

18)

Penelope: The Story Weaver

by

Willie E. Johnson, Jr.

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 28, 1996

Copyright 1996 Willie E. Johnson, Jr. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author _____
Department of Electrical Engineering and Computer Science
May 28, 1996

Certified by _____
Justine Cassell
Thesis Supervisor

Accepted by _____
F. R. Morgenthaler
Chairman, Department Committee on Graduate These

ENG
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUN 11 1996

LIBRARIES

Penelope: The Story Weaver

by

Willie E. Johnson, Jr.

Submitted to the

Department of Electrical Engineering and Computer Science

May 28, 1996

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Masters of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Penelope: The Story Weaver is a game which attempts to use the research done on how children play games and the structure of the games they play to engage, entertain and educate them. The purpose of this game is to attract children, especially girls, to technology and the world of narrative language. The social environment in America is one in which gender roles still play a large role in child development. The social pressure for girls to engage in “feminine” activities and boys to engage in “masculine” activities is still quite strong, and although girls are allowed, if not encouraged, to engage in “masculine” activities the same standard does not apply to boys. The fact that girls can play “boy” games but boys can not play “girl” games has lead most of the software houses to develop games that target boys -- construction and/or action games (Koch, 1994). Penelope is a creation game which through giving the user the power to create a story/game with world themes and realistic characters provides an alternative to the “boy” games.

Thesis Supervisor: Justine Cassell

Title: Asst. Prof., Media Arts & Sciences

TABLE OF CONTENTS

TABLE OF CONTENTS	3
TABLE OF FIGURES	6
I. INTRODUCTION	7
I.A Social Motivation	8
I.A.1 Girls and Game Play	8
I.A.2 The Oral Tradition	9
II REVIEW OF RELEVANT RESEARCH	11
II.A Gender Roles and Game Play	11
II.B Software	12
II.B.1 Story / Narrative Software	13
II.B.1.i Story Writing Software	13
II.B.1.ii Interactive Stories (books)	14
II.B.1.iii Story Games	15
II.B.2 Software that targets girls	17
II.D What was done in version 1	17
II.D.1 Creation	19
II.D.2 Relation	20
II.D.3 Conclusion	22

III WHAT I PROPOSED	23
III.A Technical Design	23
III.A.1 Language	23
III.A.2 Program Structure	24
III.A.3 Autonomous Agents	25
III.A.4 Interface	26
IV CURRENT DEMO VERSION	27
IV.A Narrative	27
IV.B Character Creation	28
IV.C Character Description	30
IV.D Autonomous Agents	30
IV.E Interface	31
IV.F Conclusion	36
V IMPLICATIONS	36
V.A Accomplishments	36
V.A.1 Gender and Game Play	36
V.A.2 Narrative language	38
V.A.3 Technical Design	39
V.B Areas Which Need Further Development	40
V.B.1 Gender and Game Play	41

V.B.2 Narrative language	41
V.B.3 Technical Design	42
V.C Conclusions	42
VI CONCLUSIONS	43
VII CREDITS	44
VIII APPENDICES	46
VII.A Users Manual	46
VII.A Users Manual	46
VII.B Technical Documentation	51
VII.B.1 Dictionary Class	51
VII.B.2 Parser Class	57
VII.B.3 Response Table Class (rtable)	64
VII.B.4 State Class	68
VII.B.5 Object Class	73
VII.B.6 Room Class	81
VII.B.7 Character Class	84
VII.B.8 Obect Handler Class (objhandler)	95
VII.B.9 Game Handler Class (game or gameplayer)	106
IX BIBLIOGRAPHY	116

TABLE OF FIGURES

<i>Figure 2.1 Version 1 “Character Creation”</i>	18
<i>Figure 2.2 Version 1 “Favorite Phrases”</i>	19
<i>Figure 2.3 Version 1 “Game page showing introduction”</i>	21
<i>Figure 2.4 Version 1 “Game page during play”</i>	22
<i>Figure 4.1 Version 2 “Game page during play”</i>	27
<i>Figure 4.2 Version 2 “Character Creation”</i>	28
<i>Figure 4.3 Version 2 “Character Description”</i>	30
<i>Figure 4.4 Version 2 “Scratchpad and Game pages”</i>	31
<i>Figure 4.5 Version 2 “Introduction”</i>	32
<i>Figure 4.6 Version 2 “Game page during stroy world introduction”</i>	33
<i>Figure 4.7 Version 2 “Create page”</i>	34
<i>Figure 4.8 Version 2 “Favorite Phrases”</i>	35

I. Introduction

Penelope: The Story Weaver is a game which attempts to use the research done on how children play games and the structure of the games they play to engage, entertain and educate them. The purpose of this game is to attract children, especially girls, to technology and the world of narrative language. The social environment in America is one in which gender roles still play a large role in child development. The social pressure for girls to engage in “feminine” activities and boys to engage in “masculine” activities is still quite strong, and although girls are allowed, if not encouraged, to engage in “masculine” activities the same standard does not apply to boys. The fact that girls can play “boy” games but boys can not play “girl” games has lead most of the software houses to develop games that target boys -- construction and/or action games (Morose, 1995, Koch, 1994, and Hanscome, 1995). Only about 30% of the girls in the US enjoy these types of games (Koch, 1994). Penelope is a creation game which targets the other 70%. Through giving the user the power to create a story/game with world themes and realistic characters, Penelope is providing an alternative to the “boy” games, an alternative that is more likely to attract that other 70%. Barbara Hanscome, the managing editor of “Software Development” magazine stated the issue quite well, “To reach girls and women with electronic entertainment (and a new audience of boys and men), it is obvious that game developers need to consider new directions in computer play. Addressing different preferences and styles in gameplay (be they gender influenced or not) will make those new directions more interesting and exciting - for male and female gamers alike.”(1995) Penelope is this developer’s step in a new direction.

I.A Social Motivation

I.A.1 Girls and Game Play

The Penelope project came about as a response to noticed trends in society, specifically, the trend in technology to develop software geared towards what is characterized as a typically male/ masculine audience. The majority of consumer software consists of games that deal with combat of some kind, sports (very much like combat), or adventure (combat with more frills). These types of games with set rules that one learns as one plays the game are typically more attractive and engaging to boys (Hanscome, 1994). Girls more often will choose to engage play that allows them to create the rules (Kennedy, 1995). This can be attributed to the differences in the way girls and boys tend to play games (Kennedy, 1995).

The main differences in the way boys and girls play is that girls tend to enjoy games in which they can create the environment and the rules. They enjoy making up the characters and actions and the locations. Creating a world and its rules is as much or more of the enjoyment they derive from playing in their world (Koch, 1994). Boys, on the other hand, tend to enjoy discovering an unfamiliar world with unknown and often unspoken rules (Koch, 1994).

This difference in game play may seem trivial but it is having a serious effect on our society.

In this age of high technology, exposure to and familiarity with technology is key to children's success. As children face a shrinking world where geographic and cultural barriers are torn down by technology, they are only limited by their knowledge of and

comfort in using this technology. Early exposure to technology and its study can only help, and the earlier the better.

The problem is that, although boys get exposure to technology and science through video games and the like, girls have no such inroads into the world of bits and baud (Kennedy, 1995, Koch, 1994). Research does not show that girls are incapable of or discouraged from using computers and other technology; what it does show is that the majority of software that is on the market has been created using a typically male approach to game play (Kennedy, 1995). Thus, girls do not get as much exposure to high technology based games as boys because the games that are out there do not interest them. This lack of exposure to high-tech games becomes a problem down the road when computer literacy, and comfort in a technological world become paramount to success in any endeavor.

One of the goals of Penelope is to address the issue of game play. Penelope was designed with a more female or feminine approach to game play in mind. It was created to get young girls interested in technology and science by allowing them to actually create and in a sense program the game, therefore appealing to their desire to create.

I.A.2 The Oral Tradition

Technology, like anything else, has two sides. It has benefited humankind by creating new forms of communication and better access to information. It has hindered humankind by destroying the personal nature of communication. We share information, we even talk at each other but we as a society are becoming less and less appreciative of the art of personal communication and of the personal nature of the exchange of ideas.

Evidence of lessening appreciation for personal communication can be seen in how society treats its older members. A large part of the role elderly people fill in our society has been as keepers of history and its wisdom. With technology such as video recordings, photographs, and even books, the role of the elderly has become almost redundant. The information older members of society have historically provided can now be obtained elsewhere. The part that is so often overlooked, however, is the personal nature the information takes on when told by a person as opposed to an impersonal agent. Our society is losing its appreciation for the personal nature information can take on through stories. In doing so we are losing a segment of our humanity.

One aspect that was important in the concept and creation of Penelope was the idea of treating storytelling as an art and an asset. Penelope was developed to teach and to help children to create personal stories. By having children decide how their story will be told, audience, narration, etc., Penelope helps them to create a personal story that has a specific audience, and is told from a specific point of view. This method of story-telling / story-creation allows children to practice creating something personal, something given of themselves to someone, and hopefully Penelope through this method can help children to recognize and appreciate the personal wisdom and beauty passed down through the oral tradition.

I.B Technical Motivation and Challenges

In order to make Penelope attractive to our target audience, young girls, it must allow them to create the rules and in a sense program the world. Once they create this

world Penelope must also follow the rules as defined by the user. From a technical point of view Penelope is not simply a game but is more like a programming tool.

There are three technical challenges in this project: creating autonomous characters, creating an engaging interface, and distinguishing and switching between the create and play modes of the game.

This project is addressing issues that are very current in the computer science industry. Research on how to create more exciting games and better interfaces have moved away from the intellectually engaging towards the visually stimulating, but only so much can be done with the graphics aspect of a game. In the end there has to be a way to keep the children interested.

II Review of Relevant Research

II.A Gender Roles and Game Play

There have been a number of articles written lately that address the issue of girls and computer games. Some of these articles, such as “Men, Women & Computers” (Kantrowitz, 1994), fall into the trap of citing myths about girls technophobia and inherent inaptitude as the cause for their disinterest in the games on the market today. Most of the articles on this subject, however, have taken a somewhat more enlightened and scientific approach, citing the fact that, although girls do not use technology for play they do use it, and use it quite well, for work and school (Kennedy, 1995).

In “Sugar and Spice ... Software for Women and Girls ???”, Beth Kennedy (1995) cites work done by E. Maccoby and C. Jacklin that dispels many of the myths. She goes on to claim that the reason that girls do not use the games that are out there and women do not like to use computers when they do not have to is that the games that are developed are not appealing to them. The games that are out there have been made to target boys and their construction type of play as opposed to girls and their creation play.

So why aren't games being made for girls? “Gammin’ for Girls” (Hanscome, 1995) claims many software houses believed that girls would play “boy” games but boys would not play “girl” games. True, in our society it is often more acceptable for a girl to push the envelope on gender issues than it is for boys. However, most girls do not seem to enjoy “boy” games and given the choice they usually chose to play a different type of game (Koch, 1994, and Hanscome, 1995). Recognizing this, many software houses have taken on the task of developing more girl or gender neutral targeted games. They are, however, still falling into the same myth based traps -- A pink Mortal Kombat will not sell better to girls, although boys may stop buying it (Hafner, 1994).

II.B Software

Knowing that Mortal Kombat only pinker is not what will sell and is not what girls want is not enough (Hafner, 1994). We must look to see what girls are asking for. Girls tend to enjoy games that involve creation, imagination, real life themes and social interaction (Koch, 1994, and Hanscome, 1995). All of these attributes are found in narrative and so one would expect narrative based software to not only have these attributes but be appealing to girls. So why is this not the case? In this section we will look at the

narrative based software that is on the market , its strengths and weaknesses and try to determine where it fails to capture the young female audience.

In this section we also look at what software is that targets girls is on the market and what Penelope's first version was. In each of these types of software we examine the strengths and weaknesses of each of their approaches to see which aspects worked and which did not.

II.B.1 Story / Narrative Software

The software on the market which uses narration in one aspect or another can, it seems, be cleanly divided into three main types; a) software that helps children write stories; b) interactive stories; and c) story games.

II.B.1.i Story Writing Software

There is a large amount of software on the market that helps children write stories. Some of the better examples of this type of software are *Creative Writer* and *3D Movie Maker* by Microsoft, *EasyBook* by Sunburst and *The Amazing Writing Machine* by Brodebund. Each of the aforementioned has its benefits and its weaknesses.

The Amazing Writing Machine is one of the better games from a creative stand point. It has a number of capabilities. One is brainstorming aids such idea generators that offer possible situations, nouns, verbs, quotes, riddles, etceteras. Another capability is *The Amazing Writing Machine's* ability to combine text with graphics, designs, page layout and font types. It also provides pre-written stories that can be filled in like Mad Libs to help stimulate the creative process. The most impressive aspect of this software is that it

outputs a printed book at the end of use. *The Amazing Writing Machine*'s biggest weakness is that it has no recognition of the users inputs. For example the following sentence could be generated if the user asked for suggestions of a person, an action and a place (the italics indicate computer generated text):

... Mark went to see *Mark ran beneath the ocean.* ...

The suggestions it generates are randomly chosen from a list and do not change based on the users input.

The 3D Movie Maker does not make suggestions. It consist of libraries of scenes, actors, sounds, camera angles, props and costumes that can be combined to create a story. It is however limited to the elements of these libraries. It does not allow for the introduction of new elements.

II.B.1.ii Interactive Stories (books)

Another type of software on the market is interactive stories. Examples of this type of software are the "*Living Books*" series by Brodebund, the "*Animated Storybook*" series by Disney and *How the Leopard Got His Spots* by Microsoft. This software tells a story with the use of the multimedia capabilities to add graphic and sound aspects to the written word. Unlike television or movie versions of a book these books are user paced and the text is also displayed along with the story which is helpful from an educational point of view.

How the Leopard Got His Spots is one of the best examples of this type of software. This software tells a story to the user. An actor reads the story and the screen displays the text so that the user can read along. The screen also displays animated

illustrations to enhance the story. The story can be paused at the end of each page and the user can flip forward or back in the story or the user can switch to a further study/ help page. On this page the user can highlight parts of the story text that was just read to them and get a pronunciation and definition of the word or words they highlighted. There are also games and historical data through various pages of the story. For example, the beginning of the game if the user clicks on the globe a map pops up displaying Africa and some statistical data about the size, location and population of the continent are read and displayed. The weakness of this software is that interaction is completely independent of story -- games and historical information are independent of the story. When using that software, to enter into one of the game or background pages the user must pause the narrative. There is actually no user interaction with the story.

II.B.1.iii Story Games

Story games, of which some examples are *Myst*, *Seventh Guest*, *King's Quest* and *Advanced Dungeons & Dragons*, have stronger iterative capabilities, meaning, the user's actions actually effect the story, and a strong narrative aspect, but they are limited by constrained decision spaces and pre-set stories.

King's Quest, while having a dynamic story that is affected by user input, is still limited to set choices of actions. In this game the underlying story is preset, there is no user influence on overall theme or plot of the story. The user is a young man that is sent to find his true love, the princess, who has disappeared under mysterious circumstances. There are other characters in this world that the user encounters and at each stage the user can choose to talk to the character, give something to the character, or simply continue past

the character. The user's decisions do affect the story by determining in what sequence things will happen and whether or not the user will find the princess, but the actual story does not change -- there just may be some sections left out. So, although the user has some influence, that influence is quite limited. The pre-set characters, goals, scenes, and action sets make this more like reading a story than creating one.

Advanced Dungeons & Dragons is different from the other story game software in the fact that it allows for user controlled character creation. It also differentiates itself by having characters with limited autonomy. *Advanced Dungeons & Dragons*, like *King's Quest*, has a dynamic story, but this story is much less constrained. In this game the user is making the decisions for a group of characters each of which the user creates at the beginning. In the creation phase the user assigns attributes such as strength, agility, race (human, elf, cleric, etc.), alignment (good, bad, lawful, etc.), etc. to each character. Then the group goes shopping and is told its task by the dungeon master, in this case the computer. As the user plays the game the characters in the group gain experience points and wealth. When non-player characters (NPCs -- characters not in the group and not player controlled) are encountered the user has the option of asking them to join the group. If the NPC joins the group then they add their strength and wealth to the group but unless the NPC commits treachery, they no longer have any effect on the story. If the NPC does not join the group then it just goes away and has no further effect on the story.

In *Advanced Dungeons & Dragons* the action space is limited -- the characters can only fight, talk, rest, run, continue, or trade when they encounter a dragon. In addition, the non-player characters (NPCs) are one dimensional -- NPCs have very limited autonomy and except as a member of a group or when first encountered they have no effect on the

story. Regardless of its weaknesses *Advanced Dungeons & Dragons* is the only software in its category that could be considered a creation game.

II.B.2 Software that targets girls

For a long time the major software houses believed that girls would play boy games but boys would not play girl games, so there were no games developed specifically with a young female audience in mind. This did not work; girls were not playing the games that were developed (Koch, 1994, and Hanscome, 1995). Recognizing this, many software houses have begun to take on the task of developing more girl or gender neutral targeted games. Many of these are still in the works.

II.D What was done in version 1

The goal of the current version of Penelope is to build a game that incorporates creating a world and characters and creating relationships between those characters in order to appeal to girls. The previous version of Penelope, designed and implemented by Teresa Alagoso and Manuel Perez, focused on much the same goal. Like all of the software that has been examined so far, the first version of Penelope, had the same goal of appealing to those who tend to like narrative based games, of appealing to girls. They, however, approached this goal from different directions.

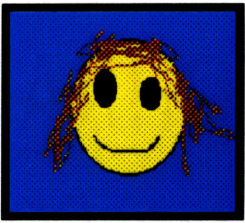

Character					
NAME: Tess		Get			
DESCRIPTION: A woman.					
MOOD: mellow		FAVORITE PHRASES			
GET OBJECTS	DROP OBJECTS	GIVE OBJECTS	HAPPY OBJECTS	SAD OBJECTS	ACTION CHANCES
ball		ball			SAY 100
					GET 100
					GIVE 100
					DROP 16
					GO 16
					NOTHING 16
				Submit	DONE

Figure 2.1 Version 1 "Character Creation"

Favorite Phrase

Penelope 2
The Story Weaver

1: Hello NAME

2:

3:

4:

5:

6:

7:

8:

9:

10:

SUBMIT DONE

Figure 2.2 Version 1 "Favorite Phrases"

II.D.1 Creation

This aspect was only just touched on in both the previous and current versions of the game. The previous version of Penelope only supported modifying a few character attributes (see Fig. 2.1) such as name, description, mood, favorite phrases (see Fig. 2.2), etceteras. These, although only giving limited control, were an excellent beginning. The possession-based nature however seemed to take away from the importance of the relationship aspect of the game. However, by creating mutable character objects Teresa

and Manny made an even more sophisticated creation design much easier because the objects were already mutable and internally portable.

II.D.2 Relation

In version one the focus on the relationship aspect took on three forms. First, they provided a facility for conducting semi-real conversations with computer agents. Next, they designed a structure to represent relationships. Finally, they created parameters to influence relationships. In order to achieve the first goal they created a parser and a dictionary which interpret the user input.

In the first version the main thing that is influenced by relationships is a character's conversation. They represented a conversation as a weighted directional graph. Each vertex represents a state, or a "point in conversation". The edges are weighted to represent a given point in conversation which way the conversation is to go. These are adjusted by how much two characters like one another. This is very much like real conversations. For example, if someone asks me a question I may be more or less helpful depending on if I like the person.

The first version of Penelope only supports one way of influencing relationships, having a list of objects that have a certain effect on a character's state graph. When a character is given such an object the weights on the character's state graph are adjusted. For example, in the first version when a character gives the cat the banana it makes the cat happy, so he becomes much nicer to that character.

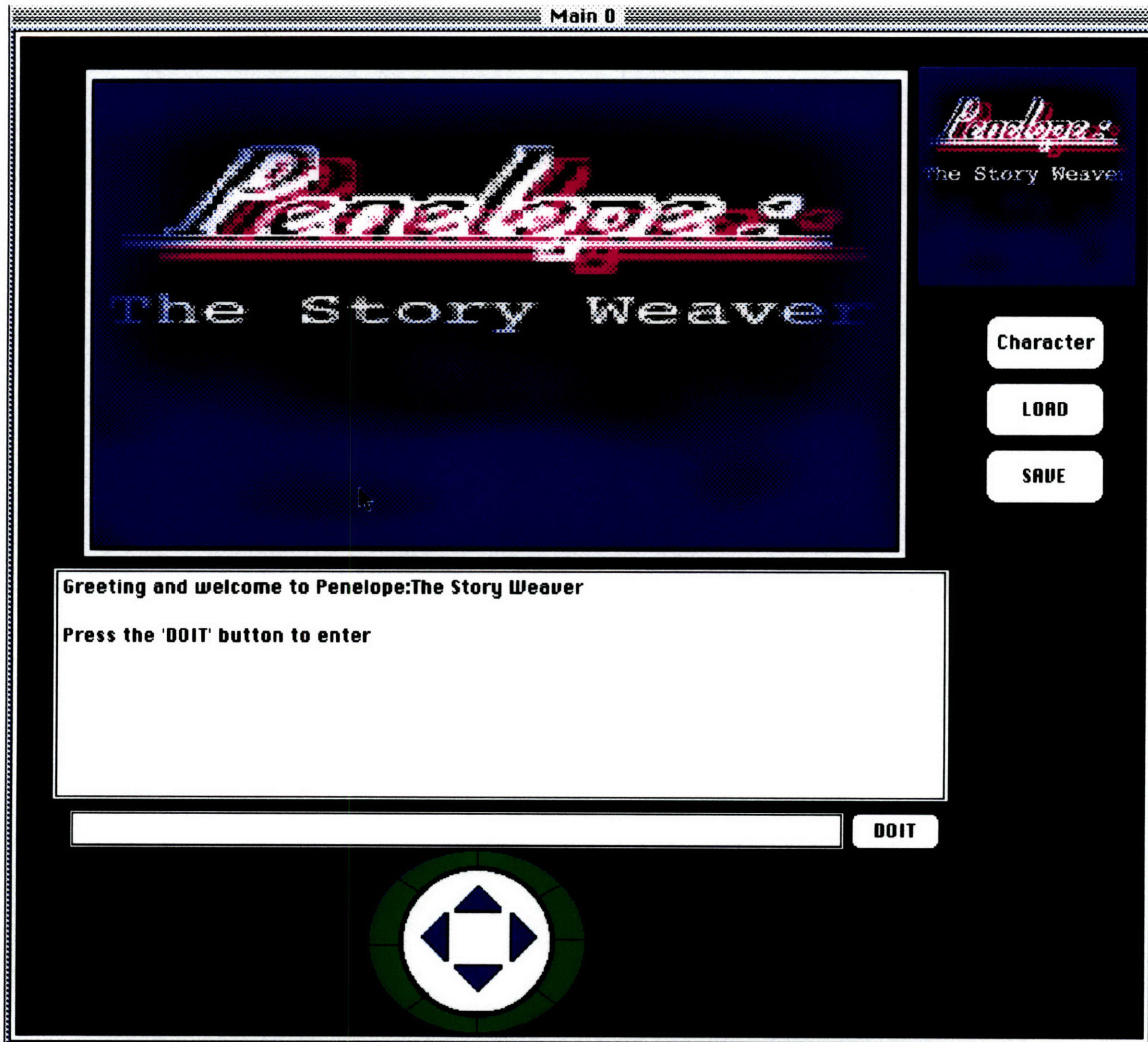


Figure 2.3 Version 1 "Game page showing introduction"



Figure 2.4 Version 1 “Game page during play”

II.D.3 Conclusion

The previous version of Penelope was focused mainly on creating discourse. The version they created is very good in the sense that it allows and encourages editing the characters in the story. Version one has many strong points: character editing, discourse, mood and limited autonomous agents are only a few. It does, however, have its weaknesses. Most of the character creation (see Fig. 2.1) is desecrate, which makes it seem somewhat mechanical and the text in the game portion is very choppy (see Fig. 2.4).

Another weakness of this version is that the creation page is a sub-page of the game play page which gives the impression that the creation aspect of the game is of lesser importance than the play portion (see Figs. 2.3, 2.4 and 2.1)

III What I Proposed

III.A Technical Design

The first version of Penelope was basically a multiple user dungeon (MUD) with panels for character design. What I proposed was taking it beyond that. In play mode the current version of Penelope may resemble its former form but in design mode it was changed greatly. Due to time constraints I chose to focus on the issues involved in the creation aspect of the game as opposed to the play aspect. There has been a great deal of work done in the area of making games more graphically appealing. In Penelope, however, I chose to put the focus on the content of the game and leave the packaging for later research and development.

III.A.1 Language

This beta version of Penelope was to be written in Mac Common Lisp using some of the code that had already been written. The logic behind using MCL was: (1) Penelope will need a great deal of the functionality, characterized by Artificial Intelligence, and MCL, being a language developed for AI, can provide that functionality; (2) MCL is object oriented, and therefor more portable, which makes it a good language for initial

development; and (3) since most elementary schools have Macintosh computers, a Mac based language, such as MCL, is the most logical to use initially.

Although I proposed writing Penelope in Mac Common Lisp, the first version was written in Semantic C++. In order to avoid duplicating effort and to take advantage of already having a working demo I chose to continue development in C++. The reasons for development in MCL were all still valid with C++ except that C++'s AI functionality is not as extensive or diverse as MCL.

III.A.2 Program Structure

I proposed designing Penelope to be portable although the beta version was not to be tested for portability. Penelope was to be written with an object oriented structure to allow for abstraction away from host specific controls. An object oriented programming structure is also useful in the technical aspect in that by using this structure eventually Penelope may be able to run on multiple platforms (i.e. Mac's, PC's, UNIX, etc.). From the game aspect, using an object oriented structure eventually could allow moving characters between stories.

I further proposed using the Model/View model for Penelope's program design. This model maintains the view or interface separately from the model (in this case the characters and the story). This model allows multiple user to look at the same object. An example of this model in use is a MUD. The same MUD can be looked at from many views. For example, if player one enters a MUD in hall 1 he looks around and sees a hall with three doors etceteras. Player two entering room six will see a book shelf and a table etceteras. If they enter into the same room they will see each other, but throughout all of

these comings and goings the dungeon object has not changed. Player one may see a different room than player two or may even see the same room differently but the room will not have changed. The model can even be applied to lower levels -- the character object does not change based on which story it is in. Using this model will make converting to a completely portable version easier. It may also, eventually, allow for multiple users and multiple views of single game (story).

III.A.3 Autonomous Agents

I proposed that character creation in Penelope be accomplished by the user in the create mode of the game. When Penelope is in create mode the user should be able to choose traits of the character that in a sense define that character. The mutable characteristics should be things that not only would distinguish characters from one another in the story, but in real life would distinguish people from one another. For example, I proposed that tendencies towards certain moods be one of the mutable characteristics.

Once characters are created there has to be a computer player who determines the actions of that character based on the characters attributes and endowments. This has been a hot area of research in the artificial intelligence (AI) field. My initial hypothesis was that an effective way to accomplish the creation and maintenance of autonomous characters was to create a character object with various slots (i.e. name, age, general disposition, whether or not the computer is playing the character, etc.) that reflect the assigned attributes, and various character methods that take conditions and provide the appropriate actions. This approach may appear frankinstienian but if the program provides for a wide range of choice it can provide a true creative vehicle.

The implementation of multiple autonomous characters was a major focus of my research around this project. In the last version the character creation panel set trait such as “like items” and “drop items”. I proposed moving away from defining characters by their possessions and instead defining them by their relationships with others. I also proposed instead of using single values of happy, sad, or mellow to define a character’s overall mood as was done in version one (see Fig. 2.1), in the new version mood should be a combination of how a character generally feels, how much a character’s mood generally swings, and how apt that is to show their feelings. I further proposed that these mood elements be continuous as opposed to discrete.

III.A.4 Interface

The game interface in the first version of Penelope was a MUD like view with added graphics to augment the text descriptions of the rooms and the speaking characters. In the first version of Penelope, upon entering a room the picture of that room and the text description of the room, its exits, and its contents would be displayed (see Fig. 2.4). I proposed changing this interface to leverage all of the advances that have been made on the graphics side of game development. I said that there would need to be two types of interface one for the play mode and one for the create mode. Time constraints, however, made the interface aspect of Penelope low priority.

IV Current Demo Version

IV.A Narrative

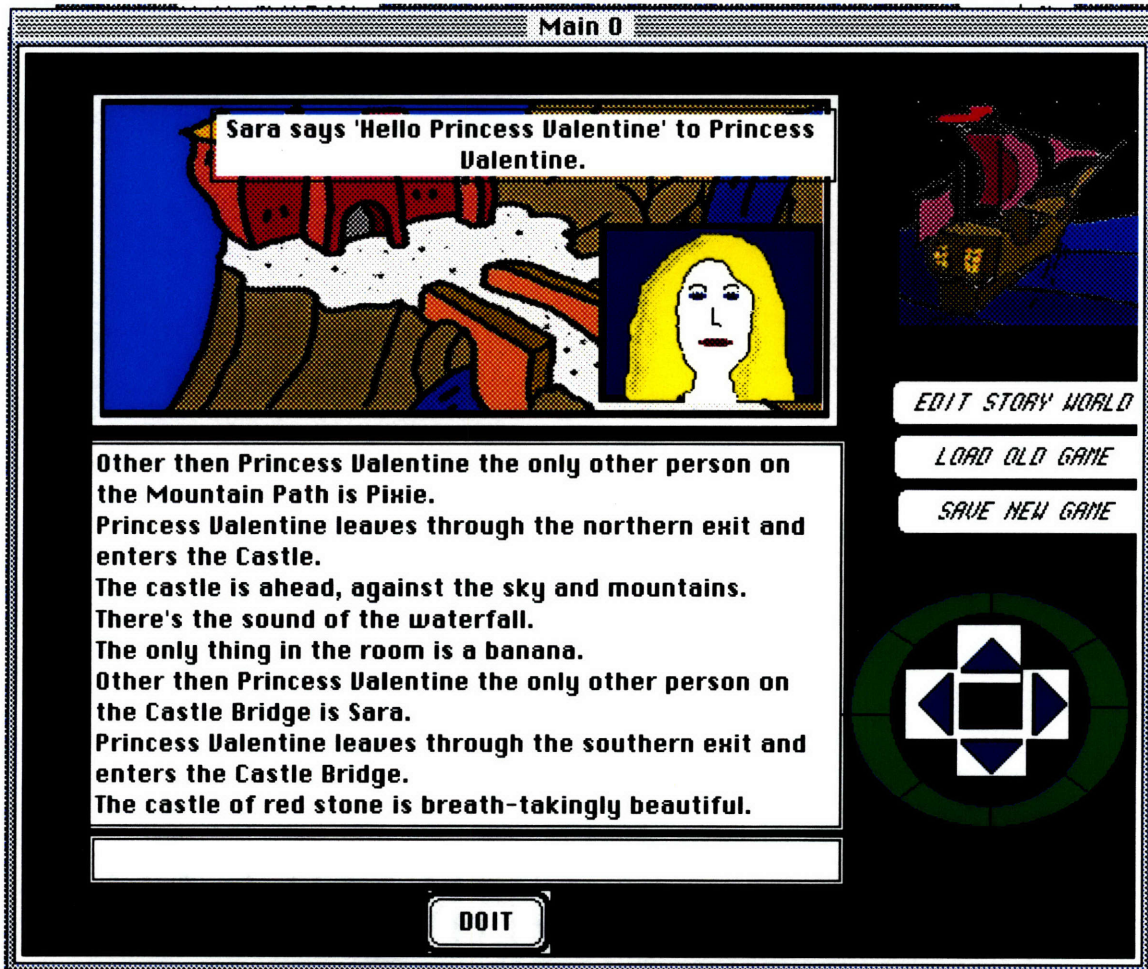


Figure 4.1 Version 2 "Game page during play

The current demo version of Penelope builds on the version that was created by Theresa and Manny. The look of the game is much the same but there are some very important though subtle differences. One of the most important differences is the narrative. Along with the shift in focus from a game to more narrative-based software, Penelope gives more of a description when you enter a room. Before the description was

somewhat mechanical (see Fig 2.4). Now, the narrative generated when a character enters a room is more natural, in part because I wrote the computer generated text to be more natural and in part because a great deal of the text is user input (see Fig. 4.1).

IV.B Character Creation

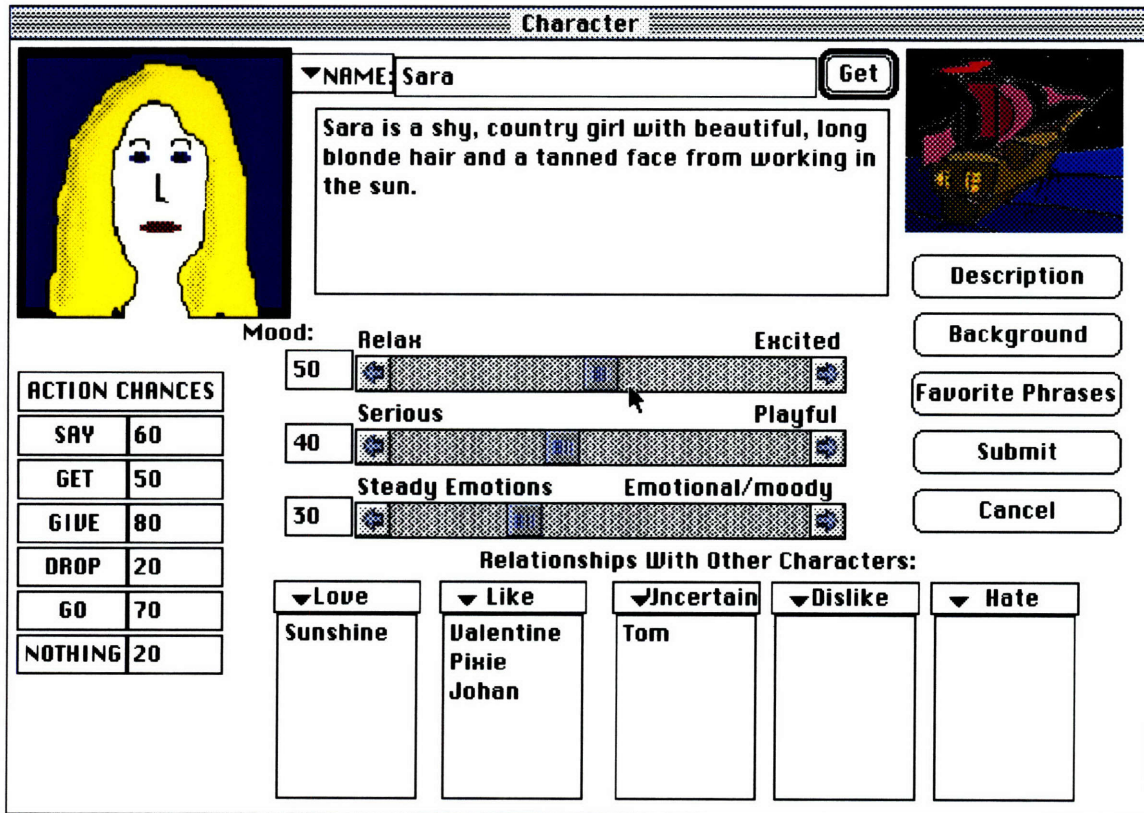


Figure 4.2 Version 2 "Character Creation"

The new version of Penelope also goes much further than the previous one to create more realistic characters. As the result of serious consideration and much discussion the definition of the characters in Penelope has moved from being more possession-based to being more inter-character relationship-based. By replacing the like, dislike, give, get and drop boxes which refer to objects from version one (see Fig. 2.1) with like, love, hate, dislike and uncertain boxes which refer to relationships with other characters in the current

version (see Fig. 4.2), Penelope no longer has the user define characters by their possessions or things they want to get but instead has the user define characters by the relationships they have with other characters.

In addition to moving to a more relationship-based character definition, the new version of Penelope now uses continuous values for mood. In describing real people one might say that someone is a generally happy person but if forced to model that person the describer would include many more parameters to describe the person's mood. In order to make the character definition more realistic I broke mood into three sections: how happy the character is, how excited it is, and what is the character's emotional inertia, i.e. how much does it take to change that character's mood (see Fig. 4.2). I made these continuous values to allow for some fuzzy logic in processing how the character's mood will affect its actions.

IV.C Character Description

▼Time Of Day	▼ Action	▼Environment	▼ Observer	▼ Location
Night				On Path

Description:

Sara, standing there on the mountain path with the wind blowing her hair and the stars dancing in her eye, smiles.

Figure 4.3 Version 2 "Character Description"

In order to focus more on the narrative, and especially the user-inputted narrative, it was important to get more user input. The way I did this was by having the user input descriptions of the characters in various situations(see Fig 4.3), much the same as what was done in the last version with the "Favorite Saying" panel (see Fig 2.2). On this new description page the user can input several different descriptions and associate them with different situations. Then later when one of those situations occurs, or a similar situation occurs, the description is put into the narrative (see Fig 4.1).

IV.D Autonomous Agents

Once I decided how I wanted to define the characters, making them act autonomously was a challenge. The creation of autonomous characters in Penelope began

by making all of the characters act independently. This meant that the characters would move around and have no effect on one another nor on their environment. Once I got the characters exhibiting somewhat expected behavior, I reintroduced the relationships between the characters. That seems to be working, in the sense that the characters act in some what predictable fashion. For example, now if Sunshine and Sara dislike one another then it is unlikely that they will be helping one another. Although neither character is user controlled (they are both NPCs), the computer still plays them according to the user their specified attributes. So, since the user specified that they dislike each other the computer does not have them cooperate.

IV.E Interface

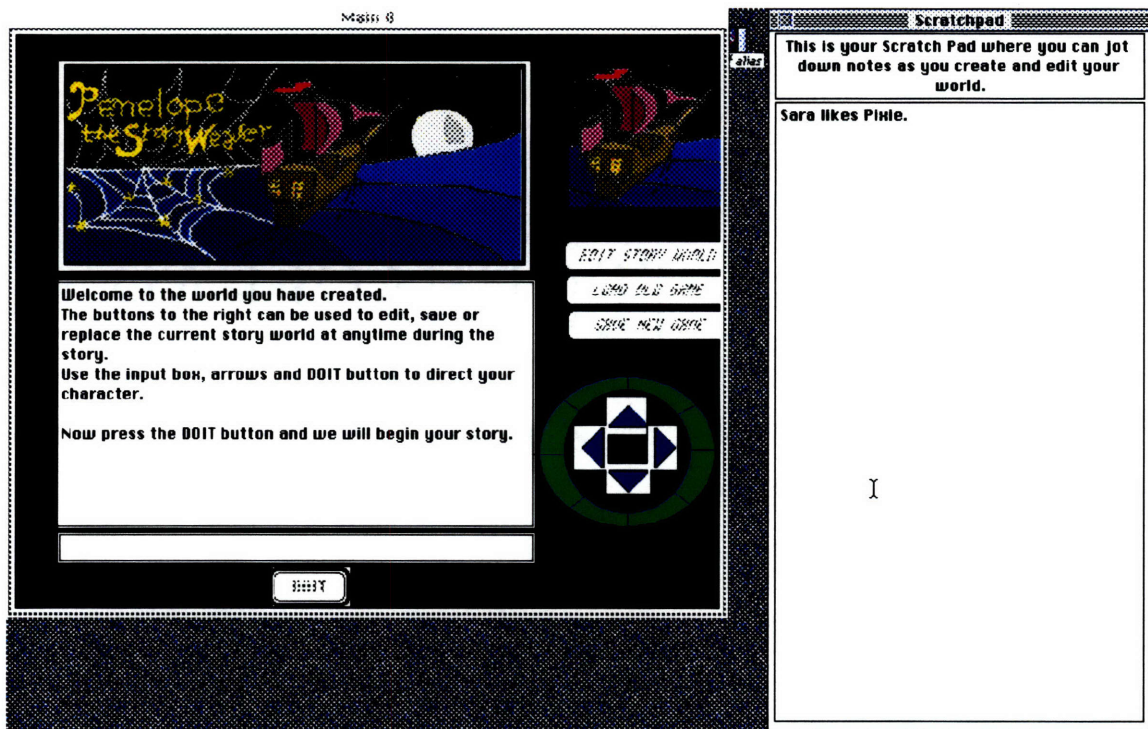


Figure 4.4 Version 2 "Scratchpad and Game pages"

The interface has not changed much. Some of the graphics, thanks to Hannes Vilhjálmsson, have improved, and there are a few more panels. The panels were added more for functionality than aesthetics. The addition of a scratch pad (see Fig. 4.4) on which users can take notes without using a separate piece of paper, was added to keep the attention focused on the screen and that world.

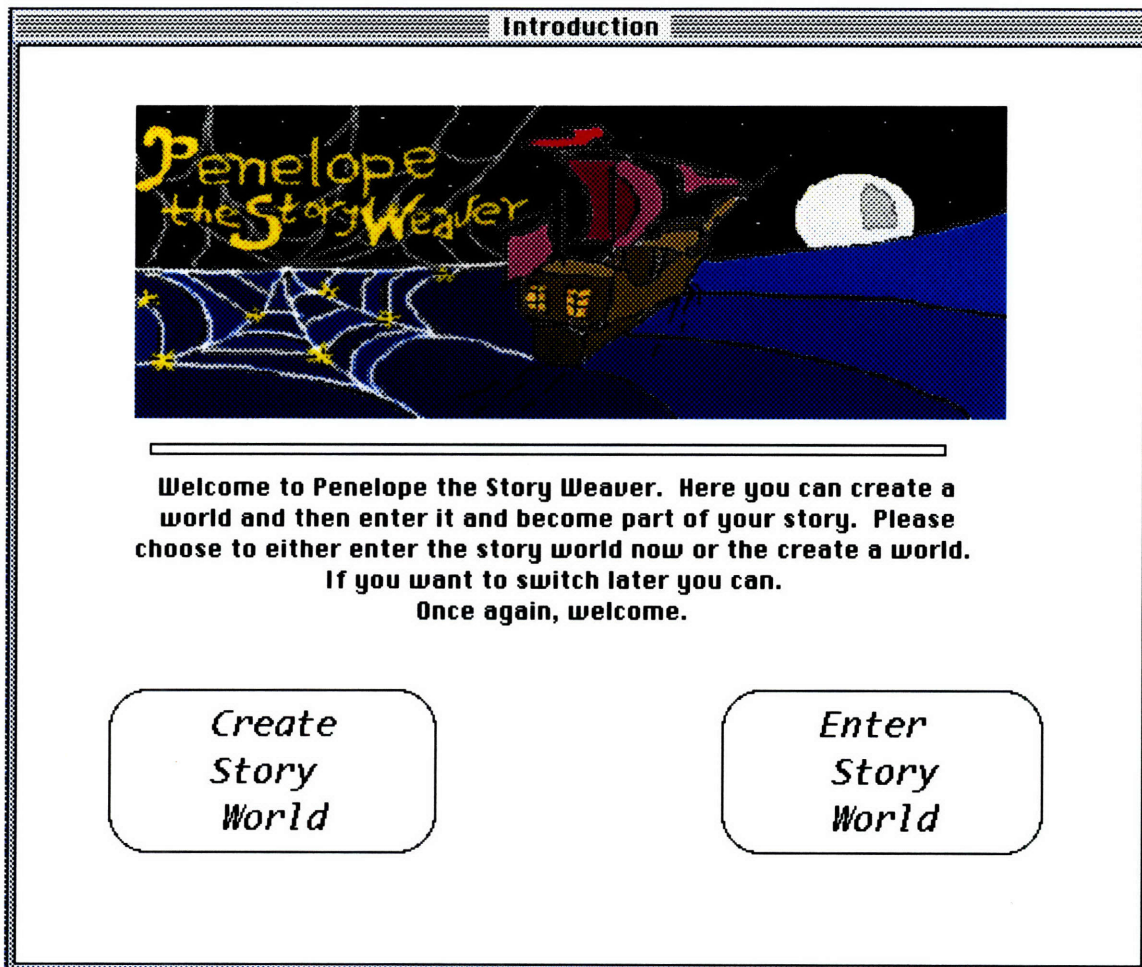


Figure 4.5 Version 2 "Introduction"



Figure 4.6 Version 2 "Game page during stroy world introduction"

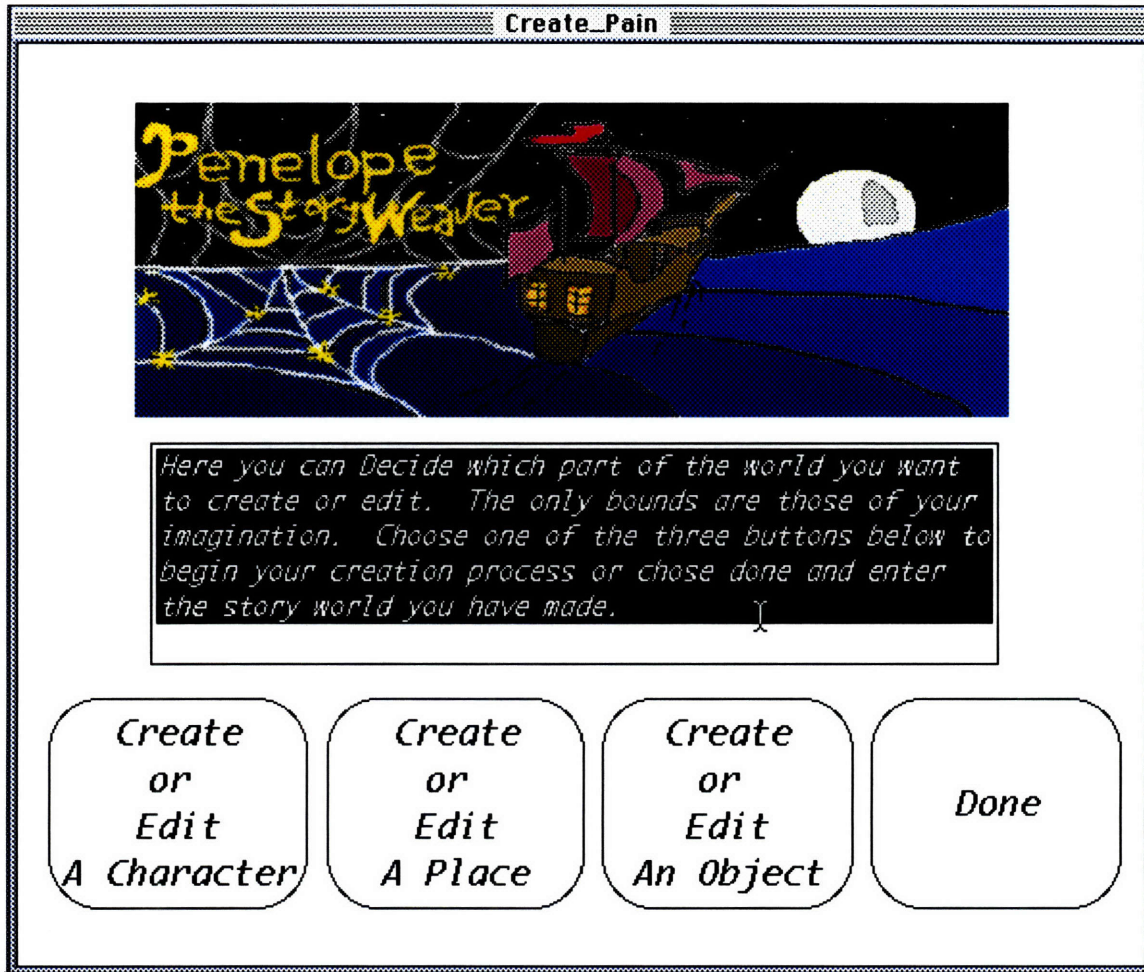


Figure 4.7 Version 2 "Create page"

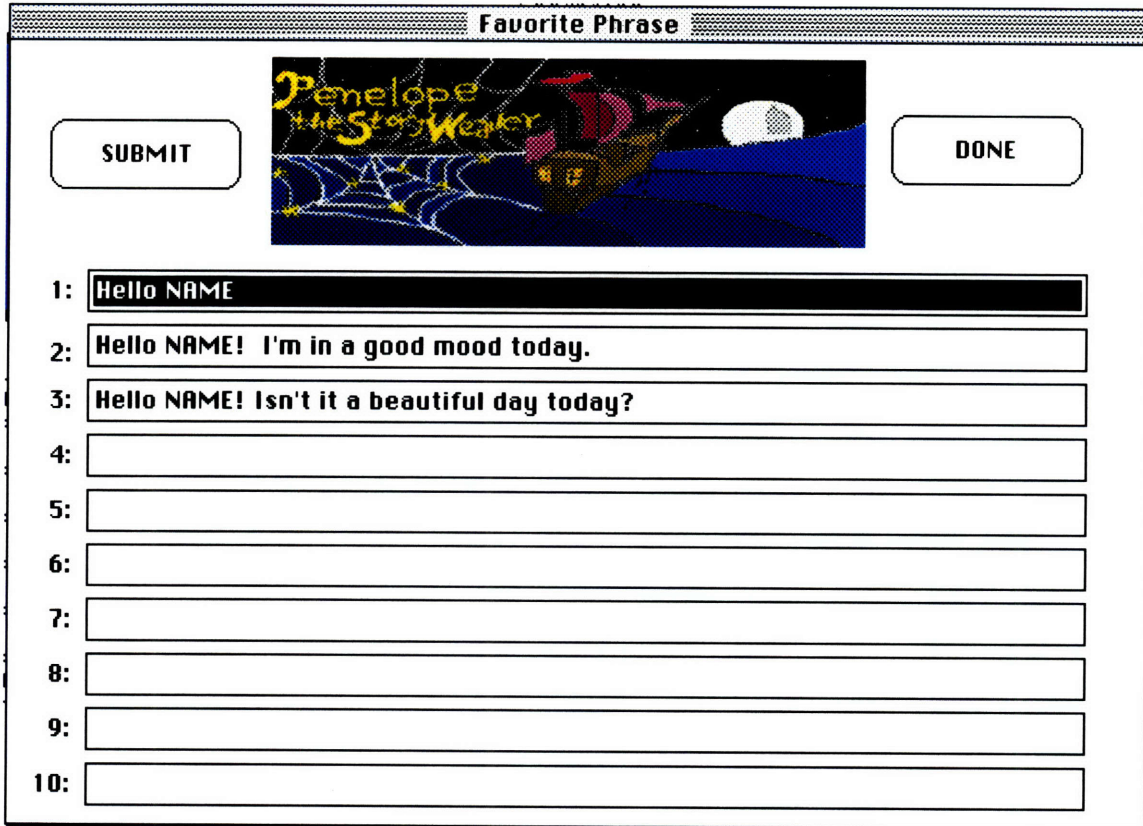


Figure 4.8 Version 2 "Favorite Phrases"

An introduction page (see Fig. 4.5) was added to create more of a separation between the game page (see Fig. 4.6) and the create page (see Fig. 4.7). The create page was created to separate the three things you can create: the character creation page (only this page is currently functional), the object creation page and the room creation page. I continued to use the favorite saying page from the previous version of Penelope (see Fig. 4.8).

Due to time constraints the graphics aspect of Penelope had to take a back seat to the other goals. Improving the graphics and other aspects of the interface would, however, make an excellent area for further research.

IVF Conclusion

The current version, although it resembles its predecessor, is under the surface quite different. The changes in the character creation and game play actually reflect the changes in the direction the Penelope project is taking. The focus of this game is moving away from game the aspect towards the narrative aspect.

V Implications

In this section I will discuss the current version and how well it accomplishes the goals set forth in the proposal. Since this is an ongoing project there are of course some goals that were not accomplished. The goals that were not completely accomplished or that need further research and development are also laid out in this section with suggestions as to next steps.

V.A Accomplishments

The Penelope project has gone through a great deal of change. Many of these changes have been in the goals of the project or more specifically the direction of the project. The overall goal has been and still is to develop story creation software that is fun and interesting to girls, thus getting them to use technology in play. How to accomplish this is the question.

V.A.1 Gender and Game Play

The trend in technology has been to develop software geared towards what is characterized as a typically male / masculine audience. The majority of consumer software

consists of games that target boys. The current version of Penelope was developed with an entirely different approach than other software that is on the market. Penelope's target audience is young girls. In the current version of Penelope, although that world is pre-set, the characters, their attributes, their description, and the narrative are all dynamic and mutable. This is a story creation game, not a game that tells the user someone else's story. The characters and most of the text is user-input (see Figs. 4.1 - 4.8).

One of the goals of Penelope was to address the issue of game play. Penelope was designed with a more female or feminine approach to game play in mind. It was created to get young girls interested in technology and science by allowing them to actually create a story. The current version does this by allowing the user to create a world. Penelope is neither a timed game nor a game that keeps score. It is a game that combines social interaction, fantasy and real world themes which are elements that girls have said are important to them in gameplay (Hanscome, 1995). When the user creates a character she is creating someone she can identify with, and since a large part of defining a character in Penelope is defining its relationships to other characters this introduces the real world themes of everyday interactions between people. Penelope also appeals to girls preference for collaborative (Koch, 1994) work in the sense that in Penelope none of the characters are alone, they are all interconnected in the sense that they are defined by their relationships to one another.

Penelope's social value is not simply to make the world more fair (boys have their games girls should have theirs too) but also to give girls a needed inroad to early exposure to technology and thus promote later interest and proficiency in science and technology

based fields. By being a fun game, attracting and retaining the interest of girls, Penelope is providing that inroad.

Penelope by being fun for girls will get them to use technology outside of work or school. By showing them that computers can be fun it will make them more receptive to doing things “on line”. The better and more comfortable they are using computers, the more prepared they will be to face the challenges of our technical society.

V.A.2 Narrative language

Penelope is a narrative generating game. In the current version Penelope generates text that is more natural. Since a large part of the text is generated by the user with only sequence determined by the game, Penelope does a good job of providing a facility for creating narrative.

In the “create character description” panel the user can write a number of descriptions of a character in different situations. This actually creates the text that will become part of the narrative. For example, if the user inputs a description of Princess Valentine when she is on the mountain path on a clear night which says, “Princess Valentine standing on the mountain in the path, the gentle breeze blowing tiny wisps of her hair around head creating a golden halo as the moonlight clings to each strand. The stars dance in her eyes accompanied by the smile which delicately dances across her face.” (see Fig 4.3) The output in the story when/if Princess Valentine was seen on that garden path at night would be “Princess Valentine walks on the mountain path. Princess Valentine standing on the mountain in the path, the gentle breeze blowing tiny wisps of her hair around head creating a golden halo as the moonlight clings to each strand. The stars dance

in her eyes accompanied by the smile which delicately dances across her face.” (see Fig 4.1)

Penelope does not simply output text that is generated by the computer. It uses the user’s input to help create a story. This is a personal story to the user in the sense that it is what the user wrote not simply something created by the computer or by the person who created the game. With the oral tradition of our society fast becoming only a part of our history, having children create personal stories will give them an appreciation for the human aspect in storytelling. They will see that Penelope will give them whatever they put into it. The more of themselves they put into their characters, the more elaborate they make their world, the better and more personal their story will be.

V.A.3 Technical Design

The current version is in one sense cutting edge and in another somewhat outdated. The graphics and interface is of the game are antiquated, but the internal workings of the game are quite far ahead of what is currently on the market. The non-player characters in Penelope make decisions based on their mood, their tendencies to do certain things, who is around and pure chance. In a vary real sense they decide how and when to act in much the same way as real people do. They are a bit closer to truly autonomous agents than most anything else on the market today.

The narration generation is handled in a very clever way. Since Penelope is a story creation game it puts the burden and responsibility of the creation of the text on the user. The story is theirs. Penelope simply aids in the sequencing.

Most of the other challenges came from that fact that it is difficult to create a software package this large and accomplish anything other than chaos. These challenges are almost always the type that can be overcome by putting in more work. For example, first the variable that held the character through whose eyes the world was seen, i.e. the narrator, was part of the game structure but later when parsing the room description called for using this data there was no way to access it. In the end it was necessary to create a new parameter to room that would be set before a description was asked for. More and more problems like this arise as the software package grows. They simply take time to trace them to their origins and resolve them.

Penelope is technically ahead of most of the other story games on the market, not because it has the best software design or because it is the most fun, but because it is really the only story game that has really put its emphasis on story creation. Penelope is the only software package that I have seen that combines story creation software with a story game. It does not solve all of the issues it addresses but it is a step closer.

V.B Areas Which Need Further Development

Penelope is an ongoing project; one which I must admit I approached with a great deal of optimism as to what I could actually accomplish in just over three and a half months. Many of the goals set at the beginning of this phase of the project have been completed but there are still, as with all major research, questions to be answered. Penelope has generated more questions than answers; some would say that that is a sign that we have been doing truly valuable work.

This section does not contain all of the questions that came up during the project but it does contain the more urgent ones and suggestions on how to proceed.

V.B.1 Gender and Game Play

The main issue that came up in relation to gender and game play was which would be most effective in getting girls to play with computers, something that is purely a game or something that is more educational. Penelope is currently the former. If we can get girls to play with computers and see them as more than tools then we will have accomplished our goal. The opposite side of the argument is that a game that does not in some way educate is not a worthwhile way for a child to spend time. I believe that play is very important in every stage of life, but especially important in the first stages, so play for play's sake is fine, but the best solution would be to find a game that is first and foremost fun but also educates. It must be fun to get and keep children interested in playing it. Being educational would be an added benefit and when education is fun and less work like it seem easier.

V.B.2 Narrative language

The language aspect of the game, since it is user inputted, is very free. This can be good or bad. One direction further development could go is to add an educational aspect to Penelope by adding an intelligent thesaurus and the definition and pronunciation guide to the game. This would allow for more elaborate stories. It could also filter out profanity and help to insure that the text generated was correct both grammatically and morally.

Another direction that proposed itself during the course of the project was the idea of story. What constitutes a story and should Penelope impose a story structure on the

game? This question is left to the next developer, because it is essential to what Penelope will become. If a story structure is imposed then one way to do it is to create a goal that the characters will try to accomplish and a beginning. This gives the story a beginning and a direction but it severely limits the players creative freedom. This is a difficult question because its answer defines what Penelope will be.

V.B.3 Technical Design

The technical next steps are more straight-forward than the design issues presented above but they are also very much dependent on how the design issues are addressed. The first area that needs to be addressed is the creation of the other parts of the world. Currently only the character creation is functional, but for Penelope to fully realize its potential the entire world must be user generated. Implementing the “create room” and “create objects” is the first step.

Another important aspect of the game that need to be dealt with no matter which direction the project takes is the issue of time. There needs to be some sort of clock in the game. The clock would not be to limit the game play but to give to the characters and the user a sense of the passage of time and a realistic time of day. This would help to make the stories and setting more believable, which makes it easier to become totally engrossed in the story.

V.C Conclusions

There are a number of changes that could be made to improve Penelope. Most of them, however, are dependent on which way the developers choose to go with the project.

If they choose to make Penelope more educational by adding spelling, grammar checking and thesaurus functions which would mean also adding parsing and editing functions to the text input boxes in the description and other parts of the game where the user inputs text, would make Penelope more educational without taking away from the creation aspect or the fun. By giving Penelope a voice, and in addition to the users text adding some programmed or computer generated text, Penelope could assist children in learning to read by highlighting the words and reading along as the story goes and by adding a dictionary and defining highlighted words Penelope could be used to help expand children vocabulary.

If increasing Penelope's educational value is not the ultimate goal but creating a fun game is then adding more advanced graphics, and more character photos to choose from would make the game even more appealing. Another idea is to save and output the text (the story) at the end of the game.

The Penelope Project is ongoing and therefore still has many options to explore. Educational, entertaining, both, neither, whichever path is chosen it promises to be a hard one. Whichever path is chosen is also bound to be an enlightening one.

VI Conclusions

Penelope: The Story Weaver is a game which attempts to use the research done on how children play games and the structure of the games they play to engage, entertain and educate them. The purpose of this game is to attract girls to technology and the world of narrative language.

The Penelope project was created in response to noted societal trends that implied that girls in general were not interested in technology in any other capacity than as a tool. Boys, however, as a majority played with computers, and later in life were still interested in technology and science. Also noticing that the technology-based games that were on the market were almost exclusively targeted at a male audience, we created Penelope to target the young female population, which by the way is the majority.

Penelope is, first and foremost, supposed to be fun for girls. Getting girls to play with a computer and see it not as something simply used for work but to also associate it with fun will make it easier and more enjoyable for them once they are introduced to using a computer for work.

VII CREDITS

For the concept of Penelope, the title, her support, and her role as thesis advisor I would like to thank Prof. Justine Cassell. Justine you made it possible.

I would like to thank Teresa Algosó and Manuel Pérez for their work on version one of Penelope. Having a working demo to begin with made life almost livable. Thank you both.

A special thank you goes to Hannes Vilhjálmsson for his awesome artwork.

To Flora Sun I give many thanks for her assistance in the coding of Penelope. Flora, your help came just when it was most needed and I greatly appreciated it.

Last, but not least, for her input and guidance during every step of my involvement with this project, I give my heartfelt thanks to Jen Glos.

VIII Appendices

VII.A Users Manual

VII.A Users Manual

Penelope

User Manual

by

Willie Johnson, Jr.

Startup

To startup Penelope first check to see if all the required files are available and in their proper location. Some “make-shift” error checking has been added to the program, so that a text box will appear if one or more of the files are missing. This error checking is very “cheap” and ugly, and should probably be changed. If the error check is to be removed or replaced the code is in the file “main.c”. If all the files are not present, Penelope can generate a bus error, which will cause your machine to crash. This is very bad!! The error checking should take care of this, but it is best to be safe.

The files that must be present are:

- (a) DATA directory

(b) dconfig.text

(c) SAVE-DATA

(d) Penelope

Now, that all the files are where they should be, begin Penelope by clicking on the “Penelope” icon.

A screen should appear with a graphical title reading, “Penelope: The Story Weaver”. If your machines crashes check to make sure ALL the files are present.

Walk Through

Once Penelope has started you will see a graphical title reading, “Penelope: The Story Weaver” and several boxes you will see some text. The text reads,

“Greetings and welcome to Penelope: The Story Weaver...”

This is the introduction page from here you can, by clicking on one of the two buttons, either enter or edit the story world.

If you choose to edit the story world the introduction page will close and anew page will open. On this page you have the options of editing the objects, places or character or exiting the edit/create phase and entering the story world. Currently only the edit/create and done buttons are active.

By choosing the edit/create button you would bring up a character creation page. Here You can bring up a character by either typing the characters name in the box next to name or by using the pulldown menu under name. Then click on get or simply hit the return key.

Now that you have a character you can edit that characters mood by use the slide bars, or you can the relationships between that character and the other characters in the story by using the pulldown menus in the boxes along the bottom of the page.

Once you have brought a character up you can also open pages to edit the characters favorite sayings and the descriptions of that character in various situations. To edit the characters favorite sayings click the “Favorite Phrases” button, which will bring up a window in which you will see 10 boxes, some of which will be filled with text. Type in what ever you wish the character to say. You can use the word “NAME” to signify a persons name. This will be replaced with the name of the person that the character is talking to in the actual game.

When you have finished click the “submit” button, to submit your changes, or the “done” button to exit without saving. Do the same in “character” window. Now when characters speak to the player they will select a phrase from this new list of phrases.

If You choose the edit the characters description you simply have to click on the “Description” button, which will bring up a window with a large text box a the bottom and five small boxes across the top. These small boxes have pull down menus which can be used to specify the situation in which this description should be used. Then in the large text box enter a description. The “OK” will submit the description. The “clear” button will clear all fields and the get button will retrieve the first description that fits the situation specified by the five smaller boxes. The cancel button exits back to the character creation page.

When done on the character creation page simply click “submit” to all of your changes or “cancel” to exit to the game world without saving.

Once you enter the story world you will see a graphical title reading, “Penelope: The Story Weaver” and several boxes you will see some text. The text reads,

“Welcome to The world that you have created

Press the 'DOIT' button and we will begin your story'''

This box is called the action-box. All information about the actions that or the computer characters perform will be displayed here.

Directly below the action-box is the input-box. The input-box is where you will type commands to interact with the world. To the far right of the input-box is a button labeled "DOIT". Click this button now.

The screen should flash a few times and then you should see mountain path. Now lets perform an action. Click on the input-box and type in work "look". Now either hit <return> or click the "DOIT" button. The action-box should update, and you should see a description of the room, the people there and other information that you would get from looking around yourself.

You will notice that the computer is waiting for your next action. Each character in Penelope gets a chance to act and no other character can act during another persons turn. If you wish to make no action simple click on the "DOIT" button. The computer characters will make their actions and then it will be your turn again. You can click the "DOIT" button as many times as you like. Click a few times and something should happen.

Sara says hello to you:-) A picture of Sara and a large white square containing her dialog will appear. In Penelope one of the main things you will be doing is talking with other characters. Lets try talking to Sunshine. To do this type, "tell Sunshine hello there cat". After awhile (click the "DOIT" button a few times) the cat will respond and say something about cats. We cannot tell you exactly what will be said, because each character has a element of randomness, such that no two conversations will be exactly the same (assuming we have developed our characters a bit more). Besides we do not know if Sunshine likes us or not.

This concludes the design features that are currently in Penelope.

Loading and Saving

Modifications to the Penelope game can be easily stored and restored through the “load” and “save” buttons on the main menu.

Commands

Here you will find a list of commands available in Penelope:

get	take	give	drop
pickup	look	examine	tell
talk	say	exit	go
exits	inv	inventory	quit
commands			

Now we will learn about moving around in Penelope. In Penelope we can move by either typing or by using the arrow button near the bottom of the screen. But first we must find out what exits exist from this room (actually in this room it is kind of obvious, but in other rooms it may not). To get the list of available exits type the word “exits” and click the “DOIT” button.

You will see that an “north” exit exists. Now type “go north” and click the “DOIT” button. You will find yourself on the “Castle Bridge”. Type “look” to see what objects are in the room. There is a banana on the ground, I bet Sunshine would love a banana. We should pick it up. To do this type “get the banana”. Now lets go give it to the cat.

Click on the “north” arrow button. You will walk through the north doorway and enter the “castle”. To give the banana to the cat type “give the banana to the cat”. After the

cat has the banana, probe him about the key again. He should tell you exactly where it is at, in the castle.

Now its your turn to create and explore.

VII.B Technical Documentation

VII.B.1 Dictionary Class

Introduction

Dictionary implements a file based dictionary, which can store and retrieve word types and keywords associated with a given word.

Heritage

Base Class	None
Derived Classes	None

Using Dictionary

Use a object of the class dictionary when you need to maintain a series of words, their types, and associated keywords.

The dictionary object can retrieve word types and keywords for words with regular English extensions:

Accepted extensions and replacements:

-ed	-s	-ing
-ies		

New entries can be added to the dictionary by using the Dictionary Maker. When using this tool be sure that the “DATA” directory and the “dconfig.txt” file are in the in the same directory as the Dictionary Maker application. Note that these two files can be specified by other names in other locations, if you inform the dictionary. How to inform the dictionary will be covered in this section. Note also that the current version of the dictionary will

inform you about faulty types and keywords, but will NOT prevent you from storing such words. This could turn out to be a problem at some later date and should be corrected.

Data Members

Though this class contains all public data members, none of them should be altered directly. For more information about the data members see the “dictionary.h” file.

Member Functions

```
dictionary(char *filename, int n);  
// effects: constructs a new dictionary based on the information  
//         contained in the file 'filename'
```

```
dictionary(char * newname);  
// effects: constructs a new dictionary and assigns it a name
```

```
dictionary(void);  
// effects: constructs a new dictionary
```

```
~dictionary(void);  
//effects: destroys a dictionary and frees up memory
```

```
void initialize(void);  
//effects: initializes all variables
```

```
char *pathdefine(char *word);  
// effects: returns the pathname where the word can be  
// found or stored
```

```
void loadfile(char *filename);  
// effects: loads a file as a set of dictionary entries
```

```
void savefile(char *filename);  
// effects: saves a set of dictionary entries as a file
```

```
void newdictionary(char *password);  
// effects: removes all directory dictionary entries
```

```
void setdirectory(char *newdirect);  
// effects: assigns the directory in which the dictionary  
// entries are stored
```

```
char *getdirectory(void);  
// effects: returns the directory path name where the  
// dictionary is stored
```

```
void setname(char * newname);  
// effects: assigns a name to the current dictionary
```

```
// signals: NAME_TOO_LONG
```

```
char * getname(void);  
// effects: returns the name of the dictionary
```

```
bool iskeyword(char *keyword);  
// effects: returns true if 'keyword' is a keyword, else  
//         returns false
```

```
char *getkeywords(void);  
// effects: returns the complete keywordlist
```

```
void addkeyword(char * newkeyword);  
// effects: adds a new keyword to the list acceptable keywords  
// signals: WORD_TOO_LONG, TOO_MANY
```

```
bool istype(char *type);  
// effects: returns true if 'type' is a type, else  
//         returns false
```

```
char *gettypes(void);  
// effects: returns the complete typeslist
```

```
void addtype(char *newtype);  
// effects: adds a new type to the list acceptable types  
// signals: WORD_TOO_LONG, TOO_MANY
```

```
int isword(char *word);  
// effects: returns true if 'word' is a word, else  
// returns false
```

```
void removeword(char *inputword);  
// effects: remove the 'inputword' from the dictionary
```

```
void addword(char *newword, char *keyword, char *type);  
// effects: adds a new word to the current dictionary and associates  
// it with a given keyword and a type.  
// signals: unknown_type, unknown_keyword, word_too_large
```

```
void setkeyword(char * inputword, char * newkeyword);  
// effects: associates a new keyword with the "inputword"  
// signals: unkown_word, unknown_keyword
```

```
char * getkeyword(char * inputword);  
// effects: returns the keyword associated with the "inputword"  
// signals: unknown_word
```

```
void settype(char * inputword, char * newtype);  
// effects: associates a new keyword with the "inputword"  
// signals: unkown_word, unknown_type
```

```
char * gettype(char * inputword);  
// effects: returns the type associated with the "inputword"  
//signals: unknown_word
```

```
char * info(void);  
// effects: Returns a human readable string decribing the  
//         the dictionary.
```

```
void inputfile(char *filename);  
// effects: converts a formatted file into a dictionary
```

```
void outputfile(char *filename);  
// effects: saves the dictionary as a formatted file with the  
//         specified 'filename'
```

DICTIONARY: STRING 2 PARSER INPUT

```
part_of_speech stop(char *input);  
// effects: turns a string type from a dictionary into  
//         an acceptable type for the parser
```

Dictionary Helper Function

```
int wordcount(dictionary *dict);  
// effects: returns the number of words that are in the dictionary  
//      'dict'
```

```
void file2dictionary(char *filename, dictionary *dict);  
// effects: adds the words stored in the file 'filename' as  
//      dictionary entries
```

```
void words2file(char *filename, dictionary *dict);  
// effects: returns the number of words that are in the dictionary  
//      'dict'
```

VII.B.2 Parser Class

Introduction

The Parser class can extract information from the sentences. The current implementation of the parser can obtain agent, action, patient, location, and temporal information.

Heritage

Base Class	None
Derived Classes	None
Required Classes	dictionary, rtable

Using Parser

The parser class is very easy to use. Simply construct a new parser in the regular manner,

```
parser *dparser = new parser();
```

Then, when you need information about the parts of a sentence just pass that sentence to the parser using the *parse* command,

```
dparser->parse("the boy ate the ball");
```

and extract the information using the *getagent*, *getpatient*, and *getaction* commands,

```
strcpy(agent, dparser->getagent());
```

```
strcpy(action, dparser->getaction());
```

```
strcpy(patient, dparser->getpatient());
```

```
strcpy(location, dparser->getlocation());
```

```
strcpy(temporal, dparser->gettemporal());
```

Data Members

The parser object has no publicly accessible data members.

Member Functions

```
parser(void);
```

```
// effects: constructs a new parser.
```

```
~parser(void);
```

```
// effects: destroys parser and frees memory.
```

```
void setname(char* newname);
```

This function assigns a new name to the parser.

```
char* getname();
```

This function returns the name of the parser.

```
char* info();
```

This function returns information about the parser as a human readable string.

```
void parse(char* inputsentence);
```

This function takes a sentence *inputsentence*, parses it and stores the results within the parser object. Information about the parsed sentence can be obtained through the *get-* and *k-* commands given below.

```
char* getagent();
```

This function returns the agent from the last parsed sentence.

```
char* getaction();
```

This function returns the action from the last parsed sentence.

```
char* getpatient();
```

This function returns the patient from the last parsed sentence.

```
char* getlocation();
```

This function returns the location marker from the last parsed sentence.

```
char* gettemporal();
```

This function returns the temporal marker from the last parsed sentence.

```
char* kagent();
```

This function returns the keyword for the agent of the last parsed sentence.

```
char* kaction();
```

This function returns the keyword for the action of the last parsed sentence.

```
char* kpatient();
```

This function returns the keyword for the patient of the last parsed sentence.

```
char* klocation();
```

This function returns the keyword for the location marker of the last parsed sentence.

```
char* ktemporal();
```

This function returns the keyword for the temporal marker of the last parsed sentence.

Member Functions: private

```
void init_variables();
```

Initializes the internal variables. Thus, when this command is called the results from the previous parse are erased.

```
void init_sentence();
```

This function clears the data structure that contains the current sentence. This function must be called before a new sentence can be parsed.

```
read sentence into data structure
```

```
void getinputsentence(char* inputsentence);
```

```
void removepunctuation(char* inputsentence);
```

```
struct word makeword(char* theword);
```

gets type of word

```
bool word_is_place_preposition(char* word);
```

```
bool word_is_time_preposition(char* word);
```

```
bool word_is_punctuation(char* word);
```

parse sentence

```
void parse_input_sentence();
```

```
int look_for_subject(int n,int subjectnumber);
```

```
int look_for_behavior(int n,int behaviornumber);
```

```
int look_for_object(int n,int objectnumber);
```

```
void look_for_place();
```

```
void look_for_time();
```

VII.B.3 Response Table Class (rtable)

Introduction

The response table class maintains a list of responses, in a table. Upon request the user can call a procedure to randomly return any one of the tables' responses.

Heritage

Base Class None

Derived Classes None

Using Response Table

The response table object is used to generate random responses from a list of stored responses. It is very easy to use. To add a response to the table simply use the *add* command. To remove a response to the table simply use the *remove* command. Then to get a random response from the current list of entries just call *rget*.

Data Members

Though this class contains all public data members, none of them should be altered directly. For more information about the data members see the “rtable.h” file..

Member Functions

```
char *inputget(char *input, int n);  
  
// effects: returns the 'n'th element from an 'input', which  
  
//has been created by "outputtable  
  
  
  
// Note: error detection needed
```

```
void setname(char* name);  
  
// effects: assigns a new name to the response table
```

```
char* getname(void);  
  
// effects: returns the name of the response table
```

```
void add(char *resp);  
  
// effects: adds a response to the list of possible  
  
//responses
```

```
char* get(int number);  
// effects: returns response located at position "number"  
//in the response table  
  
// Note: an exception should be placed here to check for out  
// of bounds cases
```

```
char* rget(void);  
// effects: returns a random response from the list  
//of responses
```

```
char* output(void);  
// effects: returns the information stored in the response table as  
//a machine readable string
```

```
void input(char *input);  
// effects: turns a machine readable string into a response table  
//
```

```
char* info(void);  
  
// effects: returns information about the response table  
  
//as a human readable string
```

```
rtable();  
  
// effects: constructs a new response table
```

```
rtable(char* name);  
  
// effects: constructs a new response table and assigns  
  
//it a name.
```

```
rtable(char* info, int n);  
  
// effects: constructs a new response table and fills it  
  
//in with the information in 'info'
```

```
void remove(char *str);  
  
// effects: removes the response 'str' from the response  
  
//table
```

```
void removeall(void);
```

```
// effects: removes all the responses from the response table
```

```
int numresponses(void);
```

```
// effects: returns the number of responses in the response table
```

HELPER FUNCTION

```
void searchreplace(char *bigstring, char *searchword, char *replaceword);
```

```
// effects: searches 'bigstring' for the 'searchword' and replaces it
```

```
//with the 'replaceword'
```

VII.B.4 State Class

Introduction

The State class is one of the more integral parts of the Penelope project. It is also a bit difficult to explain in just words.

Nodes:

A single state is one node in a weighted directional graph. It is used to mark the position in a conversation. This position could be anything from different environmental influences to markers in a story.

Edges:

Each state has an edge list. Each edge points to another state (an edge can point back to its own start state). Associated with each edge is an input-string, a color, a weight, and a response table.

The input-string is of the form:

“<agent> <action> <patient>”,

where any or all of their elements can optionally be left blank. The input-string is the first determiner of whether or not an edge can be traversed.

For example if an edge had the input-string “throw ball” and the state class was given the string “girl throw ball”. then the edge would be marked as one that could be traversed. For this same edge if the input-string was “girl eat ball”, the edge would be marked in a manner that made traversing it impossible.

Put simply in order for an edge to be traversable its input-string correlate to a given string.

The color of the edges are simply used as internal book keeping. They simply classify the edges. They can be any integer value and color of an edge does not effect its chance of being traversed. The current implementation uses the following predefined colors:

good	1
bad	0
neutral	2.

These values can be used in programs as well. To add new colors edit the file “define.h”.

Edges also have weight. Once the set of traversable edges has been determined then the edge that will be used is chosen by assigning each edge in the set a probability based on its weight and the choosing randomly.

Last but not least, each edge has an associated rtable. Once an edge is chosen then a response is chosen from its rtable.

Heritage

Base Class	None
Derived Classes	None
Required Classes	rtable

Using State

First select a list of colors to use, i.e. the good, bad, and neutral ones from above. These colors will be used to paint the edges of the state object. Like colors can be manipulated together. For example if you wanted a character to be really happy when they are happy all you have to do is increase the weights on all the good colored objects.

Next set the range of values for edge weights. From 1 to 100 is the suggested range. Now you can begin.

Data Members

Though this class contains all public data members, none of them should be altered directly. For more information about the data members see the “state.h” file.

Member Functions

```
void initialize(void);  
  
// effects: initializes internal variables
```

```
int weight(edge inedge);  
  
// effects: returns the relative edge weight
```

```
void addedge(state *next, color c, int w, char *input, rtable *resp);  
  
// effects: adds a new edge to the state
```

```
void input2valid(char *input);  
  
// effects: for all edges that have input string 'input'  
//their count parameter is set to 1. If the input  
//does not match then count is set to 0. This  
//signifies whether or not a given edge is valid  
//for further procedures
```

```
void color2valid(color input);  
  
// effects: for all edges that have input color 'input'  
//their count parameter is set to 1. If the input  
//does not match then count is set to 0. This  
//signifies whether or not a given edge is valid  
//for further procedures
```

```
void adjustweight(color input, int n);  
// effects: adjust the weights of all edges with the 'input'  
//color by a factor of n
```

```
void pickbyweight(void);  
// effects: based on the relative weights of edges an edge  
//is randomly selected and the parameters response  
//and next are set
```

```
void traverse(char *input);  
// effects: selects an output based on the 'input' and  
//the weights of edges. The ouput id stored in  
//the parameters response and next.
```

```
void traverse(char *agent, char *action, char *patient);  
// effects: selects an output based on the input and  
//the weights of edges. The ouput id stored in  
//the parameters response and next.
```

```
char *getresponse(void);  
// effects: returns a generated response
```

```
state *getstate(void);  
// effects: returns the next state
```

```
state(void);  
// effects: constructs a new state
```

VII.B.5 Object Class

Introduction

Object implements the underlining structure for all of the objects within the Penelope project. This includes characters and rooms, which are derived from this class.

Heritage

Base Class	None
Derived Classes	Character, Room

Using Objects

<needs to be filled in>

Data Members

<needs to be filled in>

Member Functions

void initvariables();

initialuzes internal variables

object();

constructor

object(object* newowner);

constructor

object(char* name_of_object);

constructor

object(char* name_of_object,object* newowner);

constructor

~object(void);

destructor

void setname(char* newname);

changes object's name to match *newname*

char* getname();

returns the name of the object

```
void setdescription(char* newdescription);
```

set the description of the object

```
virtual char* getdescription();
```

returns the description of the object.

```
int numberobjectsinpossession();
```

returns the number of objects owned by this object.

```
void setpicture(char* picturename);
```

associates the picture given by *picturename* with the object.

```
char *getpicture();
```

returns the *picturename* associated with this object.

```
char* gettype();
```

returns the type of object. i.e. room or character

```
void settype(char* newtype);
```

sets the type of the object

```
bool isownable(object* obj);
```

Returns true if the object can be owned by another object and false otherwise.

```
bool isownable(char* type_to_check);
```

Returns true if an object of type *type_to_check* can be owned by another object and false otherwise.

```
int numobjects();
```

Returns the number of possessions that object has.

```
bool addobject(object *newobject);
```

```
object* dropobject(object* object_to_be_dropped);
```

```
object* dropobject(char* object_to_be_dropped);
```

```
char* objects();
```

returns a string that contains the names of all the objects owned by this object.

```
bool returnobject(char* string);
```

each time it is called it copies into *string* the name of an object it owns once it has gone through the list once it returns the string "DONE". Returns true unless the object has no possessions.

```
char* returnobject();
```

each time it is called it returns the name of an object it owns once it has gone through the list once it returns the string "DONE".

```
char* returncharacter(int refnum);
```

```
bool isowned(char* objectname);
```

```
bool isowned(object *object_being_checked);
```

```
object* return_isowned(char* objectname);
```

```
void setowner(object* newowner);
```

```
object* getowner();
```

```
void object::setID(char *newID);
```

```
char *object::getID(void);
```

Member Functions: Private

```
private:
```

```
bool unowned(object* newobject);
```

```
void loseobject(object* object_to_be_dropped);
```

```
void changeowner(object* newowner);
```

```
void drop_possession(int n);
```

VII.B.6 Room Class

Introduction

Room implements the underlining structure for all of the rooms within the Penelope project.

<needs to be filled in>

Heritage

Base Class	Object
Derived Classes	None

Using Rooms

<needs to be filled in>

Data Members

<needs to be filled in>

Member Functions

```
room();  
// : object();
```

```
room(char* newname);
```

```
// :object();
```

```
~room(void);
```

```
char prep_desc[SENTENCE_LENGTH];
```

```
char mainact[WORD_LENGTH];
```

```
char* getprep();
```

```
void setprep(char* newprep);
```

```
virtual char* getdescription();
```

```
void addexit(char* exitdirection, room* nextroom, bool hide);
```

```
bool isexit(char *theexit);
```

```
bool is_unhidden_exit(char *theexit);
```

```
void hide(char* exitname, bool hide);
```

```
void remove_exit(char* direction);
```

```
char* getroomactor(int refer);
```

```
int numberpeopleinroom();
```

```
room* exit(char* direction);
```

```
char* exitlist();
```

```
void setowner(object* newowner);
```

Member Functions

```
void initvariables();
```

```
int numberexits();
```

```
void drop_exit(int n);
```

VII.B.7 Character Class

Introduction

Character implements the underlining structure for all of the characters within the Penelope project.

<to be filled in>

Heritage

Base Class	Object
Derived Classes	None

Using Characters

<needs to be filled in>

Data Members

<needs to be filled in>

Member Functions

```
character(void);
```

```
character(char* newname);
```

```
character(room* starting_room);
```

```
character(char* newname, room* starting_room);
```

```
~character(void);
```

```
char* getdescription();
```

```
void input_pattern(char* list);
```

```
void input_droplist(char* list);
```

```
void input_getlist(char* list);
```

```
void input_lopeople(char* list);
```

```
void input_lipeople(char* list);
```

```
void input_dipeople(char* list);
```

```
void input_hapeople(char* list);
```

```
void input_unpeople(char* list);
```

```
void inputdialog(char* newdialog,character* newspeaker);
```

```
void addstate(state *inputstate);
```

```
void reaction();
```

```
void addPDitem(char* todin, char* actionin, char* locin, char* obsin, char* environ,  
char* Pdecin);
```

```
void replacePD(int refin, char* todin, char* actionin, char* locin, char* obsin, char*  
envin, char* Pdecin);
```

```
char* getmatchPD(char* todin, char* actionin, char* locin, char* obsin, char* envin);
```

```
void clearPDs(void);
```

```
int getnumPDitems(void);
```

```
bool saysomething();
```

```
bool dropsomething();
```

```
bool getsomething();
```

```
bool givesomething();
```

```
bool movesomewhere();
```

```
void setagent2(character* thing);
```

```
character* getagent2();
```

```
void setextra(char* string);
```

```
char* getextra();
```

```
void setobj(object* newobj);
```

```
object* getobj();
```

```
void setdowhat(action newaction);
```

```
action getdowhat();
```

```
void setmood(int energylev, int moodlev, int emotionallev);
```

```
int* getmood(void);
```

```
void adjustmood(void);
```

```
void addmooditem(char *mobject, color mcolor, int mfactor);
```

```
void addgetitem(char *itemname);
```

```
void addgiveitem(char *itemname, char *towho, char *thekey);
```

```
void adddropitem(char *itemname);
```

```
void addlikepitem(char *itemname);
```

```
void addlovepitem(char *itemname);
```

```
void adddislikepitem(char *itemname);
```

```
void addhatepitem(char *itemname);
```

```
void adduncertainpitem(char *itemname);
```

```
void expandpattern(char *direction);
```

```
void setchance(action theaction, int chance);
```

```
int getchance(action theaction);
```

```
char *getpdescription();
```

```
char *getmooditems(color thecolor);
```

```
char *getgetitems();
```

```
char *getgiveitems();
```

```
char *getdropitems();
```

```
char *getlikepitems();
```

```
char *getdislikepitems();
```

```
char *getlovepitems();
```

```
char *gethatepitems();
```

```
char *getuncertainpitems();
```

```
void delete_all_items(void);
```

```
void addtalktoperson(char *name);
```

```
void setsocialfactor(int i);
```

```
int setsocialfactor(void);
```

Member Functions

```
action chooseaction();
```

```
bool can_drop_anything();
```

```
bool can_get_anything();
```

```
bool can_give_anything();
```

```
bool can_say_anything();
```

```
void doaction();
```

```
void initvariables();
```

VII.B.8 Object Handler Class (objhandler)

Introduction

The object handler class handles actions taken with any types of objects that may exist in the game. This includes ordinary object, rooms, and characters. The current functionality of the object handler class includes the following actions: look, get, drop, give, say, go, exit, inventory, and commands. It should be noted that for more complex, and less general commands, a larger portion of the functionality may need (most likely will need to be) incorporated into the objects themselves.

Heritage

Base Class	None
Derived Classes	None

Using Object Handler

The object handler will make manipulating objects an easier task for higher level procedures. It does this by "handling" much of the interaction between all types of objects.

To use an object handler simple create one in the usual manner. The add object, characters, and rooms to the object handler's control list by using the commands addobjct, addroom, and addactor. Objects that are added as rooms or characters are also added to the standard objects control list.

Once items have been added you will be able to generate interactions between the controlled objects by simply using the commands within object handler class. For example:

Consider the characters Bob and Mary, which have been declared using the standard methods for creating characters. Bob owns a ball and would like to give it to Mary. Both Bob and Mary are in the kitchen and both have been added to the object handler HANDLER. To perform this action we simple need call the give command,

```
HANDLER->give(kitchen, Bob, Mary, ball);
```

In addition to this, the object handler commands will also generate and return a string describing the action that has just taken place. Using the same example from above the following line,

```
printf("%s\n", HANDLER->give(kitchen, Bob, Mary, ball));
```

would generate the following output:

Bob gives the ball to Mary.

The object handler also supports lookup commands so that objects can be identified by their char* names, such as those set by the setname command.

Data Members

Though this class contains all public data members there are none that should be altered directly. For more information about the data member members see the "objhandler.h" file.

Member Functions

```
void initialize(void);
```

```
// effects: initializes startup variables
```

```
objhandler(void);
```

```
// effects: constructs a new object handler
```

```
void addobject(object *input);  
// effects: adds a new object to the list of objects  
//that are handled by the object handler
```

```
void addactor(character *input);  
// effects: adds a new object to the list of objects  
//that are handled by the object handler
```

```
void addroom(room *input);  
// effects: adds a new object to the list of objects  
//that are handled by the object handler
```

```
object *lookup(char *name);  
// effects: returns an object with the name 'name'
```

```
character *lookupactor(char *name);  
// effects: returns a character with the name 'name'
```

```
room *lookuproom(char *name);  
// effects: returns a room with the name 'name'
```

```
int rlookup(char *name);  
// effects: returns the position of the object with  
//the name 'name'
```

```
char* getmainactor();  
// effects: none  
//returns: name of main actor
```

```
char *look(object *theroom, object *agent, object *obj);  
// effects: returns the description of an object if  
//is present in 'theroom' or it is in the  
//agent's possession
```

```
char *look(object *theroom, object *agent, room *obj);  
// effects: returns the description of an object if
```

```
//is present in 'theroom' or it is in the
//agent's possession
```

```
char *look(object *theroom, object *agent, character *obj);
// effects: returns the description of an object if
//is present in 'theroom' or it is in the
//agent's possession
```

```
char *look(char *theroom, char *agent, char *obj);
// effects: returns the description of an object if
//is present in 'theroom' or it is in the
//agent's possession
```

```
bool bget(object *theroom, object *agent, object *obj);
// effects: attempts to places the object 'obj' into the
//ownership of agent'. if the transaction is
//possible it takes place and true is returned,
//else false is returned
```

```
bool bget(char *theroom, char *agent, char *obj);  
// effects: attempts to places the object 'obj' into the  
//ownership of'agent'. if the transaction is  
//possible it takes place and true is returned,  
//else false is returned
```

```
char *get(object *theroom, object *agent, object *obj);  
// effects: trys to perform the bget function and returns  
//a string describing the results
```

```
char *get(char *theroom, char *agent, char *obj);  
// effects: trys to perform the bget function and returns  
//a string describing the results
```

```
bool bdrop(object *theroom, object *agent, object *obj);  
// effects: attempts to drop the object 'obj' into the  
// ownership of 'theroom'. If the drop was successful  
//then the true is returned, else flase is returned
```

```
bool bdrop(char *theroom, char *agent, char *obj);  
// effects: attempts to drop the object 'obj' into the  
// ownership of 'theroom'. If the drop was successful  
//then the true is returned, else flase is returned
```

```
char *drop(object *theroom, object *agent, object *obj);  
// effects: trys to perform the bdrop function and returns  
//a string describing the results
```

```
char *drop(char *theroom, char *agent, char *obj);  
// effects: trys to perform the bdrop function and returns  
//a string describing the results
```

```
bool bgive(object *theroom, object *agent, object *agent2, object *obj);  
// effects: attempts to drop the object 'obj' into the  
// ownership of 'theroom'. If the drop was successful  
//then the true is returned, else flase is returned
```

```
bool bgive(char *theroom, char *agent, char *agent2, char *obj);
```

```
// effects: attempts to drop the object 'obj' into the
// ownership of 'theroom'. If the drop was successful
//then the true is returned, else false is returned
```

```
char *give(object *theroom, object *agent, object *agent2, object *obj);
// effects: tries to perform the bgive function and returns
//a string describing the results
```

```
char *give(char *theroom, char *agent, char * agent2, char *obj);
// effects: tries to perform the bdrop function and returns
//a string describing the results
```

```
char *say(room *theroom, character *agent, character *agent2,
char *dialog);
// effects: checkst to see if the 'agent2' owned by 'theroom'.
// If it is then this procedure passes the dialog
//to 'agent2' and returns the result.
```

```
char *say(char *theroom, char *agent, char *agent2,
```

```
char *dialog);  
  
// effects: checkst to see if the 'agent2' owned by 'theroom'.  
  
// If it is then this procedure passes the dialog  
  
//to 'agent2' and returns the result.
```

```
bool bgo(room *theroom, object *obj, char *theexit);  
  
// effects: attempts to move the object 'obj' through  
  
//the exit 'theexit'. If this is possible  
  
//then object 'obj' is placed into the  
  
//ownership of the room through the exit  
  
//and true is returned, else flase is returned
```

```
bool bgo(char *theroom, char *obj, char *theexit);  
  
// effects: attempts to move the object 'obj' through  
  
//the exit 'theexit'. If this is possible  
  
//then object 'obj' is placed into the  
  
//ownership of the room through the exit  
  
//and true is returned, else flase is returned
```

```
char *go(room *theroom, object *obj, char *theexit);
```

```
// effects: tries to perform the bgo function and returns  
//a string describing the results
```

```
char *go(char *theroom, char *obj, char *theexit);  
// effects: tries to perform the bgo function and returns  
//a string describing the results
```

```
character *getactor(void);  
// effects: returns a new chaaracter from the stored  
//character list
```

```
char *inventory(object *obj);  
// effects: returns the list of objects owned by object 'obj'
```

```
char *inventory(char *obj);  
// effects: returns the list of objects owned by object 'obj'
```

```
char *exits(room *theroom);
```

// effects: returns the list of objects owned by object 'obj'

char *exits(char *theroom);

// effects: returns the list of objects owned by object 'obj'

VII.B.9 Game Handler Class (game or gameplayer)

Introduction

The game handler class handles and coordinates all actions made by players or computer controlled agents. It brings all the other sub procedures together.

Heritage

Base Class	None
Derived Classes	Cgame

Using Game Handler

To use the game handler class, first the user must declare and define a list of objects in the usual manner. Then the should construct a new game handler object. Next the user should add all objects to the game handler's object handler, defined as a public construction under the name handler. For example:

Let us consider adding the following defined objects:

object: ball, banana, key

characters: manny and a cat

rooms: hall, closet

Then we define a new game handler as follows:

```
game *the_game=new game();
```

Now lets add them to the game handler:

```
the_game->handler->addactor(manny);
```

```
the-game->handler->addroom(hall);
```

```
...
```

```
the_game-.handler->addobject(ball);
```

Next you should decided which character will be controlled by human player (note: the ability support more than one player is supported if needed). This is done with the setview function. The user should also decide on some opening text using the setstarttext function.

With all this done you cannot play the game using only the interface to the game handler functions.

```
void playgame(game *ME)
```

```
{
```

```
    char word[15]="",setence[1000]="",space='a';
```

```
    //this parts setup the players move
```

```

ME->setupnext();
while(ME->setupnext()) {}
printf("\n\n%s\n\n", ME-> getstarttext());
while (ME->.play);
{
    // set up player prompt
    printf("\nWhat Next: ");

    //This section reads in a new typed in sentence
    scanf("%s", &word);
    strcat(sentence, word);
    while(space != '\n')
        {
            strcat(sentence,"");
            scanf("%s", &word);
            scanf("%c", &space);
            strcat(sentence, word);
        }

    // This section resolves a new player action
    printf("\n\n%s\n", ME->donext(sentence));
    while(ME->setupnext())
        {
            printf("\n\n%s\n", ME->donext(""));
        }
}

```



```
    }  
}
```

This is just a simple example, but it can actually be used to play an entire game!! Of course other functionality, such as graphics, must be layered over this design. It should be noted, however, that lower level functions do support pictures.

Data Members

Though this class contains all public data members there are only a few, which are actually meant to be manipulated. The one which must be used is the object handler defined as handler. For example on how this data member is used see the section above and the object handler class for more information about the data members see the "game.h" file.

Member Functions

```
void initialize(void);  
  
// effects: initializes stuff for a game
```

```
game(void);  
  
//effects: constructs a new game
```

```
char *act(character *agent, character *agent2, action dowhat,  
object *obj, char *extra);  
// effects: handles all character actions (text based)
```

```
void setview(character *input);  
// effects: selects through whose eyes you see the game  
// sets handeler mainactor to name of input.
```

```
action stringaction(char *input);  
// effects: returns that action associated with the  
//'input' string
```

```
void playerreact(character *player, char *actionstring);  
// effects: conducts a reaction based on keyboard inputs  
//(visual based)
```

```
char *donext(char *instring);  
//plays a game
```

```
bool setupnext(void);
```

```
char *getplayerinv(void);
```

```
// effects: returns a player's inventory
```

```
bool getpicture(void);
```

```
// effects: returns the picture number of an observed
```

```
//object, if the player looked at an object,
```

```
//else it returns false
```

```
void setstarttext(char *inputtext);
```

```
// effects: sets the startup text for a game
```

```
char *getstarttext(void);
```

```
// effects: returns the startup text for the game
```

```
void game::savecharacters(void);
```

```
// effects: saves all characters to the disk
```

```
void savegame(void);
```

```
void loadgame(void);
```

```
void loadcharacters(void);
```

```
// effects: loads all characters from the disk
```

```
void setpath(char *newpath);
```

```
// effects: sets the path where files should be saved
```

```
void addcpicture(char *newpicture);
```

```
// effects: adds a possible picture to the list of cpictures
```

```
char *nextcpicture(void);
```

```
// effects: returns the next character picture
```

Member Functions: Private

```
int takeout_extra(char myarray[MAX_INPUT_STRING][WORD_LENGTH], int i);
```

```
// effects: removes extra words
```

```
void initplayerreact(character *player);
```

```
// effects: initializes player reaction variables
```

```
int resolve_input_string(char *actionstring, char  
myarray[MAX_INPUT_STRING][WORD_LENGTH], int size);
```

```
// effects: converts the input string into an array and stores it in myarray
```

```
int resolve_player_action(character *player, char  
myarray[MAX_INPUT_STRING][WORD_LENGTH], int size);
```

```
// effects: resolves a players action based on the information stored in 'myarray'
```

```
bool isNULLobj(character *player, char *name, action p_action);
```

```
// effects: returns true if the action object of 'player' is the NULL object
```

```
bool isNULLagent(character *player, char *name);
```

```
// effects: returns true if the action object of 'player' is the NULL object
```

```
int resolvecap(character *player, char  
myarray[MAX_INPUT_STRING][WORD_LENGTH], int i);
```

```
// effects:: checks for alternate capitalized object names
```

```
int resolve_agent_cap(character *player, char  
myarray[MAX_INPUT_STRING][WORD_LENGTH], int i);
```

```
// effects:: checks for alternate capitalized actor names
```

```
int setobj(character *player, char myarray[MAX_INPUT_STRING][WORD_LENGTH],  
action myaction, int i);
```

```
// effects: sets the object for a player's action
```

```
int setagent2(character *player, char  
myarray[MAX_INPUT_STRING][WORD_LENGTH], int i);  
// effects: sets the agent for a player's action
```

IX Bibliography

- Dangelmaier, Heidi. "Gender and the Art of Designing Interactive Media". Computer Game Developers. (January 4, 1995).
- Glos, Jen. "Computer-Generated Narrative: Statement of Problem". (April 9, 1995)
- Hafner Katie. "What is Little Girls' Software Made of?". PC, (October 1994).
- Hanscome, Barbara. "Beyond Chrome and Sizzle", Game Developer, (Feb 1995).
- _____. "Gammin' for Girls". Game Developer. (Oct. 1995).
- Huff, Charles, and Joel Cooper. "Sex Bias in Educational Software: the effect of designers' stereotypes on the software they design." Journal of Applied Social Psychology. Vol 17 (6) p. 519-532.
- Jenkins, Henry. "'x Logic': Repositioning Nintendo in Children's Lives". Quarterly Review of Film & Video, Vol 14 (4) pp. 55-70, (1993).
- Jones, Amanda, and Sheila M. Glenn. "Gender Differences in Pretend Play in a Primary School Group". Early Child Development and Care. (1991). Vol 77. p. 127-135.
- Kantrowitz, Barbara. "Men, Women and Computer Games". Newsweek. (May 16, 1994)
- Kennedy, Beth. "Sugar and Spice ... Software for Women and Girls??!" . Excerptes from a three part series of articles: "Opportunities for the Present and Future: Productes and Niche Demographic Markets". (1995)
- Koch, Melissa. "No Girls Allowed!". Technos. (Fall 1994). Vol 3, No. 3.
- Morse, Susan. "Why Girls Don't Like Computer Games". AAUW Outlook. (Winter 1995).