An Adaptive Automatic Pilot for Role Playing Games 5.3

Rob Collaris

November 30, 2004
Contents

1 Introduction  
1.1 RPG/online games ........................................ 5  
1.2 AI research in the field of commercial computer games .... 7  
1.3 Research subject ........................................... 9  
1.4 Significance of the research ................................ 10  
1.5 Problem statement and Research questions .................. 10  
1.6 Thesis structure ............................................. 10

2 User Modeling in Games  
2.1 How to define an RPG user .................................. 11  
2.2 Related research ............................................. 12  
2.2.1 Multi Agent Systems ................................. 12  
2.2.2 Rule Based Systems ................................ 13  
2.2.3 Neural Network .................................... 13  
2.2.4 Finite State Machines ............................... 13  
2.3 Possible solutions .......................................... 14  
2.3.1 Finite State Machine ................................. 14  
2.3.2 Rule based ........................................... 18  
2.3.3 Neural network .................................... 19  
2.4 Conclusions ................................................. 20

3 Implementing an Automatic Pilot in NWN  
3.1 The predefined user models ................................ 22  
3.2 The environment ........................................... 24  
3.3 The perceptron .............................................. 25  
3.3.1 Learning ............................................. 25  
3.3.2 The update function ................................ 26  
3.3.3 Validating ............................................ 27

4 Experiments  
4.1 Success criteria ............................................ 29  
4.2 Set-up of the experiments ................................ 30  
4.2.1 The chosen perceptron inputs ....................... 30
List of Figures

1.1 A screenshot of Never Winter Nights an example RPG. . . . 6
2.1 The stage transition flow. . . . . . . . . . . . . . . . . . . . . . . . 17
2.2 A neural net to choose the correct user action . . . . . . . . . . . 20
2.3 A perceptron for choosing a user action . . . . . . . . . . . . . . . 20
3.1 Pseudocode snippet from the ‘Coward model’ . . . . . . . . . . 23
3.2 The environment used for testing. . . . . . . . . . . . . . . . . . . 24
3.3 The opponents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
3.4 Pseudocode perceptron learning . . . . . . . . . . . . . . . . . . . . 27
List of Tables

2.1 The 5 stages and their transition conditions. . . . . . . . . . . 14
2.2 An example rule base . . . . . . . . . . . . . . . . . . . . . . . 18
2.3 A percentage based rule base . . . . . . . . . . . . . . . . . . 19

3.1 Predefined User Models . . . . . . . . . . . . . . . . . . . . . . 23
3.2 The user settings . . . . . . . . . . . . . . . . . . . . . . . . . . 25

4.1 Input variable importance table . . . . . . . . . . . . . . . . . 30
4.2 Predefined User Models . . . . . . . . . . . . . . . . . . . . . . 31
Chapter 1

Introduction

"In the Forgotten Realms region of Faerun, the northern city of Neverwinter is stricken with a deadly plague. The sickness is spreading rapidly from its origin in the beggars’ quarter, and despite a quarantine being imposed, more and more citizens are falling victim to what has become known as the Wailing Death. A plea for help is sent to the allied city of Waterdeep where the wizard Khelben Blackstaff Arunsun arranges for the delivery of some rare magical creatures that might provide a cure. They reach the troubled city in a heavily guarded caravan, but unknown assailants attack the Academy where they are held and release them. At the behest of Lord Nasher and the young Tyrist paladin, Aribeth de Tylmarande, the adventurers of the city set out to recapture the dangerous creatures of Waterdeep, cure the Wailing Death, and unravel the conspiracy of Neverwinter’s mysterious betrayers."

Thus, the story of Neverwinter Nights, a Role Playing Game, begins. In this chapter we will give an introduction of the domain in which the research will take place. First, an explanation of RPG games is given. Second the reader will be introduced into the world of commercial gaming. Third the research subject will be explained followed by the importance of the research.

1.1 RPG/online games

Role Playing Games (RPG) are commercial computer games, that are mostly based upon fictional worlds and stories. An RPG enables a player to become someone he or she would like to be in real life but due to real world constrications is not possible. The particular game used in this thesis is Never

---

1Bioware’s Co-Lead Designer for Neverwinter Nights, Rob Bartel
Winter Nights (NWN)\textsuperscript{2}, a game that contains all RPG characteristics. Figure 1.1 shows a screenshot from NWN. In this screen the user is casting a spell named "Burning Hands", this creates a wave of fire that burns the opponents. Typically, in a RPG the user is centered in the middle of the screen and remains there. At the bottom of the screen you can see the "Quickslots" which are used to quickly access spells or other inventory items. On the right side are the user characteristics, option menu and the user’s "friends".

\textbf{Figure 1.1: A screenshot of Never Winter Nights an example RPG.}

When starting an RPG you normally begin with choosing your character’s race, such as Human, Elf or Dwarf. Each race has its own special abilities e.g. Strong, healthy, Smart, Dexterity and Charisma. These differences between the races give the game a more natural feeling. Humans differ from dwarves not only in body shape but also in character and abilities. Besides the race, the user has to choose what type of role he wishes to play. He\textsuperscript{3} can, for example, choose between types such as Wizard, Mage, Druid or Swordsman. By choosing the type of role, the user defines what kind of actions and spells are available to him. When a user chooses to be a Wizard he will be able to cast spells but will not be able to handle a sword as

\textsuperscript{2}A game developed by Bioware, http://nwn.bioware.com/
\textsuperscript{3}Even though research pointed out that most RPG players are female [10] the user and the enemies are referred to as 'he' but should be read as he or she, from here on.
skilled as a Swordsman. Deciding what type to play influences the gaming experience of the user and the game tactics used. By game tactics we mean the manner in which a user approaches an enemy and tries to defeat him.

Each game starts with introducing the user to the environment by a tutorial, this makes the user more comfortable with spells and weapons. The goal of NWN is to solve a mystery which is to find out who infested the capital city with the plague. To reach that goal, the player has to fulfill all sorts of quests. Quests are assignments, such as riddles and rescues, received from Non Player Characters (NPCs) during the game. An NPC is a pre-programmed character placed in the game which has the objective to either attack or to aid the player. These quests help to bit by bit solve the mystery which can be seen as the main quest. To complete these assignments the user must fight with enemies in some cases. When such a fight is over and a foe is defeated, the user gains experience points. As soon as a certain threshold of experience points is gained the player advances one level, which makes it possible to use higher level spells and/or weapons.

Most RPGs, like Neverwinter Nights, provide an option to play the game online. Playing online gives the user the chance to connect to a game server on the internet and play with other human players. This gives the advantage of dynamic game play and interaction between players, due to the fact that they can construct a strategy by talking to each other.

1.2 AI research in the field of commercial computer games

In the field of AI-research there are many fields of investigation, traditional type of games such as boardgames. But with the rise of commercial computer games, new types such as RPGs, RTS, FPS and Simulation games, the demand for AI changes.

Defeating only low level enemies gets boring, being a high-level character, so the level of the enemy should rise together with the player. Just increasing the level of the enemy does not increase the entertainment in game play, but making a foe "smarter" might do so. An example of how to make a NPC more "intelligent" is given by Pieter Spronck [9]. Spronck explains a way to adjust the NPCs intelligence to match the players intelligence by way of dynamic scripting. We do not claim that the game is as smart or as creative as a human it only learns what the best counteractions are for a given user. Thus a solution is to adjust the game to the user making sure he or she is challenged but not always defeated and/or victorious. Imagine playing a game in which you do not know if an opponent is an NPC or another human player, which is a sort of a Turing test, it would certainly improve the gaming experience [10]. Some AI techniques are already applied in commercial
computer games; we will give some examples.

AI developers are getting interested in Commercial Computer Games (CCGs) such as RPGs, because these games often involve game AI. There is a difference however between scientific AI and game AI. Game AI means that the artificial NPC opponents react to the user actions in a way that is perceived as being intelligent. Many of the techniques used to improve the illusion of intelligence have nothing to do with true ”intelligence”, but involve ”cheats,” such as giving NPCs the ability to see through walls, or ”pretend” by letting NPCs ”talk” to each other but completely ignore what is said.

However, changes are coming up [5]: AI researchers are getting more interested in game AI as the gaming industry is willing to use AI techniques more. Since the CPU is releaved of secondary tasks like graphics and sound, due to fast graphics cards, there is more calculating power for good algorithms to be evaluated real-time. More and more game programmers are starting to put an experiment environment in their games. Users can make own modules using scripts and AI implementations. Another change comes from AI students, like myself, who are used to playing computer games, and have all kind of ideas how to make the NPCs smarter. The main and most important driving force for developing smart game AI still is commercial. A game may look as real as possible, if nobody likes to play it due to weak or unconvincing NPCs it does not get sold and no money is earned.

While concentrating in this thesis on RPGs, we want to mention that AI is not only applied in RPGs but to other fields of computer games as well. These other types are Real Time Strategy (RTS), First Person Shooters (FPS) and Simulation games.

Real Time Strategy Games

Real-time strategy games, such as Warcraft III: Frozen Throne, form a category of strategy game that is played in ”realtime”. There are no ”turns” in a true RTS - players all move at once. The user has to command an army and conquer the enemy by defeat. In these games several AI techniques are used, for instance path finding [6, 18], the user can select a points on the map to which his army has to move. However, the AI needs to find the shortest path between the current and the selected point, based on the 3D information included in the map. Using the pathfinding technique lets the armies move across the map in a logical sense. In an RTS the NPC can surprise as well and does not wait for the user find and destroy him, but actively searches the opponents with the intention of destroying them. Moreover, planning algorithms [17] are used to make sure the right sequence of actions is carried out with the right post conditions. Other used techniques from AI are flocking [15], used to keep the group together in a logical movement, and Finite State Machines [15] to decide what the next action should be.
The latest introduced technique is Multi Agent System [11] this is used for combing forces and learning from the user.

First person Shooter

First Person Shooter games such as Unreal Tournament 2004, are domains in which Adaptive Team AI [3, 12] is investigated. An FPS is a game played from the first-person point of view. This means that you typically see what is around you, but you do not see yourself, except perhaps for the barrel of your gun. This genre lets the player be a kind of Rambo using all sorts of weapons to eliminate the enemy with various techniques. A user can hide, sneak, distract, throw explosives, and can use pistols and machine guns. In this type of games the enemy NPC should work as a team, effectively seek and use cover, know how to suppress and flank, to use sight and sound convincingly for hunting down their target, to retreat when necessary and not be too predictable. This is done by use of several AI-techniques [16] such as neural networks, agents, scripting, extensible AI, finite state machines and fuzzy logic.

Simulation games

Simulation games such as Roller Coaster Tycoon and The Sims 2 are games that, for example, let the user build an amusement park and control it. Setting a finishing threshold at the beginning of the game such as an amount of money, the success of the park, number of rides and a time period. The user has to create an amusement park ride and maintain it for a given period trying to fulfill the goal of the game level. The AI techniques used in this genre are path finding, decision trees, fuzzy logic and artificial life.

1.3 Research subject

Besides the advantages of playing RPGs online as described above there actually are some disadvantages too. There is the unlikely threat of a ‘connection interrupt’. However, a more common problem is a player who needs to pick up a telephone or go to the toilet; you can’t pause an online game. When a user is disconnected or logged out, his character is normally cut off from the game server. However, most games have a rule to make sure players do not quit during an intense battle and leave the character in the game for several minutes. The reason for introducing this rule is to avoid the abuse of logging out. Unfortunately for the honest players, it creates a sense of irritation. Imagine being left helpless, defenseless and open for attacks by NPCs that do not take into account you having a connection interrupt. The only thing the user can do is to reconnect as fast as possible and to
hope he is still alive by the aid of other human players. Playing online is a very important feature in game development at present, so every new game that hits the market will surely offer this option. To prevent the problems mentioned above, an Automatic Pilot would be helpful. It could replace the user while on the phone, toilet or disconnected.

1.4 Significance of the research

The threats of online gaming are clear, only the solution has to be found. For a gamer it would be great to have an automatic pilot at hand. It increases the value of the game simply by improving the gaming experience. However the Automatic Pilot, if introduced, should be invisible to the user whilst learning and should show results quickly. Main interest for AI research lies in optimizing the learning time and modeling the user. Besides the interest of the user another part is relevant. Modeling an user in an RPG has the advantage that you learn from, understand and perhaps can aid the user during game time.

1.5 Problem statement and Research questions

In 1.3 the research subject is given, from this context the following problem statement is constructed:

How can, with the aid of AI techniques, an Automatic Pilot be constructed that mimics a user behavior in an RPG.

This problem statement leads to eight research questions. The first four concern creating an Automatic pilot: What should it do? How should it do that? What is expected from the Automatic Pilot? And, what AI techniques are available to fulfill this job. The second four defines what is necessary for modeling the user: What characterizes the user? What actions are available to him? And why does he take certain actions? How can we mimic the user with an Automatic Pilot.

1.6 Thesis structure

Here the outlay of the thesis is explained.

In chapter one the environment of the thesis is commented, followed by chapter two in which the research subject is emphasised, then the research structure is explained in chapter three chapter four investigates the results of the research and finally in chapter five the conclusions are given.
Chapter 2

User Modeling in Games

As stated in section 1.2 there is plenty AI research going on in the field of commercial gaming. Much is done on developing the opponent AI to improve the gameplay. However, we want to model the user by online learning, which is new for RPGs. Therefore we will elaborate on the different approaches used on the subject of user modeling. We want to create a model of the user, to do so we need to define a user, his actions and motivations. To realize this we divided this chapter into three parts, in section 2.1 we try to explain what a user is then in section 2.2 we show what research on user modeling is done already and finally section 2.3 gives some solutions to our research questions.

2.1 How to define an RPG user.

A user can be defined by its actions and reasons for his actions. To model a user one has basically two options, use the history of actions taken by the user or create a basic profile of several user types and classify a user based on current perceptions. Trying to reason about why a user took certain actions, is hard to do in a non-deterministic, continuous environment. A gamer has some characteristics that make him unique and in time he will grow form a newbie to an expert. Therefore, we want to use a method that learns from the user, follows this development, never stops learning and adapts to new strategies.

Modeling an user is depends on the tasks the model should fulfill, there are many different parts of a user that can be modeled, e.g., user interface preferences, reasoning in games, a psychological model, emotions or information interest. For this thesis we want to model the actions of an RPG user based on a given environment, which constraints the modeling part to action modeling. Therefore it is necessary to define what actions are available to a user and what action is chosen. In Neverwinter Nights the user can choose between 33 basic actions at any time. However there are only
five combat related actions, ‘flee’, ‘attack’, ‘cast spell’, ‘summon help’ and ‘heal’, each action has an identifier number. Detecting what action is taken should not be difficult.

2.2 Related research

2.2.1 Multi Agent Systems

Bandini, Manzoni and Vizzari, [13] use an RPG to create a user model for hiring new employees. In their approach, instead of learning from the user, they interpret every answer a user gives and accordingly change the score of level descriptors. Level descriptors, such as ‘courage’ and ‘aggression’, are used to assign a understandable value to abstract archetypes, such as orphan or warrior. Changing these levels can be interpreted the same way as adjusting weights in a Neural Network. Level descriptors are introduced to prevent users to influence the outcome by controlling their answers, and being evaluated as the needed archetype for the job. Meaning that by knowing what answers to give to become the right archetype, has no influence on the level descriptors. Finally, every descriptor has a value and by use of this value each archetype is scored. Depending on the selected archetype a suitable applicant is chosen for the vacancy. The authors use a Multi Agent System (MAS) to communicate between different parts of their system. Four different agents are used: an user interface wrapper, database wrapper, user profiler and a story flow manager. Currently, even though only developed as prototype, the MAS profiler already showed potential for selecting suitable candidates.

Ojo, Rahman and Longe, [1] use seven agents to model users: Personal Agent (PA), Interface Agent (IA), Control Agent (CA), Service Agent (SA), Result Agent (RA), User Knowledge Base Agent (UKBA) and the User Modeling Agent (UMA). According to the authors, user models may be constructed in two basic ways. The first is through the acquisition of stereotypical information while the alternative is through the analysis of user-system interaction history. To create a user model they combine these two ways with the aid of the agents. Each agent has a distinct role in the process of modeling, following a communication structure. The result is that the seven agents combined provide the user with an user adapted interface, environment and personalized services. The concept of MAS oriented user modeling, is described but not implemented. The authors them self conclude that apart from technical challenges, which involve the development of supervised and unsupervised learning components, ethical issues like right of users to privacy of their personal information may demand users access to user-models developed by the system.
2.2.2 Rule Based Systems

A. J. Champandard [4], states that Rule Based Systems (RBS) are a useful tool in modeling an user, because they are flexible, capable of performing low-level control as well as decision making. Moreover that, RBS are very simple to implement and understand, which makes them easy to extend, to a certain amount of rules, and customize according to the problem at hand. They are also data driven (declarative knowledge), which makes them an ideal lightweight alternative for scripting languages, as long as not to many language features are required. RBS are at their best when they retain syntactic and semantic simplicity.

2.2.3 Neural Network

Besides implementing an RBS, A. J. Champandard [4], implemented a Neural Network (NN) as well. The author claims that a NN gives an even better outcome then an RBS. But they used the results of the RBS to train the NN and it took them about 1500 runs and using 26 hidden layers to get it as good as possible, so it would not function as a real-time algorithm. A great advantage of an NN is that it can deal more fluently with fuzzy input, making it more continues and smoother behavior.

Sklar [14] uses a multilayer perceptron to train agents which copy or teach a users online Tron game playing behavior (called clones, peers, and instructors) using backpropagation. Advantages, the author claims, of NNs are that they offer methods for huge data sets, weak background knowledge and sometimes even unlabeled data. As inputs she takes the environment variable and the output gives the action to take, the goal was to reach a predefined winning rate. Training was done with results of 250 games played by 58 different players. The result was that NNs are capable of simulating complex behavior of the players, using simpler functions. One note of the author is that implicit knowledge as modeled by a trained NN can only partially be explained and therefore remains rather incomprehensible to the user.

2.2.4 Finite State Machines

A. LaMothe [8], uses Finite state machines, or FSMs, to recognize a stage a user is in. This are abstract models that can be implemented either in hardware or software. FSMs are not really machines in the mechanical sense of the word, but rather abstract models with a set of ‘states’ that can be traversed. Within these states, the FSM not only has a special set of outputs but remembers the state and can transition to another state, if and only if a set of inputs or premises are met.

Gmytrasiewicz and Lisetti [7] explain how to model a user’s emotions by use of an FSM. The authors claim that in general, a users personality is
said to be a set of user specific characteristics which persist through time, while emotions are rather short termed. They want to use the FSM state machine to describe an Agents personality by use of transition between emotions. Each state is an emotion and each transition is a change in emotion. This could be useful for human computer interaction when a user is getting angered the agent knows what to do to relax the user again. A problem with this approach, the authors found, is the difference between stages, it is hard to express the difference between some emotions and therefore though to correctly change states.

2.3 Possible solutions

In this section we will evaluate the different parts of how we could use the approaches from section 2.2 to model an user in an RPG. Modeling a user is divided into 2 parts based on our findings in the research section. The first part is to select what stage a user is in, this means that we want to find out what the user is doing at the moment, e.g., resting, fighting or walking. The second part is selecting the right action at the right moment if mimicking the user.

2.3.1 Finite State Machine

To model an user in the game we first have to detect what stage the user is in. This detecting can be done by use of a Finite State Machine (FSM) as discussed in section 2.2.4. An FSM can separate different stages in an encounter with use of predefined classes. These stages will be one of the following ‘No action’, ‘Preparations’, ‘Place a character’, ‘Combat’ and ‘Flee’ (Table 2.1). When these stages are combined they constitute a strategy which the player utilizes. An advantage of setting the user states to five

<table>
<thead>
<tr>
<th>0:No action</th>
<th>1:Preparation</th>
<th>2:Placing</th>
<th>3:Attack</th>
<th>4:Flee</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Buying or selling weapons</em></td>
<td><em>Healing</em></td>
<td><em>Citing enemy</em></td>
<td><em>Casting a spell (aggressive)</em></td>
<td><em>Transporting</em></td>
</tr>
<tr>
<td><em>Standing still</em></td>
<td><em>Resting</em></td>
<td><em>Walking (remain in sight of enemy)</em></td>
<td><em>Fighting</em></td>
<td><em>Walking away from enemy</em></td>
</tr>
<tr>
<td><em>Talking to NPCs</em></td>
<td><em>Casting a spell (defensive)</em></td>
<td><em>Looking for traps</em></td>
<td></td>
<td><em>Leave battle</em></td>
</tr>
<tr>
<td><em>Disabling traps</em></td>
<td><em>Pausing</em></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: The 5 stages and their transition conditions.
stages, is that by selecting a stage the user actions are separated. Meaning that each stage has its own ‘most likely actions’ to be taken by the user. Therefore each stage has a small set of actions to learn. Instead of putting all actions in one model, there are five different models to be made, one for each stage. This shortens the learning time for each stage and by that the total learning time for the user model.

In table 2.1 an overview of the five stages and their actions is given, meaning that an action in a column transitions any stage to the corresponding stage column. For example, when an user is in the ‘Attack’ stage and uses the action ‘transport’ he immediately is placed in the ‘Fleeing’ stage. As can be seen, each stage has his own specific characteristics that makes it possible to identify the different stages. The stages are explained in the following paragraphs.

Stage 0: No action

When a player is wondering around in the game just investigating and walking he is in stage 0, he is just moving from one place to another. In this stage there are several different options for the player, he can keep walking or go into preparations mode (stage 1). He can encounter a surprise attack an needs to place his character (stage 2), attack the enemy (stage 3) or run for his live and flee (stage 4). By doing one of these actions a typical surprised reaction of the player is modeled and kept in the Rule Base.

Stage 1: Preparation

When a player is in the first action stage, Preparation, he is getting ready for surprise attacks or maybe for a planned attack. Usually, a player starts preparing by resting the character to restore the health points, regain ‘mana’ or prepare other spells. But other ways of preparation are also available such as creating a companion, hiring a henchmen or equipping stronger weapons and better armor. The diversities of preparation methods makes it a hard stage to recognize but it will certainly be useful. Because every user prepares in his own way for a battle. From this stage a player can go into one of the 4 other stages, going back to stage 0 can be done by a timeout when a user takes no other action. Stage 2 can be reached by moving toward a foe without attacking yet, stage 3 is making an aggressive move toward an enemy by way of spells or weapons. Going to the flee stage is not an option because no danger was present, so, the user will go back into stage 0.

Stage 2: Placing

Reaching the second stage, Placing, a user, based on the assumption that this stage is normally preceded by the preparation stage, is ready for action
and only needs to determine what the best strategy is. The initial step of a good strategy is finding a good place to initiate an attack and make sure it is in your advantage. Thus placing a character gives a user the advantage of surprise and the possibility to use destructive spells and long-range weapons. Another important part is the ability to see what type, amount and strength the enemy has, to make sure you attack the right person first. Reaching this stage could be done during any time in the game, for example after a surprise attack or when fighting and the character is not made for close combat. This is an important stage because this determines the main strategy planned by the user during the following combat.

Stage 3: Attack

In the most active stage, Attack, the user unleashes all his fury on the enemy trying to bring him down as soon as possible. This may be done by use of several ways, he could try to kill the strongest person in the group, meaning the person who inflicts the most damage upon the player. Or first start with the quickly destroyed ‘rubbish’ to have more time and concentration for the ‘big boss’. In this stage the user applies all his combat strategies such as spells, weapons and moves to lose as less health points as possible and meanwhile killing the NPCs. Perhaps he chooses to lure a foe with him so he has only one enemy to fight in stead of 3 or 4. Every user has a distinct strategy and this part combined with stage 2 define a user strategy and make a user recognizable thus these two stages need the most attention and right classification. Reaching this stage may be preceded by several different stages such in which ‘No action’ and ‘Preparation’ are consequences of a surprise attack. In this case the user will probably try to distract or flee to go into stage 1 or 2. Reaching the attack stage from fleeing means the enemy is not shook of during escape and the only option left is fighting.

Stage 4: Fleeing

Finally, the last stage in a battle, Fleeing, is only useful if the enemy is to strong or the user is left without weapons, spells or health restore options. Thus this stage can be reached from every other stage but normally will be preceded by the ‘Attack’ stage. In some cases the NPC can surrender thus making an end to the battle without killing the enemy, this would be seen as fleeing because the foe is still alive. But because of a time threshold the user will get into stage 0 after a while.

In figure 2.1 the flow is drawn of the stage transitions. The bold arrows indicate a normal flow a user will follow during the game. However sometimes unexpected events occur and this changes the flow, that is why we have to create a stage recognition system. Stage recognition To identify the different stages explained above, there are several options. Let the player
tell in which stage he is, so the pc only needs to register the strategies, a sort of supervised learning. When utilizing this option the player is somewhat forced to actively be an active part of the learning cycle. This leaves the possibility open for errors because a player can forget to name a stage and the wrong rule base is updated. A better way of stage recognition is to let the Auto Pilot learn the differences between the stages and recognise them by use of a classifier. These stages will then be bound to four different rule bases each with its own description of the rules complying with the right stage. These rule bases will be filled while the player is in the corresponding stage with data about the used tactics and environmental conditions. For each stage, a timer is activated such that when a threshold is reached the player automatically goes into the next stage or to stage 0. This timer is used to make sure that some action is always taken even when a player is in the midst of combat. The Automatic Pilot must know what stage to go to. Another transition manager, besides the timer, is a set of actions that is taken by the Automatic Pilot. This action belonging to a stage make the FSM automatically go to the corresponding stage. Thus to recognize a stage the characteristics of a stage transition have to be analysed, these are basically shown in Table 2.1.

To get these stage specifics, it is possible to use environmental conditions
of the game such as the player health, casting spells, in combat or enemy sightings. But these have to be identified to belong to a stage. This can either be done by hand or by use of a classifier, such as a Neural Network. When a stage is properly recognized, it is time to learn the user tactics. The two main requirements for this are learning should be fast and accurate. Of what use would an Automatic Pilot be if it takes hours or even days to learn from the user. Why should the Automatic Pilot learn from the user, if the user does not recognize himself in the actions taken.

### 2.3.2 Rule based

There are several different options to make a computer learn from a user. The first one is to use a rule base, as discussed in section 2.2.2, meaning a database which contains information about environmental conditions needed to judge a situation. The intention is to keep the number of variables as small as possible without losing valuable information, e.g. to keep the result as good as possible. As an example, imagine a character playing a wizard, while wandering around he encounters a foe and finds himself under attack. On this moment some variables are stored in the rule base such as the health (how many life points a player has) and mana points (points needed to use a magic spell) of the player and the type (the role a foe plays), health and strength (the maximum number of health points a strike would cost the player) of the enemy (Table 2.2). When facing a foe the player must act swiftly and thus given the conditions the player applies a tactic also stored in the rule base.

<table>
<thead>
<tr>
<th>Player health</th>
<th>Player mana</th>
<th>Enemy type</th>
<th>Enemy health</th>
<th>Enemy strength</th>
<th>Tactic</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>300</td>
<td>Knight</td>
<td>100</td>
<td>25</td>
<td>Blizzard</td>
</tr>
<tr>
<td>425</td>
<td>0</td>
<td>Knight</td>
<td>100</td>
<td>25</td>
<td>Flee</td>
</tr>
</tbody>
</table>

Table 2.2: An example rule base

This rule base gives two examples of environmental conditions that vary only in mana points of the player, resulting in different tactics as shown in the last column. The next time when an encounter takes place and the player uses the Automatic Pilot, it will check the rule base for equivalent situations, and apply these tactics to mimic the user. When no comparable environmental conditions are found the most resembling one will be chosen to make sure the Automatic Pilot does take action. A different possibility to fill the rule base is to use percentages (Table 2.3), this has the advantage of being dynamic and flexible. In every RPG a player gains points for winning a battle or completing a quest, when a player reaches a set amount of points he will advance a level. Gaining a level mostly means that the health and
Player health | Player mana | Enemy type | Enemy health | Enemy strength | Tactic
---|---|---|---|---|---
100% | 100% | Knight | 100% | 25 | Blizzard chain
100% | 0% | Knight | 100% | 25 | Flee

Table 2.3: A percentage based rule base

mana points increase and extra spells are available. This would make the rule base in table 2.2 useless because now the player has different values for the environmental conditions. That is why in table 2.3 percentages are introduced, when gaining a level the rule base does not need to change. When engaging in an encounter the rule base is checked for similar situations and will produce the corresponding tactic. Besides that the rule base is easily divided into chunks, meaning that every variable is placed in a range. For example when an environmental condition is set at 87% and in an encounter that same variable ranges between 85% and 90% it is seen as the same value. A problem which could occur if filling the rule base this way is what to do when a player has the same encounter again but uses a different tactic. The solution is quite simple, give it an evaluation function so only the best survives. This evaluation function could for example check what strategy leaves the most healthpoints and/or wins the battle.

However using a rule base for modeling the user, is not applicable for our research. This is due to the complexity of the environmental inputs and the continuously changing of the selection values. Moreover, what should the rulebase look like in the beginning, should there be standard actions predefined or is the Rule Base empty. A disadvantage of a rule base is the offline learning demand.

### 2.3.3 Neural network

Besides the RBS from section 2.3.2 we evaluated the possibility to use a neural network (fig. 2.2) for choosing a strategy. To use a neural network you have to define possible inputs, the number of hidden layers and outputs of the network. The input will exist of the stage and the environmental conditions defined by variable selected by hand. The stage is recognized with the FSM and thus does not have to be found by the neural network. The outputs point to a set of actions the user can take. Combining this then the user is mimicked by correctly pointing to the corresponding action in the right stage.

Another idea for a neural network is a simple perceptron (Fig. 2.3), linking the inputs directly to the outputs. The advantage of this is that learning takes place quicker and more efficient. Here the weights will be adjusted by use of action prediction, the model predicts the outcome of a
user action given a situation, award the one who does it right. To select the best action, there are several options. The first option, would be to select the action with the largest value. The second option, is to select randomly by throwing a dice in which the weights stand for a percentage, giving high weights a bigger chance of being selected. The third option, is using a voting system [2] in which each high weighted action gets a vote, leading to executing the action with the highest vote. Possibly a multiplier could be introduced based on the weight of the model for in case several actions get the same amount of votes. The approach of using a perceptron for modeling an user, is the most appealing one. Due to the fact of the fast learning rate, and online learning capabilities.

2.4 Conclusions

In this chapter, we evaluated current research on user modeling in Role Playing Games. Moreover, we discussed the possible applicability of these ideas on our research, dividing the user modeling tasks into two parts, ‘Stage recognition’ and ‘Learning’. After evaluation of the stage recognition part,
we decided to leave this out of our user modeling. This can be justified, because we only want to concentrate on the user modeling part in which the user has to survive, that is during combat. Therefore we assume that the user is always in combat when training the user model. Implementing the FSM to recognize the stage a user is in, should be possible but is regarded as not relevant for this thesis.

Our conclusion is based on the criteria learning rate, best possible results in a short time and it should be an online learning algorithm. We evaluated the ideas of a rule base, Finite State Machine (FMS) and a neural network. We are aware of other possibilities but these seem the most suitable for this thesis. Choosing among these three was done by our selection criteria, a rule base has the disadvantage that it should be trained offline, a FSM does not learn at all it only selects states and a Neural Network learns online but slow.

We concluded that using a perceptron for user modeling in an RPG, is likely to be successful. However, using the perceptron to select an action is not sufficient we decided to select a predefined user model. The idea of using a user model is based on the archetypes discussed in section 2.2.1. This has the advantage of always having an action to perform, and not select the most used action for each encounter but the most appropriate one.
Chapter 3

Implementing an Automatic Pilot in NWN

This chapter explains the choices we made for implementing the Automatic pilot. First, section 3.1 will explain what predefined model are. Second, section 3.2 will explain the setting of the learning environment. Third, section 3.3 will explain the perceptron used and how it is implemented.

3.1 The predefined user models

For our perceptron outputs, we decided to create standard user models. Selecting models in stead of an action, is based on the idea of archetypes used by Bandini, Manzoni and Vizzari [13]. Their use of adjusting ‘Weights’ for selecting an archetype inspired us to use a similar approach. But in stead of selecting only a model we had to select an action, thus we implemented predefined models which, when chosen, select an appropriate action.

For defining these models we used three basic types from an RPG, ‘Warrior’, ‘Wizard’ and ‘Coward’ and their combinations (Table 3.1). These models completely differ from each other, however to make the models overlapping we introduced three extra model combinations. Each model chooses an action based upon the environmental inputs fitting its profile. The amount of six models used is chosen on base of empirical results, it should be sufficient to mimic the user correctly but not slow down the learning process.
Table 3.1: Predefined User Models

<table>
<thead>
<tr>
<th>Model nr.</th>
<th>Model type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Warrior</td>
<td>Only fights with weapons</td>
</tr>
<tr>
<td>2</td>
<td>Warrior Wizard</td>
<td>Prefers to fight with weapons, can cast spells</td>
</tr>
<tr>
<td>3</td>
<td>Wizard</td>
<td>Fights with spells</td>
</tr>
<tr>
<td>4</td>
<td>Wizard Coward</td>
<td>Fights with spells but is scared easily</td>
</tr>
<tr>
<td>5</td>
<td>Coward</td>
<td>Only fights if sure of winning</td>
</tr>
<tr>
<td>6</td>
<td>Coward Warrior</td>
<td>Fights with weapons but is scared easily</td>
</tr>
<tr>
<td>7</td>
<td>Test User</td>
<td>Used for testing experiments</td>
</tr>
</tbody>
</table>

For an impression of how a model looks like here is a snippet of the ‘Coward model’ code:

```
1 IF(enemy sighted) DO
2   IF(distance ≤ 20 feet) DO
3     IF(my health ≤ 90%) DO runAway(enemy)
4     ELSE rangedAttack(enemy)
5     ELSE
6       runAway(enemy)
7     ENDIF
8 ELSE
9   runAway(enemy)
10 ENDIF
11 ENDIF
```

Figure 3.1: Pseudocode snippet from the ‘Coward model’

As can be seen a model is basically an RBS, in this rule base it is easy to add or remove an IF statement resulting in a new model. The ‘Coward’ model has two IF statements which can be found in line 2 and line 3. Keeping the models easy to expand leaves the option to add extra predefined user models if needed. The advantage of this Rule Based construction is that fuzzy input is accepted. Meaning that the RBS ‘Archetypes’ always selects the same action given the same inputs within a range value. An example of this is given by line 3, if the health drops below 90% then runAway. This results in taking different fuzzy inputs, to activate the same rule. The last ELSE statement, line 8 in the code snippet, is added to ensure that if the
model is selected it will always perform an action in this case: IF none of 
the above complies then runAway.

In Table 3.1 a seventh model is shown ‘Test user’. This model is added 
for testing the perceptron, it is used to create actions for a given input. 
Moreover, Neverwinter Nights is a game played in realtime which means 
that a test run should be done manual. This is the reason we introduced this 
seventh model, so we know always the same action is selected for an input. 
The definition of this test user is constructed according to the Archetypes 
code, however new extra rules are added to ensure diversity of the predefined 
models. An advantage of knowing every action the test user takes, is that 
we can calculate a succesrate for our learning algorithm. A disadvantage of 
using a test user, is the transparency of the actions and the reasons for these 
actions. A real user does not always take clear actions, which can lead to 
different result sets for testing the test user and a real user.

3.2 The environment

The environment (Fig. 3.2) we created for learning and testing our perceptron is kept as basic as possible. Meaning that we only implemented and placed what we need in the area. At start the user stands in front of a sign looking over a pit. This ‘shingle’ is used to control the settings of the area. When the users speaks to the sign he can set the parameters used for training. The available options are shown in Table 3.2. Option 2.1 is used for setting all parameters for testing/validating. Option 2.3, is used for setting the runtime. This is can be anything ranging 6 to 60 minutes with an 6 
minutes interval. Selecting, e.g., 30 minutes means that the perceptron has 
15 minutes for learning and 15 minutes for validating. Meaning the runtimes 
available from the conversation menu, represent the real runtime.

Figure 3.2: The environment used for testing. On the left a screenshot from the environment in the game, on the right a top view of the screenshot. A) The opponent B) The testmodel C) The shingle to set parameters D) The user
Table 3.2: The user settings

<table>
<thead>
<tr>
<th>Option nr.</th>
<th>Option</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select model</td>
<td>Select a model for default usage, used for testing.</td>
</tr>
<tr>
<td>2</td>
<td>Set parameters</td>
<td>Set the testing parameters.</td>
</tr>
<tr>
<td>2.1</td>
<td>Set learning</td>
<td>Set the perceptron to learning or validating.</td>
</tr>
<tr>
<td>2.2</td>
<td>Reset weights</td>
<td>Set the values of the weights to basevalue.</td>
</tr>
<tr>
<td>2.3</td>
<td>Select runtime</td>
<td>Choose a learning and validating time, for testing.</td>
</tr>
<tr>
<td>2.4</td>
<td>Ready setting</td>
<td>Return to main menu.</td>
</tr>
<tr>
<td>3</td>
<td>Start fight</td>
<td>Begins a new fight.</td>
</tr>
<tr>
<td>4</td>
<td>End fight</td>
<td>Ends the current fight.</td>
</tr>
<tr>
<td>5</td>
<td>Exit</td>
<td>Closes the conversation.</td>
</tr>
</tbody>
</table>

For learning and validating our perceptron we needed enemies as well, therefore we created four different types of foes as shown in Fig. 3.3. The opponents are selected based on their strength, that is healthpoints and maximumdamage in one blow. This is done for having a range of enemy types available, which represent the game NPCs a user will encounter. Each one of the enemies has his own class descriptor, set to level 5. The dwarf for instance has the class ‘Fighter’ this means he is a level 5 fighter, enhancing his weapons skills. For selecting what opponent the test user will face we used a random generator. This was done for mimicking a real ingame situation as good as possible and to prevent overtraining on one type of foe.

Now when a battle is started a foe and test user are created. As soon as they sight each other they will engage in combat. When one of these two characters dies, the other is destroyed as well and the fight is over. However, as long as the runtime is not passed, a new random opponent and a new test user is created. This will continue until the runtime is over or the user ends the battle manually. The advantage of this cycle is that it enables the perceptron to keep learning and validating, without the need of user intervention.

3.3 The perceptron

*Implementing the perceptron is divided into two parts, in section 3.3.1 we will look at the learning part. Section 3.3.2 covers the update function for the perceptron learning. In section 3.3.3 the validating part is reviewed.*

3.3.1 Learning

We implemented the learning procedure of the perceptron in such a way that it could be easily expanded for testing purposes. It should be easy to add or change an input type. When learning, the perceptron perceives the environmental inputs such as health, size and distance between the user and the
enemy. With these inputs all predefined models and the test user present an action to be taken, line 3 and 4. These actions are chosen from the predefined model profile, as explained in section 3.1. The next step is comparing the model actions with the action the test user chose, line 5. When these actions correspond, the weight of the connection between the input and the correct model is raised, if the actions differ the connection weight is lowered.

### 3.3.2 The update function

For updating the connection weights we constructed a function that uses all inputs and gives them impact power according to their value. What we would like to see is that a weight value is 1 if the model correctly predicted the user action and the weight should be 0 if not. This is to ensure that with the given input the right model is chosen.

First we normalize the inputs:

\[ \mu = \frac{I_i}{I_i(\text{max})} \]  

(3.1)

in which \( \mu \) is the normalized value for input \( i \), \( I_i \) is the current input and \( I_i(\text{max}) \) is the maximum value the input can have. This calculation ensure that each input value has a value between 0 and 1 and has as much impact as possible without overrating the connection.

In case the action predicted and used correspond, raise the weight to approach the wanted value of 1 as close as possible:

\[ W_i(\text{New}) = W_i(\text{Old}) + \lambda * ((1 - W_i(\text{Old})) * \mu) \]  

(3.2)

if the actions do not correspond, lower the weights to approach 0 as close as possible:

\[ W_i(\text{New}) = W_i(\text{Old}) - \lambda * (W_i(\text{Old}) * \mu) \]  

(3.3)
1 IF (Detect enemy) DO
2 FOR( model = 0 TO model = n) DO
3    modelAction := predict model action(environmental values)
4    userAction := user action taken
5    IF( modelAction == userAction) DO
6       adjustWeight(raise)
7    ELSE IF( modelAction $\neq$ userAction) DO
8       adjustWeights(lower)
9    ENDIF
10  ENDFOR
11 ENDIF

Figure 3.4: Pseudocode perceptron learning

In which $W_i(new)$ is the weight being updated, $W_i(Old)$ is the weight before update, $\lambda$ is a preset update factor and $\mu$ is the normalized inputvalue.

### 3.3.3 Validating

Option 1: Is to sum all the inputs multiplied with their weights, and set a value for a model. Selecting the highest value to be the winning model, meaning this model is chosen to select an action.

$$M_j = \sum I_i * W_i$$  \hspace{1cm} (3.4)

in which $M_j$ means the value of Model $j$, $I_i$ stands for input $I$ and $W_i$ is the weight corresponding to input $i$.

$$Max = M_j \ IF \ M_j \geq \ Max$$  \hspace{1cm} (3.5)

The highest model value $Max$ is selected by comparing each model value $M_j$ with max, if a higher value is found then that is set as max.

Option 2: Sum all the inputs multiplied with their weights, and set a value for a model. Using these model values as percentages for selecting a model by chance. For instance if models 1, 2 and 3 have the values of 0.3, 0.6 and 0.1, means that the chance of letting model 1 select an action is 30%.
Option 3: Sum all the inputs multiplied with their weights, and set a value for a model. Using these model values as votes for an action. For instance, if models 1, 2 and 3 have the values of 0.3, 0.4 and 0.3 and these models have selected an action. Then the selected action gets the corresponding model value as vote. Continuing the example, model 1, 2 and 3 have chosen ‘Spell attack’, ‘Melee attack’ and ‘Spell attack’. Every action that is chosen multiple times is awarded the total number of votes. In this example it would be ‘Spell attack’ = 0.6 and ‘Melee attack’ = 0.4 resulting in selecting a model that uses the action with the highest value, thus ‘Spell attack’.

tion gets means that the chance of letting model 1 select an action is 30%.
Chapter 4
Experiments

Implementing an Automatic Pilot is be successful if the user is well mimicked. However what should be the criteria for accuracy in mimicking the user, we will give the success criteria and the set-up of the experiments.

4.1 Success criteria

Measurement of success is a very important part of experimenting, if it is unknown what results imply they are useless. What we had to do was to find a way to qualify the results of the experiment and give them a measurement.

To find if our experiments are successful we defined a way to measure the success rate. Every model makes a prediction for a battle of what action it would choose, the test user we implemented chooses an action as well. To measure if an automatic pilot performs well, is merely a process of comparing the actions chosen by the test-user and the selected user-model. What we hope to find is a success rate above 90% because that would imply that we modelled the user correct. An accuracy of 90% would suffice due to the random factors in the game which are not controllable, such as the amount of damage done by one blow.

As a second test we could try an approach which uses a "Turing" like technique. We could place two human players in a room, and let them tell whenever they think the automatic pilot of the other player is activated. To make it more convincing there are no set time intervals for swapping and not signals are given to the other players. If they flawlessly are able to point out the Automatic Pilot and the human user time, then the Automatic Pilot is a complete disaster. However if the human players are not able to pinpoint who is active it could mean a succesfull implementation of the user model.

For this thesis we used the success criteria generated by comparing the chosen actions, even thought it would be useful as well to do the "Turing" testing we did not find the time to implement the extra code needed.
4.2 Set-up of the experiments

For testing our autopilot we created an environment as described in section 3.2 but what we used for testing is important as well. We had to choose the inputs, the model types and settings, and the amount of learning time/runs.

4.2.1 The chosen perceptron inputs

The inputs used in the experiments should be visible to the user as well. So we could not use invisible inputs such as exactly what spell is the opponent casting or how many spells has he remaining. That would be cheating, especially considering the fact that we are trying to mimic the user. However what input is available to the user and what is used differs among users so we had to choose our own types of input mostly used to judge a situation. From our own experience as RPG players we derived a list of importance for selecting an action.

<table>
<thead>
<tr>
<th>Rating</th>
<th>Input type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User Health</td>
<td>The current health of the user</td>
</tr>
<tr>
<td>2</td>
<td>Distance</td>
<td>The user to enemy distance</td>
</tr>
<tr>
<td>3</td>
<td>Enemy Health</td>
<td>The current health of the enemy</td>
</tr>
<tr>
<td>4</td>
<td>Challenge rating</td>
<td>Based on level, race and type</td>
</tr>
<tr>
<td>5</td>
<td>Enemy type</td>
<td>Wizard, Fighter or Monsters</td>
</tr>
<tr>
<td>6</td>
<td>Enemy race</td>
<td>Race specific characteristics</td>
</tr>
<tr>
<td>7</td>
<td>#enemies</td>
<td>The amount of enemies attacking</td>
</tr>
</tbody>
</table>

Table 4.1: Input variable importance table

As seen in table 4.1 we chose the user health first. The first concern of a user is to keep himself alive, so every action taken will be based on the current health status of the user. Ranked number two on our list is the distance between the user and the foe, used to judge what action should be taken if, to close, to far or far enough. This is necessary to select the attack action, close range means spells are useless, to far means moving closer and far enough means ranged weapons or healing oneself are usable. On the third place we used the health of the enemy, important to know if one last blow will suffice or that recharging is required. The remaining inputs are not considered to be useful in the automatic pilot, however are added for illustrating the environment inputs used by users. The inputs we used are the top three of this list, if needed expanding the inputs is possible.

4.2.2 The predefined models used

The number of predefined user models is set to seven, representing different user types including a random action user. We chose seven user models and
an eight test model with distinct motives as shown in table 3.1.

<table>
<thead>
<tr>
<th>Model nr.</th>
<th>Model type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Melee attack</td>
<td>Prefers to attack with melee weapons</td>
</tr>
<tr>
<td>2</td>
<td>Spell attack</td>
<td>Prefers to use spells in combat</td>
</tr>
<tr>
<td>3</td>
<td>Flee</td>
<td>Fights if 99% sure of victory else run away</td>
</tr>
<tr>
<td>4</td>
<td>Random action</td>
<td>Selects a random action to use</td>
</tr>
<tr>
<td>5</td>
<td>Only spells</td>
<td>Can choose only spell as action</td>
</tr>
<tr>
<td>6</td>
<td>Only melee</td>
<td>Can choose only melee as action</td>
</tr>
<tr>
<td>7</td>
<td>Only flee</td>
<td>Can choose only to flee as action</td>
</tr>
<tr>
<td>8</td>
<td>Test</td>
<td>Used for testing experiments</td>
</tr>
</tbody>
</table>

Table 4.2: Predefined User Models

The intention of our research is to let the perceptron choose the right model at the right time to represent the user. Therefore we think it is justified to select seven predefined user models, because they differ enough to select an appropriate action at the right time.

4.2.3 The testing settings

For testing our Automatic Pilot we had to run the simulations in Neverwinter Nights this means real time testing. Automated testing is done by setting a time step and a number of runs. Each time step, e.g. 12 minutes, half the time the Automatic Pilot learns from the actions of the user, the other half of the given time the learned actions are validated resulting in a success rating. However one time step is performed several runs, five times to be exact this is done to minimalize error due to negative and positive results. We used time steps of six minutes up to a maximum of one hour. These time limits are set because a demand of the Automatic pilot is that it should learn fast, besides that it takes a lot of time to test runs longer then one hour. So due to time restrictions and prerequisites of the learning algorithm we did not involve these values.
Chapter 5

Results

Taking a closer look at the results of the experiments

5.1 Results explanation

How are the results measured when is it a success and when a failure.

5.2 Research results

The results with an explanation

5.3 Discussion

Discussion about the results
Chapter 6

Conclusion

6.1 Recommendations

6.2 Further investigation
Bibliography


