
0.0010753808131337, //cure critical wounds
0.145049563966445, //divine power
0.0010753808131337, //dismissal
//level 5 (19-23)
0.0010753808131337, //heal
0.0010753808131337, //slay living
0.0010753808131337, //flame strike
0.0010753808131337, //cirle of doom
0.0471079483502052, //spell resistance
//level 6 (24-27)
0.0010753808131337, //harm
0.0010753808131337, //blade barrier
0.0010753808131337, //banishment
0.0010753808131337, //control undead
//level 7 (28-30)
0.195523784206128, //destruction
0, //greater restoration
0.0010753808131337, //regenerate
//level 8 (31-32)
0.195523784206128, //fire storm
0.00691610017735052, //earthquake
//misc 33
0.0010753808131337 //melee
};
```

Appendix D: Results from NWN

Below are the results from a team of CCWW facing a team of FFCW. The FFCW team used the default logic that comes with the NWN module. The columns on the left show the results when the CCWW team used the default logic, the columns on the right show the results when that team used the starting point synthesized from the dynamic scripting trials. The tables show the number of wins for the CCWW team, along with the number of draws, and the average remaining hit points for the opposing team. The opposing team had 434 hit points to start.
Each trial had 100 fights.

	default starting point			dynamic scripting starting point		
trial #	wins	draws	enemy hp	wins	draws	enemy hp
1	2	2	113.54	6	0	70.3
2	3	1	135.41	8	0	71.49
3	3	0	128.22	4	1	71.97
4	4	1	122.05	11	0	68.47
5	3	2	121.9	4	0	73.23
6	1	2	130.52	5	0	68.97
7	3	0	111.36	10	0	70.95
8	4	0	112.33	6	0	72.03
9	7	2	109.21	8	0	64.64
10	7	2	96.89	10	0	70.09
11	2	1	116.92	6	2	70.5
12	2	2	118.45	12	0	64.59
13	3	3	95.25	5	0	76.33
14	2	0	120.25	10	0	63.95
15	5	0	115.56	12	1	67.14
average	3.4		116.524	7.8		69.64333
variance	2.91		114.15	7.63		11.08
standard deviation	1.7		10.68	2.76		3.33

Table 17: Results from CCWW v FFCW

Appendix E: Code Samples from NWN

The following samples come from scripts that shipped with the NWN game.

Code Sample 1:

```
// Tries to cast a summoning spell as high as possible, at least when there
// is not yet as creature summoned, unless that creature almost dead and
// currently inactive. Gate will not be cast this way, because for Gate the
// whole party should be protected against evil. This can be checked and added,
// of course.
int CastHighSummon( object oEnemy )
{
   location locTarget = GetLocation( OBJECT_SELF );

   object oSummoned = GetAssociate( ASSOCIATE_TYPE_SUMMONED );

   if (GetIsObjectValid( oSummoned )) // There is a summoned creature
   {
      if (!GetIsEnemy( oSummoned ))
      {
         if (GetCurrentHitPoints( oSummoned ) * 4 < GetMaxHitPoints( oSummoned )) // And it has
at least 25% of HP left
            return FALSE;

         if (!(GetHasEffect( EFFECT_TYPE_SLEEP, oSummoned ) || // And it is useful
            GetHasEffect( EFFECT_TYPE_STUNNED, oSummoned ) ||
            GetHasEffect( EFFECT_TYPE_TURNED, oSummoned ) ||
            GetHasEffect( EFFECT_TYPE_CHARMED, oSummoned ) ||
            GetHasEffect( EFFECT_TYPE_CONFUSED, oSummoned ) ||
            GetHasEffect( EFFECT_TYPE_DOMINATED, oSummoned ) ||
            GetHasEffect( EFFECT_TYPE_FRIGHTENED, oSummoned )))
            return FALSE;
      }
   }

   if (GetIsObjectValid( oEnemy ))
   {
      vector vTarget = GetPosition( oEnemy );
      vector vSource = GetPosition( OBJECT_SELF );
      vector vDirection = vTarget - vSource;
      float fDistance = VectorMagnitude( vDirection ) / 2.0f;
      vector vPoint = VectorNormalize( vDirection ) * fDistance + vSource;
      locTarget = Location( GetArea( OBJECT_SELF ), vPoint, GetFacing( OBJECT_SELF ) );
   }

   if (GetHasSpell( SPELL_ELEMENTAL_SWARM ))
   {
      ClearAllActions();
      sRuleText += " (Elemental Swarm)";
      ActionCastSpellAtLocation( SPELL_ELEMENTAL_SWARM, locTarget );
      return TRUE;
   }
```

```
if (GetHasSpell( SPELL_SUMMON_CREATURE_IX ))
{
   ClearAllActions();
   sRuleText += " (Summon Creature IX)";
   ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_IX, locTarget );
   return TRUE;
}

if (GetHasSpell( SPELL_CREATE_GREATER_UNDEAD ))
{
   ClearAllActions();
   sRuleText += " (Create Greater Undead)";
   ActionCastSpellAtLocation( SPELL_CREATE_GREATER_UNDEAD, locTarget );
   return TRUE;
}

if (GetHasSpell( SPELL_GREATER_PLANAR_BINDING ))
{
   ClearAllActions();
   sRuleText += " (Greater Planar Binding)";
   ActionCastSpellAtLocation( SPELL_GREATER_PLANAR_BINDING, locTarget );
   return TRUE;
}

if (GetHasSpell( SPELL_SUMMON_CREATURE_VIII ))
{
   ClearAllActions();
   sRuleText += " (Summon Creature VIII)";
   ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_VIII, locTarget );
   return TRUE;
}

if (GetHasSpell( SPELL_MORDENKAINENS_SWORD ))
{
   ClearAllActions();
   sRuleText += " (Mordenkainen's Sword)";
   ActionCastSpellAtLocation( SPELL_MORDENKAINENS_SWORD, locTarget );
   return TRUE;
}

if (GetHasSpell( SPELL_SUMMON_CREATURE_VII ))
{
   ClearAllActions();
   sRuleText += " (Summon Creature VII)";
   ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_VII, locTarget );
   return TRUE;
}

if (GetHasSpell( SPELL_PLANAR_BINDING ))
{
   ClearAllActions();
   sRuleText += " (Planar Binding)";
   ActionCastSpellAtLocation( SPELL_PLANAR_BINDING, locTarget );
   return TRUE;
}
```

```
    if (GetHasSpell( SPELL_SHADES_SUMMON_SHADOW ))
    {
        ClearAllActions();
        sRuleText += " (Summon Shadow)";
        ActionCastSpellAtLocation( SPELL_SHADES_SUMMON_SHADOW, locTarget );
        return TRUE;
    }

    if (GetHasSpell( SPELL_SUMMON_CREATURE_VI ))
    {
        ClearAllActions();
        sRuleText += " (Summon Creature VI)";
        ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_VI, locTarget );
        return TRUE;
    }

    if (GetHasSpell( SPELL_LESSER_PLANAR_BINDING ))
    {
        ClearAllActions();
        sRuleText += " (Lesser Planar Binding)";
        ActionCastSpellAtLocation( SPELL_LESSER_PLANAR_BINDING, locTarget );
        return TRUE;
    }

    if (GetHasSpell( SPELL_GREATER_SHADOW_CONJURATION_SUMMON_SHADOW ))
    {
        ClearAllActions();
        sRuleText += " (Greater Shadow Conjuration)";
        ActionCastSpellAtLocation(
SPELL_GREATER_SHADOW_CONJURATION_SUMMON_SHADOW, locTarget );
        return TRUE;
    }

    if (GetHasSpell( SPELL_SUMMON_CREATURE_V ))
    {
        ClearAllActions();
        sRuleText += " (Summon Creature V)";
        ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_V, locTarget );
        return TRUE;
    }

    if (GetHasSpell( SPELL_ANIMATE_DEAD ))
    {
        ClearAllActions();
        sRuleText += " (Animate Dead)";
        ActionCastSpellAtLocation( SPELL_ANIMATE_DEAD, locTarget );
        return TRUE;
    }

    if (GetHasSpell( SPELL_SUMMON_CREATURE_IV ))
    {
        ClearAllActions();
        sRuleText += " (Summon Creature IV)";
        ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_IV, locTarget );
        return TRUE;
    }

    if (GetHasSpell( SPELL_SUMMON_CREATURE_III ))
    {
        ClearAllActions();
        sRuleText += " (Summon Creature III)";
        ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_III, locTarget );
        return TRUE;
    }

    if (GetHasSpell( SPELL_SUMMON_CREATURE_II ))
    {
        ClearAllActions();
        sRuleText += " (Summon Creature II)";
        ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_II, locTarget );
        return TRUE;
    }

    if (GetHasSpell( SPELL_SUMMON_CREATURE_I ))
    {
        ClearAllActions();
        sRuleText += " (Summon Creature I)";
        ActionCastSpellAtLocation( SPELL_SUMMON_CREATURE_I, locTarget );
        return TRUE;
    }

    return FALSE;
}
```

Code Sample 2:

```
    // HIGH Priority rules. These rules are fairly specific, but can be used
    // in general at any time during combat. They are executed in specific
    // circumstances or use talents in a specific way, to ensure that
    // the talents a creature has, when they are employed, are employed as
    // effectively as possible.

    // Still added must be: CastHighBreach (to break through enemy magical
    // defenses), CastHighBless (to add good effects to friends),
    // Shapechanging spells, Domination spells, Curse removal spells, and
    // Invisibility.
    if (iRule < HIGHEST_PRIORITY_COUNT + HIGH_PRIORITY_COUNT)
    {
        iSwitchValue = iRule - HIGHEST_PRIORITY_COUNT;

        switch (iSwitchValue)
        {
            case 0:
<several lines removed for clarity>

            case 1:
                if (IsMage( oCreature ) ||
                    IsPriest( oCreature ) ||
                    HasClass( oCreature, CLASS_TYPE_UNDEAD ) ||
                    HasClass( oCreature, CLASS_TYPE_OUTSIDER ) ||
                    HasClass( oCreature, CLASS_TYPE_CONSTRUCT ) ||
                    HasClass( oCreature, CLASS_TYPE_ELEMENTAL ) ||
                    IsGeneralClass( oCreature ))
                {
                    if (iJustCheck)
                        return TRUE;
                    sRuleText = "CastHighSummon(oNearest)";
                    return CastHighSummon( oNearest );
                }
                break;

            case  2:
                if (IsMage( oCreature ) ||
                    IsPriest( oCreature ) ||
                    HasClass( oCreature, CLASS_TYPE_UNDEAD ) ||
                    HasClass( oCreature, CLASS_TYPE_OUTSIDER ) ||
                    HasClass( oCreature, CLASS_TYPE_CONSTRUCT ) ||
                    HasClass( oCreature, CLASS_TYPE_ELEMENTAL ) ||
                    IsGeneralClass( oCreature ))
                {
                    if (iJustCheck)
                        return TRUE;
                    if (GetIsObjectValid( oNearestWizard ))
                    {
                        sRuleText = "CastHighSummon(oNearestWizard)";
                        return CastHighSummon( oNearestWizard );
                    }
                    else if (GetIsObjectValid( oNearestPriest ))
                    {
                        sRuleText = "CastHighSummon(oNearestPriest)";
                        return CastHighSummon( oNearestPriest );
                    }
                }
                break;

            case  3:
                if (IsMage( oCreature ) ||
                    IsPriest( oCreature ) ||
                    HasClass( oCreature, CLASS_TYPE_UNDEAD ) ||
                    HasClass( oCreature, CLASS_TYPE_OUTSIDER ) ||
                    HasClass( oCreature, CLASS_TYPE_CONSTRUCT ) ||
                    HasClass( oCreature, CLASS_TYPE_ELEMENTAL ) ||
                    IsGeneralClass( oCreature ))
                {
                    if (iJustCheck)
                        return TRUE;
                    sRuleText = "CastHighDamageAreaEffect(oNearest)";
                    return CastHighDamageAreaEffect( oNearest );
                }
                break;
<several lines removed for clarity>
```

Code Sample 3:

```
    // If the fight is still going on, update the counters of those that
    // are still allive. I originally did this in the heartbeats of the
    // opponents themselves, but the heartbeats don't fire if they are
    // dazed or something.
    if (nFight)
    {
       // Execute curse if necessary
       int iDoCurse = GetLocalInt( OBJECT_SELF, "DO_CURSE" );

       if (iDoCurse)
       {
          SetLocalInt( OBJECT_SELF, "DO_CURSE", 0 );

          string sTarget = "BLANCHE";

          // Addition
          if (iDoCurse < 2)
             sTarget = "NERA";

          oWhite = GetNearestObjectByTag( sTarget, OBJECT_SELF );
          effect eCurse = EffectCurse( CURSE_PENALTY, CURSE_PENALTY,
             CURSE_PENALTY, CURSE_PENALTY, CURSE_PENALTY, CURSE_PENALTY );
          effect evisStun = EffectVisualEffect( VFX_DUR_MIND_AFFECTING_DISABLED );
          effect evisStunImp = EffectVisualEffect( VFX_IMP_STUN );
          int k = 1;
          while (GetIsObjectValid( oWhite ))
          {
             ApplyEffectToObject( DURATION_TYPE_INSTANT, evisStunImp, oWhite );
             ApplyEffectToObject( DURATION_TYPE_TEMPORARY, eCurse, oWhite, 60.0 );
             ApplyEffectToObject( DURATION_TYPE_TEMPORARY, evisStun, oWhite, 60.0 );
             ++k;
             oWhite = GetNearestObjectByTag( sTarget, OBJECT_SELF, k );
          }
       }

       oWhite = GetNearestObjectByTag( "BLANCHE" );
       object oSign;
       int i = 1;

       while (GetIsObjectValid( oWhite ))
       {
          oSign = GetNearestObjectByTag( "SIGN_" + GetResRef( oWhite ) );
          if (GetIsObjectValid( oSign ))
             SetLocalInt( oSign, "END_ROUND_NUMBER", GetLocalInt( oSign,
"END_ROUND_NUMBER" ) + 1 );
          ++i;
          oWhite = GetNearestObjectByTag( "BLANCHE", OBJECT_SELF, i );
       }

       i = 1;
       while (GetIsObjectValid( oBlack ))
       {
          oSign = GetNearestObjectByTag( "SIGN_" + GetResRef( oBlack ) );
          if (GetIsObjectValid( oSign ))
             SetLocalInt( oSign, "END_ROUND_NUMBER", GetLocalInt( oSign,
"END_ROUND_NUMBER" ) + 1 );
          ++i;
          oBlack = GetNearestObjectByTag( "NERA", OBJECT_SELF, i );
       }
```

```
}

// Check how the current fight is going, but do this only once
// a minute, otherwise it might take too much time.
if (nFight && ((nHeartbeat%10) == 0))
{
   // Igor checks nothing if it's a fight from the player against one
   // of the opponents.
   object oGong = GetObjectByTag( "GONG" );
   int iFightWhite = GetLocalInt( oGong, "FIGHT_WHITE" );
   int iFightBlack = GetLocalInt( oGong, "FIGHT_BLACK" );

   if (iFightWhite || iFightBlack)
      return;

   // First check if there is at least one of the opponents visible.
   int i = 1;
   int iSomeoneVisible = FALSE;
   oWhite = GetNearestObjectByTag( "BLANCHE" );
   while (GetIsObjectValid( oWhite ))
   {
      if (!(GetHasSpellEffect( SPELL_INVISIBILITY, oWhite ) ||
         GetHasSpellEffect( SPELL_IMPROVED_INVISIBILITY, oWhite )))
      {
         iSomeoneVisible = TRUE;
         break;
      }

      ++i;
      oWhite = GetNearestObjectByTag( "BLANCHE", OBJECT_SELF, i );
   }

   if (iSomeoneVisible)
      return;

   i = 1;
   oBlack = GetNearestObjectByTag( "NERA" );
   while (GetIsObjectValid( oBlack ))
   {
      if (!(GetHasSpellEffect( SPELL_INVISIBILITY, oBlack ) ||
         GetHasSpellEffect( SPELL_IMPROVED_INVISIBILITY, oBlack )))
      {
         iSomeoneVisible = TRUE;
         break;
      }

      ++i;
      oBlack = GetNearestObjectByTag( "NERA", OBJECT_SELF, i );
   }

   if (iSomeoneVisible)
      return;

   // Since everyone is invisible, Igor will remove the effects from everyone.

   ClearAllActions();
```

```
    // Flag that Igor is busy and should not be interrupted.
    SetLocalInt( OBJECT_SELF, "BUSY", 1 );

    SpeakString( "You are unable to see each other, ladies. Let me correct that." );

    i = 1;
    oWhite = GetNearestObjectByTag( "BLANCHE", OBJECT_SELF, 1 );
    while (GetIsObjectValid( oWhite ))
    {
        RemoveSpecificEffect( EFFECT_TYPE_INVISIBILITY, oWhite );
        RemoveSpecificEffect( EFFECT_TYPE_IMPROVEDINVISIBILITY, oWhite );
        ++i;
        oWhite = GetNearestObjectByTag( "BLANCHE", OBJECT_SELF, i );
    }

    i = 1;
    oBlack = GetNearestObjectByTag( "NERA", OBJECT_SELF, 1 );
    while (GetIsObjectValid( oBlack ))
    {
        RemoveSpecificEffect( EFFECT_TYPE_INVISIBILITY, oBlack );
        RemoveSpecificEffect( EFFECT_TYPE_IMPROVEDINVISIBILITY, oBlack );
        ++i;
        oBlack = GetNearestObjectByTag( "NERA", OBJECT_SELF, i );
    }
```

Code Sample 4:

```
// Tries to cast a cursing spell as high as possible at a single enemy. The
// lowest level spells are not used here because they usuallu have less effect
// than a simple physical attack.
int CastHighCurse( object oEnemy )
{
    if (!GetIsObjectValid( oEnemy ))
        return FALSE;

    if (!GetIsEnemy( oEnemy ))
        return FALSE;

    float iDistance = GetDistanceToObject( oEnemy );
    if (iDistance < 0.0)
        return FALSE;

    if (GetHasSpell( SPELL_ENERGY_DRAIN ) &&
        !GetIsImmune( oEnemy, IMMUNITY_TYPE_DEATH ) &&
        !ProtectedAgainstSpells( oEnemy, 9 ))
    {
        ClearAllActions();
        ActionCastSpellAtObject( SPELL_ENERGY_DRAIN, oEnemy );
        return TRUE;
    }

    if (GetHasSpell( SPELL_POWER_WORD_STUN ) &&
        GetCurrentHitPoints( oEnemy ) <= 150 &&
        !GetIsImmune( oEnemy, IMMUNITY_TYPE_STUN ) &&
        !GetIsImmune( oEnemy, IMMUNITY_TYPE_MIND_SPELLS ) &&
    !ProtectedAgainstSpells( oEnemy, 7 ))
{
    ClearAllActions();
    sRuleText += " (Power Word: Stun)";
    ActionCastSpellAtObject( SPELL_POWER_WORD_STUN, oEnemy );
    return TRUE;
}

if (GetHasSpell( SPELL_FEEBLEMIND ) &&
    HasClass( oEnemy, CLASS_TYPE_WIZARD ) &&
    !GetIsImmune( oEnemy, IMMUNITY_TYPE_MIND_SPELLS ) &&
    !ProtectedAgainstSpells( oEnemy, 5 ))
{
    ClearAllActions();
    sRuleText += " (Feeblemind)";
    ActionCastSpellAtObject( SPELL_FEEBLEMIND, oEnemy );
    return TRUE;
}

if (GetHasSpell( SPELL_ENERVATION ) &&
    iDistance < DISTANCE_SHORT &&
    !GetIsImmune( oEnemy, IMMUNITY_TYPE_DEATH ) &&
    !ProtectedAgainstSpells( oEnemy, 4 ))
{
    ClearAllActions();
    sRuleText += " (Enervation)";
    ActionCastSpellAtObject( SPELL_ENERVATION, oEnemy );
    return TRUE;
}

if (GetHasSpell( SPELL_BESTOW_CURSE ) &&
    iDistance < DISTANCE_SHORT &&
    !GetIsImmune( oEnemy, IMMUNITY_TYPE_CURSED ) &&
    !ProtectedAgainstSpells( oEnemy, 3 ))
{
    ClearAllActions();
    sRuleText += " (Bestow Curse)";
    ActionCastSpellAtObject( SPELL_BESTOW_CURSE, oEnemy );
    return TRUE;
}

if (GetHasSpell( SPELL_HOLD_PERSON ) &&
    !GetIsImmune( oEnemy, IMMUNITY_TYPE_MOVEMENT_SPEED_DECREASE ) &&
    !ProtectedAgainstSpells( oEnemy, 2 ))
{
    ClearAllActions();
    sRuleText += " (Hold Person)";
    ActionCastSpellAtObject( SPELL_HOLD_PERSON, oEnemy );
    return TRUE;
}

if (GetHasSpell( SPELL_GHOUL_TOUCH ) &&
    iDistance < DISTANCE_SHORT &&
    !GetIsImmune( oEnemy, IMMUNITY_TYPE_DEATH ) &&
    !GetIsImmune( oEnemy, IMMUNITY_TYPE_PARALYSIS ) &&
    !ProtectedAgainstSpells( oEnemy, 2 ))
```

```
{
   ClearAllActions();
   sRuleText += " (Ghoul Touch)";
   ActionCastSpellAtObject( SPELL_GHOUL_TOUCH, oEnemy );
   return TRUE;
}

if (GetHasSpell( SPELL_BLINDNESS_AND_DEAFNESS ) &&
   IsMage( oEnemy ) &&
   !GetIsImmune( oEnemy, IMMUNITY_TYPE_BLINDNESS ) &&
   !GetIsImmune( oEnemy, IMMUNITY_TYPE_DEAFNESS ) &&
   !ProtectedAgainstSpells( oEnemy, 2 ))
{
   ClearAllActions();
   sRuleText += " (Blindness and Deafness)";
   ActionCastSpellAtObject( SPELL_BLINDNESS_AND_DEAFNESS, oEnemy );
   return TRUE;
}

if (GetHasSpell( SPELL_SCARE ) &&
   GetHitDice( oEnemy ) <= 5 &&
   !GetIsImmune( oEnemy, IMMUNITY_TYPE_FEAR ) &&
   !GetIsImmune( oEnemy, IMMUNITY_TYPE_DEATH ) &&
   !ProtectedAgainstSpells( oEnemy, 1 ))
{
   ClearAllActions();
   sRuleText += " (Scare)";
   ActionCastSpellAtObject( SPELL_SCARE, oEnemy );
   return TRUE;
}

if (GetHasSpell( SPELL_RAY_OF_ENFEEBLEMENT ) &&
   GetAbilityScore( oEnemy, ABILITY_STRENGTH ) > 14 &&
   !GetIsImmune( oEnemy, IMMUNITY_TYPE_DEATH ) &&
   !ProtectedAgainstSpells( oEnemy, 1 ))
{
   ClearAllActions();
   sRuleText += " (Ray of Enfeeblement)";
   ActionCastSpellAtObject( SPELL_RAY_OF_ENFEEBLEMENT, oEnemy );
   return TRUE;
}

if (GetHasSpell( SPELL_DOOM ) &&
   !GetIsImmune( oEnemy, IMMUNITY_TYPE_MIND_SPELLS ) &&
   !ProtectedAgainstSpells( oEnemy, 1 ))
{
   ClearAllActions();
   sRuleText += " (Doom)";
   ActionCastSpellAtObject( SPELL_DOOM, oEnemy );
   return TRUE;
}

if (GetHasSpell( SPELL_ENTANGLE ) &&
   !(IsMage( oEnemy ) || IsPriest( oEnemy )) &&
   !GetIsImmune( oEnemy, IMMUNITY_TYPE_ENTANGLE ) &&
   !ProtectedAgainstSpells( oEnemy, 1 ))
```

```
    {
        ClearAllActions();
        sRuleText += " (Entangle)";
        ActionCastSpellAtObject( SPELL_ENTANGLE, oEnemy );
        return TRUE;
    }

    return FALSE;
}
```

Code Sample 5:

```
  switch ( GetLastSpellCastClass() )
  {

  case 10: //wizard

    if (GetHasSpell( SPELL_HORRID_WILTING ))
    {
        ClearAllActions();
        ActionCastSpellAtLocation( SPELL_HORRID_WILTING, locTarget );
        PrintString(" cast horrid wilting ");
        return ;
    }

    if (GetHasSpell( SPELL_CHAIN_LIGHTNING ))
    {
        ClearAllActions();
        ActionCastSpellAtObject( SPELL_CHAIN_LIGHTNING, oEnemy );
        PrintString(" cast chain lightning ");
        return ;
    }

    if (GetHasSpell( SPELL_ICE_STORM ))
    {
        ClearAllActions();
        ActionCastSpellAtLocation( SPELL_ICE_STORM, locTarget );
        PrintString(" cast ice storm ");
        return ;
    }

    if (GetHasSpell( SPELL_EVARDS_BLACK_TENTACLES ))
    {
        ClearAllActions();
        ActionCastSpellAtLocation( SPELL_EVARDS_BLACK_TENTACLES, locTarget );
        PrintString(" cast evards black tentacles ");
        return ;
    }

    if (GetHasSpell( SPELL_MELFS_ACID_ARROW ))
    {
        ClearAllActions();
        ActionCastSpellAtObject( SPELL_MELFS_ACID_ARROW, oEnemy );
        PrintString(" cast melf's acid arrow ");
        return ;
```

```
}

break;

}//end switch
```

Appendix F: Wizard and Sorcerer spells

The following are the default spells available to the two character classes. There is a large amount of overlap in the spells available to these classes, which allows a script that is created in from class be used by the other.

Spell Level	Wizard (Level 15)	Sorcerer (Level 15)
0	Daze Light Ray of Frost Resistance	Daze Light Ray of Frost Resistance
1	Mage Armor Summon Creature I Magic Missile Identify Sleep	Mage Armor Summon Creature I Magic Missile Identify Sleep
2	Ghostly Visage Melf's Acid Arrow Summon Creature II Invisibility Knock	Ghostly Visage Melf's Acid Arrow Summon Creature II Invisibility Knock
3	Fireball Clarity Summon Creature III Haste Dispel Magic	Fireball Clarity Summon Creature III Haste
4	Stoneskin Elemental Shield Evard's Black Tentacles Ice Storm Minor Globe of Invulnerability	Stoneskin Elemental Shield Evard's Black Tentacles Ice Storm
5	Lesser Spell Mantle Summon Creature V Cloudkill Greater Shadow Conjuration	Lesser Spell Mantle Summon Creature V Cloudkill Greater Shadow Conjuration
6	Chain Lightning Greater Spell Breach Summon Creature VI	Chain Lightning Greater Spell Breach Summon Creature VI
7	Mordenkainen's Sword Spell Mantle	Mordenkainen's Sword Spell Mantle
8	Horrid Wilting	(none)

Table 18: Comparison of Wizard and Sorcerer spells

Appendix G: Bayesian Networks in Games

Many games use static terrain with fixed reconstruction of areas that a player leaves and later revisits. This can result in noticeable differences between the reconstructed area and the player's recollections (or expectations). These differences can lessen a player's immersion in the game, or the usefulness of the simulation. We propose an approach for environmental reconstruction that uses Bayesian Networks to quickly and easily calculate likely effects that external factors have on the environment. The reconstruction of revisited areas becomes less disconcerting and permits the incorporation of plausible changes based on unobserved, yet reasonably expected events that could have occurred during the player's absence. Once these Bayesian Networks have been created, we show how they can also be used to model an agent's beliefs on what happened in the environment, and to possibly help control the agent's actions. We conclude with screenshots and a discussion of an implementation we created that uses Bayesian Networks to recreate different environments.

Introduction

The motivation behind this section is to have a "Level-of-Detail Artificial Intelligence" (LODAI) [Bro02] approach to determine what an area should look like when a player revisits an area in a game. Many games allocate no CPU time for events unfolding outside a player's view. For example, if you break windows in the game "Fable"[1], and then leave the zone and come right back, the windows will all be replaced. If you were to stay in the area and watch, the windows will not get fixed. These games can have the result that if the player is watching, one thing happens, and if the player is not watching, a different set of events occur. This disconnect can be undesirable in games and simulations. At the other extreme, games like "Oblivion"[2] keep track of a great number of changes in the environment. For instance, an apple dropped in the Market District will remain there throughout the game, unless the player picks it up. It never rots or gets removed by another game character. A LODAI approach might use a small amount of CPU time to update the world that is outside the player's view periodically. Using such an approach in the window-breaking example, if the player came back into the area immediately, there would be no change (all of the windows would still be broken). After a long period of time, everything might be back to normal (all of the windows would be replaced). At an intermediate point in time, a plausible intermediate state could be represented (after a few days, the windows on the expensive homes could be replaced, while the other windows would still be broken, or boarded up). In the case of a dropped perishable item (or a valuable item dropped in a public place), the chance of it remaining would decrease with time. In addition to coming up with a quick way of determining what a region that a player is returning to might look like, we will show that it will also be possible to re-use the structures that made this possible to have different agents in the game reason about and act on what caused the current state of their environment.

1 http://lionhead.com/Fable/Fable/
2 http://www.bethsoft.com/eng/games/games_oblivion.html

Bayesian Networks

A Bayesian Network (BN) is a data structure that can be used to represent dependencies among variables and to give a concise specification of a joint probability distribution. A BN is a directed acyclic graph with the variables represented as nodes in the graph, with an edge between a parent node and a child node if the parent node directly influences the child node. It is usually easy for a domain expert to decide what influences exist in the domain. A BN representation can take much less space to specify than a joint probability distribution, and can also be easier to represent. Charniak [1991] provides a good introduction to BNs. A Dynamic Bayesian Network is a BN that represents a temporal probability model. A Dynamic Bayesian Network can be used model the environment over many time slices. At each time slice, the variables have parent nodes that are in its own time slice, or in the time immediately preceding it. We discuss our BN later in this paper.

We chose to use a BN to reconstruct the environment since it is easy to model many independent variables efficiently, and it is quick to calculate the desired probabilities given the available information. Once the BN has been created, we realized that it could be also be used to determine what an agent believes about the local environment and events that may have recently occurred there.

Using Bayesian Networks to Reconstruct the Environment

We used the Microsoft Bayesian Network Editor[3] to model the BN, which can subsequently be used to determine what type of terrain to draw when a player walks into an area that he or she has not visited for a while. Several nodes could affect the terrain that would be ultimately drawn. Nodes can directly affect other nodes (a fire in an area would affect the amount of plants in the area), or indirectly affect other nodes (a dragon in the area can set fires, which affect the plants). Precipitation, the temperature, and the creatures that are present in an area, etc. can all have an effect on what the region should look like when the player enters an area.

As a simple example, a game could have a BN that has rainfall and temperature as nodes that each could be in one of three states (lower than normal, normal, higher than normal). There could also be a node for whether or not there was a dragon in the area. A node for fires could exist that has precipitation, temperature, and dragon as parent nodes. Generally speaking there will more likely to be fires in an area if there is a dragon around, if the temperatures are higher than normal, or if the precipitation is lower than normal. There could also be a node for plant growth with three states (lower than normal, normal, and higher than normal). The plants in the area would grow best with higher temperatures, lots of precipitation, and no fires in the area. The final node in this example indicates whether or not animals have gone missing in the area. Animals are more likely to go missing in a region if there is a dragon in the area. The network is shown below in Figure 21, and the values used in one of the nodes are shown in Figure 22.

3 http://research.microsoft.com/en-us/um/redmond/groups/adapt/msbnx/

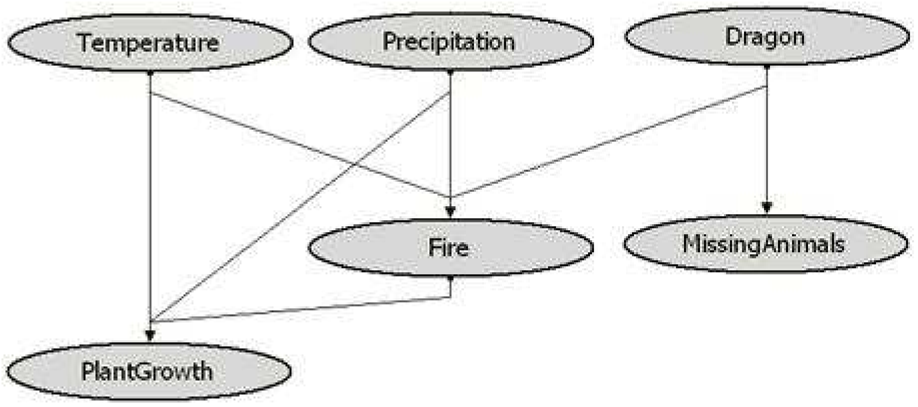
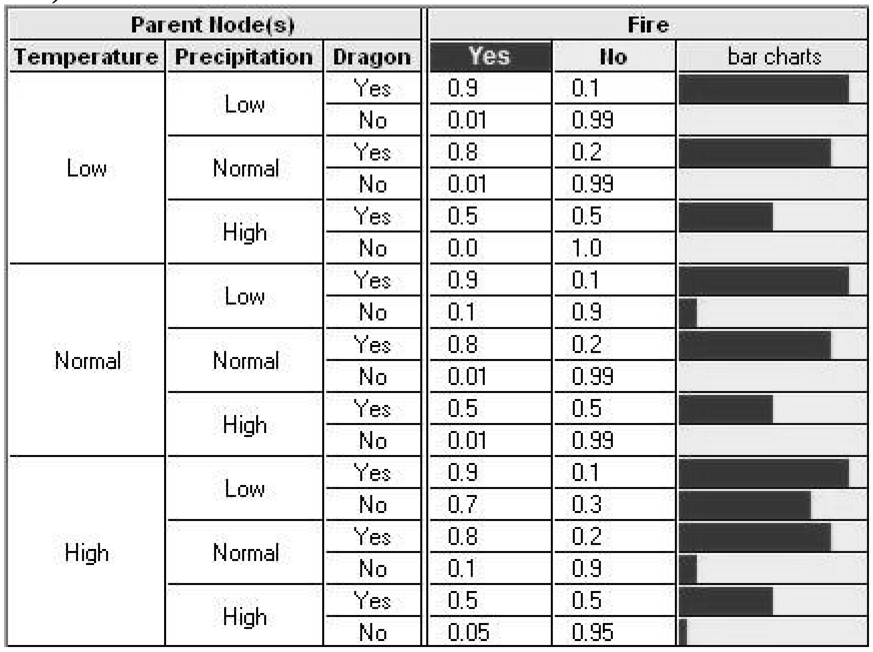


Figure 21: A sample Bayesian Network

Once this network has been set up in a game, the environment in the area can quickly and easily be updated while the character is not in the area. The server can either provide the top-level states of what happened, or they can be randomly generated. For example, if the player's character has not ventured into a certain area in a given month, the server could spend a small amount of CPU time to update the area. First it could examine the last state of that area, and then determine what the states in the BN should be. It would determine what the temperature and precipitation was, and whether or not there was a dragon in the area. In this example, the temperature and precipitation might be randomly determined, but for reasons to advance the storyline of the game, the 'dragon present' node might be set to 'yes'. Once these nodes are given values, the remaining nodes can easily be determined by looking at the probability tables associated with each node. For example if the temperature and precipitation were normal, and there was a dragon in the area, there would be an 80% chance of a fire occurring in that region (look near the middle of Figure 2 for this value).



Parent Node(s)			Fire		
Temperature	Precipitation	Dragon	Yes	No	bar charts
Low	Low	Yes	0.9	0.1	
		No	0.01	0.99	
	Normal	Yes	0.8	0.2	
		No	0.01	0.99	
	High	Yes	0.5	0.5	
		No	0.0	1.0	
Normal	Low	Yes	0.9	0.1	
		No	0.1	0.9	
	Normal	Yes	0.8	0.2	
		No	0.01	0.99	
	High	Yes	0.5	0.5	
		No	0.01	0.99	
High	Low	Yes	0.9	0.1	
		No	0.7	0.3	
	Normal	Yes	0.8	0.2	
		No	0.1	0.9	
	High	Yes	0.5	0.5	
		No	0.05	0.95	

Figure 22: Probabilities used for "Fire" node in Figure 21

Once we determined if there were fires in the region (for this example, we will assume that we picked a number less than 80% so there were fires), we can determine what kind of plant growth occurred in the area. In our network, if there is normal temperature, normal precipitation, and fires are in the area, the plant growth in the area is 60% likely to be low, 30% likely to be normal and 10% likely to be high. All that is required to determine the amount of plant growth in the area is to pick a random number and a lookup the value in the table.

Once the states that the environment should be in are calculated, the game could use that information to determine what the player will actually see. As games can vary quite significantly, an exact implementation of how to update what the player sees would have to depend on the individual game, and is beyond the scope of this paper; however, a possible implementation follows. In a given area under normal conditions, the ground can be covered with healthy green grass and a healthy tree could be drawn. In times of little rainfall and high temperatures, the grass could be drawn using a yellow colour, and instead of using a 'healthy-tree' model, an 'unhealthy-tree' model could be used. If we determined that there was low plant growth in an area, we might not update the environment at all, whereas if there was high plant growth in an area, we might scale the tree models by a small amount, so they appear to have grown, and we might draw some small bushes in an area to show new growth. In a very cold winter with high precipitation, snow can be covering the ground instead of grass, and a model of the tree with snow on the branches can be used. It would make it simple to integrate an approach like this into a game, since there would be little chance of negative interactions with other parts of the game. For example, if the non-player characters in a game use waypoints to find their way around, the same waypoints could be used to navigate around the tree, regardless of what model of the tree the player sees.

This approach does not have to be limited to 'realistic' reconstruction of the environment. In the game Fable, your character's appearance changes based on your actions – a very evil person will have horns growing out of his head. There is nothing stopping a game that uses a BN approach to give clues to the player through the environment. For example, if there is an evil wizard living in a tower, the vegetation in the surrounding area could appear sick and dying. Once the evil wizard has been driven away, the plants can appear healthy and vibrant again.

This approach could also take into account the actions of the characters in the game when determining the environment to draw. If there was a brawl in a tavern recently, a player who wanders into the room might see evidence of the recent fighting (broken glass and furniture). After a period of time, however, the tavern would go back to normal, and there would be little evidence for the player to see that there was a fight. If there was a battle recently between a mad wizard who likes to cast fireballs and a bunch of goblins, a player wandering into the area would see that the ground is blackened, and that there are some corpses on the ground.

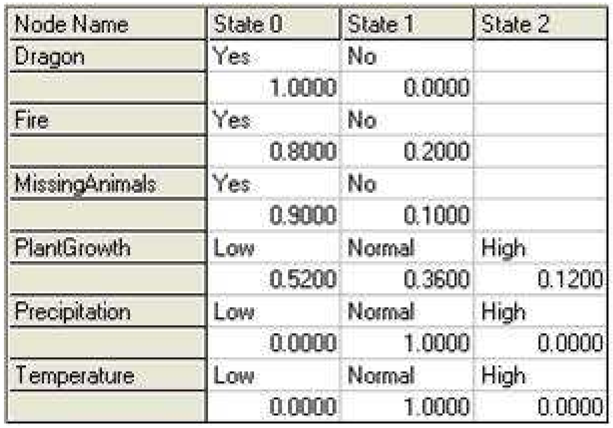
Difficulties with this approach

Implementing this kind of solution in a game will come with additional costs. There is the knowledge engineering cost with using this kind of approach: many game programmers will be unfamiliar with BNs, so there will be a learning curve to overcome. Coming up with the structure of the network can be relatively easy by examining the effects various factors have on each other; however, it might take a fair bit of tuning to get realistic or reasonable probabilities. It would detract from a game if the BN that was used creates an

environment that varies greatly from what the character expects. Testing must be done with the BN to ensure that a small drought does not turn an ancient forest into a desert. The values that we assigned to the nodes in our BN were generated by hand. Depending on how complex you make your network, it might take a considerable amount of time to enter in all of the values and test them. The good news is, once you have acceptable results for one area, it should be easy to reuse your BN in similar areas with little modification. An area in a game that should be desert will require a different BN than an area that should be grasslands, but once you have a BN that gives you suitable results for one grasslands area, you should be able to use it in grassland areas throughout your game.
In addition to any difficulties in getting your BN to produce acceptable outputs, there will be additional work for the game's artists to come up with additional models. Instead of a single tree for a certain location in the game, several trees would need to be made. Based on the BN, the game engine would pick which one the player would see. In addition to the extra time and effort to come up with the models, there would be additional storage requirements to consider when installing the game. This might not be a very large cost, since many models can likely be reused in different areas of the game. For example, if an artist has to make several different models for each tree in a certain area, those models can also be used for the trees in different areas.
There might also be additional costs and other difficulties in keeping track of the player's actions. In a multiplayer game, several players would be able to simultaneously affect the environment. It could be difficult to store all of the data, so that each player can revisit an area at a later date to see their effect on it. However, the data should not need to be stored for very long. If a player is involved in a big battle on a certain day, the details of the area can be aggregated when a certain time passes after the player leaves the area. For example, after a few days, the exact placement of the bodies can be discarded, and only the number and the relative area can be kept. If the player returns after a week, he or she will likely remember that there was a battle in the area, but not the exact placement of the bodies. Even if the player notices that the bodies are in a slightly different orientation, it might be explained away by something else (perhaps someone coming by to search for treasure).
In a single-player game, it might be possible for several years to go by in the game before a player returns to an area. The BN should be able to relatively easily recreate an environment that the player believes is realistic. In a massively multiplayer game, it is unlikely that an area will remain unvisited for that long of a time. In the worst case, a certain area might constantly have one player character or another in the area. In this situation, other approaches will likely have to be taken, as using a BN to update the environment would be inappropriate (since it would be disturbing to the players if the environment they are in suddenly changed for no apparent reason).

Using a Bayesian Network to model an agent's beliefs

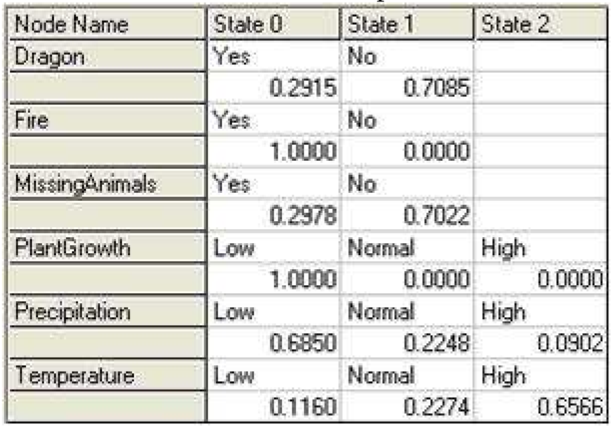
Once the BN has been created, it can also be used to modify an agent's beliefs of what has actually happened in an area while they were away. For example, we can use the BN from Figure 1 in our game we want to recreate a zone that an agent has not visited for a while. In this area, we calculate that the temperature and precipitation has been in the normal regions. Using the BN, we determine that there are likely missing animals in the region (90% probability), evidence of recent fires (80%), and lower than normal overall plant growth (52%). Figure 23 shows the probabilities the server uses when recreating the zone.



Node Name	State 0	State 1	State 2
Dragon	Yes	No	
	1.0000	0.0000	
Fire	Yes	No	
	0.8000	0.2000	
MissingAnimals	Yes	No	
	0.9000	0.1000	
PlantGrowth	Low	Normal	High
	0.5200	0.3600	0.1200
Precipitation	Low	Normal	High
	0.0000	1.0000	0.0000
Temperature	Low	Normal	High
	0.0000	1.0000	0.0000

Figure 23: The probabilities the server uses

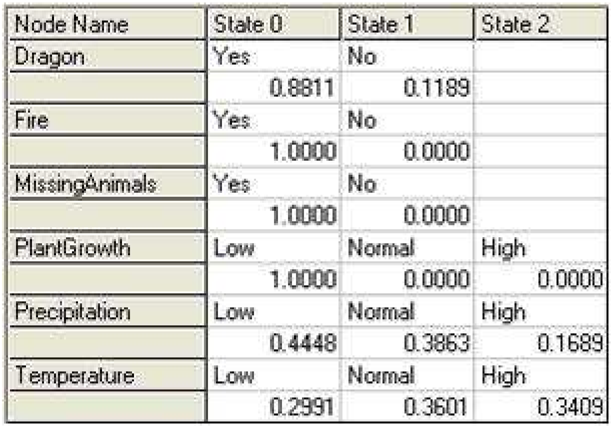
When the agent enters the zone, it sees that the evidence of recent fires and the sparse vegetation. Using the same network, the agent can use inference to calculate that there is likely not a dragon in the area (approximately 71% chance), and that the fires were likely caused by low precipitation and high temperatures (approximately 69% and 66%, respectively). The agent's initial beliefs of the zone are shown in Figure 24. At this point, the agent is only sure that there were fires in the area recently and that there was not much plant growth. Inference was used to determine the probabilities of the other nodes.



Node Name	State 0	State 1	State 2
Dragon	Yes	No	
	0.2915	0.7085	
Fire	Yes	No	
	1.0000	0.0000	
MissingAnimals	Yes	No	
	0.2978	0.7022	
PlantGrowth	Low	Normal	High
	1.0000	0.0000	0.0000
Precipitation	Low	Normal	High
	0.6850	0.2248	0.0902
Temperature	Low	Normal	High
	0.1160	0.2274	0.6566

Figure 24: The agent's initial beliefs

Later on, after talking to a farmer who mentions that some of his livestock has been disappearing lately without a trace, the agent updates his beliefs. Starting with the same evidence as above, and then setting the "Missing Animals" node to true, the agent infers that the chance of a dragon being in the area jumps up to just over 88%, which makes the agent a little nervous. The agent's updated beliefs are shown in Figure 25.



Node Name	State 0	State 1	State 2
Dragon	Yes	No	
	0.8811	0.1189	
Fire	Yes	No	
	1.0000	0.0000	
MissingAnimals	Yes	No	
	1.0000	0.0000	
PlantGrowth	Low	Normal	High
	1.0000	0.0000	0.0000
Precipitation	Low	Normal	High
	0.4448	0.3863	0.1689
Temperature	Low	Normal	High
	0.2991	0.3601	0.3409

Figure 25: The agent's updated beliefs

The BNs that the agents use to infer what happens in an area do not have to exactly match the networks used to update the environment. For example, the agents in an area might have a simpler model of the world than what you use for reconstructing the environment. A character might not believe in dragons, so when presented with evidence of increased fires, he might explain it away either by dry, hot weather, lightning strikes, or even that humans might have started them. A different character might use the same nodes as the BN used to reconstruct the environment, but have different beliefs about what the probabilities should be. Using the same example from Figure 1, if a character overestimates the chance of a dragon being in the area, that character is much more likely to blame any fires that happen on dragons. Although it is possible to have several different networks for different agents to reason about the world, creating and testing them would involve extra work, which detracts somewhat from the simplicity of this approach.

One additional benefit of this approach is that different agents might come to different conclusions when they view the same evidence. If two agents were presented with the same evidence as in Figure 25, one might believe that a dragon is in the area, but the other might not (the server picks a random number of under 88% for the first agent, so he believes that there is a dragon in the area, and a random number of over 88% for the second one, so he does not believe there is a dragon in the area). By having the agents in the game act in different ways because of their different views might make the player think they are doing much more complex reasoning than they actually are. The differences in the ways that the agents react to the situations around them may also add to the replayability to a game, as the player will not be able to completely predict how everyone will react to a certain event.

Controlling Agents with Bayesian Networks

At a high level, a BN can be used to model an agent's beliefs. After an agent views its surroundings and updates its beliefs on what has happened in the area, the agent can also use a probabilistic model to determine its actions. Instead of using a finite state machine (FSM) or similar tool as is common in games to determine the actions of a character, a BN can be used not only to calculate the characters beliefs, but also to decide what actions the character takes. Using a BN to model beliefs and control behaviours has the advantages of being quick to compute, easy to model, and can provide behaviours that are realistic without being recurring or predictable.

As an example, a game has a guard patrolling an area. The goal of the player is to get into a building in the area that the guard is patrolling. One approach that a game might take would be to have the guard controlled by a finite state machine or even a push down automata (both described in [Yis04], with patrol and investigate states. The benefits of this approach are that it is easy to understand, code, and debug; however it does have drawbacks. A guard can easily get caught doing the same thing over and over again, which can detract from the realism of the game. You can imagine the following scenario occurring in a game: A character makes a loud noise while trying break into the building. The guard (using the FSM in Figure 26) hears this, switches from "patrol" to "investigate". The player hides since he hears the guard coming to investigate, after a while, the guard decides there is no one there, and goes back to "patrol" from "investigate". The problem with a finite state machine is this scenario can occur several times, and after a while one would likely think that this behaviour is unrealistic. After the fifth time the player makes a noise in an area, most people would not think it is realistic for a guard to investigate the area for the same amount of time as the previous four occurrences, and most people would expect the guard to do a more thorough investigation of the area, or even perform a different action. While an improved behaviour can certainly be modeled with a more sophisticated finite state machine, the purpose of the approach proposed in this paper is to be able to represent more sophisticated behaviours easily.

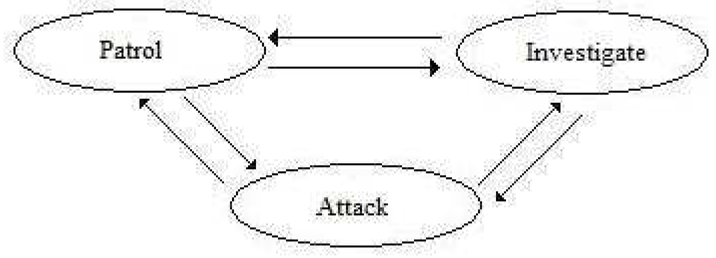


Figure 26: A FSM to control a guard

Let us use our approach in the example. The player makes a noise which the guard hears. This raises the suspicion level in the guard. The guard investigates, finds nothing, and returns to patrolling. When the guard hears noises during subsequent rounds, the guard can end up investigating longer, since his suspicion level was already raised. This can be modeled by having a simple Bayesian Network with a "suspicion" state. The suspicion level can rise whenever a noise is heard (or if the guard notices something else out of the ordinary), and if the guard is suspicious enough he will investigate. The states in this BN will depend only on the states from the previous time slice and any current stimuli (similar to the way the environment in the first section can be calculated simply by using the states of the environment in the previous time slices, along with current factors). After a while, the heightened suspicion would set the guard to investigate on smaller and smaller stimuli, and it could take longer and longer for the guard to have his suspicion lowered enough to return to patrolling.

Implementation

Part of the BN described in Figure 1 was implemented by the authors in the "Neverwinter Nights"[4] (NWN) environment. NWN is an award-winning commercial game that comes with a toolset that allows modifications. There were no noticeable effects on load time

[4] http://nwn.bioware.com/

when this BN was implemented, and it provided different environments to the player based on the player’s actions. The module that was created has a tower that contains magic that allows the player to change the temperature and precipitation in the area, as well as advance time by one or more years.
The following screenshots show some of the screens experienced in this module. Figure 27 shows the initial terrain, which is grassland.

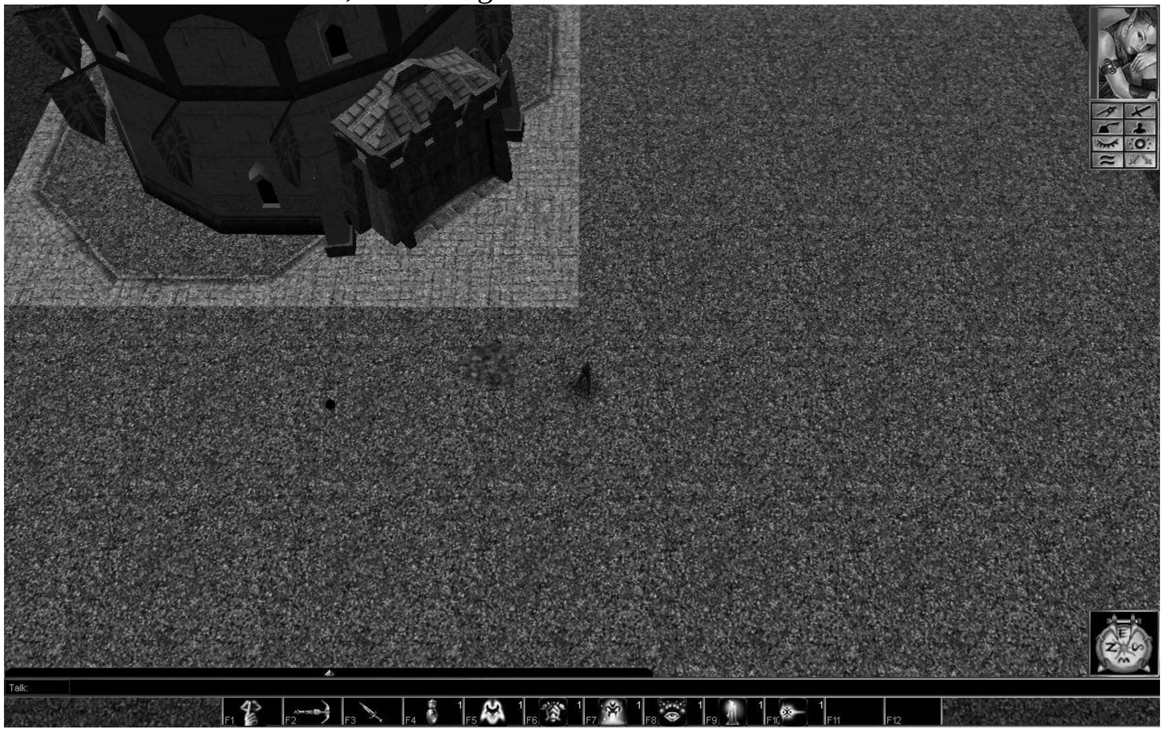


Figure 27: Initial terrain

After going into the tower and setting the temperature and precipitation to very favourable levels and advancing time by several years, the player leaves the tower and views the terrain seen in Figure 28.

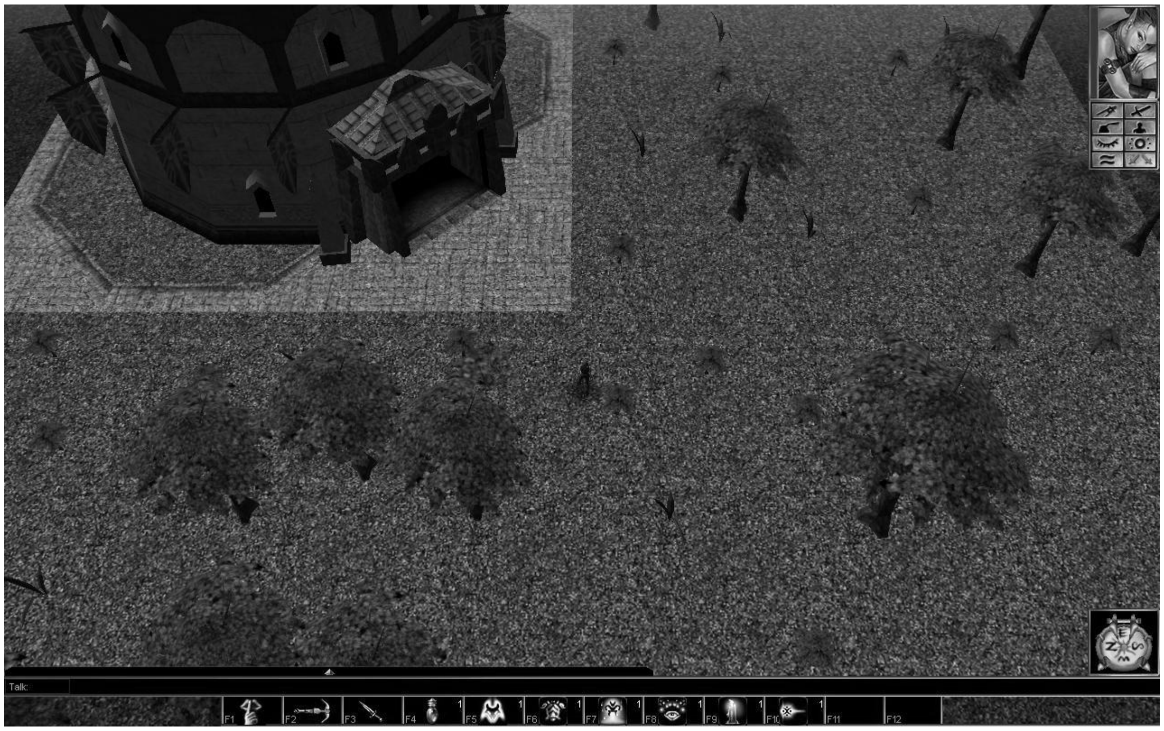


Figure 28: Increased plant growth

In this scenario, the player goes back in the tower, sets the temperature and precipitation levels to unfavourable conditions, and then advances time a few more years. The result is the screenshot in Figure 29.

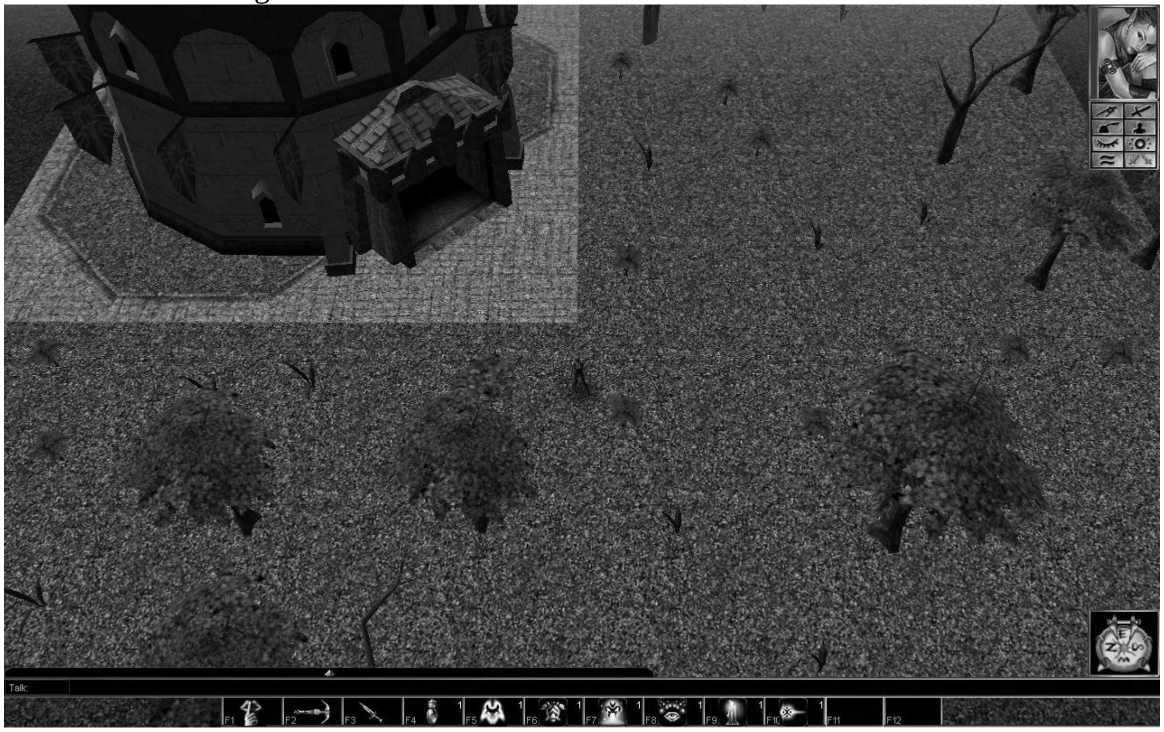


Figure 29: Less favourable conditions

Each time the player advances time by a year, a BN is checked against the temperature and precipitation variables to come up with the amount of plant growth for that year.

Once that is chosen, the world is updated based on the growth rate and the previous state of the world. A fully-grown tree will not pop up after one year, if there was nothing in an area before, only a small bush or patch of grass can occupy that area. If favourable conditions continue, the model can be replaced by a smaller tree model, and then finally by a larger tree model. Similarly, if there is a large tree in an area, it will not disappear after a single poor year. Bushes and grass might disappear, but for every tree in the area a random number will be generated and checked against the results from the BN to see if the tree survived or not. If the tree did not survive, a model of a dead tree would replace the living tree, and would stay for a few years regardless of the subsequent conditions – it would be disconcerting for a tree to be dead one year and the alive the next. Since the model that would appear in any given area at a certain year depends on the conditions and the plant that appeared there in the previous year, a player will not see odd behaviour like an area that is empty one year followed by a dead tree in that area the very next year. The models will only be rendered when the player returns to the area, which helps the computation very quick, necessary for the LODAI approach. If several years pass before a player returns, only the final state of the area will be observed. This behaviour was relatively easy to implement, runs quickly, and could be changed to suit a game's requirements.

Future Work

This method for generating terrain could benefit from applying any techniques that simplify the calculations in any BN. For example, we could explore using only singly connected networks, or polytrees. A method for joining nodes in a BN into a tree of clusters is found in [Huang and Darwiche, 1996]. There also might be a benefit to using a simpler representation of the BN. For example, if the node "fire", which represents the probability of a forest fire occurring, has two parent nodes: "rainfall" and "temperature", which represent the amount of recent rainfall and the current temperature, it might be advantageous to have a representation where we can specify the probability of a large forest fire starting is zero if there was very high amounts of rainfall recently regardless of temperature, rather than having to specify the probability for each temperature. Tools that would allow developers to easily view, modify, and test the networks that they create would also greatly facilitate the incorporation of this idea into commercial games. In addition, advances in BN technology continuously appear in the literature, which will inevitably inspire improvements in our proposed approach.

Conclusions

This paper has demonstrated that it is possible and quite simple to use a BN to help determine what type of terrain to render in a simulation or game. This could be used to make games and simulations more realistic than their counterparts that simply use static terrain. The contribution of this paper is to demonstrate that determining what type of terrain to draw can be done quickly and easily, and can add to the realism in a game, while not taking up much CPU time. The importance of this LODAI approach is that it might give the player the impression that the game has modeled the world more accurately than it actually has, and that the game is doing more work than it actually is. For a relatively small cost, the believability of the game or simulation can be immensely improved.

Additionally, if networks have been created and tested for environmental reconstruction, we have shown how they could be adapted for use by the agents to infer what has happened recently in the area. This can lead to agents appearing to do complex reasoning about the environment, and different agents coming to different conclusions based on what they observe, which might lead to interesting and entertaining interactions for the player.

References

[Ada07] Adams, Ernest, and Andrew Rollings. “Fundamentals of Game Design”, Pearson Education Inc., pp. 650-658, 2007.

[Atk09] Atkinson, John, and Dario Rojas. “On-the-fly generation of multi-robot team formation strategies based on game conditions”, Expert Systems With Applications, Vol. 36, pp. 6082-6090, April 2009.

[Bae06] Baekkelund, Christian. “A Brief Comparison of Machine Learning Methods”, AI Game Programming Wisdom 3. Steve Rabin, ed., pp.617-631, 2006.

[Bai05] Bailey, Christine, and Michael Katchabaw. “An Experimental Testbed to Enable Auto-Dynamic Difficulty in Modern Video Games”, Proceedings of the 2005 GameOn North America Conference, Montreal, Canada, 2005.

[Ban00] Banerjee, Bikramjit, and Sandip Sen. "Selecting partners", Proceedings of the Fourth International Conference on Autonomous agents, pp 261 - 262, 2000.

[Bar07] Barber, K. Suzanne, Jaesuk Ahn, Suratna Budalakoti, David DeAngelis, Karen K. Fullam, Chris L. D. Jones, Xin Sui. "Agent Trust Evaluation and Team Formation in Heterogeneous Organizations", Proceedings of AAMAS-07, pp 1356-1357, 2007.

[Bar02] Barnes, Jonty, and Jason Hutchens. “Testing Undefined Behavior as a Result of Learning”, AI Game Programming Wisdom. Steve Rabin, ed., pp.615-623, 2002.

[Ben09] Benton, J., Minh Do, and Subbarao Kambhampati. “Anytime Heuristic Search for Partial Satisfaction Planning”, Artificial Intelligence, Vol. 173, pp. 562-592, April 2009.

[Bou96] Boutilier, Craig. “Planning, Learning and Coordination in Multiagent Decision Processes”, Theoretical Aspects of Rationality and Knowledge, pp. 195-201, 1996.

[Bow02] Bowling, Michael, Rune Jensen, and Manuela Veloso. “A Formalization of Equilibria for Multiagent Planning”, AAAI Workshop on Planning with and for Multiagent Systems, 2002.

[Bow04] Bowling, Michael, Brett Browning, and Manuela Veloso. "Plays as Effective Multiagent Plans Enabling Opponent-Adaptive Play Selection", Proceedings of the International Conference on Automated Planning and Scheduling, 2004.

[Bow06] Bowring, Emma, Milind Tambe, Makoto Yokoo. “Multiply-Constrained Distributed Constraint Optimization”, Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems, 2006.

[Boy09] Boyle, Liz, Fiona Hancock, Matt Seeney, and Laz Allen. “The Implementation of Team Based Assessment In Serious Games”, 2009 IEEE Conference in Games and Virtual Worlds for Serious Applications, pp. 28-35, 2009.

[Bra09] Brafman, Ronen I., and Carmel Domshlak. “Preference Handling-An Introductory Tutorial”, AI Magazine, Vol. 30, pp. 58-86, Spring 2009.

[Bra07] Brandt, Felix, Felix Fischer, Paul Harrenstein, and Yoav Shoham. “A Game-Theoretic Analysis of Strictly Competitive Multiagent Scenarios”. IJCAI-07, pp.1199-1206, 2007.

[Bro02] Brockington, Mark. “Level-Of-Detail AI for a Large Role-Playing Game”. AI Game Programming Wisdom, pp. 419–425, Charles River Media, Hingham, Massachusetts, 2002.

[Bry06] Bryant, Bobby D. “Evolving Visibly Intelligent Behavior for Embedded Game Agents”, PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2006.

[Bui97] Bui, Hung H., Svetha Venkatesh, and Dorota Kieronska. “A Framework for Coordination and Learning among Team of Agents”, Proceedings of the Third Australian Workshop on Distributed AI, 1997.

[Cha04] Charles, Darryl, and Michaela Black. “Dynamic Player Modelling: A Framework for Player-Centred Digital Games”, Computer Games: Artificial Intelligence, Design and Education, pp.29-35, 2004.

[Cha91] Charniak, Eugene. “Bayesian Networks without Tears”, AI Magazine, Vol. 12 No. 4, pp. 50-63, 1991.

[Che08] Chevaleyre, Yann, Ulle Endriss, Jérôme Lang, and Nicolas Maudet. “Preference Handling in Combinatorial Domains: From AI to Social Choice”, AI Magazine, Vol. 29, pp. 37-46, Winter 2008.

[Cla94] Clarke, Edmund M., Orna Grumberg, and David E. Long. “Model Checking and Abstraction” ACM Transactions on Programming Languages and Systems, Vol. 16 No. 5, pp. 1512-1542, 1994.

[Cut08] Cutumisu, Maria, and Duane Szafron. “A Demonstration of Learning Behaviours for Non-Player Characters in Computer Role-Playing Games”, 18th Canadian Conference on Intelligent Systems, Windsor, Canada, May 27-31, 2008.

[Daw02] Dawson, Chad. "Formations", AI Game Programming Wisdom, Steve Rabin, ed., pp.272–281, 2002.

[Dia04] Dias, M. Bernardine, "TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments", Ph.D. Dissertation, Robotics Institute, Carnegie Mellon University, January, 2004.

[Fal03] Falke, William Joseph II, and Peter Ross. "Dynamic Strategies in a Real-Time Strategy Game", Lecture Notes in Computer Science, Vol. 2724, pp.1920-1921, 2003.

[Fri07] Friedman, Thomas L. “The World is Flat: A brief history of the twenty-first century”, Version 3, Douglas & McIntyre Ltd, Vancouver, BC, 2007.

[Fur01] Fürnkranz, Johannes. “Machine learning in games: a survey”, Machines that learn to play games, Advances in Computation: Theory and Practice, Vol. 8, pp. 11-59, 2001.

[Gas05] Gaston, Matthew E., and Marie desJardins. “Agent-Organized Networks for Dynamic Team Formation”, Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 230-237, 2005.

[Gas04] Gaston, Matthew, E., John Simmons, and Marie desJardins. "Adapting Network Structures for Efficient Team Formation", Proceedings of AAMAS-04, Workshop on Learning and Evolution in Agent-based Systems, 2004.

[Gil07] Gilpin, Andrew, and Tuomas Sandholm. “Lossless Abstraction of Imperfect Information Games”, Journal of the ACM, Vol. 54, No. 5, 2007.

[Gol05] Gold, Aliza. “Academic AI and Video Games: A Case Study of Incorporating Innovative Academic Research into a Video Game Prototype”, Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games. Piscataway, NJ: IEEE, 2005.

[Gru05a] Gruenwoldt, Leif, Stephen Danton, and Michael Katchabaw. “Creating Reactive Non Player Character Artificial Intelligence in Modern Video Games”, Proceedings of the 2005 GameOn North America Conference, Montreal, Canada, 2005.

[Gru05b] Gruenwoldt, Leif, Stephen Danton, and Michael Katchabaw. “A Realistic Reaction System for Modern Video Games”, Proceedings of the Digital Games Research Association Conference, Vancouver, Canada, 2005.

[Hei08] van der Heijden, Marcel, Sander Bakkes, and Pieter Spronck. "Dynamic Formations in Real-Time Strategy Games", IEEE Symposium on Computational Intelligence and Games, pp. 47-54, 2008.

[Her05] van den Herik, Jaap, Jeroen Donkers, and Pieter Spronck. “Opponent modelling and commercial games”, IEEE 2005 Symposium on Computational Intelligence and Games, pp. 15–25, 2005.

[Hou04] Houlette, Ryan. “Player Modeling for Adaptive Games”, AI Game Programming Wisdom 2. Steve Rabin, ed., pp.557-566, 2004.

[Hua96] Huang, Cecil, and Adnan Darwiche. “Inference in Belief Networks: A Procedural Guide”. Journal of Approximate Reasoning, Vol. 15, No. 3, pp. 225–263, 1996.

[Huy06] Huynh, Trung Dong, Nicholas R. Jennings, and Nigel R. Shadbolt. "An Integrated Trust and Reputation Model for Open Multi-Agent Systems", Journal of Autonomous Agents and Multi-Agent Systems, Vol. 13, No. 2, pp 119–154, 2006.

[Jen95] Jennings, Nicholas R. "Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems Using Joint Intentions", Artificial Intelligence, 1995.

[Jon06] Jones, E. Gil, Brett Browning, M. Bernardine Dias, Brenna Argall, Manuela Veloso, and Anthony Stentz. "Dynamically Formed Heterogeneous Robot Teams Performing Tightly-Coordinated Tasks", Proceedings of the International Conference on Robotics and Automation, 2006.

[Jun05] Jung, Hyuckchul and Milind Tambe. “On Communication in Solving Distributed Constraint Satisfaction Problems”, International Central and Eastern European Conference on Multi-Agent Systems, 2005.

[Koe04] Koenig, Sven. “A Comparison of Fast Search Methods for Real-Time Situated Agents”, Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 864-871, 2004.

[Kra03] Kraus,Sarit, Onn Shehory, and Gilad Taase. "Coalition Formation with Uncertain Heterogeneous Information", Proceedings of AAMAS-03, pp 1-8, 2003.

[Lai00] Laird, John E. and Michael van Lent. “Human-level AI's Killer Application: Interactive Computer Games”, AAAI Fall Symposium Technical Report, North Falmouth, Massachusetts, pp. 80-97, 2000.

[Lad08] Ladebeck, Manuel. “Applying Dynamic Scripting to "Jagged Alliance 2"”. Diploma thesis, TU Darmstadt, Knowledge Engineering Group, 2008.

[Lan10] van Lankveld, Giel, Pieter Spronck, Jaap van den Herik, and Matthias Rauterberg. “Incongruity-Based Adaptive Game Balancing”, Advances in Computer Games. 12th International Conference, pp. 208-220, Springer-Verlag, Berlin Heidelberg, Germany, 2010.

[Lar02] Laramee, Francois Dominic. “Genetic Algorithms: Evolving the Perfect Troll”, AI Game Programming Wisdom. Steve Rabin, ed., pp.629-639, 2002.

[Lid04] Liden, Lars. “Artificial Stupidity: The Art of Intentional Mistakes”, AI Game Programming Wisdom 2. Steve Rabin, ed., pp.41-48, 2004.

[Lu05] Lu, Hongen. “Team Formation in Agent-Based Computer Games”, ACM International Conference Proceeding Series, Vol. 123, pp. 121-124, 2005.

[Lud08] Ludwig, Jeremy. “Extending Dynamic Scripting”, PhD thesis, Department of Computer and Information Science, University of Oregon, 2008.

[Lud09] Ludwig, Jeremy, and Arthur Farley. “Examining Extended Dynamic Scripting in a Tactical Game Framework”, Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference, 2009.

[Mac09] Mackworth, Alan K. “Agents, Bodies, Constraints, Dynamics, and Evolution”, AI Magazine, Vol. 30, Iss. 1, pp. 7-28, Spring 2009.

[Mah04] Maheswaran, Rajiv T., Jonathan P. Pearce, and Milind Tambe. “Distributed Algorithms for DCOP: A Graphical-Game-Based Approach”, 17th International Conference on Parallel and Distributed Computing Systems, 2004.

[Man06] Manslow, John. “Practical Algorithms for In-Game Learning”, AI Game Programming Wisdom 3. Steve Rabin, ed., pp.599-616, 2006.

[Mar08] Marecki, Janusz, Tapana Gupta, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. “Not All Agents Are Equal: Scaling up Distributed POMDPs for Agent Networks”, Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems, 2008.

[Mar93] Markovitch, Shaul, and Yawn Sella. “Learning of Resource Allocation Strategies for Game Playing”, IJCAI-93, Vol 2, pp. 974-979, 1993.

[McN04] McNaughton, Matthew, Jonathan Schaeffer, Duane Szafron, Dominique Parker, and James Redford. "Code Generation for AI Scripting in Computer Role-Playing Games", Challenges in Game AI Workshop at AAAI, pp. 129-133, 2004.

[Mii06] Miikkulainen, Risto, Bobby D. Bryant, Ryan Cornelius, Igor V. Karpov, Kenneth O. Stanley, and Chern Han Yong. “Computational Intelligence in Games”, In Gary Y. Yen and David B. Fogel (editors), Computational Intelligence: Principles and Practice. IEEE Computational Intelligence Society, 2006.

[Mod03] Modi, Pragnesh Jay. “Distrubuted Constraint Optimization for Multiagent Systems”, PhD thesis, Faculty of the Graduate School, the University of Southern California, 2003.

[Nar05] Narayek, Alexander, Robert Fourer, Eugene C. Freuder, Enrico Giunchiglia, Robert P. Goldman, Henry Kautz, Jussi Rintanen, and Austin Tate. “Constraints and AI Planning”, IEEE Intelligent Systems, Vol. 20, No. 2, pp. 62-72, 2005.

[Pea07] Pearce, Jonathan P. “Local Optimization in Cooperative Agent Networks”, PhD thesis, Faculty of the Graduate School, the University of Southern California, 2007.

[Pea06] Pearce, Jonathan P., Rajiv T. Maheswaran, and Milind Tambe. “Solution Sets in DCOPs and Graphical Games”, Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems, pp. 577-584, 2006.

[Pol10] Policarpo, Daniel, Paulo Urbano, and Tiago Loureiro. “Dynamic Scripting Applied to a First-Person Shooter”, Proceedings of the Fifth Conference on Information Systems and Technologies, 2010.

[Pon07] Ponsen, Marc, Pieter Spronck, Hector Muñoz-Avila and David Aha. “Knowledge Acquisition for Adaptive Game AI”. Science of Computer Programming, pp. 59-75. Springer. The Netherlands, 2007.

[Pon04] Ponsen, Marc. “Improving Adaptive Game AI with Evolutionary Learning”, MSc thesis. Delft University of Technology. Delft, the Netherlands, 2004

[Pri10] Price, Robert G., and Scott D. Goodwin. “Team Formation with Heterogeneous Agents in Computer Games”, Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI Press, pp. 1957-1958, 2010.

[Pri05] Price, Robert G. “Forward Checking in the Primal and Dual Constraint Graphs”, MSc thesis, Department of Computer Science, The University of Windsor, 2005.

[Pyn03] Pynadath, David V., and Milind Tambe. “An Automated Teamwork Infrastructure for Heterogeneous Software Agents and Humans”, Journal of Autonomous Agents and Multi-Agent Systems, Vol. 7, pp. 71-100, 2003.

[Pyn02] Pynadath, David V., and Milind Tambe. "The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models", Journal of Artificial Intelligence Research, Volume 16, pp 389-423, 2002.

[Ric09] Rich, Charles, and Candace L Sidner. “Robots and Avatars as Hosts, Advisors, Companions, and Jesters”, AI Magazine, Vol. 30, Iss. 1, pp. 29-41, Spring 2009.

[Ros97] Rosin, Christopher Darrell. “Coevolutionary Search Among Adversaries”, PhD thesis, Department of Computer Science and Engineering, The University of California, San Diego, 1997.

[Ros08] Rossi, Francesca, K Brent Venable, and Toby Walsh. “Preferences in Constraint Satisfaction and Optimization”, AI Magazine, Vol. 29, pp. 58-68, Winter 2008.

[Sce05] Scerri, Paul, Alessandro Farinelli, Steven Okamoto, and Milind Tambe. “Allocating tasks in extreme teams”, Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, The Netherlands, pp. 727-734, 2005.

[Sch09] Schwalb, Robert J. “Party Building: From Basics to Themes”, Dragon Magazine, Issue 373, pp. 24-34, 2009.

[Sha10] Sharifi, Amir Ali, Richard Zhao, and Duane Szafron. “Learning Companion Behaviours Using Reinforcement Learning in Games”, Proceedings of the Sixth Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 69-75, 2010.

[Smi09] Smith, Michael J., and Marie desJardins. "Learning to trust in the competence and commitment of agents", Journal of Autonomous Agents and Multi-Agent Systems, Vol. 18, Number 1 pp 36-82, 2009.

[Spr09] Spronck, Pieter. “Pieter Spronck's Neverwinter Nights Modules”. http://ticc.uvt.nl/~pspronck/nwn.html as viewed September 22, 2009.

[Spr06] Spronck, Pieter, Marc Posen, Ida Sprinkhuizen-Kuyper, and Eric Postma. “Adaptive Game AI with Dynamic Scripting”, Machine Learning, Vol. 63, No. 3, pp. 217-248, 2006.

[Spr05] Spronck, Pieter. “Adaptive Game AI”, PhD thesis, Maastricht University Press, Maastricht, The Netherlands, 2005.

[Spr04] Spronck, Pieter, Ida Sprinkhuizen-Kuyper, and Eric Postma. “Enhancing the Performance of Dynamic Scripting in Computer Games”. Entertainment Computing - ICEC, Lecture Notes in Computer Science Vol. 3166: pp. 296–307, 2004.

[Sta06] Stanley, Kenneth O., Bobby D. Bryant, Igor Karpov, and Risto Miikkulainen. “Real-Time Evolution of Neural Networks in the NERO Video Game”, Proceedings of the Twenty-First National Conference on Artificial Intelligence, Meno Park, CA: AAAI Press, pp. 1671-1674, 2006.

[Sta05] Stanley, Kenneth O., Bobby D. Bryant, and Risto Miikkulainen. “Evolving Neural Network Agents in the NERO Video Game”, Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games. Piscataway, NJ: IEEE, 2005.

[Stu06] Sturtevant, Nathan, and Michael Bowling. “Robust Game Play Against Unknown Opponents”, Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 713–719, 2006.

[Szi09] Szita, István, Marc Ponsen, and Pieter Spronck. “Effective and Diverse Adaptive Game AI”, IEEE Transactions on Computational Intelligence and AI in Games, Vol. 1, No. 1, pp. 16-27, 2009.

[Tam00] Tambe, Milind, David V. Pynadath, Nicolas Chauvat, Abhimanyu Das, and Gal A. Kaminka. “Adaptive Agent Integration Architectures for Heterogeneous Team Members”, Proceedings of the International Conference on MultiAgent Systems, pp. 301-308, 2000.

[Tam97] Tambe, Milind. “Towards flexible teamwork”, Journal of Artificial Intelligence Research, Vol. 7, pp.83-124, 1997.

[Tha06] Thawonmas, Ruck, and Syota Osaka. “A Method for Online Adaptation of Computer-Game AI Rulebase”, Proceedings of the 2006 ADM SIGCHI International Conference in Computer Entertainment Technology, 2006.

[Tim07] Timuri, Timor, Pieter Spronck, and Jaap van den Herik. “Automatic Rule Ordering for Dynamic Scripting”, The Third Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 49-54, 2007.

[Tim06] Timuri, Timor. “Automatic Rule Ordering for Dynamic Scripting”, MSc thesis, Maastricht, The Netherlands, 2006.

[Tou02] Touzour, Paul. “The Evolution of Game AI”, AI Game Programming Wisdom. Steve Rabin, ed., pp. 3-15, 2002.

[Tsa93] Tsang, Edward. “Foundations of Constraint Satisfaction”, Academic Press Inc, San Diego, 1993.

[Whi06] White, Chris, and David Brogan. “The Self Organization of Context for Learning in Multiagent Games”, Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 92-97, 2006.

[Yis04] Yiskis, Eric. “Finite-State Machine Scripting Language for Designers”. AI Game Programming Wisdom 2, pp. 319—325, Charles River Media, Hingham, Massachusetts, 2004.

[Yon07] Yong, Chern Han, and Risto Miikkulainen. “Coevolution of Role-Based Cooperation in Multi-Agent Systems”, Technical Report AI07-338, Department of Computer Sciences, The University of Texas at Austin, 2007.

[Zha09] Zhao, Richard, and Duane Szafron. "Learning Character Behaviors using Agent Modeling in Games", Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 179-185, 2009.

[Zin06] Zinkevich, Martin, Michael Bowling, Nolan Bard, Morgan Kan, and Darse Billings. “Optimal Unbiased Estimators for Evaluating Agent Performance”, Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI), pp. 573–578, 2006.

Vita Auctoris

NAME:	Robert George Price
PLACE OF BIRTH:	Windsor, Ontario.
YEAR OF BIRTH:	1975
EDUCATION:	General Amherst High School, Amherstburg, Ontario
	1989-1994
	University of Windsor, Windsor, Ontario
	1994-1998 B.Sc.
	University of Windsor, Windsor, Ontario
	2002-2005 M.Sc.
	University of Windsor, Windsor, Ontario
	2005-2011 Ph.D.