

**INTELLIGENT BEHAVIOR FOR COMPUTER GAME
CHARACTERS USING PREDICTION AND LEARNING**

CS 297 Report
by
Leo Lee

Advisor: Dr. Chris Pollett
Department of Computer Science
San Jose State University
December 2004

ABSTRACT

This report details the work done and knowledge learned during the first semester of the author's thesis (CS 297). Although video game graphics has improved tremendously, artificial intelligence in games has taken a much smaller step forward. One of the most promising new prospects in game AI is for the computer to learn and adapt to the environment, and in particular to the player. The thesis aims to create a fighting game with an AI system capable of learning and adapting to the player. Four deliverables were attempted this first semester. All were completed with good results with the exception of deliverable three, which will be completed before next semester. Deliverables one and four explored three AI techniques: n-grams, string-matching prediction, and Hidden Markov Models in detail. Deliverable two was to generate a document recording the concept and ideas for the game, called Alpha Fighter. Deliverable three was to create character models and animations. From the work completed this semester, the author is equipped with the assets and knowledge necessary for implementation of the game next semester.

TABLE OF CONTENTS

Introduction	1
Motivation for Advanced Game AI	1
Overview of Thesis	1
Deliverable 1: Prediction Using N-grams and String-Matching	2
Deliverable 2: Game Concept	7
Deliverable 3: Character Modeling and Animation	9
Deliverable 4: Prediction Using Hidden Markov Models	11
Summary of Work Completed	14
Future Work	14
References	15

LIST OF TABLES AND FIGURES

Tables

Table 1. RPS game with string-matching prediction effectiveness survey.	6
Table 2a. Attack-attack payoff matrix.	8
Table 2b. Attack-defend payoff matrix.	8

Figures

Figure 1. N-gram word predictor main window.	3
Figure 2. N-gram word predictor model display window.	4
Figure 3a. RPS game with string-matching prediction, MFC version.	5
Figure 3b. RPS game with string-matching prediction, Java applet version.	5
Figure 4. Female character model.	10
Figure 5. Partially shown HMM used for HMM move predictor.	12
Figure 6. HMM move predictor applet.	13

1. INTRODUCTION

This report provides comprehensive details on the work performed in CS 297 for the author's thesis entitled, "Intelligent Behavior for Computer Game Characters Using Prediction and Learning." Section 2 presents the motivation behind the thesis, in particular, why the topic of choice is new and relevant. Section 3 describes the goal of the thesis and the expected final product. Sections 4 through 7 explain in detail the four CS 297 deliverables. Each of these sections is broken down into four subsections. The first subsection explains the relevance of the deliverable in the context of the thesis. The second subsection presents the goal of the deliverable. The third subsection details the methodology behind the deliverable, as well as the outcome. Lastly, the fourth subsection gives some concluding thoughts on the results of the deliverable. Section 8 summarizes all of the work performed this semester in CS 297. Finally, Section 9 outlines the work to be done next semester in CS 299.

2. MOTIVATION FOR ADVANCED GAME AI

Computer games have come a long way, especially in the area of graphics. However, artificial intelligence in games has not advanced nearly as far. Many games today still use the simplest techniques such as finite state machines (FSMs) and decision trees [11,12]. These simple methods have the advantage of being easy to develop and debug. However, as gamers demand more sophisticated AI, these techniques simply cannot keep up. A major drawback of these traditional AI techniques is that the programmer predefines all behavior. A FSM can be made extremely complex by adding tons of states, or even by introducing a hierarchy. However in the end, the behavior is still static. If a non-player character (NPC) is controlled by a FSM, its behavior cannot change to adjust to the environment of which the player is most notably a part of. In other words, the player is simply choosing preset paths, as opposed to being able to define new paths.

One of the most exciting prospects on the horizon in AI for games is the ability for NPCs to learn and adapt to their environment [13]. The ability to learn from mistakes and change one's strategy is the cornerstone by which a human gamer plays. The human player cannot cheat as the AI system can, by processing the opponent's input before making a decision. The human player can only observe the video and audio output of the game. In creating NPCs that can dynamically adapt to the changing environment, in particular to the player, the game would in turn become a more realistic and enjoyable experience for the player.

3. OVERVIEW OF THESIS

The final product of this thesis is a 3d fighting game for the PC called Alpha Fighter. This is a single player game. The game puts two fighters against each other. The fighters can damage each other by performing various attacks, as well as defend oneself using various blocking and dodging techniques. The first fighter to lose all of his or her health is the loser.

The NPC will be backed by an AI system that can learn and adapt to the player's actions. This will provide an opponent who is always at about the same level as the player, which makes the game's difficulty at an enjoyable level. More importantly, the NPC will act more like a human player would, learning from mistakes and developing new strategies based on observation. One can view this project as a Turing Test in the context of video games; that is, the goal is to make the NPC indistinguishable from a human controlled character. Some may argue that passing the Turing Test does not conclude the agent is intelligent. However in the realm of video games, for the NPC to exhibit human-like behavior is the mark by which the AI system is judged, since it is after all a human who is playing the game.

4. DELIVERABLE 1: PREDICTION USING N-GRAMS AND STRING-MATCHING

4.1 Motivation

The main objective at this point was to gain a broad perspective on the AI techniques available, in particular focusing on the methods with good potential for incorporating into the design of the Alpha Fighter AI system. Before starting this deliverable, extensive literature on game AI and AI in general were studied [1,3,5,6,7,8,9,14]. Afterwards, it was decided that two methods would be investigated further at this point, n-grams [1,6] and string-matching [9]. Being simple to implement, both of these techniques were good candidates as a first wave of experiments in designing the AI system. It should be kept in mind that in choosing an AI technique for a game, the designer must factor in the execution time. In many cases, accuracy must be sacrificed for the benefit of speed, as games must run in real-time. Despite their simplicity, both n-grams and string-matching prediction are both used in the games industry. This showed they were practical and relevant to Alpha Fighter. Therefore, knowledge of these techniques would be useful in designing the final AI system of Alpha Fighter.

4.2 Goals

The goal was to produce two small programs, one using n-grams and the other using string-matching. The n-gram program was to perform word prediction. To keep the program simple, neither the meanings nor parts of speech of the words were considered. The words were simply to be interpreted as a string of consecutive alphabetic characters. Using this program, the user would be able to input training text, construct an n-gram model with the input, and receive the program's prediction of the next word based on the statistical probability defined in the n-gram model.

The second program was to use string-matching to predict the next move in a Rock-Paper-Scissors (RPS) game. This would be a one player game against the computer, where the player could input one of three moves: rock, paper, or scissors. The computer would then respond with its choice. Rock defeats scissors, scissors defeats paper, and paper defeats rock. It is important to note that the computer does not use the player's

current move in deciding its move, doing so would constitute as cheating. The purpose of this program is to predict the player's next move, and based on that prediction, the computer would do the single counter-move that defeats the predicted move.

4.3 Implementation and Results

The n-gram word predictor was implemented using Microsoft Foundation Classes (MFC) as a Windows program. Figure 1, shows the main window of the program.

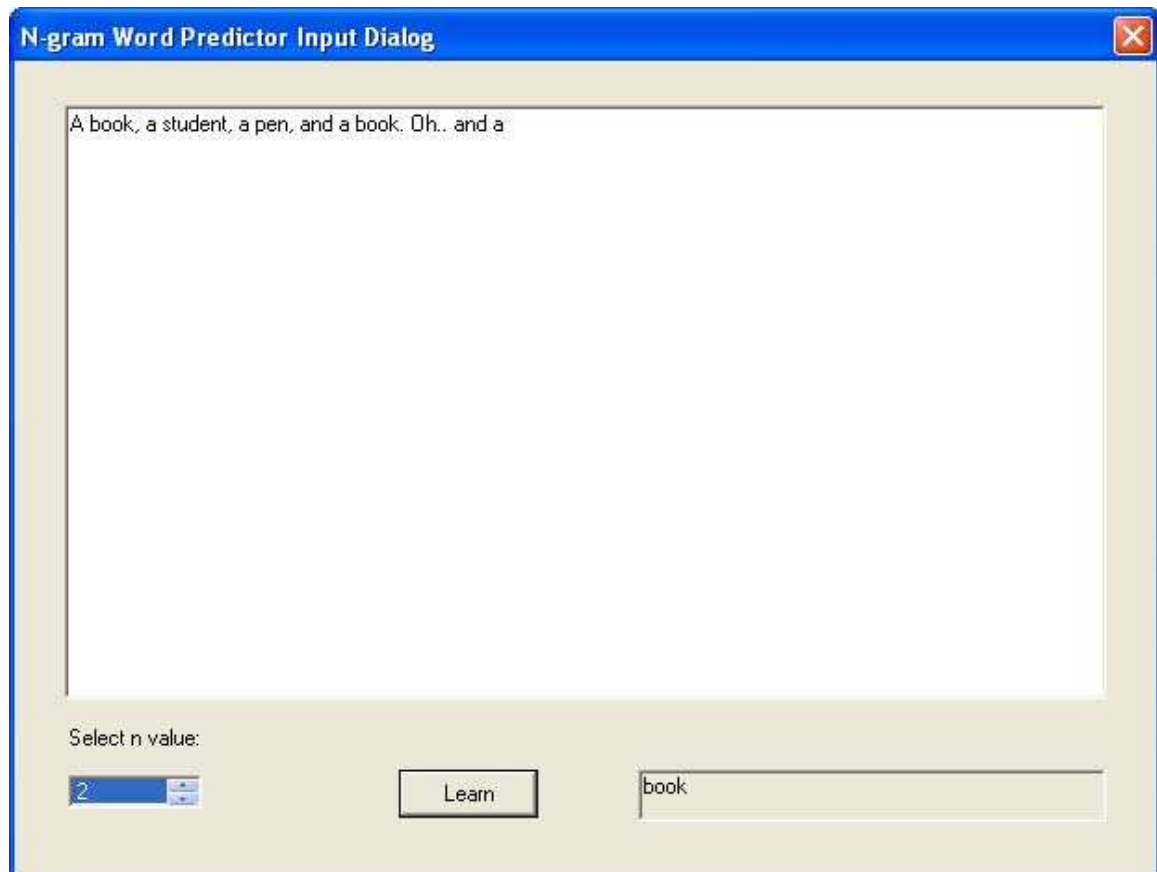


Figure 1. N-gram word predictor main window.

As shown, there is a large area for the user to input text. On the lower left, the user can choose the value of n . For example, choosing two creates a bigram, three a trigram, and so forth. The higher the value of n , the more precise the prediction will be. However, lower n values are more robust as higher values can suffer from overfitting (see section 7.1 for more detail). On clicking the “Learn” button, the program will train the existing n -gram model with the text in the text area. The program periodically updates the prediction, which is shown in the lower right. In the screenshot, it is clear that with a bigram model, the most likely word statistically is “book.” A bigram only takes into consideration the previous word, in this case, “a.” The model was trained using the text shown in the input text area. The word “book” appears twice after the word “a,” while all

other words only appear once. Therefore, the word “book” is twice as likely to come after “a” according to the statistical bigram model.

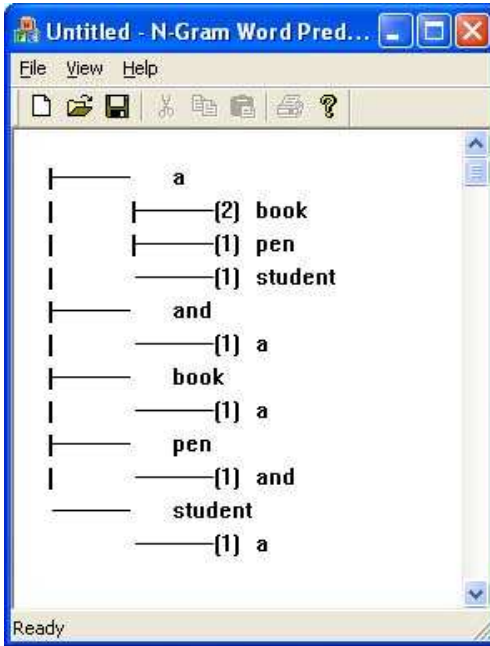


Figure 2. N-gram word predictor model display window.

In addition to the main window in Figure 1, the program also displays a model display window as shown in Figure 2. This window depicts the current n-gram model. As it corresponds with the training text and n value of Figure 1, the model is a bigram populated with the words in Figure 1’s input training text. The display, as well as the underlying model is represented in a tree structure. The tree is always complete and has a depth of n, where n is the n-gram n value. The display of the tree in Figure 2 is shown such that the deepest level, in this case 2, is furthest to the right. To predict the next word, the program does the following. It takes as input the previous n – 1 words. So in our example, that would be the single word “a.” It takes the first of these words and starting at the root traverses down the branch leading to that word. For higher values of n this process would be repeated for all words. In our bigram case, there is only level of traversal. At this point, the next level down contains leaf nodes. Each leaf node has a count as well as the word. Going back to the example, these leaf nodes would be “book,” “pen,” and “student.” These represent the words that have followed the word “a” in the training text. The count is the number of times that particular sequence has occurred. Therefore, the word with the highest count, in this case “book,” has the highest probability of coming after the input sequence of words, in this case “a.” Note that the actual probability does not need to be calculated.

The RPS program was initially done as an MFC application, but later converted to Java applet. Thus there are two fully functional versions, but the applet is more polished. Figure 3a shows the MFC version and Figure 3b shows the applet version.

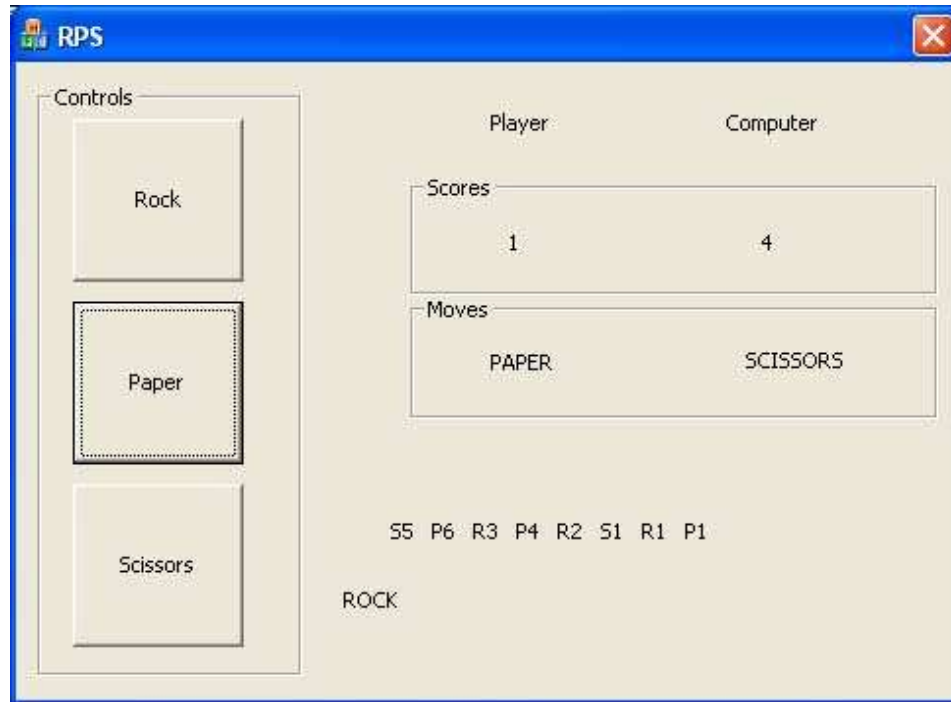


Figure 3a. RPS game with string-matching prediction, MFC version.

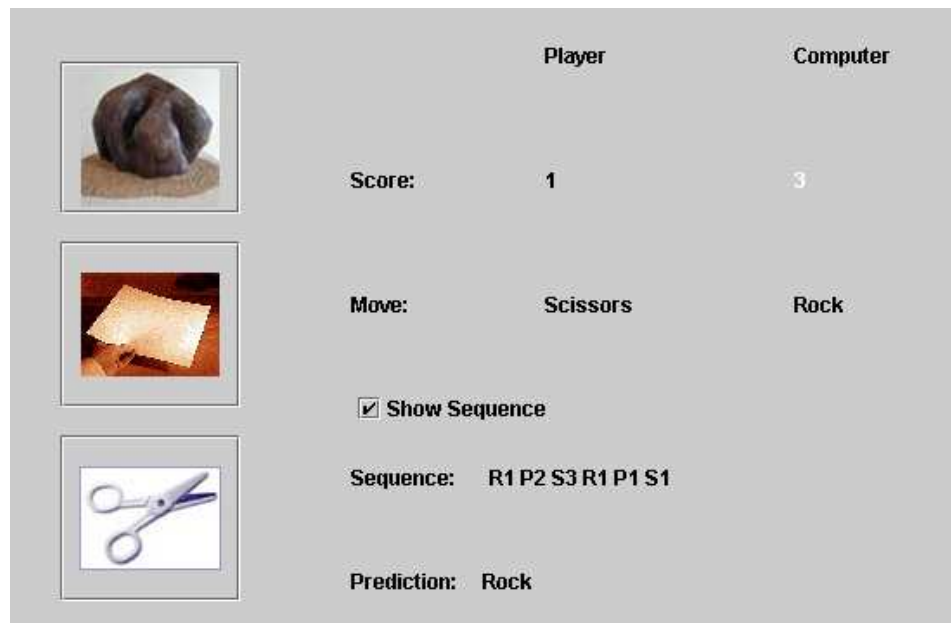


Figure 3b. RPS game with string-matching prediction, Java applet version.

The discussion will focus on the applet version since it is more refined. The user chooses a move by clicking the respective button on the left. On the right, the scores and last move of the player and computer are displayed. When checked, the “Show Sequence” checkbox will allow the display of the underlying string using in the string-matching

prediction scheme. It will also allow the display of the computer's prediction of the player's next move.

The string-matching algorithm from [9] finds the longest substring within a given string s that matches the end of s . This algorithm can be used for prediction simply by finding the end of the string match, and taking the next character as the prediction. For example, given the string *abcabc* the algorithm would find the longest match to be the first substring *abc*. The prediction would then be *a*, the immediately next character. In the RPS applet, the first character of the move name is used to represent the move. For example, *R* stands for rock. The sequence in Figure 3b shows the player has made the sequence of moves: rock, paper, scissors, rock, paper, and scissors. See [9] for details on the algorithm uses the numbers displayed. The algorithm finds the longest substring match to be the first rock, paper, scissors sequence, thus it predicts the next move to be rock. This in turn will cause the computer to choose paper for its next move, the counter to rock.

4.4 Remarks

Both of the programs were completed satisfactorily. The effectiveness of both prediction schemes was evident in the respective programs. Although these two methods for prediction may not be directly incorporated into the final Alpha Fighter AI system, the knowledge gained from this deliverable will undoubtedly have impact on the design of the system.

A small survey was taken using the RPS MFC program. Three people were asked to play around 20 rounds of RPS with the computer then to report the scores of both the player and computer. The results are summarized in Table 1. Two of the three participants decided to play multiple matches; that is, restart the program after a certain number of rounds. The match number is indicated in the first column following the name.

Table 1. RPS game with string-matching prediction effectiveness survey.

Player	Player Score	Computer Score	# Rounds	% Computer Won
Michelle 1	7	14	21	66.67
Michelle 2	15	10	25	40
Si Si 1	9	8	17	47.06
Si Si 2	6	9	17	52.94
Si Si 3	8	5	12	41.67
Young 1	11	9	20	45

On average, the computer won 49.1% of the time. If a strategy of random guessing were used instead, the expectation would be that the computer wins 33.3% of the time. That means there is a 15.8% improvement with the string-matching prediction strategy over random guessing.

5. DELIVERABLE 2: GAME CONCEPT

5.1 Motivation

Having focused on the AI system in Deliverable 1, it was important to take a broader look at the game as a whole. As such, a game concept document was needed to lay down the foundations for Alpha Fighter. The document would help to define the key elements of the game as well as paint an overview of the work to be done. Later work could simply build on the elements as described in this document.

5.2 Goal

The purpose of this deliverable was to write down the game concept. This is different from a game design document, which has much greater detail and can be between 50 to 300 pages long [15]. Due to the limited time frame of this thesis, writing a complete game design document was simply not feasible. Furthermore, since the focus is on the game AI, the game itself serves just as a medium to exhibit the AI system. In other words, although as much effort as possible will be put into creating a good game, the development of the AI system holds higher priority.

5.3 Implementation and Results

The document of game concept consisted of the following sections: concept, game-play, characters, style, payoff matrix, and concept art. In addition, a means of research section was subsequently added to explain the method by which the payoff matrix was created. The concept section presents the story behind the game. The game-play section describes what type of game Alpha Fighter is and its main feature, the AI system. The characters and style sections presents the characters and fighting styles that are planned to be included in the game.

The most interesting section is the payoff matrix section. A payoff matrix outlines the outcome of performing a certain action in a certain state. Two matrices were created, an attack-attack matrix (Table 2a), and an attack-defend matrix (Table 2b). In both matrices, the possible actions of one fighter, Fighter A, are labeled along the leftmost column, while the possible actions of the other fighter, Fighter B, are labeled along the top row. Each cell, c_{ij} , where i is the row and j is the column, shows how much damage each fighter sustains as a consequence of Fighter A performing the action of row i and Fighter B performing the action at column j . The number following the letter A indicates the damage to Fighter A, and similarly for Fighter B. Low numbers are favorable.

Table 2a. Attack-attack payoff matrix.

B A	high long fast	high long slow	high short fast	high short slow	low long fast	low long slow	low short fast	low short slow
high long fast	A1 B1	A0 B1	A2 B1	A0 B1	A1 B1	A0 B1	A2 B1	A0 B1
high long slow	A1 B0	A3 B3	A2 B0	A5 B3	A1 B0	A3 B3	A2 B0	A5 B3
high short fast	A1 B2	A0 B2	A2 B2	A0 B2	A1 B2	A0 B2	A2 B2	A0 B2
high short slow	A1 B0	A3 B5	A2 B0	A5 B5	A1 B0	A3 B5	A2 B0	A5 B5
low long fast	A1 B1	A1 B0	A1 B2	A1 B0	A1 B1	A1 B0	A2 B1	A1 B0
low long slow	A1 B0	A3 B3	A2 B0	A5 B3	A1 B0	A3 B3	A2 B0	A5 B3
low short fast	A1 B2	A0 B2	A2 B2	A0 B2	A1 B2	A0 B2	A2 B2	A0 B2
low short slow	A1 B0	A3 B5	A2 B0	A5 B5	A1 B0	A3 B5	A2 B0	A5 B5

Table 2b. Attack-defend payoff matrix.

B A	high long fast	high long slow	high short fast	high short slow	low long fast	low long slow	low short fast	low short slow
block high	A0 B0	A0 B0	A0 B0	A0 B0	A1 B0	A3 B0	A2 B0	A5 B0
block low	A1 B0	A3 B0	A2 B0	A5 B0	A0 B0	A0 B0	A0 B0	A0 B0
strafe left	A0 B0	A0 B0	A0 B0	A0 B0	A0 B0	A0 B0	A0 B0	A0 B0
strafe right	A0 B0	A0 B0	A0 B0	A0 B0	A0 B0	A0 B0	A0 B0	A0 B0
jump	A1 B0	A3 B0	A2 B0	A5 B0	A0 B0	A0 B0	A0 B0	A0 B0

The attack-attack payoff matrix shows what would happen if the two fighters simultaneously performed the same attack. This shows the effectiveness of each attack relative to every attack. The attack labels along the top and left are not specific fighting moves, but rather three attributes. These attributes are target, range, and speed. Each attribute has a binary value. Target can be high or low, range long or short, and speed fast or slow. So for example, a high, long and fast attack by Fighter A will always damage Fighter B by 1 point if Fighter B is doing a low, long and slow attack, regardless of the actual fighting move.

There were several reasons why attacks were put into classes as opposed to using the actual move. One reason was to simplify the game mechanics, since dealing with several classes is simpler than dealing with dozens of moves. Secondly, since the characters and their animations were not yet developed, it would have been unwise to settle on a set of moves at this point. Thirdly, classifying the moves would allow easier incorporation into the AI system later, as the attributes point out the important aspects of the movements. Finally, by generalizing multiple moves into a single class, the efficiency of the game logic and AI system will undoubtedly be increased.

Research was done to derive the two payoff matrices. Video of live martial arts tournaments and commercial fighting games were studied. It should also be noted that the author has eleven years of background in martial arts, so some of his experience and knowledge also influenced the final outcome of the matrices. The three attributes mentioned earlier: target, range, and speed, were determined by the research. It was observed that these three factors were the most significant in classifying the effectiveness of the attack relative to various other attacks and defenses.

5.4 Remarks

The resulting document was satisfactory in explaining the high level idea of the game. The most significant portion, the payoff matrices, will likely be used in the final game with little or no modification. At this stage, the design of the fighting mechanics of Alpha Fighter was basically complete.

6. DELIVERABLE 3: CHARACTER MODELING AND ANIMATION

6.1 Motivation

Having laid down the foundations for Alpha Fighter with the game concept document in Deliverable 2, it was time to begin building resources to be used in the game. It was important to begin the process of creating graphic resources, as the author had no prior experience with 3d modeling or animation.

6.2 Goal

For this deliverable, the task was to create the characters for Alpha Fighter as described in the game concept document. The characters would have to be modeled, textured, and animated.

6.3 Implementation and Results

Since the game will be using the Direct3D graphics library, it was important to ascertain the finished models could be loaded into a Direct3D environment. The native format for Direct3D is the x-file format. The author found a freely available x-file exporter from Maya.

Having no experience with 3d modeling, the author followed a training video for modeling in Maya to construct a female character. After two weeks, it was realized that the task was much more difficult than anticipated, and the model was nowhere near completion. It was decided that a much simpler model be created, sacrificing visual quality for the sake of time. A low quality model of a female character was finished (Figure 4), and work began on texturing the model. However at this point, the time allotted for working on this deliverable was over, and it was necessary to move on to the next task. It was decided that completion of the models was not crucial at this point, and that they could be completed over the winter break.

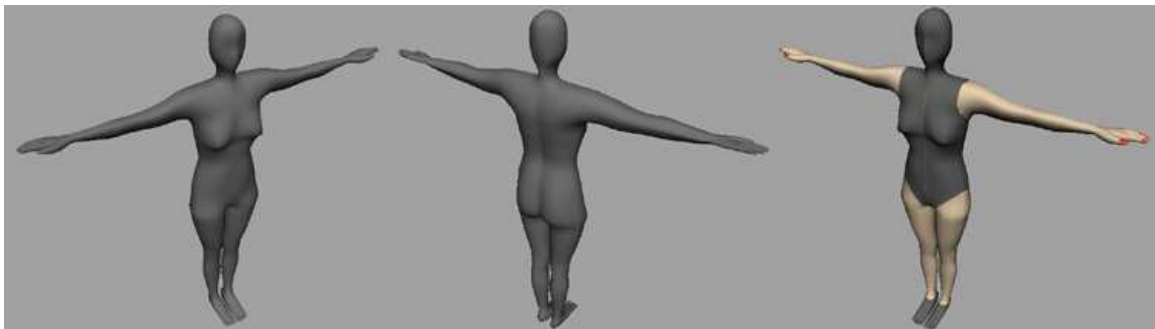


Figure 4. Female character model: untextured front (left), untextured back (middle), and partially textured front (right).

6.4 Remarks

This deliverable was not completed due to time constraints and the unpredicted difficulty of the task. The female character must still be completely textured. Then, animation for all of the moves must also be done. It is expected that once the first character is completed, other characters can be created easily by simply adjusting the first model. The plan is to complete the character modeling and animation during the winter break.

7. DELIVERABLE 4: PREDICTION USING HIDDEN MARKOV MODELS

7.1 Motivation

At the start of this deliverable, more AI research was performed, in particular on Hidden Markov Models (HMMs) and Dynamic Bayesian Networks (DBNs) [1,10,14].

Using high order n-grams can lead to overfitting, where a prediction cannot be made due to the basis of the prediction being too specific. For example in a trigram model, the basis for prediction is the sequence of the two previous states. If the model comes across a particular sequence that was never before seen, the model would not be able to make a prediction at all. To avoid overfitting, one could use lower order n-grams. It is clear that by using a unigram, where the prediction is simply the state that occurred the most number of times, a non-empty model will always be able to generate a prediction. However, by lowering the value of n, the model loses precision.

It was presented in [1] that the robustness of n-grams could be improved by combining multiple n-grams into a single HMM. This solves the overfitting problem by starting at the highest possible n-gram, then falling back to lower ones if a prediction cannot be made. In the worst case a unigram will be used. HMMs were a prime candidate for the AI system of Alpha Fighter. It was important to gain a solid understanding of them.

7.2 Goal

This purpose of this deliverable was to become familiar with HMMs by developing a prediction program similar to the first deliverable. This program would differ in two points. First, it would use a HMM as opposed to n-grams, although the HMM is actually made up of multiple n-grams. Second, the program would be put into the context of Alpha Fighter. Therefore, instead of predicting words, the program would predict the next move by the player.

7.3 Implementation and Results

The HMM used is based off of one in [1], and is partially shown in Figure 5. The valid moves are the characters *o*, *p*, *l*, and *;*. These correspond to high-weak, high-strong, low-weak, and low-strong attacks respectively. For the sake of readability, Figure 5 only shows the transitions from the *op* state; that is, what can happen after a player has pressed *o* followed by *p*. The full HMM has equivalent transitions for each possible, non-lambda state. As shown, from the *op* state, the player can go into the *po*, *pl*, or *p;* states. The three λ states in between are in effect how the n-grams are encapsulated inside this HMM. Each transition is denoted by an output followed by a colon, followed by the probability of taking that transition. The symbol ϵ is the null symbol and outputs nothing. We can see that from the λ_1 state, all the probabilities on the outgoing transitions are unconditional probabilities. This corresponds to the unigram model. Likewise, the λ_2

state corresponds to the bigram model (conditioned on one previous state) and λ_3 the trigram model (conditioned on two previous states).

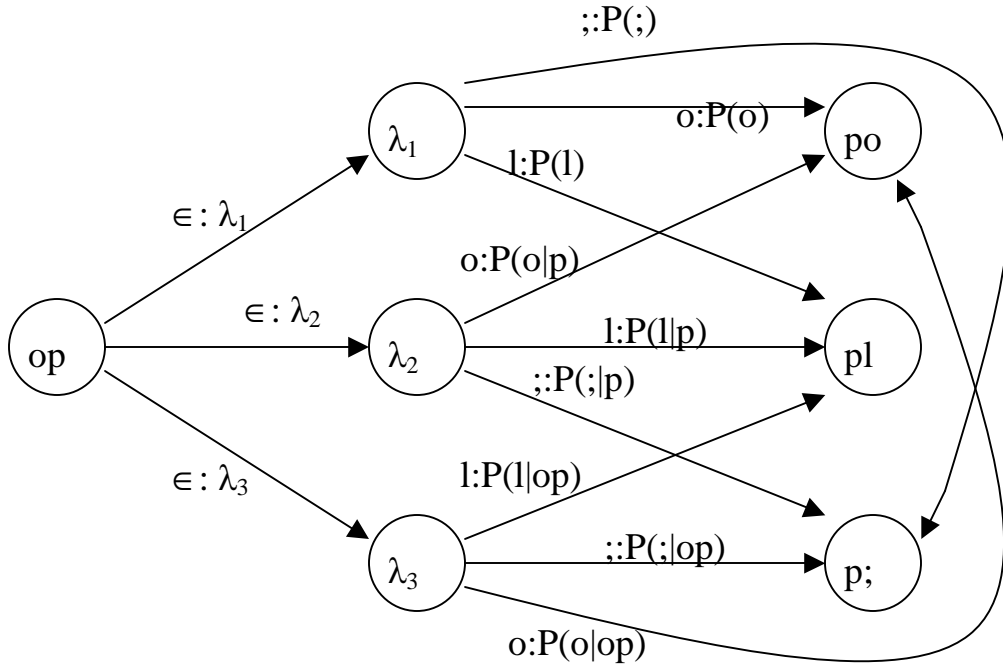


Figure 5. Partially shown HMM used for HMM move predictor.

The three lambda values are the weights that each of their respective n-gram model is given, and they should sum to one. By making λ_3 the largest, say 0.6, λ_2 the second largest, say 0.3, and λ_1 the smallest, the remaining 0.1, the HMM will favor higher order n-grams. The beauty of this model lies in how it deals with overfitting. Each of the three n-grams will make its own prediction and in doing so determines a probability, or the likelihood of that prediction. In other words, each n-gram will find the statistically most probable next state given its constraints of previous states. The probabilities of these next states are multiplied by the corresponding lambda weights. So if the trigram is suffering from overfitting, it will report an unknown next state with a probability of zero. This will in effect make the HMM fall back to the bigram's result. In the event that the bigram also happens to suffer from overfitting, the unigram's result will be used. The unigram, since its result is independent of prior states, will always return a prediction except when the model is empty.

This program was done as another Java applet as shown in Figure 6. The model display output on the left is similar to that of the n-gram word prediction program. The only exception is that the nodes at each level now has a count, as opposed to only the leaves in the n-gram program. The first level corresponds to the unigram model, and these counts are used in the unigram to calculate the probability of each state and determining the

highest such probability. The second level is likewise used for the bigram and the third for the trigram. There is a small input text field on the bottom right for the player to input moves. As noted before, the moves are limited to the *o*, *p*, *l*, and *;* keys, which corresponds to the four different attack types. The history section shows the last two moves performed with an appropriate picture. These pictures were taken from the fighting game Tekken 4 at the ign.com website [16]. Under prediction, there is a similar visual display of what the HMM's prediction of the next move will be.


Model

```


o(3)
  p(2)
    l(1)
p(1)
  l(1)
    p(1)

```

History



Prediction



Input:

Figure 6. HMM move prediction applet.

As shown in the text field of Figure 6, the player has done the sequence of moves, *opl*. If a simple trigram were used, the program would not be able to predict the next move, as it has never before seen the move *l* followed by *p* before. Since the trigram fails in this case, the HMM falls back to the bigram case; that is, making a prediction based only on

the last move, p . It finds that the most probable next move is l , a low-weak attack as depicted in the prediction area.

7.4 Remarks

This program showed promising results as a start for Alpha Fighter's AI system. This will likely be part of the low level learner (LLL). The LLL is responsible for learning the patterns of the player and using that knowledge to make predictions of the player's next move. It is important to realize that the lambda values in this HMM applet were static. In such a case, the player may eventually catch on to the learning mechanism of the AI and use that knowledge against the NPC. As a simple example, if the player realizes that it takes four moves for the NPC to adjust to the player's pattern, the player simply change his pattern every four moves. To avoid this, the AI system must have a way to adjust the LLL, a high level learner (HLL). In the case of the HMM, the HLL could simply adjust the lambda weights, so that the prediction would fluctuate between various n-grams.

8. SUMMARY OF WORK COMPLETED

At the end of this semester, most of the groundwork for Alpha Fighter has been established. The concept of the game has been documented, and development of character models has begun. Most importantly, a good understanding of various AI techniques has been learned that could be incorporated into the game's AI system. At this time, it seems the AI system will be based on a HMM, or possibly the more general model of a DBN. As noted in [14], the relationship between a HMM and a DBN is analogous to the relationship between a full joint probability table and a Bayesian Network. In both cases, the latter model provides a more concise and efficient representation. However more work is required upfront to create the model. In particular, the dependency relations between nodes must be identified.

9. FUTURE WORK

Next semester in CS 299, full production of Alpha Fighter will begin. Most of the time will be allotted to implementation of the game. The game will be built upon an existing game framework written by the author. Completion of character modeling and animation is expected before the start of next semester.

Key parts of the implementation will include the following.

- AI system (LLL) – The NPC must be able to recognize the player's patterns.
- AI system (HLL) – The NPC must adapt and formulate a new strategy if the current strategy is not working.
- AI system (planner) – The NPC must formulate offense and defense based on its knowledge.
- Character animation (key framed) – The completed character models will need to be properly imported and animated using the DirectX API.

- Character animation (IK) – Some research was done on inverse kinematics [2,4]. Some animations such as getting hit may use the technique detailed in [4].
- Controls – A key component of a fighting game is a responsive and well-timed control. For example, certain moves are prohibited during the execution of other moves.
- Physics – Accurate collision detection will be necessary to determine what parts of the body are colliding between the two fighters.

10. REFERENCES

- [1] Charniak, E. (1996). *Statistical language learning*. Cambridge, MA: MIT Press.
- [2] Elias, H. (2004). *Inverse kinematics – improved methods*. Retrieved December 6, 2004, from http://freespace.virgin.net/hugo.elias/models/m_ik2.htm
- [3] Evans, R. (2002). Varieties of learning. In S. Rabin (Ed.), *AI game programming wisdom* (pp. 567-578). Hingham, MA: Charles River Media.
- [4] Jakobsen, T. (2003, January). *Advanced character physics*. Retrieved December 6, 2004, from http://www.gamasutra.com/resource_guide/20030121/jacobson_01.shtml
- [5] Kaukoranta, T., Smed, J., & Hakonen, H. (2004). Understanding pattern recognition methods. In S. Rabin (Ed.), *AI game programming wisdom 2* (pp. 579-589). Hingham, MA: Charles River Media.
- [6] Laramée, F. D. (2002). Using n-gram statistical models to predict player behavior. In S. Rabin (Ed.), *AI game programming wisdom* (pp. 596-601). Hingham, MA: Charles River Media.
- [7] Manslow, J. (2002). Learning and adaptation. In S. Rabin (Ed.), *AI game programming wisdom* (pp. 557-566). Hingham, MA: Charles River Media.
- [8] Manslow, J. (2004). Using reinforcement learning to solve AI control problems. In S. Rabin (Ed.), *AI game programming wisdom 2* (pp. 591-601). Hingham, MA: Charles River Media.
- [9] Mommersteeg, F. (2002). Pattern recognition with sequential prediction. In S. Rabin (Ed.), *AI game programming wisdom* (pp. 586-595). Hingham, MA: Charles River Media.
- [10] Tozour, P. (2002). Introduction to Bayesian networks and reasoning under uncertainty. In S. Rabin (Ed.), *AI game programming wisdom* (pp. 345-357). Hingham, MA: Charles River Media.
- [11] Tozour, P. (2002). The evolution of game AI. In S. Rabin (Ed.), *AI game programming wisdom* (pp. 3-15). Hingham, MA: Charles River Media.
- [12] Rabin, S. (2004). Common game AI techniques. In S. Rabin (Ed.), *AI game programming wisdom 2* (pp. 3-14). Hingham, MA: Charles River Media.

- [13] Rabin, S. (2004). Promising game AI techniques. In S. Rabin (Ed.), *AI game programming wisdom 2* (pp. 15-27). Hingham, MA: Charles River Media.
- [14] Russell, S, & Norvig, P. (2003). *Artificial intelligence: a modern approach* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.
- [15] Sloper, T. (2001). *Lesson #2: sample outline for a game design*. Retrieved December 6, 2004, from <http://www.sloperama.com/advice/specs.htm>
- [16] http://media.ps2.ign.com/media/016/016600/imgs_1.html?fromint=1