

INTERACTIVE STORYTELLING ENGINES

A Dissertation

by

TEONG JOO ONG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2006

Major Subject: Computer Science

INTERACTIVE STORYTELLING ENGINES

A Dissertation

by

TEONG JOO ONG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	John J. Leggett
Committee Members,	Richard K. Furuta
	Frank M. Shipman III
	Hugh D. Wilson
Head of Department,	Valerie E. Taylor

August 2006

Major Subject: Computer Science

ABSTRACT

Interactive Storytelling Engines.

(August 2006)

Teong Joo Ong, B.S., University of Oregon;

M.C.S., Texas A&M University

Chair of Advisory Committee: Dr. John J. Leggett

Writing a good story requires immense patience, creativity and work from the author, and the practice of writing a story requires a good grasp of the readers' psychology to create suspense and thrills and to merge the readers' world with that of the story. In the digital writing space, authors can still adhere to these rules of thumb while being aware of the disappearance of certain constraints due to the added possibility of narrating in a nonlinear fashion.

There are many overlapping approaches to interactive storytelling or authoring, but each of the approaches has its own strengths and weaknesses. The motivation for this research arises from the perceived need for a new hybrid approach that coalesces and extends existing approaches. Since each of the approaches empowers certain aspects of the storytelling and narration process, the result forces a new research direction which eliminates certain weaknesses exhibited by a single approach, due to the synergistic nature of the various approaches. We have developed: 1) a Hybrid Evolutionary-Fuzzy Time-based Interactive (HEFTI) storytelling engine that generates dynamic stories from a set of authored story constructs given by human authors; 2) a set of authoring tools that allow authors to generate the needed story constructs; and, 3) a storytelling environment for them to orchestrate a digital stage play with computer agents and scripts.

We have conducted a usability study and system evaluation to evaluate the performance of the engine. Our experiments and usability study have shown that the authoring environment abstracted the complexity of authoring an interactive, dynamic story from the authors with the use of windows-based interfaces to help them visualize various

aspects of a story. This reduces the amount of learning and knowledge required to start having the pleasure of authoring dynamic stories. The studies also revealed certain features and tools that may be reflected by authoring tools in the future to automate various aspects of the authoring process so that the authors may spend more time thinking rather than writing (or programming) their stories.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my advisor, John Leggett. His sharp insight, quick wit and patient guidance made this research endeavor full of fun and challenges. Dr. Leggett has taught me well; especially that we may have the best algorithms and software in the world, but they are worthless if no users can use them. I would like to thank Dr. Furuta, Dr. Shipman and Dr. Wilson, who gave me many useful suggestions and inspired so many research ideas. I would like to also thank them for taking time in reviewing this dissertation and giving valuable comments.

I have grown academically and personally from interactions with many other people from the group of Digital Flora of Texas and Humanities Informatics projects. I would like to thank Dr. Haowei Hsieh for his exciting discussions with me and especially his help for my experiments in this research. I would like to also thank Dr. Luis Francisco-Revilla, Deng Jie, Konstantinos Meinkos, Unmil P. Karadkar, Rui Li, Yi Yang and other students in the Center for the Study of Digital Libraries for cooperative works, exciting discussions, help and fun moments.

Last, but not least, I would like to thank my parents and my girl friend, Siang-Chin Tan, for their support, their encouragement and their love throughout all these years.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
ACKNOWLEDGEMENTS.....	v
TABLE OF CONTENTS.....	vi
LIST OF FIGURES	ix
LIST OF TABLES.....	xiv
1. INTRODUCTION	1
1.1 Background and Literature Review	2
1.2 Combining Evolutionary Algorithms and Fuzzy Logic for Dynamic Story Generation	6
1.3 Motivation and Scope	7
1.4 Scenarios of Use	9
1.5 Objectives of Research	11
2. ARCHITECTURE OF THE INTERACTIVE STORYTELLING ENGINE.....	12
2.1 Genetic Algorithm and Fuzzy System	15
2.1.1 Genetic Algorithm	15
2.1.2 Fuzzy Logic	18
2.2 Design of the Evolutionary Fuzzy System.....	22
3. CONSTRUCTING STORIES	30
3.1 Encoding the Story Components.....	30
3.2 Decoding the Story Components	34
3.3 Stories Controlled by Rules	35
4. THE INTERACTIVE AUTHORING ENVIRONMENT (IAE).....	39
4.1 Agent Characters	39
4.2 Text-to-Speech (TTS) Engines	43
4.3 The IAE Interfaces.....	44
4.3.1 The Authoring Interface.....	44
4.3.2 The Reading Interface.....	45

	Page
4.4 Syntax and Semantics of the XML Tags	47
4.4.1 Replaying a Story.....	47
4.4.2 Actions.xml.....	48
4.4.3 Agents.xml.....	49
4.4.4 Objects.xml.....	50
4.4.5 Rules.xml.....	51
4.4.6 Scripts.xml.....	53
4.4.7 Story.xml.....	53
4.4.8 Templates.xml.....	55
4.4.9 Textstrings.xml.....	58
5. AUTHORING AN INTERACTIVE, DYNAMIC STORY	59
5.1 Case Study: A Simple Three Little Pigs' Story.....	61
5.2 Editing the Story Elements	72
5.3 Authoring Stories with the Drag-and-Drop Interface.....	75
6. STORY STATES AND BRANCHES IN A DYNAMIC STORY	79
7. AUTHORING A TEXT BASED STORY	85
7.1 The Interface.....	85
7.1.1 Authoring a Story.....	87
7.1.2 Story Element Distribution	89
7.1.3 Comparing the Story Elements.....	91
7.1.4 Visualizing Distribution of Story Elements.....	93
7.1.5 Affinity Measure of Story Elements and Story Threads.....	96
7.1.6 Hamming Distance Affinity Measure.....	98
7.1.7 Multiple Contiguous Bit Affinity Measure.....	100
7.1.8 Run-time Complexity	101
8. EVALUATION	104
8.1 Evaluation of the Interactive Authoring Environment (IAE).....	104
8.1.1 Profile of the Test Subjects.....	105
8.1.2 Research Procedures	106
8.1.3 Observations and Analyses.....	107
8.1.4 Desirable Features.....	109
8.1.5 Undesirable Features.....	110
8.1.6 Future Improvements.....	111
8.1.7 Conclusions.....	112
8.2 Evaluation of the Storytelling Engine.....	113

	Page
8.3 Affinity Measures	114
8.4 Setup	117
8.4.1 Linear Story	118
8.4.2 Non-linear Story	127
9. CONCLUSION AND FUTURE WORK	134
APPENDIX A STORY AND AGENT SCRIPTING FUNCTIONS	148
APPENDIX B TASK SHEET FOR THE USABILITY STUDY	151
APPENDIX C QUESTIONNAIRE FOR THE USABILITY STUDY	156
APPENDIX D USABILITY STUDY RESULTS	163
APPENDIX E TEST RESULTS FROM EVALUATION	164
VITA	187

LIST OF FIGURES

	Page
Figure 1. The overall system architecture and subsystems' relationships	12
Figure 2. Generating a story component	13
Figure 3. The standard GA crossover operators	17
Figure 4. Mutation that changes the encoding of a gene	17
Figure 5. A partially true and false situation in fuzzy logic	18
Figure 6. Fuzzy sets for laundry softness	20
Figure 7. Fuzzy sets for laundry quantity	20
Figure 8. Fuzzy sets for washing cycle	21
Figure 9. The inference and defuzzification processes of the fuzzy controller ..	22
Figure 10. Symmetric fuzzy sets in HEFTI's FDBS	23
Figure 11. Regions in the domain of an input variable	24
Figure 12. Adjusted fitness function with $R = 0.1$	26
Figure 13. Adjusted fitness function with $R = 0.5$	26
Figure 14. Adjusted fitness function with $R = 0.75$	27
Figure 15. Encoding story elements into a chromosome	32
Figure 16. Gene representation of story elements	33
Figure 17. Decoding a chromosome into story scripts	34
Figure 18. Hierarchical combination of story elements at various levels with a story template to form a story component	35
Figure 19. Microsoft agent character editor's properties tab page	40
Figure 20. The word balloon tab page	41

	Page
Figure 21. The voice tab page	41
Figure 22. Assigning image frames to an animation	42
Figure 23. Building a character agent	42
Figure 24. The speech properties dialog box	44
Figure 25. The authoring interface	45
Figure 26. The reading interface	46
Figure 27. A high level story flow graph for the three little pigs' story	59
Figure 28. Story's introduction	70
Figure 29. Wolf terror	70
Figure 30. Wolf attack	71
Figure 31. Wolf's plan	71
Figure 32. Pigsconclusion	71
Figure 33. Wolfsconclusion	72
Figure 34. A collapsible sub-tree embedded within a template numbered 1	72
Figure 35. Tree view of the action editor	73
Figure 36. Tree view of the agent editor	73
Figure 37. Manipulating a top level tree node	74
Figure 38. Manipulating a tree leaf	74
Figure 39. Attribute editor and the list of story elements	75
Figure 40. Visualizing a story component as a collection of story elements	76
Figure 41. Placing a story element at an indicated location	76

	Page
Figure 42. Deleting and rearranging a story element in the drag and drop interface	77
Figure 43. Attribute box that allows authors to edit a story component attribute	78
Figure 44. Several variants of the three little pigs' story	80
Figure 45. The pigs are having fun together with the mother pig	81
Figure 46. The wolf agent shows up in front of the pigs and the pigs scramble to safety	81
Figure 47. The straw house of the pink pig collapses!	82
Figure 48. The pig failed to escape from the wolf and becomes wolf's lunch	83
Figure 49. The gray pig cooks the wolf after he falls down the chimney of the wooden house	84
Figure 50. The pigs celebrate as the wolf rises to the heavens	84
Figure 51. Authoring interface of the text-based IAE	87
Figure 52. Timestep menu	88
Figure 53. Template menu	88
Figure 54. Element menu	88
Figure 55. Editing a story element	89
Figure 56. The statistical tool that complements the functionalities of the authoring interface	90
Figure 57. Advanced parameters for the generate statistics tool	90
Figure 58. Story distribution form with its ordered list view	92
Figure 59. The story distribution form's affinity list visualization	93

	Page
Figure 60. Story elements distribution bar graph	94
Figure 61. Story elements distribution bar graph for all of the story threads	95
Figure 62. The story thread differences dialog box	96
Figure 63. Process of computing the affinity measure for subsequence of length 1 and 2 between two story threads	99
Figure 64. Process of computing the multiple contiguous bit affinity measure for story threads A and B	101
Figure 65. Sequences of story elements grouped by story timesteps	114
Figure 66. Affinity for story elements considering different alignments	116
Figure 67. A non-linear story with multiple branching points	118
Figure 68. Graphing the series in Table E.1	121
Figure 69. Graphing the series in Table E.2	121
Figure 70. Graphing the series in Table E.3	122
Figure 71. Graphing the longest contiguous element measure in Table E.4	123
Figure 72. Graphing the longest contiguous element measure in Table E.5	123
Figure 73. Graphing the longest contiguous element measure in Table E.6	124
Figure 74. Graphing the average contiguous measure in Table E.7	125
Figure 75. Graphing the average contiguous measure in Table E.8	126
Figure 76. Graphing the average contiguous measure in Table E.9	126
Figure 77. Number of forward branches for different story stages.	127
Figure 78. Graphing the series in Table E.13	129
Figure 79. Graphing the series in Table E.14	130

	Page
Figure 80. Graphing the series in Table E.15	130
Figure 81. Graphing the series in Table E.16	131
Figure 82. Graphing the series in Table E.17	131
Figure 83. Graphing the series in Table E.18	132
Figure 84. Graphing the series in Table E.19	132
Figure 85. Graphing the series in Table E.20	133
Figure 86. Graphing the series in Table E.21	133

LIST OF TABLES

	Page
Table 1. Story elements referenced in template one	62
Table 2. New story elements referenced in tmplt_2.....	65
Table 3. Story elements referenced in tmplt_3	66
Table 4. Story elements referenced in tmplt_4	67
Table 5. Story elements referenced in the conclusion templates	69
Table D.1 Minutes it took the test subjects to complete the assigned tasks	163
Table E.1 Matching elements measure for 1,000 story elements set	164
Table E.2 Matching elements measure for 2,500 story elements set	164
Table E.3 Matching elements measure for 5,000 story elements set	164
Table E.4 Longest contiguous element measure for 1,000 story elements set.....	165
Table E.5 Longest contiguous element measure for 2,500 story elements set.....	165
Table E.6 Longest contiguous element measure for 5,000 story elements set.....	165
Table E.7 Average contiguous measure for 1,000 story element set	166
Table E.8 Average contiguous measure for 2,500 story element set	166
Table E.9 Average contiguous measure for 5,000 story element set	166
Table E.10 Lower and upper bounds of the 1,000 story element set	167
Table E.11 Lower and upper bounds of the 2,500 story element set	167
Table E.12 Lower and upper bounds of the 5,000 story element set	168

	Page
Table E.13 Matching element measure for 1,000 story elements set (25% forward branching probability).....	168
Table E.14 Matching element measure for 1,000 story elements set (50% forward branching probability).....	169
Table E.15 Matching element measure for 1,000 story elements set (75% forward branching probability).....	169
Table E.16 Matching element measure for 2,500 story elements set (25% forward branching probability).....	169
Table E.17 Matching element measure for 2,500 story elements set (50% forward branching probability).....	170
Table E.18 Matching element measure for 2,500 story elements set (75% forward branching probability).....	170
Table E.19 Matching element measure for 5,000 story elements set (25% forward branching probability).....	170
Table E.20 Matching element measure for 5,000 story elements set (50% forward branching probability).....	171
Table E.21 Matching element measure for 5,000 story elements set (75% forward branching probability).....	171
Table E.22 Longest contiguous element measure for 1,000 story elements set (25% forward branching).....	171
Table E.23 Longest contiguous element measure for 1,000 story elements set (50% forward branching).....	172
Table E.24 Longest contiguous element measure for 1,000 story elements set (75% forward branching).....	172
Table E.25 Longest contiguous element measure for 2,500 story elements set (25% forward branching).....	172
Table E.26 Longest contiguous element measure for 2,500 story elements set (50% forward branching).....	173

	Page
Table E.27 Longest contiguous element measure for 2,500 story elements set (75% forward branching)	173
Table E.28 Longest contiguous element measure for 5,000 story elements set (25% forward branching)	173
Table E.29 Longest contiguous element measure for 5,000 story elements set (50% forward branching)	174
Table E.30 Longest contiguous element measure for 5,000 story elements set (75% forward branching)	174
Table E.31 Average contiguous measure for 1,000 story element set (25% forward branching)	174
Table E.32 Average contiguous measure for 1,000 story element set (50% forward branching)	175
Table E.33 Average contiguous measure for 1,000 story element set (75% forward branching)	175
Table E.34 Average contiguous measure for 2,500 story element set (25% forward branching)	175
Table E.35 Average contiguous measure for 2,500 story element set (50% forward branching)	176
Table E.36 Average contiguous measure for 2,500 story element set (75% forward branching)	176
Table E.37 Average contiguous measure for 5,000 story element set (25% forward branching)	176
Table E.38 Average contiguous measure for 5,000 story element set (50% forward branching)	177
Table E.39 Average contiguous measure for 5,000 story element set (75% forward branching)	177
Table E.40 Lower and upper bounds of the 1,000 story element set (25% forward branching)	178

Table E.41	Lower and upper bounds of the 1,000 story element set (50% forward branching)	179
Table E.42	Lower and upper bounds of the 1,000 story element set (75% forward branching)	180
Table E.43	Lower and upper bounds of the 2,500 story element set (25% forward branching)	181
Table E.44	Lower and upper bounds of the 2,500 story element set (50% forward branching)	182
Table E.45	Lower and upper bounds of the 2,500 story element set (75% forward branching)	183
Table E.46	Lower and upper bounds of the 5,000 story element set (25% forward branching)	184
Table E.47	Lower and upper bounds of the 5,000 story element set (50% forward branching)	185
Table E.48	Lower and upper bounds of the 5,000 story element set (75% forward branching)	186

1. INTRODUCTION

Storytelling in the form of narrative writing is a rigorous practice that requires discipline, inspiration, creativity, and hard work. Histories, philosophies, concepts and ideas can all be treated as a form of narration or expression, and only through a careful choice of words and sentences (signs and structures) can the author appropriately convey intended information to the reader. Before the advent of information technology, the main medium used for this type of expression was paper in all of its various forms. Even after the invention of computing environments, writing spaces still mimic these age old, two dimensional (2D) static media. For example, popular word-processing suites such as Microsoft's Office and Lotus Smart-suite are modeled on the writing spaces spanned by paper or similar media that have been in use since the dawn of human civilization. Several researchers have opened new horizons to this art form by experimenting with different ways of telling a story. Well-known examples include non-linear storytelling [14] and interactive storytelling using artificial intelligence (AI) techniques [54, 64] in which the readers play some role in unfolding the story. These techniques are particularly suited for digital media that are no longer confined to 2D writing spaces. Digital writing spaces offer more alternatives to express (abstract) ideas and concepts through visualization, simulation and animation techniques [13]. This new generation of writing spaces mark the beginning of a new era in information conveyance and storytelling.

In the digital age, authors can use computer graphics to stage a play in a virtual world. Authors are now freer to narrate their stories in ways they deem preferable or explore alternatives that can deviate from traditional representations of a story. Authors who enjoy writing fantasy role-playing novels might start by writing electronic book versions of the genre, whereas those who are more adventurous might create executable scripts, interactive story components or indeterminate events that jointly form constructs of the narration depending on the user's responses [1]. However, in either case, writing a good story still requires immense patience, creativity and work from the author, and the practice of writing a story still requires a good grasp of the readers' psychology in order to create suspense and thrills and to merge the readers' world with that of the story.

This thesis follows the style of Artificial Intelligence.

Although there are no universal rules on how to write a good story for all genres that appeal to all readers, influential work of writers such as Edward Morgan Forster [38] and David Lodge [50] outline the foundational elements of narrative texts derived from the structure used by most writers. From reminiscing to essay writing, a story has a topic that informs the reader of what the story is about, an introduction that provides background and character information for the story, a series of events that ultimately leads to development of the characters and interaction of these characters and, finally, a closing that ends the story or presages to a continuation of the story some time in the future. The above description provides a high level view of the most common story components used in the traditional writing space. In the digital writing space, authors can still adhere to these rules of thumb while being aware of the disappearance of certain constraints due to the added possibility of narrating in a nonlinear fashion and the increase in dimensionality of the writing space.

1.1 Background and Literature Review

Interactive storytelling is a major endeavor, with a long history of previous research [21, 77, 78]. The area of synthetic actors is especially important because they play a critical role in the implementation of future interactive storytelling systems [55]. Many overlapping approaches are currently being used to explore interactive storytelling, such as: immersive storytelling [58, 71], emergent storytelling [4, 29], plot-based systems [41, 51, 68], interactive authoring of stories [11, 27, 51] and character-based systems [12, 42, 71]. These approaches present different design paradigms, aiming not only at the user experience but also different technical solutions to the problem of generating interactive narrative itself and user intervention on the unfolding story.

Most approaches used in interactive storytelling fall into the field of narrative intelligence. This term is commonly used to refer to research on human narratives and storytelling, as well as the development of software or robotic agents that either support human storytelling or are themselves storytellers and/or story-listeners. Mateas and Sengers [54, 64] provide an introduction to narrative intelligence and its study in artificial intelligence.

Applications of narrative intelligence include autobiographic agents that try to understand the ‘world’ by constructing autobiographies, a term co-opted by Mateas and Sengers to refer to histories and experiences collected by the agents while exploring the world [30, 67]. Based on this foundation, researchers at the University of Reading, UK experimented with applying autobiographical memory [24] to virtual environments in the hope of creating a more interesting virtual environment that met the cognitive needs of its users (agents or avatars) in representing and encoding memories in terms of stories [19].

Other projects focus on creating authoring tools for writers to use virtual reality technologies in interactive storytelling. Glassner [39, 40] discusses relevant issues for designing interactive fiction, the need for a story structure, and an idea called the story contract that describes some important traits of successful fictive experiences. Related work done by Sgouros [66, 67] addresses the lack of appropriate direction and execution environments for interactive media and proposes a solution based on CHOROS, a Java-based environment for authoring, direction and control of narrative performances. In particular, CHOROS allows the story author to annotate the performance script with stage directions while offering an augmented reality interface for planning the behavior of the actors. The system uses vision-based tracking methods and behavior-based control for adjusting the behavior of the robotic actors according to the director’s instructions during the performance [65].

Newer techniques, such as using fuzzy cognitive maps [52] to model and implement believable agents’ behaviors by virtue of their self-perception in the storytelling context, serve as one of the keys for the autonomy of virtual entities’ decision making. Researchers at the University of Teesside, suggest that making plan-based representations and flexible character-based systems that rely on narrative formalisms and representations can satisfy the real-time requirements of interactive storytelling, while still being compatible with the narrative formalization they are pursuing [20, 22, 23]. One of their papers presents a short episode generated by the system, which illustrates both high-level results and technical aspects, such as re-planning and user intervention [23]. However, much work remains to be done on developing more complex narrative representations and

investigating the relations between natural language semantics and narrative structures in the context of interactive storytelling [22].

A number of research projects at the MIT Media Lab have focused on creating electronic media that exhibit features and characteristics which cannot be offered by traditional media. The Electronic Publishing project addresses the issues of professional production and distribution of news and providing the news consumer with tools that facilitate gathering, access, and use of news in both individual and communal contexts, while facilitating two-way communication between writers and readers [7]. Thus, contrary to the traditional news publishing process, these tools allow the creation of an environment in which the news-as-a-service involves the consumer of the news being an active, engaged participant. Similarly, using ISIS, a programming language for responsive media, research projects directed by Dakss and Bove [26] demonstrate the capability to deliver interactive media contents to a wide variety of platforms, from high-powered workstations and servers to set-top boxes and Personal Digital Assistants (PDAs). For instance, “An Interactive Dinner At Julia's” focuses on delivering interactive contents of television programs to produce “hyperlinked video” programs that enable viewers to navigate a “web” of video clips featuring Julia Child. Using the remote control, viewers can “click” on entrees and decorative items at the dinner table and be shown video clips in which Julia creates these items. Commercial applications using the ISIS programming language include the “HyperSoap” [25], a soap opera program produced in conjunction with JCPenney in which a viewer can select clothing, props and scenery and see purchasing information, such as the item's brand and price. The authoring software creates statistical models of objects' color, texture, motion and spatial position and uses these models to both track objects automatically and perform searches among identified objects. This technology is now being licensed to a Boston-based company named WatchPoint Media [75] which provides hyperlinked video as a service for interactive entertainment and a portal to electronic commerce.

The Gesture and Narrative Language Group at MIT's Media Lab design technology to enable and enhance natural forms of communication and linguistic expression, focusing

on face-to-face conversation and storytelling for adults and children [15]. Tools created by this group that are related to this research are BEAT, the Behavior Expression Animation Toolkit [18], Animal Blocks [2], Avatars [17] and Sam[62]. The BEAT toolkit automatically inserts gesture and facial expression into character dialogue or monologue scripts for animators. Animators input typed text to be spoken by an animated human figure and obtain output of synchronized nonverbal behaviors and synthesized speech in a form that can be sent to a number of different animation systems. The toolkit uses nonverbal behaviors that are assigned on the basis of actual linguistic and contextual analysis of the typed text, relying on rules derived from extensive research into human conversational behavior. A related project is the Animal Blocks collaborative storytelling environment for children. Animal Blocks can retell stories told by children who have played with them previously. With the toolkit, the child can write her own stories in the book by using the physical animal blocks in addition to the keyboard. Complementing the functionalities of the previous toolkits, the Avatars toolkit provides facilities to represent the user's communicative intentions non-verbally in embodied, animated avatars for graphical online societies [16]. Bringing together research results from the aforementioned projects, the SAM agent is an embodied conversational storyteller for children. In SAM, the character and the child share a castle play space and a set of story-evoking toys that can magically exist in both participants' worlds [62].

In addition to exploring the possibilities of paper-based and electronic media, researchers of the Interactive Cinema Group at MIT's Media Lab seek to increase immersion of audiences in theatrical settings using electronic sensing devices, computer animations, electronic collage, computational story engines, and communication networks [31]. The Metalinear Cinematic Narrative project [14] proposed a new narrative form that is a collection of small related story pieces designed to be arranged in many different ways, to tell many different linear stories from different points of view, using the aid of a storytelling engine [32]. Similar in several aspects to my approach discussed in later sections, Metalinear Cinematic Narrative has: a structural environment that describes the narrative structure using narrative primitives, a representation environment that captures knowl-

edge of the various story components in the form of relationships between clips and story granules, a presentational environment in which story agents narrate the story in their own individual stylistics, and an agent scripting environment that allows the author to create new agents, thus altering the narrative construction process. The resultant software is a writing tool which offers the author knowledgeable feedback about narrative construction and context during the creative process and is essential to the task of creating metalinear narratives of significant dimension. Furthermore, work done by Barbara Barry of the Interactive Cinema Group defines a specific storytelling process known as "Transactional Storytelling" which is the construction of story through trade and re-purposing of images and image sequences [6]. Barry's "StoryBeads" are wearable computers designed as a tool for constructing image-based stories by allowing users to sequence and trade story pieces of image and text. StoryBeads are modular, wearable computer necklaces made of tiny computer "beads" capable of storing or displaying images. Beads communicate by infrared light, allowing the trade of digital images by beaming from bead to bead or by trade of a physical bead containing images.

1.2 Combining Evolutionary Algorithms and Fuzzy Logic for Dynamic Story Generation

Synergy between evolutionary algorithms and fuzzy logic can occur in three complementary forms [72]:

1. Exploit the optimum search ability of evolutionary algorithms to synthesize and optimize fuzzy systems [73];
2. Use a fuzzy knowledge base [3] to detect the emergence of a solution and dynamically tune the parameters of the evolutionary algorithms [8, 48];
3. Embed fuzziness into the evolutionary algorithm itself, such as sacrificing precision in the calculation of fitness to save computational resources by defining a fuzzy fitness criteria or fuzzifying the genetic operators [63].

Generation of dynamic stories with emergence properties cannot be easily accomplished with the first approach because the system has to rely on evolutionary algorithms to optimize parameters or rules in a fuzzy system that has only a finite number of rules and fuzzy sets. Stories generated from such a rule-based system are always bounded by the number and range of the parameters in the fuzzy rules since the story generation process is driven by a fuzzy system.

The second approach utilizes the power of fuzzy logic to approximate a solution to tune the parameters of the evolutionary algorithms in order to achieve faster convergence, better population diversity or better solutions. Hybrid systems that use this approach rely heavily on evolutionary algorithms to search the problem space, while the fuzzy knowledge base fine-tunes their search capabilities at regular intervals or under certain conditions. Such systems maintain the desired characteristic that the evolutionary algorithms can “discover” potentially promising solutions from various regions in the problem space, thus exhibiting emergence characteristics required in the generation of dynamic stories.

Lastly, the third approach “fuzzifies” the genetic operators or encodings of genes in evolutionary algorithms. This approach is mainly used in situations when it is hard to assess the quality of candidate solutions with precision [73] or the fitness value computed is inherently noisy. This could incur unnecessary runtime complexity since authors have already provided a set of evaluation schema that guide the evolutionary algorithms in assigning fitness or manipulating the gene to achieve better fitness. Fuzzifying these inputs will not help in the evolutionary algorithm search process.

From the above discussion, the second approach matches most of the features required in dynamic story generation without incurring additional costs in “fuzzifying” inputs or being constrained by a limited set of fuzzy rules. Therefore, the design of our storytelling engine adheres closely to the second approach.

1.3 Motivation and Scope

Each of the approaches to interactive storytelling discussed above has its own strengths and weaknesses. The motivation for this research arises from the perceived need

for a new hybrid approach that coalesces and extends existing approaches. The result forces a new research direction which eliminates certain weaknesses exhibited by a single approach, due to the synergistic nature of the various approaches. Each approach empowers certain aspects of the storytelling and narration process and it is important to inherit these many strengths of the traditional approaches. But graceful juxtaposition of these approaches is not possible without modification to their constructs, and characteristics of these approaches will have to be discarded in pursuit of simplicity and understandability. This new approach also introduces the use of a global search technique from artificial intelligence that samples the problem space by using story construct templates created by human authors that serve implicitly as rules for a Genetic Algorithm (GA) [56]. Such template-based search methods offer greater flexibility and robustness than other approaches, because the narrative rules can be controlled and customized by authors to fit into the story context. Furthermore, the templates can be used to create rules that are conceptually similar to logic-based reasoning and rules used in plot-based storytelling [41, 51, 68], character-based storytelling [12, 42, 71] and interactive authoring of stories [11, 51] by using the power of Fuzzy Decision-Based Systems (FDBS). Instead of modeling agent perceptions directly with fuzzy logic as in the case of Maffre, et al. [52], fuzzy logic is used to monitor and adjust system parameters to alter the GA's sampling process of the problem space and to monitor the performance of the other subsystems.

While using a stage play in a virtual environment to tell a story is not a new idea [46], current research approaches [19, 20, 52, 67] led to the inspiration to create embedded scripts constructed from "atomic actions" that will generate complex agent behaviors in the virtual narration environment. Governed by the FDBS, synchronization of these scripts during a storytelling session and activation and modification of the scripts at run time by the author or storytelling engine are made to conform to the rules created specifically for the story context. Although this architecture has the potential shortcoming of being bounded by the story contexts written by the human author and tacitly implied in the story templates, less time and fewer resources are committed in the attempt to generate rules or templates that can be used in many story contexts.

Traditional approaches that are not being emulated by the template approach proposed here are immersive storytelling and emergent storytelling. Due to their heavy reliance on 3D graphics and rendering techniques, virtual reality modeling, and large demands on processing power, I have decided not to incorporate them into the architecture to ensure the complexity of the resultant system is within acceptable limits. However, these approaches can be easily incorporated into the overall schema through slight modifications in the story component constructs. In this scenario, the learned interactions of users and agents can be reflected in the knowledge base used by the FDBS for autobiography reconstruction. Immersive storytelling is also discarded due to the scope of this research and the various agent controls, synchronization of user actions and path finding issues exhibited by this approach.

1.4 Scenarios of Use

During the process of creating a story, the human author engages in various activities that collectively form a narration. New media for the expression of ideas and concepts no longer constrain the user since the new generation of authoring tools proposed here provides authors with the capability to orchestrate various story events, characters and rules with immense flexibility [36]. A few modifications of the engine's rule sets and parameters can create completely different plots, story lines and relationships of story characters, thus reducing the need to reorient one's mindset in narrative writing.

Writers using the new generation of storytelling engines cannot be considered solely as the story writers since they are not the sole creator of their stories, but instead play the dual roles of writer and conductor in the narration symphony *performed* by the agent characters through the storytelling engine with interaction from the reader. In such an environment, the term cyborg author [1, 43] is a more accurate term that describes the role of the human because he/she is not only relying upon the writing space for creative activities, but also combining multimedia and graphical materials with simulations and visualizations produced by the digital writing environment in order to author the composition. Unlike traditional story writing, authors using storytelling engines are augmented

by computers that can generate works that are less biased by authors' opinions and points of view, allowing reader involvement at an unprecedented level in unfolding the story through mutual exploration of alternatives in story development between reader and the storytelling engine in a pre-authored story environment.

Traditional stories and novels do not always have re-readability because the story elements are familiar to the reader. However, the cyborg author can improve on this aspect because the storytelling engine allows dynamic stories to be generated from the current story context. For example, an intended murder mystery that focuses on how a detective (role-played by the reader) goes about finding the murderer of a victim can yield different conclusions depending on the reader's interaction. If the "player" consistently engages in activities that increase the level of mistrust among agent characters, the rules created by the author can make the agent characters unwilling to cooperate with the detective. Without the information provided by eyewitnesses, the story can end with the conclusion that the detective was unable to solve the case and what exactly happened to the victim still remains a mystery. On the other hand, if the "player" attempts to solve the mystery by commingling with the agent characters and acting virtuously, the agent characters will start to trust the "player" and cooperate with the player. With so many variables embedded in the story, it is hard for the "player" to predict the composition of the next story component. Consequently, the storytelling engine allows the murder mystery story to be retold (reread) over and over again without exhausting the reader's patience [47].

Applications of the storytelling engine are not restricted to the entertainment world. A teacher could use the storytelling engine to teach children math. With the students assuming the role of a tomb raider, various agent characters in the disguise of monsters will spring out of their hiding places and challenge the raider with various mathematical puzzles [60]. These puzzles can be retrieved from a knowledge base or dynamically generated from the templates created by the teacher with the appropriate rules. As the student becomes accustomed to these puzzles, the teacher can alter the difficulty of the puzzles by simply modifying the knowledge base or templates. The integration of a storytelling

engine with educational materials encourages children to learn as they are playing a fully-interactive game created by the teacher.

1.5 Objectives of Research

These observations, in part, lead to the objectives of this research: 1) create a Hybrid Evolutionary-Fuzzy Time-based Interactive (HEFTI) storytelling engine that generates dynamic stories from a set of authored story constructs given by human authors; 2) provide various authoring interfaces for authors to generate the needed story constructs for dynamic stories; 3) create a storytelling environment that enables authors to narrate the story and orchestrate a digital stage play, using computer agents and scripts to demonstrate a certain level of creativity to aid authors in the authoring process [35]; 4) evaluate the performance of the HEFTI storytelling engine in the context of dynamic story generation.

The benefits of such a system are many: 1) it allows authors the freedom to create their own stories while providing a cyborg authoring environment that augments the author's natural storytelling capabilities by adding facilities for executable scripts, interactive story components, interactive events, and new media; 2) it encourages readers' involvement in the story through textual and graphical representations in an interactive environment, thus offering a greater level of immersion [43] than traditional paper-based media; and, 3) it uses a new, hybrid evolutionary-fuzzy technique for story development with the potential to offer endless possible combinations in length, depth of story, and character development.

2. ARCHITECTURE OF THE INTERACTIVE STORYTELLING ENGINE

A layered and modularized architecture is proposed for HEFTI that isolates the authors from the underlying mechanisms that run the storytelling engine, thus freeing authors from the need to learn the convoluted syntax of programming languages in order to begin authoring a story that can be played in the Stage Play Subsystem (SPS) (Figure 1). Furthermore, various authoring tools (described in section 5.3) collectively form an Integrated Authoring Environment (IAE) that allows authors to monitor system status and modify system parameters. These tools are linked directly to the knowledge base shared by the Evolutionary-Fuzzy System (EFS), which consists of the Genetic Algorithms (GA) and Fuzzy Decision-Based Subsystem (FDBS) (Figure 1). The EFS is mainly responsible for creating content out of story templates, adjusting and monitoring system parameters, and generating action scripts for the character agents.

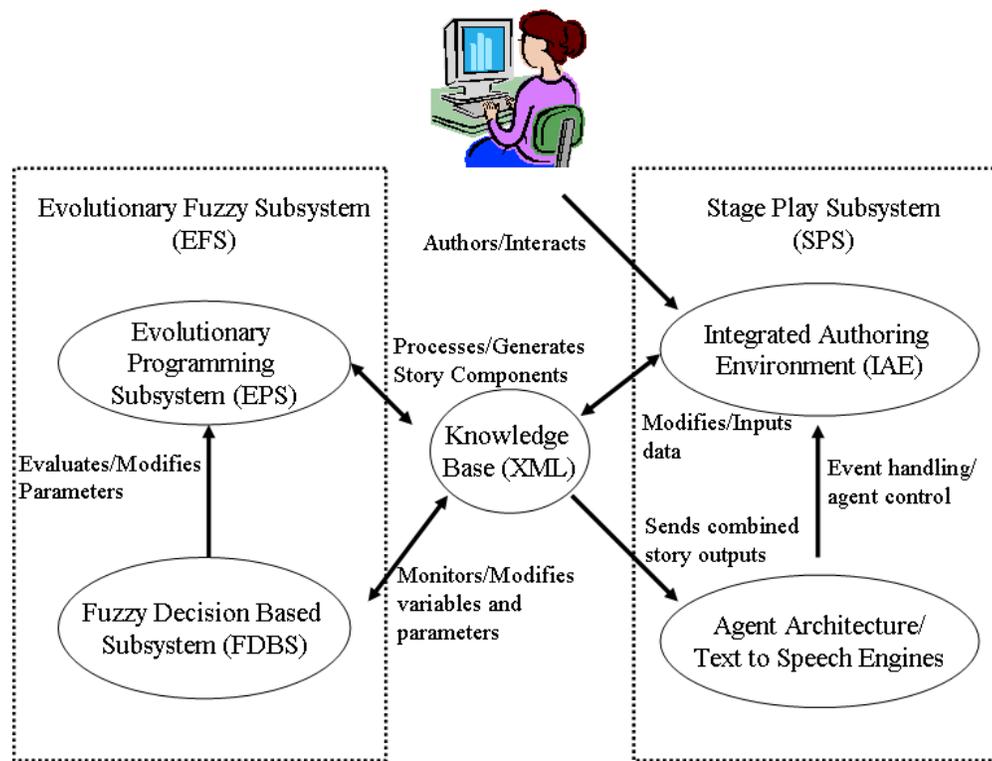


Figure 1. The overall system architecture and subsystems' relationships

To better facilitate information exchange between the storytelling engine and users of this form of storytelling, the IAE will support two different modes, namely the read-only and authoring modes. The read-only mode is a slender version of the environment without the authoring tools to prevent readers from modifying properly configured story content distributed by authors. The read-only mode will initialize the EFS using the authored environment, context and parameters distributed by authors and begin dynamically narrating a story according to the reader's interaction and system parameters. The authoring mode, on the other hand, gives users full control over all aspects of the storytelling system, allowing rules and parameters to be adjusted accordingly to suit the preferences of the author or reader.

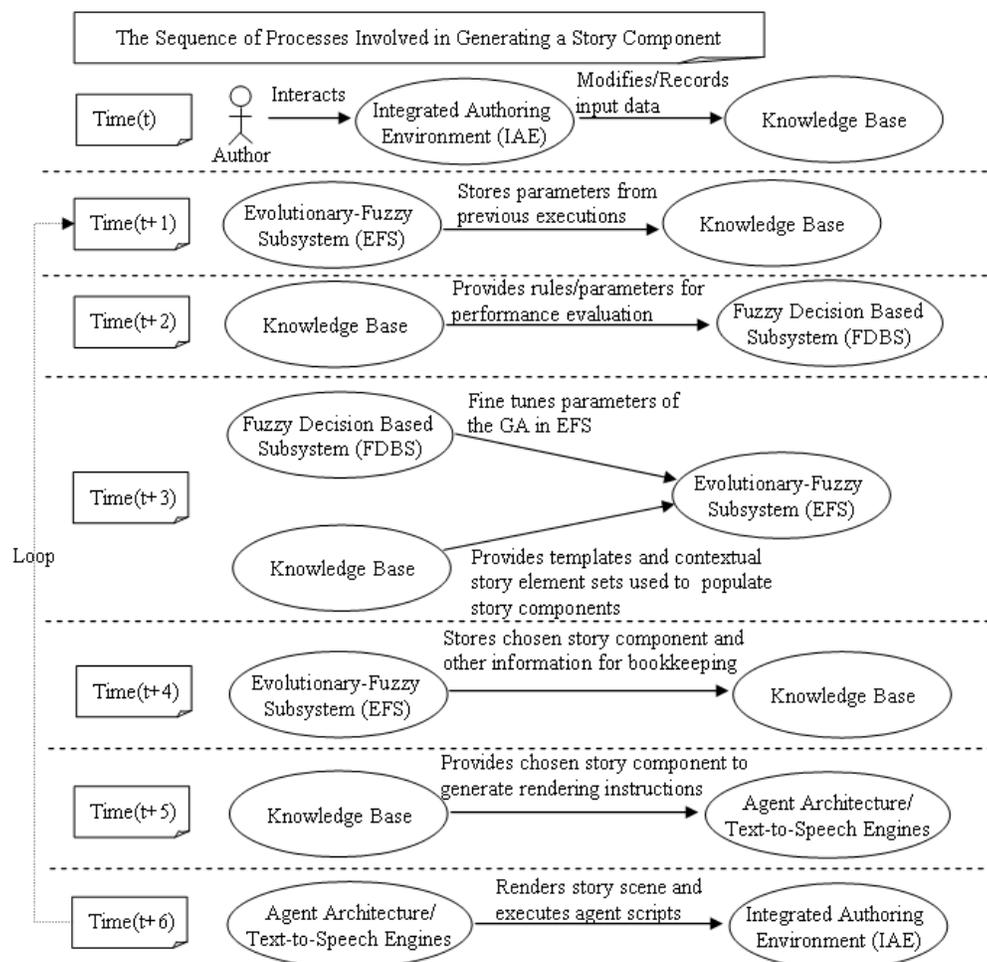


Figure 2. Generating a story component

The story component is the single most important entity representing story contents, parameters and action scripts of the agents in the Stage Play Subsystem. Individual story components are generated through the integration of rules, global events, and interactions of agent characters with the reader and story component templates (Figure 2).

Figure 2 provides an overview of the authoring process, interactions and relationships of the various submodules in the story generation process and the workings of the storytelling engine. The life cycle of a dynamic story begins with:

1. At $Time(t)$, The author manipulates/creates story building blocks that are stored in the knowledge base with the authoring tools in the Integrated Authoring Environment (IAE):
 - i. A main story thread that serves both as a guideline and evaluation schema for the storytelling engine (Section 4.4.7, Story.xml)
 - ii. Story templates that describe how story elements, such as agent actions (Section 4.4.2, Actions.xml) can be combined to form story elements (Section 4.4.8, Templates.xml)
 - iii. Rules (Section 4.4.5, Rules.xml) and story variables (Section 4.4.4, Objects.xml) that are used to generate emergent properties of a dynamic story
 - iv. Dialogues (Section 4.4.9, Textstrings.xml) to be spoken by the agent characters (Section 4.4.3, Agents.xml) and their corresponding behavior scripts (Section 4.4.6, Scripts.xml)
2. With all of the necessary story building blocks available from the knowledge base, the story generation process starts with initializing or storing story parameters (if available). This step, $Time(t+1)$, initializes various aspects of the storytelling engine, such as setting/resetting the parameters of the GA within the EFS system (Section 2.1, Genetic Algorithm and Fuzzy System) and storing the story state in the knowledge base.
3. At $Time(t+2)$, Fuzzy rules stored in the knowledge base are resolved by the FDBS based on performance of the GA in the current execution cycle (Section 2.1.2, Fuzzy Logic).

4. At $Time(t+3)$, state of the story is checked based on rules and story conditions provided from the knowledge base to determine execution order or termination of the story. At the same time, FDBS fine tunes/adjusts parameters of the GA based on the results in step 3 if necessary and new sets of story building blocks, such as story templates and story elements, are provided to the EPS to generate the next part of the story in the form of a story component (Section 3, CONSTRUCTING STORIES).
5. At the end of the generation process ($Time(t+4)$), state of the story is recorded in the knowledge base for bookkeeping purposes. The information is recorded to allow readers to “rewind” the story to any arbitrary point in time.
6. At $Time(t+5)$, results from the generation process are sent to the agent architecture and text-to-speech engines for processing.
7. Lastly, at $Time(t+6)$, the agent architecture and text-to-speech engines execute the appropriate agent scripts or rendition of the story scene in IAE.
8. Go back to step 2.

2.1 Genetic Algorithm and Fuzzy System

The genetic algorithm (GA) and fuzzy logic forms an integral part of EFS. GA belongs to the evolutionary algorithms category which imitates nature’s way of evolving complex biological systems and structures under the influence of environmental factors [56], whereas fuzzy logic mimics our brain’s ability to perform approximate reasoning in the face of incomplete information and uncertainty [76].

2.1.1 Genetic Algorithm

GA has found its use in many applications when other algorithms are too costly to implement. It does not always generate an optimal solution to a given problem but, aided by good evaluation functions, GA can discover satisfactory solutions or, sometimes, an optimal solution. In contrast to other heuristics-based algorithms and random search algorithms, the GA is simple to implement, requires less memory usage, and is more intuitive

because it employs simple data structures (such as a collection of linked lists to represent individual chromosomes in a population and arrays to store cumulative frequency values during the selection process and fitness of chromosomes).

AI researchers derived the GA from observations made by biologists concerning how stronger species in a population tend to outperform others in the long run due to the natural selection process [28, 79]. From their work, we know that the process of evolution is a slow process since its effects can be observed only after a long period of time. However, we do not have that much time to wait for the effects of the evolutionary process to manifest itself naturally. Instead computer algorithms that mimic the process of evolution are being used to allow programs or solutions to evolve by themselves, albeit in a much simpler manner than what is being done by nature [56]. GA is one of the evolutionary algorithms that try to emulate the natural selection process through the use of various genetic operators and selection processes in the evolutionary process. The genetic operators try to imitate the processes used by nature to generate the myriad species of flora and fauna found on this planet with various adaptations to the environment. GA's selection, crossover and mutation operators are the computer counterparts of the selection and breeding process of organisms in the real world to pass on their traits to the next generation. Therefore, when a small number of strong individuals dominate an entire population after a certain number of evolution cycles, the evolutionary algorithm is said to have reached the convergence stage where new variability is less likely to develop.

In GA terminology, genes in a chromosome represent part of the solution instance, and a chromosome is a collection of genes which represents a solution instance in the problem space. A collection of several chromosomes form a population that represents the solution instances currently discovered by GA. There are many ways that one can encode a solution instance in the genes, however, the most popular encoding methods are binary, floating point and string representations. Most encoding methods use a random initial population at the start of the algorithm and, depending on the chosen encoding method, a correction algorithm might be used to correct any mistakes in the generation or

selection routines so that the encoding for a chromosome represents a valid solution instance in the problem space.

The standard crossover operator facilitates exchange of genetic materials between two chromosomes. It specifies a point of exchange between two parents and genetic materials, after the specified offset, are swapped between the parents. The multipoint crossover operator allows multiple points of exchange between two parents (Figure 3). Many versions of crossover operators are mentioned in the literature [5, 56].

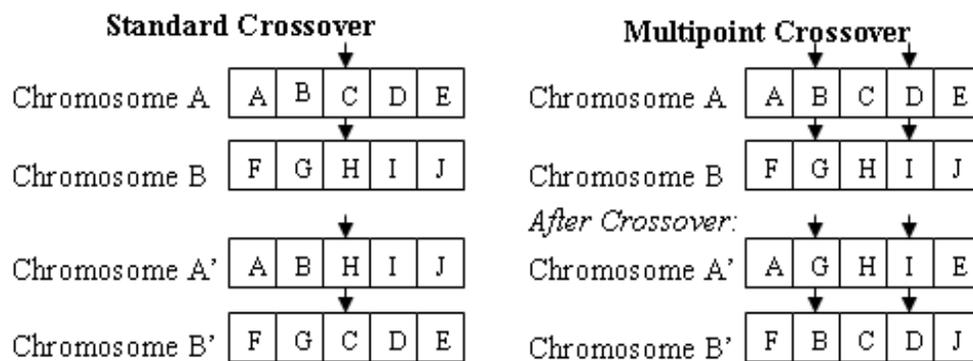


Figure 3. The standard GA crossover operators

The traditional mutation operator in GA introduces variations into a population by overwriting a particular gene's encoding with a random number. This is conceptually similar to the AI search technique called simulated annealing that resets the search process to a different area in the problem space randomly to reduce the probability of tethering the search process around a local maxima peak [70] (Figure 4)

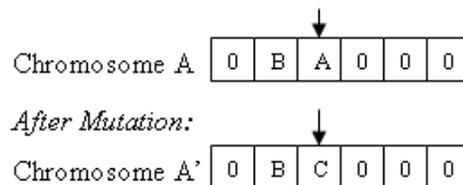


Figure 4. Mutation that changes the encoding of a gene

2.1.2 Fuzzy Logic

Fuzzy logic has been used in a number of areas where traditional modeling techniques are too expensive to implement or formulate. Fuzzy logic mimics the approximate reasoning capability of the human brain to evaluate rule conditions and consequences. In contrast to traditional Boolean logic that operates on true and false values, fuzzy logic evaluates rules in a continuous membership domain between 0 and 1.

By representing the correctness/membership of conditions in a continuous domain, a fuzzy rule can be both partially true and false if its membership values in these domains are greater than 0 (Figure 5). Such uncertainty is later resolved by the defuzzification process (described later) to derive fuzzy rule consequence(s).

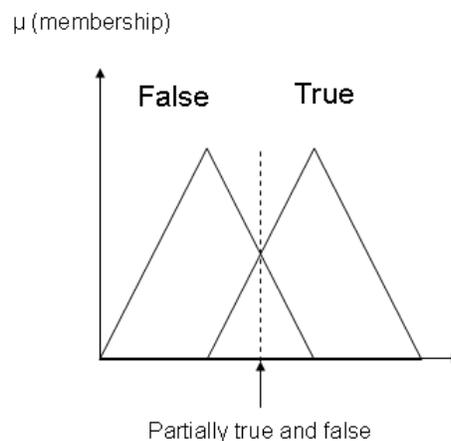


Figure 5. A partially true and false situation in fuzzy logic

Greater convenience is achieved when certain ranges of the domain are associated with linguistic labels, such as “accurate”, “a bit accurate”, “minimally accurate,” so that a fuzzy rule can be written in an easily readable form such as:

If condition A is accurate then perform action C

Therefore, analogous to most Boolean rule systems, multiple fuzzy rules can be used in conjunction to describe certain phenomena allowing the controller to exert appropriate behaviors based on the observed input values.

Assuming that *conditionA* is evaluated to be somewhat inaccurate, the firing strength of this rule will be adjusted accordingly so that *actionC* can be invoked accordingly. In contrast to a Boolean rule where everything is discrete in nature, the control action *actionC* is either invoked or not invoked at its full strength. The power of fuzzy rules is more apparent when we attempt to represent the range of firing strengths of the above fuzzy rule into a set of Boolean rules. To accomplish such a task, we will have to take into account the range of possible input values of the single fuzzy rule and formulate a set of Boolean rules to handle them individually:

If *conditionA* is accurate then perform *actionC* at firing strength

If *conditionA* is a bit accurate then perform *actionC* at 50% firing strength

If *conditionA* is minimally accurate then perform *actionC* at 25% firing strength

One can quickly observe how tedious writing the Boolean rules can be in this situation because the writer not only has to formulate Boolean rules to handle every possible set of input values, but the rule set above is still inadequate to handle situations such as “If *conditionA* is somewhere between accurate and minimally accurate,” which can easily be evaluated by fuzzy logic to be a membership value that falls somewhere in the range of the fuzzy set *Accurate*.

Operators used in the inference process and defuzzification method used by the fuzzy system can be based on several fuzzy rule-based models that are frequently used in practice, such as the Mamdani model or the Takagi-Sugeno-Kang (TSK) model [76]. A washing machine example based on the Mamdani model is provided below to give the reader an overview of the fuzzy logic reasoning process:

Using a fuzzy washing machine with two fuzzy rules as an example, we want the controller to monitor laundry quantity and softness to adjust the washing cycle accordingly. The fuzzy rule set can consist of the following rules:

1. If Laundry Quantity is Large AND Laundry Softness is Hard then Wash Cycle is Strong
2. If Laundry Quantity is Medium AND Laundry Softness is Soft then Wash Cycle is Normal

We can setup the fuzzy sets for the input (Figure 6, Figure 7) and output (Figure 8) variables as follow:

μ (membership)

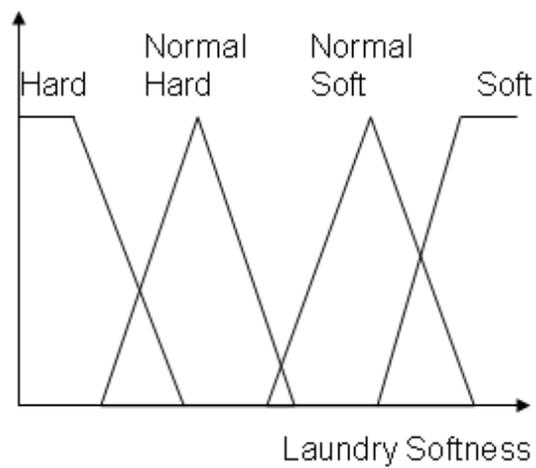


Figure 6. Fuzzy sets for laundry softness

μ (membership)

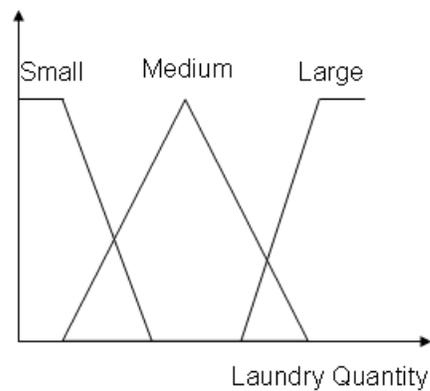


Figure 7. Fuzzy sets for laundry quantity

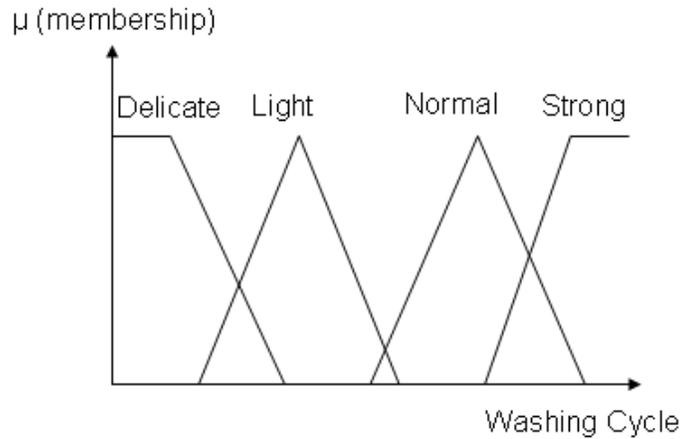


Figure 8. Fuzzy sets for washing cycle

As seen in Figure 6 and Figure 7, the input domain of the two variables, laundry softness and laundry quantity, are separated into four and three fuzzy sets respectively, whereas the output domain consists of four fuzzy sets. Mapping of the input variables to the appropriate fuzzy sets can be based on the weight of the laundry (laundry quantity) and what was chosen by the user (laundry softness). After the mapping phase, the controller is going to perform fuzzy inference based on the Mamdani model and combine the output of the fuzzy rules using the minimum operator that extracts the lowest value from a set of values (Figure 8). The lowest values from multiple rules are then used to generate fuzzy subsets from the output variable whose centroid is then computed using the formula:

$$(\sum W_i \cdot D_i) / (\sum W_i)$$

where W_i stands for the weight of D_i from 0 to 1 (y-axis) in a fuzzy set, and D_i stands for the value of the input variable (x-axis). In essence, the centroid formula computes the weighted average of the fuzzy subsets to produce an appropriate output value (Figure 9).

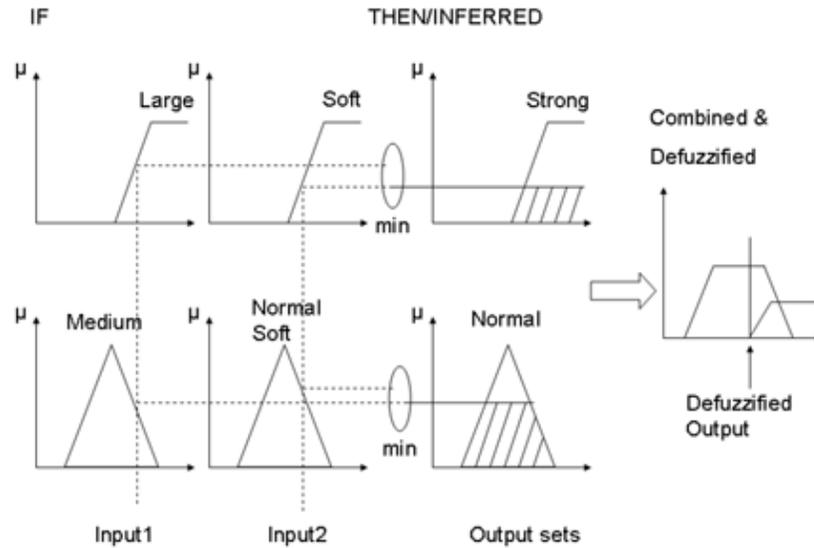


Figure 9. The inference and defuzzification processes of the fuzzy controller

2.2 Design of the Evolutionary Fuzzy System

The robustness of fuzzy rules makes them ideal for control related applications. The FDBS of the storytelling engine uses a set of fuzzy rules to monitor various aspects of the EPS to accomplish the following objectives: 1) fast convergence rate of the GA; 2) good variability in story components; and, 3) good chromosome selection criteria. The application environment makes achieving the first goal important because the IAE is a real-time application that has to generate many story components for the reader. The difference in speed might not be immediately apparent for a small story with few story elements, but larger stories with many story elements can make the GA converge rather slowly due to the search space complexity spanned by these story elements. The second objective is related to the first objective because they are influenced by similar factors, namely, population number, mutation rate and crossover rate. Lastly, the chromosome selection criteria of the GA is also monitored by the FDBS because chromosomes selected from a previous population play an important role in determining subsequent searches of the GA. In other words, if the current evolution cycle of the GA is converging slowly, selection criteria of the GA can be altered to be more competitive. This will increase the odds of selecting

stronger individuals from the population, thus shortening the convergence time of GA because variability in the population is reduced and less of the problem space is explored.

The problem of sub-optimality does not have much influence in HEFTI because the constraints setup by the story rules (please refer to the “Stories Controlled by Rules” in section 3.3 for details) generate story components that are valid solution instances, i.e. logically correct and coherent partitions of the complete story, therefore, a suboptimal solution in this context would be a solution instance which is highly similar to a previous run of the same story. Since HEFTI is driven by a random number generator and readers can provide a random seed during story initialization, this problem can be alleviated by increasing the number of story templates and elements in any particular set to significantly increase story variability in the chromosomes.

Based on the observations and objectives given above, we can construct a FDBS that invokes fuzzy rules according to the performance of the GA. In this framework, the GA parameters are adjusted based on FDBS evaluations of the previous story components’ generation cycle, so that GA can use the adjusted parameters in the next generation cycle. Since fuzzy rule-based systems try to focus on capturing the generality or pattern of a problem [73], we try not to formulate too many rules for the FDBS to avoid “overfitting” [76] the controller, because this can result in poor performance of the GA. For simplicity, HEFTI maps and scales all of the input and output values into three fuzzy sets: WEAK, AVERAGE and STRONG (Figure 10).

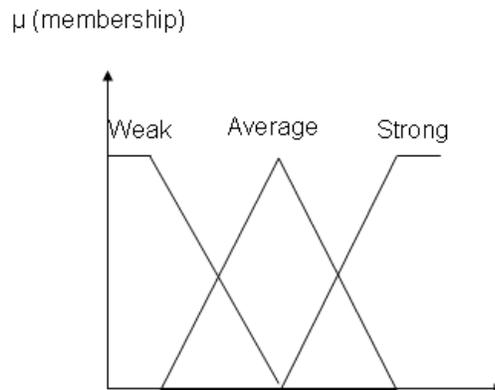


Figure 10. Symmetric fuzzy sets in HEFTI's

Mapping an input value to the ranges of the fuzzy sets is done by noting the six regions in the domain of the input variable (as indicated by the dotted lines in Figure 11) that divides the range of an input value into six even partitions. Functions that describe the fuzzy sets can be determined using equations for straight lines and diagonal lines in geometry, such as the equations that describe the shape of the fuzzy set *Weak* in Figure 10:

1. If $i \leq 3$ then $\mu(i) =$
2. > 3 & $i \leq 9$ then $\mu(i) = (9 - i)/(9 - 3)$
3. If $i > 9$ then $\mu(i) = 0$

Where $\mu(i)$ represents the membership value of the input value i .

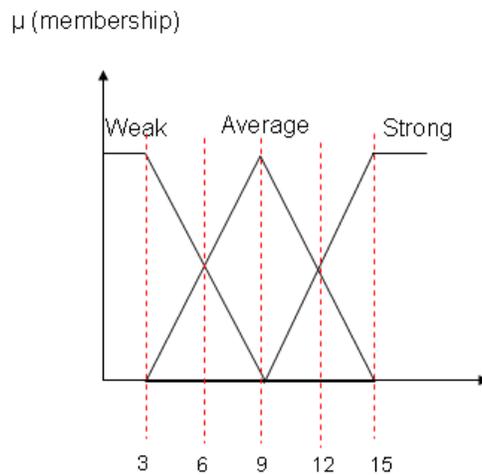


Figure 11. Regions in the domain of an input variable

HEFTI adheres to the inference scheme of the Mamdani model due to how its rules are described in the “If ... Then ..” form instead of the TSK form “If ... Then $f_n = ax+bx+c$.” The fuzzy rules used in HEFTI consist of the following (See below for symbol definitions):

1. If a is STRONG and b is AVERAGE then y is STRONG
2. If a is WEAK and b is WEAK then c is STRONG and s is STRONG and y is WEAK

3. If a is STRONG and b is STRONG then c is WEAK and s is WEAK and y is WEAK
4. If c is WEAK and s is WEAK and a is AVERAGE then c is AVERAGE and s is AVERAGE and y is AVERAGE

Similar to the example demonstrated in Figure 9, the values of these rules are combined with the minimum fuzzy operator and defuzzification of the subsets is computed with the Centroid method.

Various symbols are used in the rule sets above for convenience. We will go through their definitions one by one. Symbol a measures the given number of epochs that GA can use to generate a story component. b measures population variability that is computed by comparing the differences in individual genes (each representing a set of story elements) among the chromosomes in a population. Comparison between genes can be accomplished easily by comparing the element indices encoded in the genes, if they are different, we will increment b by 1 otherwise we will leave b as it is. This computation can be viewed as a process where each chromosome is matched to a different pair, thus the number of comparisons being made to a population can be computed by the equation $N * (N - 1)$, which translates into selecting a random chromosome from the population and pairing it with the next chromosome without replacement.

The symbol y stands for the selection criteria of individuals from a population. Although not explicitly stated in the rule set, the resultant firing strength of the fuzzy rules (in the range between 0 to 1) is mapped to a function whose value serves as a scalar that modifies individual fitness of genes that collectively contributes to the overall fitness of a chromosome. The adjusted fitness of the genes is computed by the equation below:

$$F = f \cdot e^{-R}$$

Where e = natural log, R = firing strength of the rule, f = fitness of the gene before normalization and F = fitness of the gene after normalization.

For low values of R (indicates low selection criteria), the function is *concave down* to encourage variability in a population by promoting chromosomes with average fitness, as shown by the graph below ($R = 0.1$):

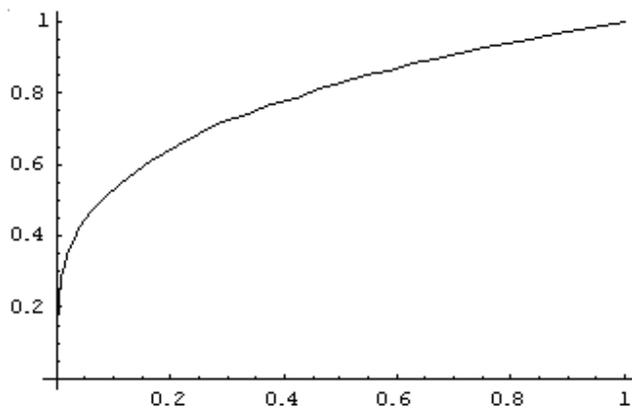


Figure 12. Adjusted fitness function with $R = 0.1$

However, as the value of R increases (which indicates a stronger selection criteria), the function gradually becomes a *concave down* function that serves to increase the difference in fitness between weaker and stronger chromosomes, thus gradually reducing population variability since strong individuals are now more likely to survive the selection process due to a larger boost in their fitness, as shown by the graphs below ($R = 0.5$),

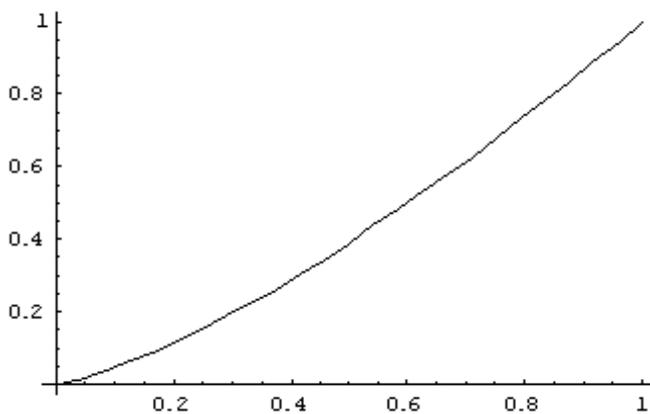


Figure 13. Adjusted fitness function with $R = 0.5$

and ($R=0.75$):

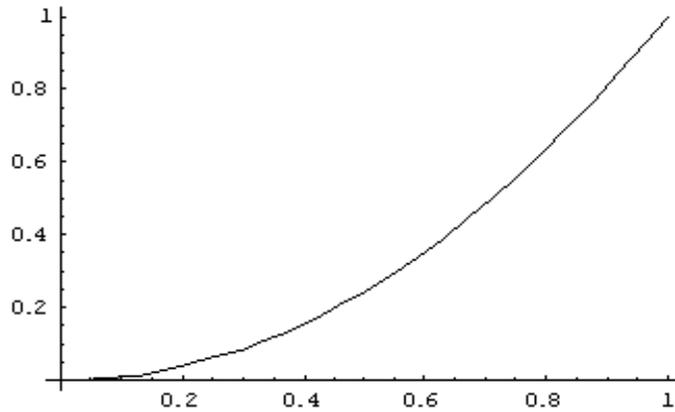


Figure 14. Adjusted fitness function with $R = 0.75$

In the rule set, c stands for GA's mutation rate. The firing strength of the fuzzy rules is scaled according to the standard recommended range of mutation rate between 0 and 0.1. For example, if we assume that the fuzzy rules are evaluated to a value of 0.75, the mutation rate will be modified to be $0.1 * 0.75 = 0.075$. Lastly, y stands for the crossover rate of the chromosomes and adjustments made by the fuzzy rules are similar to that of c .

As explained in the previous paragraphs, the rules are designed to monitor convergence, variability and selection criteria of the GA. Instead of using heuristic functions that can be less readable and sometimes difficult to formulate, fuzzy rules can be easily understood because of their linguistic labels and explicit expression of the relationships between variables. The rest of this section goes through the process of how the rules are formulated:

1. Rule 1 is used to control the selection criteria to avoid the event of too much population variability (b is AVERAGE) in the chromosomes near the end of the evolution cycle (a is STRONG). This rule is used to avoid having a population with many different average individuals such that the strong individuals are difficult to

determine because their differences are so small. Therefore, the remedy to the problem is to increase the selection criteria (y is STRONG) such that stronger individuals have better odds in dominating the population during subsequent evolution cycles since their competitiveness has been boosted significantly in the selection process by the F function.

2. Rule 2 is used to control the mutation and crossover rate of GA. Mutation and crossover rates [72] are one of the factors that differentiate the search behavior of the GA from random search. Higher values in these rates encourage greater variability in the chromosomes, thus closing the gap between GA and random search. Since we want greater population variability when GA begins its search in the problem space, we can monitor where it is in the evolution process (a is LOW) and the population variability measure (b is LOW) to adjust the mutation and crossover rate accordingly. The rule also controls the selection criteria of the GA (y is WEAK) such that weaker individuals have a better chance to pass the natural selection filter during early stages of the search because we do not want to confine the search process to a particular search region (represented by stronger individuals in the population) since this can potentially lead to local maxima solutions. Thus, better variability in the population encourages more uniform search across the problem space.
3. Rule 3 is just the complement of Rule 2, and it is used to increase the convergence rate of GA.
4. Rule 4 is used to adjust the mutation and crossover rates and the selection criteria of the GA when it is somewhere in the middle of the evolution process, such that effects of the fuzzy rules have a much smoother transition between the start and end of the entire evolution cycle. The absence of this rule would generate cliffs in the GA parameters (mutation rate, crossover rate and the selection criteria) for regions not covered by rule 2 and rule 3. Therefore, this rule ensures gradual adjustments to the GA parameters as the state of the system changes over the course of the entire evolution cycle.

The process of designing fuzzy sets and rules is an art [76]. No fixed set of guidelines exist that designers can follow to design a good fuzzy system. However, experience and detail knowledge about the problem domain can better acquaint engineers in designing fuzzy rules for various systems.

3. CONSTRUCTING STORIES

Under the story construction paradigm of dynamically building stories from generated story components, a story can be divided into various story elements, namely actions, agents, stage objects, music and action scripts. Relationships of these elements are described by story templates and background story thread to determine their interactions. Furthermore, there are a set of rules provided by the author to control how these story elements can be combined dynamically with each other at execution time while the story is being told. Although story elements are not as flexible as Lego™ blocks where children can form arbitrary objects from certain categories of geometry shapes, the story construction process of the HEFTI storytelling engine is conceptually similar to how children construct various variants of an object using building blocks under certain guidelines/templates.

3.1 Encoding the Story Components

Only certain types of story elements are encoded in chromosomes for stochastic sampling: 1) *actions*, which can generate various story events, alter the state of other objects and agents, and have scripts and music associated with them; 2) *agents*, which are the characters that carry out the plots, actions and events in the story; and, 3) *objects*, which are as important as actions since they are the props of a given story and sometimes their existence sets the stage for an appropriate story atmosphere and mood. In most stories, the mere existence of an agent or object in a particular scene of a story can spark many different events and plots. The following example from the three little pigs' story illustrates the importance of these elements in the process of story construction. Assuming that we are at the story time step when the wolf sees one of the pigs. If the pig has no house to hide in, the wolf can just eat the pig without devising a method to tear down the house. On the other hand, the existence of a house object between the wolf and pig can spark various other events and actions on these agents since: 1) the wolf will have to think of an alternative of action to remove the house object that prevents him from getting to the pig; 2) the wolf might choose the wrong alternative(s) and get himself killed in the

process; and, 3) attributes of the house object can also generate various other scenes or actions where the wolf manages to tear down the house or fails miserably in his attempt. If the wolf manages to tear down the house, he can eat the pig for lunch or the pig can escape from his claws by hiding in a different house. On the other hand, if the wolf fails to tear down the house, he will have to think of a different alternative of action so that the entire loop of decision making and event generation repeats itself again. Thus, from this short example, we can see easily how different the three little pigs' story can be with or without the existence of a house object.

The encoding process of the aforementioned story elements into chromosomes is accomplished by generating valid sets of story elements for each gene in the chromosome during each story time step. The purpose of this process is to generate a constraint chromosome that holds only valid story elements throughout the evolution cycle. This eliminates the need for a correction algorithm to check for inconsistent or redundant story elements. Under such an encoding framework, valid story elements consist of story elements that can be combined with each other according to the story conditions and rules provided by the author. This encoding process can only evaluate genes in a chromosome sequentially since the previous story element will have to be evaluated and changes in the story state applied before it can continue with subsequent story elements in the chromosome (Figure 15).

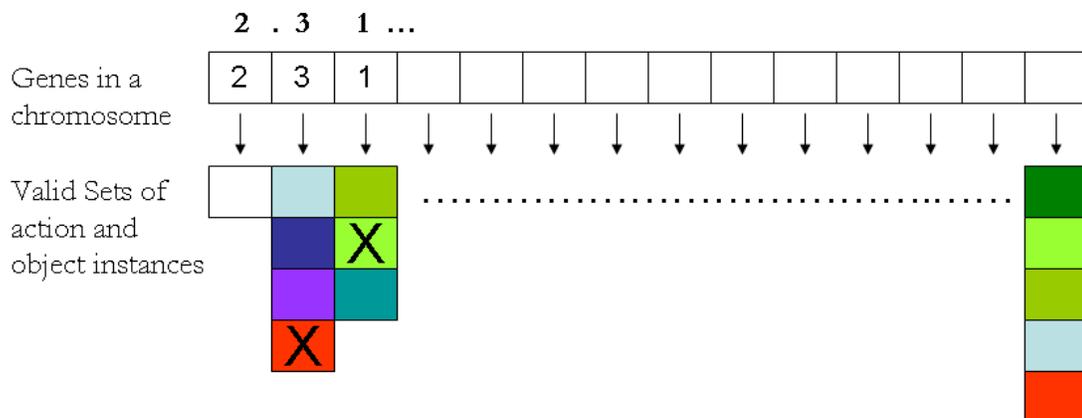
The example below illustrates how the chromosome encoding scheme generates story components for the next time step of the story and demonstrates how rules, actions and events conjunctively decide how a story should progress. Assume that we are starting a new three little pigs' story. HEFTI will choose a story template from the template set provided by the author to serve as a general guideline on how the story elements and text can be generated. Now let us look at how to encode one story sequence of the given template into a gene within the chromosome:

Template 1:

Sequence 1:

Once upon a time, there was an old pig with three little pigs {action_1}
 {currentpig1} is an {currentpig1.personality}, so {pig_action}
 {currentpig2} is an {currentpig2.personality}, so {pig_action}
 {currentpig3} is an {currentpig3.personality}, so {pig_action}

Note: {} depicts reference to a particular attribute, story element or a variable.



For example, 2.31.. above indicates with X's showing the object instances referenced

Figure 15. Encoding story elements into a chromosome

Depending on how the various action elements are authored, they can generate other events and actions according to the rules associated with them. The reference {pig_action} is a random variable that refers to a set of action elements: {action_2}, {action_3} and {action_4}. We have also associated a rule with {action_2} such that it invokes {action_5} when a certain condition is fulfilled, otherwise it will invoke {action_6}. Based on the example, the GA can encode these story elements into different genes based on the participating story elements (Figure 16).

Therefore, if a story template has more than one story sequence, it will have more gene selections and many different possible combinations based on the rule conditions, variables and action elements referenced in the template. A *story thread* that is created by the author is used to inform HEFTI about the various story stages, story templates to use

at each story stages in the evolutionary search process and specifies which of the story elements are preferred in a particular time step of the story over others. Author may assign a positive floating point number (to indicate preference) or a negative floating point number (to indicate avoidance) in the story thread towards a particular story element. When the GA selection process chooses a chromosome, the overall fitness of the chromosome is based on the existence of story elements such that 0.5 fitness value is assigned to story elements that are not specified by the story thread, while it assigns the manually given fitness values to story elements referenced in the story thread. Since fitness of an individual plays an important role in the natural selection process in GA, fitness values assigned by authors through the story thread can significantly boost or reduce the competitiveness of chromosomes in a population, thus allowing authors to exert a certain degree of control on the composition of the chromosomes and regions of the story space searched by the GA. Since strong individuals occupy larger intervals in the cumulative frequency domain than weak individuals, chromosomes with higher fitness values will have greater odds of becoming members of the new population and vice versa. After a certain number of evolution cycles (which can be specified by the author/reader), the strongest individual (the chromosome with the highest fitness value) in the population will be selected to decode into the appropriate story scripts to continue the story.

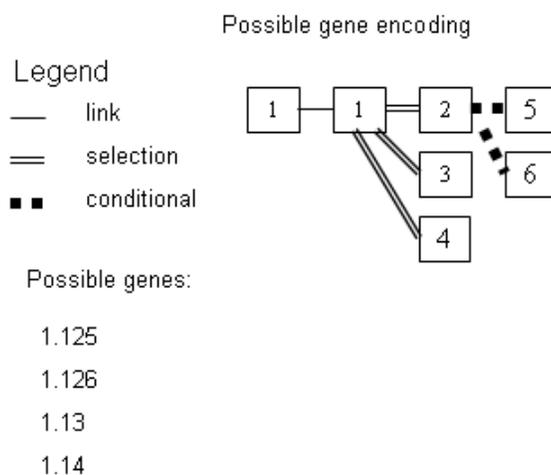


Figure 16. Gene representation of story elements

3.2 Decoding the Story Components

The decoding process is essentially the reverse of the encoding process since this process steps through each of the genes in the selected chromosome and decodes the genes according to their story context to generate a story component in order to continue the story (Figure 17).

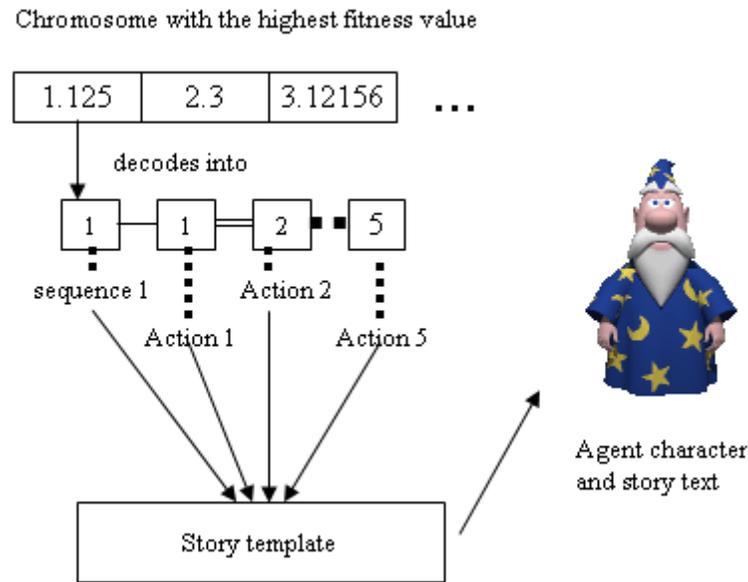


Figure 17. Decoding a chromosome into story scripts

It is possible for us to encode everything about a story into a chromosome regardless of the story granularity level. However, this is not always an efficient or practical approach because such redundancy may influence the performance of the evolutionary algorithm. This can further complicate the encoding, decoding and selection processes in the GA, since the GA will have to resolve any redundant information (such as, story text that does not influence the flow of the story) before performing searches in the story space. Since the GA is used by HEFTI to generate story components based on a given set of story templates instead of evaluating and resolving story elements to be included in the search process, the GA should concentrate on finding valid sets of possible story compo-

ment combinations. The responsibility of resolving ambiguities and creating story building blocks rests on HEFTI and the author.

Under the GA's scheme of recombination and mutation, chromosomes become collections of various story elements that can be combined to form a particular story component, and the story elements are combinations of even lower level story elements such as the story scripts, dialogues spoken by agents, scene objects and music. This hierarchical representation (Figure 18) allows music, story scripts, scene objects and text spoken by agents to be associated to a story element so that they can be combined with other story elements to convey the intended story mood to the reader. While various plots are acted out by the agents, the story atmosphere is reinforced by the existence of these complementary story elements in the scene.

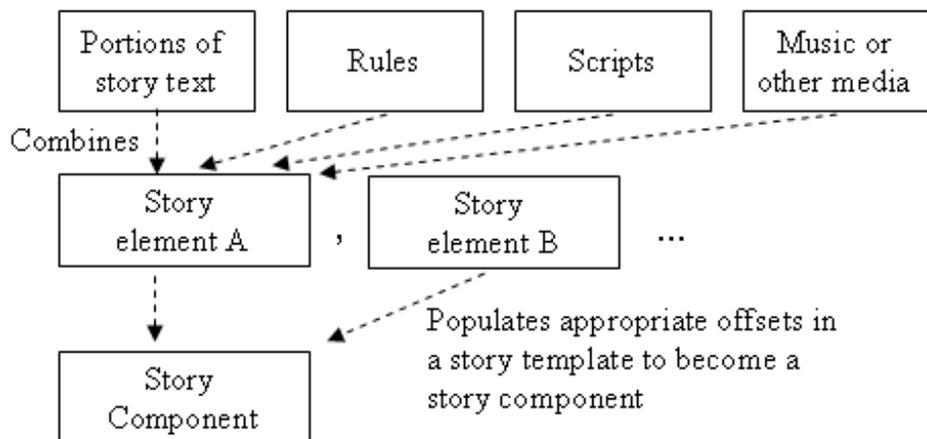


Figure 18. Hierarchical combination of story elements at various levels with a story template to form a story component

3.3 Stories Controlled by Rules

Analogous to writing a program, authors have to write rules that govern execution orders and decision making processes of agents that might arise when a program is loaded into the system memory for execution. The story authoring process requires authors to create rules that govern the generation of: 1) dynamic sub-stories that branch out of the

current story; 2) how agents react to a particular story context when certain conditions are met; and, 3) how the engine should recombine story elements to generate the next story sequence. HEFTI allows authors to create story rules and invocation rules to drive the story generation process.

Story rules govern the state of the story itself. They are provided by authors to control the inclusion of a particular set of story templates into a story based on certain conditions. These rules provide a mechanism for the engine to swap out irrelevant templates from the story generation cycle as necessary. Using the XML encoded story thread from the three little pigs' story as an example:

```
<timestep order="2" name="wolf terror" sequence="and" condition =
  "{ag_2.isEaten} == false OR {ag_3.isEaten} == false OR {ag_4.isEaten} ==
  false">
  <set type="wolf terror"/>
</timestep>
```

*Note: {ag_3.isEaten} indicates reference to a variable named **ag_3** with the attribute **isEaten***

The bolded text “{**ag_2.isEaten**} == false OR {**ag_3.isEaten**} == false OR {**ag_4.isEaten**} == false” depicts story conditions that HEFTI tests before the template set enclosed by the *timestep* tag can be used to construct new story components. In this example, the author tests the attribute *isEaten* of the three pig agents referred to as **ag_2**, **ag_3**, and **ag_4** before the template set *wolf terror* is used in the story generation process. Observing how the story is laid out in Section 5, we note that the pig agents (**ag_2**, **ag_3** and **ag_4**) may be eaten by the wolf before they can get to their houses, so execution of the story should stop when the pigs are gone from the scene because the big bad wolf would have nothing else to eat. Therefore, the absence of the conditional test in the template tag above generates inconsistency in the story because its absence allows HEFTI to generate inconsistent story components using an invalid template set that may result in laughable scenes such as the wolf trying to blow away a pig’s empty house. Instead of writing templates to handle every possible scenario, which can be infinitely large, a short

conditional test in a template tag (such as the example above) can eliminate a large number of invalid story templates from the template set.

On the contrary, invocation rules are rules that introduce branching, combinations of new story elements and uncertainty in the story components. They are more powerful than story rules because any number of them can be invoked in an action element to include additional story elements or rules to further alter the course of the story. Invocation rules are processed sequentially. An example from the three little pigs' story will be used to describe the invocation rule in greater detail (given an action element in the actions.xml file):

```
<action id="act_3" invoke="rule1">
    {currentpig.value.name} runs to his house |scene_12, {currentpig.value.agent-
    Name} |
</action>
```

The *invoke* attribute of this story element *act_3* invokes a rule *rule1* that has the following definition:

```
<rule id="rule1"
    type="normal"
    set="wolf terror"
    comments="rule1 = runs to his own house (if successful, hides in his own house,
    otherwise invoke wolf action 5)"
    precondition="$randf() > 0.25"
    then="$do(act_9)"
    else="$set(currentpig.value.isEaten,true);$remove(currentpig.value,ran-
    dompig);$do(act_5)"
/>
```

The precondition of this rule generates a random floating point number to determine how likely it is that a certain pig can escape from the wolf. According to the precondition clause, there is a 25% probability that the pig will be eaten before he/she can get to the house. If the pig is eaten, the else clause records the state of the agent and story before

invoking *act_5*, which shows the wolf eating the pig. On the other hand, if the pig managed to get away from the wolf, he/she will hide safely in his/her house (as described by the then clause *\$do(act_9)*). The story elements, such as *act_5* and *act_9*, involved in the clauses can each have their own association with certain rules so that they can trigger other rules or action elements in the knowledge base. Thus, under certain conditions, a chain of rules are triggered simultaneously that generate a very different story from the pre-authored set of story templates (Appendix A, Story Functions and Agent Scripting Functions, provides detailed descriptions of the various story functions and agent scripting functions that can be employed in various story elements).

4. THE INTERACTIVE AUTHORING ENVIRONMENT (IAE)

The IAE consists of a collection of editing tools and the Stage Play Subsystem (SPS). The editing tools provide high level access to the various story element definition files that describe various aspects of the story, elements and agents, whereas SPS uses various modules from the Microsoft Agent Character engine [57] to support agent animation, text-to-speech synthesis and event handling.

The story definition files are encoded in the ubiquitous and structured XML document format so that story elements, events and action instances can be represented in hierarchies. Editing tools in the IAE allow authors to define various story elements: 1) agent, action and object instances (agents.xml, actions.xml, objects.xml); 2) set of rules (rules.xml); 3) story thread (story.xml); 4) templates (templates.xml); and, 5) scripts (scripts.xml). The agent, action and object elements described in the XML files form the main building blocks of a story because they carry associations of rules, storyline, templates and scripting elements with them. Therefore, a rule is unlikely to be invoked without an agent carrying out certain actions, which can subsequently trigger other rules or actions that can alter the course of the story. Scripts that control how the agents will act out a particular plot graphically are executed only after the aforementioned rules are resolved.

4.1 Agent Characters

The SPS delegates character animations, text-to-speech synthesis and event handling to the Microsoft Agent Character engine. This engine is freely available for download on Microsoft's agent character website along with a graphical editing tool that allows the user to create and modify agent characters (Figure 19).

Most of the text input boxes in the editor are self-explanatory, thus, users should be able to get acquainted with this tool rather quickly. The properties tab page provides access to an agent character's properties (Figure 19), the word balloon tab page allows the user to customize the size of the balloon used to display text (Figure 20), and the voice tab page presents a selectable list of any text-to-speech engines that are installed on the

local computer (Figure 21). The tree view control to the left lists the animation frame set that is assigned to this particular agent character. Since an animation is formed by displaying multiple image frames one after the other within a certain time frame, each of the frames should represent certain minute changes in an agent's posture. Thus, assigning a new animation to an agent requires several image frames to create the illusion that an agent character is executing a certain action (Figure 22).

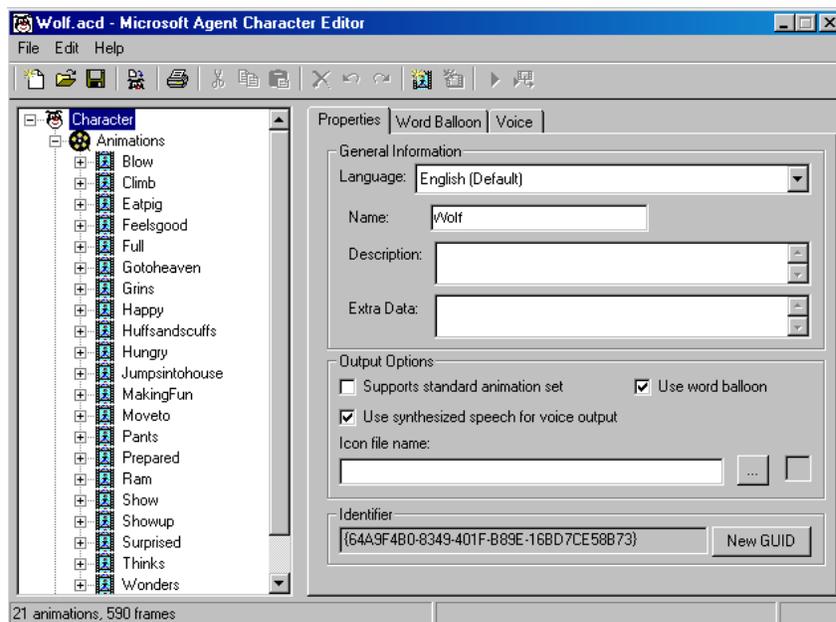


Figure 19. Microsoft agent character editor's properties tab page

After generating animations for an agent character, the user can now build the agent character for the engine by clicking on the *build character* menu item under the *file* menu (Figure 23). This command generates an agent character specification (.acs extension) file that can be linked into any windows applications as an agent.

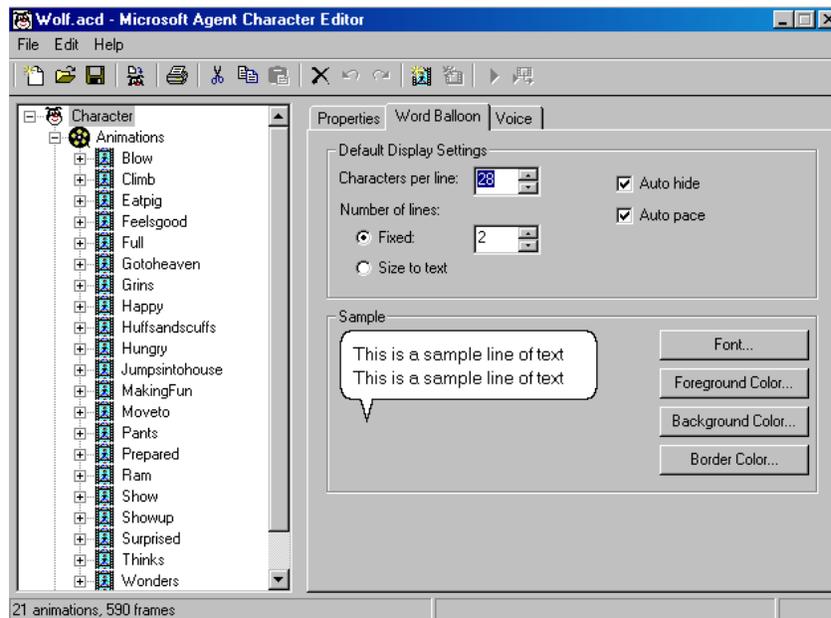


Figure 20. The word balloon tab page

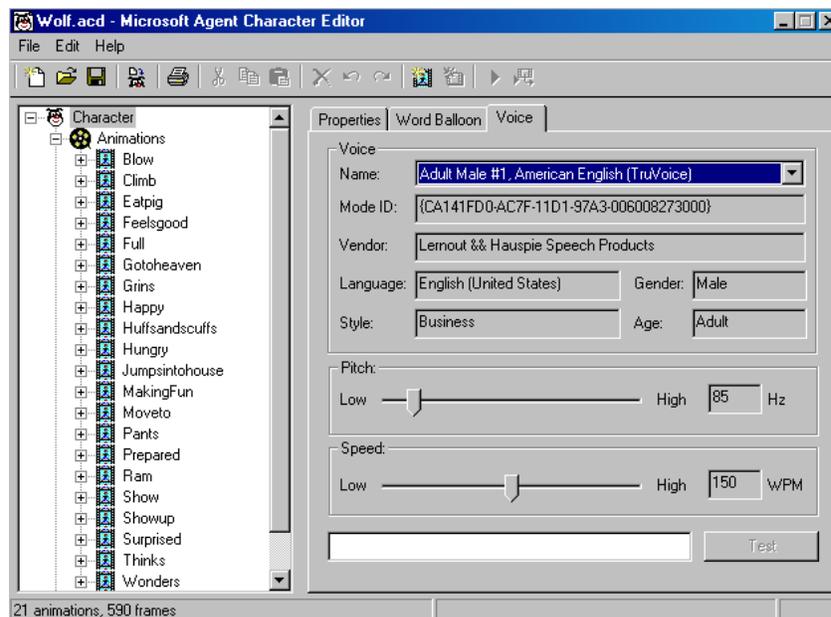


Figure 21. The voice tab page

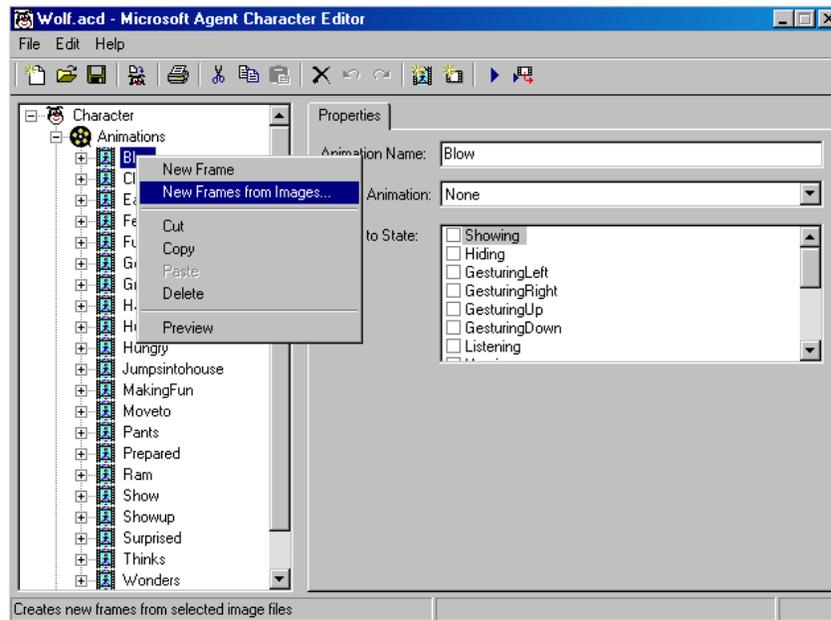


Figure 22. Assigning image frames to an animation

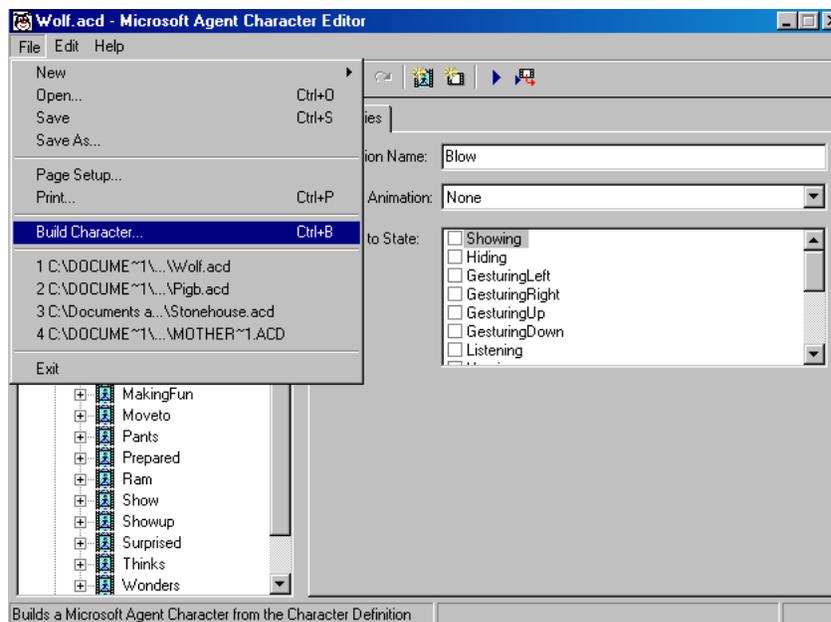


Figure 23. Building a character agent

4.2 Text-to-Speech (TTS) Engines

The agent architecture supports a myriad of speech input (voice recognition) and speech output (text-to-speech) engines. Although speech input is not used by the IAE, HEFTI uses speech output extensively in the narration process since he text-to-speech engine can bring the auditory experience of a stage play to a higher level of realism than mere story text. As dialogues between the characters and the tone of the narrator become part of the narration, authors have the capability to adjust agents' voices to reflect their personalities and complement their voice with the appropriate background music to buttress the atmosphere of the story.

The Microsoft agent character architecture offers support for numerous text-to-speech (TTS) engines, and they are available for download on the Microsoft Agent's website. This architecture offers an immensely robust and flexible application environment due to its support of many different TTS engines and languages. Therefore, stories authored with the IAE can be distributed in many different languages. The reader can choose the language or it can be chosen according to the reader's locale. The tasks of encoding, decoding and pronouncing the words are delegated to the TTS engines since the authors no longer have to provide a different set of voice recordings for each of the languages. Languages supported by the TTS engines are immense and English, Chinese, Japanese, Russian and Portuguese are just a small set of the languages supported.

Although manually configuring the TTS engines is not necessary, users can control the voice type, voice speed and other options via the *Speech* control panel item. Double clicking on this panel item brings up the Speech Properties dialog box that is populated with controls to alter the parameters of an agent character (Figure 24).

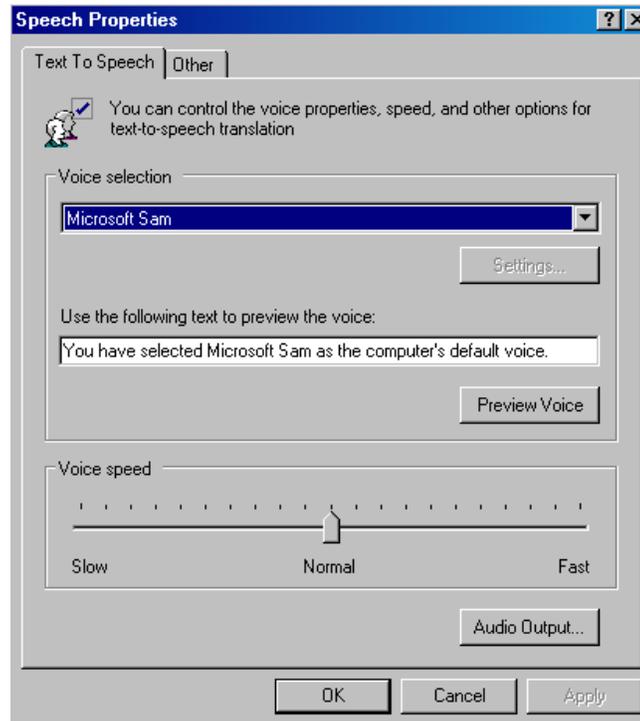


Figure 24. The speech properties dialog box

4.3 The IAE Interfaces

The IAE provides various authoring tools to create story elements, and it also comes with a reading interface that is populated with track bars, story log controls and control panel for readers to choose their favorite story branches and manually override certain system parameters while reading a story. Separation of the IAE into two different modes differentiates its functionalities in two areas so that the authoring interface is used solely to generate story elements, whereas the reading interface can be used to read a distributed story or test and debug a story generated in the authoring interface.

4.3.1 The Authoring Interface

The authoring mode gives authors full control over all aspects of the storytelling engine while allowing rules and parameters to be adjusted accordingly to suit their preferences. It consists of eight editors that modify the corresponding XML documents which

represent the story contents and context in HEFTI's knowledge base. The editors are: 1) *storyline*, which corresponds to the story thread provided by authors as an evaluation and story creation guideline; 2) *template*, which modifies story templates that HEFTI uses to generate dynamic stories; 3) *action*, which allows authors to create action elements that can be used by agent characters to act out the plots; 4) *object and variable*, which modifies the corresponding system state, list and agent state variables; 5) *rule*, which displays a set of rules that govern the deletion, combination and inclusion of story elements; 6) *story resources*, such as agent dialogues and multimedia contents embedded in the story; 7) *agent*, which enables authors to associate any attributes to the agent characters, such as personality and behaviors to be used by rules defined with the rule editor; and, 8) *agent scripts*, which associates scripts to various agent actions or scenes in the story so that they can be combined dynamically by HEFTI (Figure 25).

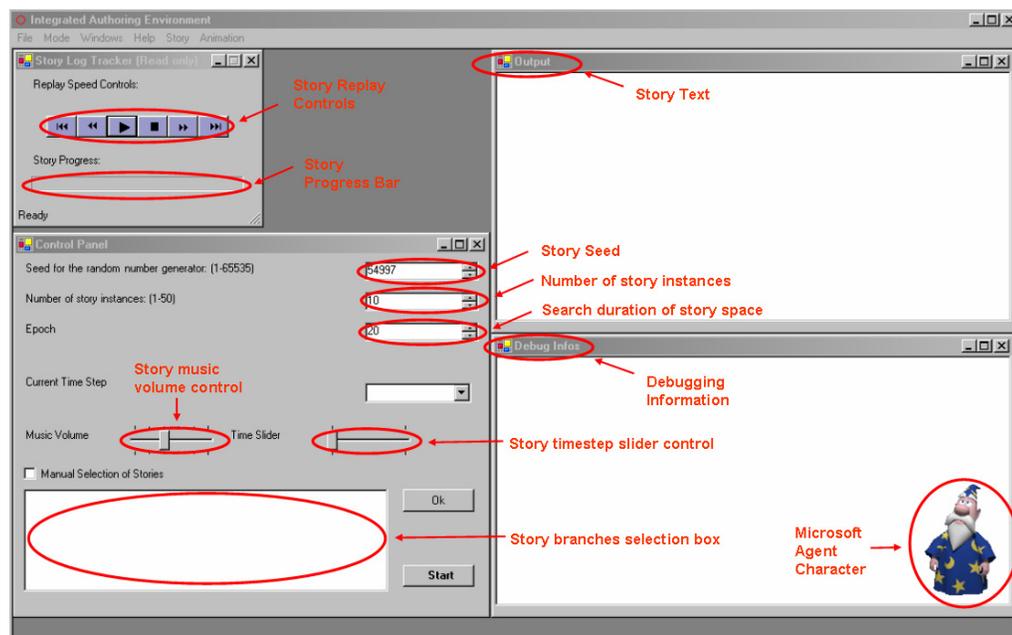


Figure 25. The authoring interface

4.3.2 The Reading Interface

The Reading mode, on the other hand, is a slender version of the environment without the various authoring tools to prevent readers from modifying properly configured story

content distributed by authors. The read-only mode initializes the EFS with the pre-authored story context and parameters distributed in a set of XML files and begins dynamically narrating the story according to the reader's interaction and system parameters (Figure 26). It consists of four windows: 1) *Story log tracker* that records the story as it progresses so that readers can export their stories into an XML file to be shared with others or replay a particular variant of the story that they enjoyed; 2) *debugging information window* that provides information about the events and exceptions that are generated by the system as the story is being told due to missing audio files or other story resources; 3) *Control panel* that allows the user to manually control various parameters of the GA. The story seed is a random number generator seed that drives the HEFTI storytelling engine so that a reader can use the same story seed to reread the same story again. Similarly, the number of story instances and search duration controls determine the number of chromosomes and duration that the EPS will search the story space. Readers do not have to worry about setting appropriate values for these controls because they are automatically controlled by the FDBS, however, this interface allows them to indirectly choose a particular way that the story is being generated by HEFTI; and, 4) the *story text window* displays the actual story text that is generated by HEFTI.

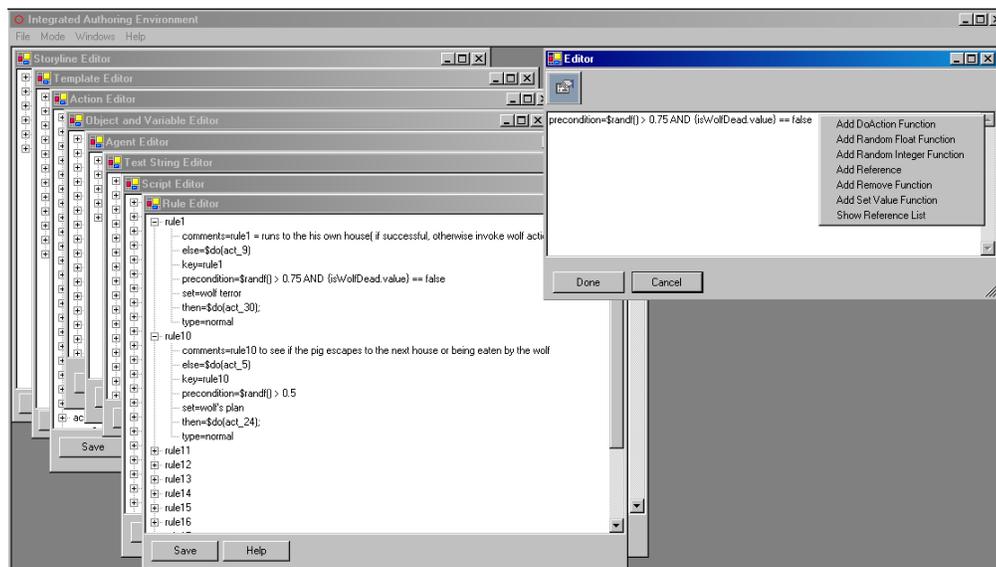


Figure 26. The reading interface

4.4 Syntax and Semantics of the XML Tags

4.4.1 Replaying a Story

The Reading Interface allows readers to read a story with different story seeds repeatedly. However, the Story Log Tracker window shown in the top left corner of Figure 22 allows readers to save the story they have generated into an XML file for safe keeping or distribution. Since this file is stored in the XML format, any computer user can view its contents with a text editor. The format of the replay file is grouped according to scene ids and played in sequential order. An example is as follows:

<ScriptLogs>

<scriptgroup id="scene_3"

synchronized="true"

textoutput="The three little pigs lived with their mother, they play and play day in and day out until one day ,"

scripts="\$setbackground(c:\\documents and settings\\mike\\desktop\\IAE\\bin\\debug\\imports\\images\\back.bmp); \$playsong(c:\\documents and settings\\mike\\desktop\\IAE\\bin\\debug\\imports\\music\\intro.wav);

\$show(MotherPig,0,0); \$say(MotherPig,'Hi, I am the mother pig');

\$show(PigA,300,0); \$say(PigA,'Hi, I am Angela'); \$show(PigB,300,80);

\$say(PigB,'Yawn, I am Barney'); \$show(PigC,300,160); \$say(PigC,'Howdy!

I am Charlene'); \$relativemove(MotherPig,-20,0);" />

<scriptgroup

id="scene_33"

synchronized="false"

textoutput=""

scripts="\$anim(PigA,0,Play,yes,2);\$anim(PigB,0,Play,yes,2);

\$anim(PigC,0,Play,yes,2); " />

...

</ScriptLogs>

The **scriptgroup** tag consolidates all of the functions invoked in a story scene as a group. In contrast to a story template, the parameters and references of variables (such as *{currentPig.id}*) are “expanded” to refer to actual agents (such as *PigA*). Various scripting functions are separated by the ‘;’ character, but they still have the same definitions as that described in Appendix A, Story Functions and Agent Scripting Functions. Therefore, one can treat the replay scripts as merely a collection of agent scripts and functions that are used to control agents and media elements for a particular variant of the story.

4.4.2 *Actions.xml*

XML tags in the *Actions.xml* file describe various agent action elements that can be used in a story. This file has to be a well formed XML document. Subsequently, individual action tags are enclosed by the `<actions>` tag that supports a comment attribute, this tag functions analogously to the `<document>` tag used in most XML documents, but since we are not dealing with ordinary text documents, the tag “actions” provides better description than “document” in this context. For example:

```
<actions comments="stores various action elements">
  <action .../>
  ...
</actions>
```

The **actions** tag encloses various action elements that can be performed by the agent characters. Due to complexities stemming from possible story branches, actions, agent interactions and conditions, the action tag also allows invocation of various functions, variable references and scripting elements other than plain story text. We will explain such mechanisms using an example from the three little pigs’ story:

```
<action id="act_5" invoke="rule9">
  The wolf eats {currentpig.value.name} $set(currentpig.value.isEaten,true)
  |scene_14|
</action>
```

The **action** tag has a unique attribute called *id* and an invocation attribute called *invoke*. The *id* attribute uniquely identifies this action instance within the story knowledge base, so that HEFTI can reference this story element by its id. Analogously, the *invoke* attribute invokes a particular rule instance by its unique *id* whenever this particular action instance is invoked. References to a particular variable are indicated by enclosing their ids in curly braces, such as “{” and “}”. In our example, {*currentpig.value.name*} references a global variable called *currentpig*, the next dot operator, in turn, references an attribute, *value*, of *currentpig*, and the last dot operator references the attribute *name* of an actual pig agent. The referencing process is resolved as follows: 1) the global variable *currentpig* and the attribute *value* are processed to reference an actual object instance, such as *agt_2*; 2) the next attribute name now references *agt_2*'s *name* attribute, so its value is retrieved from the knowledge base; and, 3) the locations where this reference is used are replaced by the result and become part of the story text.

A small set of functions are supported by the scripting engine to allow authors to keep track of the story states, control story execution, set the value of variables and remove story elements from a particular set. In our example, the *\$set* function is one such function that assigns a particular value to a variable. In our example, the *\$set* function assigns the value *true* to an agent's *isEaten* attribute referenced by the text *currentpig.value*. A thorough discussion of the functions used in HEFTI is presented in Appendix A, Story Functions and Agent Scripting Functions.

4.4.3 *Agents.xml*

Agents play an important role in the IAE. Due to the customizability of the story environment, they can fulfill various roles such as narrators, actors and even background objects according to the wishes of the author.

There are two types of agents used in the three little pigs' story, normal agents such as the pigs and wolf, and object agents that can be background objects or props in the story. For example, object agents are used for the houses of the three little pigs with their own set of animations, such as collapsing.

The `agents.xml` file has to be a well formed XML document where agent specification begins with an `<agents>` tag that encloses descriptions of the agents:

```
<agents>
...
  <role id="ag_5" name="Wolf" personality="Agressive"/>
</agents>
```

The `<role>` tag enclosed by the `agents` tag provides various attributes of a particular agent in the story. Other than the `id` attribute, which uniquely identifies an agent for the system, authors are free to define their own attributes for their stories. In the three little pigs' story example, the attributes `name`, `personality`, `owns` and `isEaten` are used:

```
<role id="ag_1" name="Old Pig" personality="Neutral"/>
  <role id="ag_2" name="Pig A" personality="extremely lazy" owns="obj_3"
    isEaten="false"/>
  <role id="ag_3" name="Pig B" personality="not so lazy" owns="obj_2"
    isEaten="false"/>
  <role id="ag_4" name="Pig C" personality="hard working" owns="obj_1"
    isEaten="false"/>
  <role id="ag_5" name="Wolf" personality="Agressive"/>
```

The `name` attribute is used to name the agents. The `personality` attribute is used to set an agent's personality. The `owns` attribute specifies the owner of a particular house object, for instance, Pig A owns the straw house as he is an extremely lazy pig who does not want to work hard. Lastly, the `isEaten` attribute serves as a state variable which allows rules to determine if this pig is eaten by the wolf.

4.4.4 *Objects.xml*

Various variables are used to keep track of the story's progress and states of the agents, such information is treated as objects implicitly by the engine. There are two types of variables supported by HEFTI: single value variables and random variables. A

single value variable holds the value of a particular attribute in a story element, for example:

```
<object id="obj_1" name="stoneHouse" collapsable="false" material="stone"/>
```

The single value variable *obj_1* defined above is actually a reference to the *stoneHouse* object. It holds various attributes of the *stoneHouse* object that jointly describe the house's composition, how it should behave in face of a wolf's attack and its building material. The attributes described above are custom named to serve the purpose of the three little pigs' story, and authors are free to define their own attributes in their stories as long as the variables can be uniquely identified based on the *id* attribute.

The random variable provides a list data structure for authors to introduce randomness into the story. Various story elements can be added into the random variable so that they can be selected randomly by HEFTI. For example:

```
<object id="randompig" name="randompig" type="random" elements="
  ag_2,ag_3,ag_4" />
```

The variable *randompig* is defined to be a random variable by its *type* attribute, it has three elements associated to it, namely the *ag_2*, *ag_3* and *ag_4* agents. By associating these elements to a random variable, the engine can randomly choose an element out of this list whenever the random variable is referenced. Such a variable can prove to be very convenient in generating unpredictability, since the wolf can now choose a random pig to attack instead of adhering to a static story script. Unpredictability introduced by a random variable can generate many different story branches from the original story while obeying the rules of the story.

4.4.5 Rules.xml

Rules play an important role in dynamic stories because they control various aspects of the story while determining the alternative actions of agents under various conditions. The **rules** tag encloses various individual rules that might or might not be invoked in the story and associates the rules to a unique id.

Rules in HEFTI are meant to provide conditional reasoning for authors, such as assigning the value true to a global variable when the wolf shows up in the scene. There are several attributes that can be used in the rule tag: 1) the *set* attribute associates a rule to a particular story template set, in other words, a rule associated with a particular template set will only be in force when story components are constructed out of this set; 2) the *comments* attribute allows authors to associate comments to the rules they have written to improve readability; and, 3) *precondition*, *then* and *else* attributes allow authors to associate premises of the rules with various global variables, agent states and attributes in the if-then-else statements. The example below illustrates a story rule being encoded in the described format:

```
<rule id="rule1"
  set="wolf terror"
  comments="current pig runs to his own house( if successful, otherwise invoke
  wolf action 5, which makes the wolf eat the pig)"
  precondition="$randf() > 0.25"
  then="$do(act_9)"
  else="$set(currentpig.value.isEaten,true);$remove(currentpig.value,ran-
  dompig);$do(act_5)"/>
```

The rule above has the unique id *rule1*, and it belongs to the template set *wolf terror*. The precondition of this rule uses a random number generator to make a pig escape from the claws of the wolf 75% of the time by hiding in his/her house, while 25% of the time the pig will be eaten by the wolf before he/she could get to safety. In the event that the wolf eats the pig, the attribute *isEaten* of this pig is set to true and the pig's id is removed from the random variable *randompig* so that the wolf cannot inflict further pain on the same pig. Lastly, agent scripts that control the scene where the wolf eats the pig are acted out by the agents on the screen by carrying out the appropriate actions.

4.4.6 *Scripts.xml*

Agent characters in the story are controlled by scripts that govern how they act out their roles at run time. However, agent scripts are used only in stories told in the graphical IAE (Section 4, THE INTERACTIVE AUTHORING ENVIRONMENT (IAE)). Stories told in the text-based IAE do not require any agent scripts (Section 7, AUTHORIZING A TEXT BASED STORY). The structure of the agent scripts.xml file should adhere to the following layout:

```
<agentscripts>
  <script id="scene_1" comment="The Angela pig is a happy pig" synchron-
    ized="true">
    All agent scripting functions are provided here
  </script>
  ...
</agentscripts>
```

The *id* attribute uniquely identifies the script within the story, the *comment* attribute allows authors to provide their own descriptions of the script for future references and the *synchronized* attribute indicates to the storytelling engine whether the scripting functions should be “synchronized” (executed in order) or “non-synchronized” (simultaneous execution). There are many potential uses for this attribute, for instance, a virtual chorus can be simulated by assigning the value *false* to the *synchronized* attribute so that multiple dialogues of the agent characters are synthesized by the text-to-speech engines (Section 4.2, Text-to-Speech (TTS) Engines) at the same time.

4.4.7 *Story.xml*

The story.xml file describes the general structure of the story to HEFTI. Within the enclosing **storyline** tag are sets of story templates, object, action and agent elements defined by the author. Authors can include any set of story template into a particular time step of the story by referencing them within the “timestep” tag and assigning the proper time value to the order attribute, for instance:

```

<timestep order="1" name="introduction" >
  <set type="introduction">
    <element name="act_10" addfitness="1"/>
  </set>
</timestep>

```

The **timestep** tag above has its *order* attribute set to 1, which makes it the first set executed by the storytelling engine. Also enclosed by this tag is the **set** type *introduction* which specifies the name of the set of templates, objects, actions and agent instances to choose from. The **element** tag specifies preference or avoidance towards certain story elements. In the above example, the storytelling engine is instructed to increase the overall fitness of a chromosome that holds the story element *act_10*. An author can also penalize the fitness of a chromosome within a particular story element by assigning a negative floating point value to the *addfitness* attribute.

Looping and conditional statements are also supported by the story thread. These mechanisms allow authors to “program” a dynamic story with a huge set of valid story variants without engaging in the ordeal of determining their validity manually. Functions of the looping and conditional statements are similar to most programming languages, such that when a given condition is met, the body of the statement will be executed and vice versa. The only difference between a looping and a conditional statement is that the body of a loop is executed repeatedly while the condition is valid, on the other hand, a conditional statement will be executed only once.

A looping example:

```

<timestep order="4" name="wolf's plan" loop="{isPigStillAvailable.value} ==
  true AND {isWolfDead.value} == false">
  <set type="wolf's plan" >
    <element name="act_13" addfitness="2"/>
  </set>
</timestep>

```

A conditional statement example:

```

<timestep order="5" name="Pigsconclusion" condition="{isWolfDead.value} ==
true AND {isPigStillAvailable.value} == true">
  <set type="Pigsconclusion" />
</timestep>

```

In order to indicate to HEFTI that it should enter into a loop, the author has to create conditions to be tested in the *loop* attribute. The loop example above has its *loop* attribute set to check for the value of the global variables *isPigStillAvailable* and *isWolfDead*. This loop will terminate whenever no more pigs are available for the wolf to devour or the wolf is murdered by one of the pigs. Analogously, setting the *condition* attribute indicates a tag to be a conditional statement. The conditional statement example above will be executed if and only if the wolf is dead and one of the pigs is still alive.

Organizing the general structure of a story into an XML hierarchy has the advantage that an author familiar with the format can quickly recognize the flow of the story. Detail discussions about the operators supported by the conditional and loop statements are provided in the Appendix A, Story Functions and Agent Scripting Functions.

4.4.8 *Templates.xml*

Story templates form an integral ingredient in HEFTI's story construction. Each of the templates defined in the *Templates.xml* file belongs to a particular template set referenced in the story thread. Associated with these template sets are rules, scripts, actions and agent instances such that each of them plays a role in forming the final story that the reader sees on the screen. Correlations between these story elements make story templates the most difficult for authors to construct since authors have to be vigilant in detecting conflicting rules that might cause undesirable or unpredictable results, while at the same time concentrating on the congruence of the story text embedded in the story templates. In addition, authors have to consider the appropriate time and location to insert story scripts that drive the agents so that the actors will not perform an action ahead of the narrator's speech.

Templates allow authors to break up a story into any granularity. The smaller pieces that form a template are known as story sequences. These represent a series of smaller time steps. Support for varying levels of granularity in the story templates allows authors to map a particular story sequence to a simple action, such as an agent lifting his hand, or a highly involved story plot where a detective is interrogating the suspect in a murder mystery. The templates file has to adhere to the hierarchical structure below:

```

<templates>
  <template id type>
    <sequences>
      <sequence order>
        statements, story text, refereces to object instances and others.
      </sequence>
      <sequence order>
        ...
      </sequence>
    </sequences>
  </template>
</templates>

```

The *id* attribute of a story template uniquely identifies the template for HEFTI. The *type* attribute defines the template set to which the story template belongs. Assigning a template to a template set allows the engine to identify which templates to use during a particular stage of the story, as described in the Story.xml section. The Story.xml file uses the *set* attribute to name stages in the story, and associate rules and various object instances to a particular time step in the story.

The listing below is an excerpt from the three little pigs' story:

```

<template id="tmplt_2" type="introduction" >
  <sequences>
    <sequence order="1" >

```

The three little pigs lived with their mother, they play and play day in and day out until one day , |scene_3| {act_1}

</sequence>

<sequence order="2" >

\$set(currenthouse.value,obj_3.name) \$set(currentpig.value,ag_2.id) The {ag_2.personality} {ag_2.name} pig |scene_4, {currentpig.value} | {act_10}

</sequence>

<sequence order="3">

\$set(currenthouse.value,obj_2.name) \$set(currentpig.value,ag_3.id) The {ag_3.personality} {ag_3.name} pig | scene_4, {currentpig.value} | {act_10}

</sequence>

<sequence order="4">

\$set(currenthouse.value,obj_1.name) \$set(currentpig.value,ag_4.id) The {ag_4.personality} {ag_4.name} pig | scene_4, {currentpig.value} | {act_10}

</sequence>

</sequences>

</template>

The text enclosed by the sequence tags represent the story text. Although some of them are actually function calls, object references and embedded scripts, all of them will be resolved to strings at execution time. For example, the string generated from the first element of this sequence could be “The three little pigs lived with their mother, they play and play, day in and day out until one day, the mother pig grows old and asks the little pigs to live by themselves.” Scripting elements enclosed in “|”s will not generate any story text, whereas references (enclosed in “{” and “}”), might generate story text depending on the rules being invoked, and the state of agents and variables. Similarly, function calls (preceded by the “\$” character) may or may not generate story text since certain functions have a return value while others do not.

4.4.9 *Textstrings.xml*

All of the spoken text, and references to file and music objects are stored in the *textstrings.xml* file. Each of these is treated as an XML element enclosed by the **string** tag such as the following:

```
<textstrings>
  <string id="PigA_Greeting" locale="USEnglish" text="Hi, I am Angela|Eh, I
    am Angela"/>
  <string id="PigB_Greeting" locale="USEnglish" text="Hi, I am Barney|Yawn, I
    am Barney"/>
  ...
</textstrings>
```

Authors cannot customize the attributes of the XML tags in this file and attribute values will have to be assigned in order to make them accessible to HEFTI. The attribute *id* is a globally unique identifier for that resource, whereas the *Locale* attribute allows authors to provide different languages or files to be used according to the locale setting of the IAE. To provide randomness in agent dialogues, authors can use the “|” operator to form a list of texts that can be selected arbitrarily by HEFTI as the story is being told. For example, the tag “<string id="PigA_Greeting" locale="USEnglish" text="Hi, I am Angela|Eh, I am Angela"/>“ provides two possible text strings that can be chosen by HEFTI whenever the reference *PigA_Greeting* is used in the story. Authors can assign any names to the IDs of these resources but it will be convenient to name them systematically.

5. AUTHORIZING AN INTERACTIVE, DYNAMIC STORY

In contrast to traditional stories that are overlaid upon two dimensional media, story branches in an interactive and dynamic story are similar to programs generated by the GA that can respond according to conditions or inputs specified by the author. Conventional approaches for visualizing story elements in a static story are no longer adequate due to the dynamic nature of the story. Story variations stemming from interactions and relationships between story elements that are driven by the GA can introduce high degree of unpredictability in the resultant story that even the author cannot foresee.

A dynamic story has constructs that are similar to a computer programming language. For instance, a rule in the story is similar to a conditional statement in a program because it tests certain conditions to decide the possible courses of action. We can construct a story flow graph out of a dynamic story in the same fashion as we derive an informal flow graph out of a well-written computer program. A high level overview of the three little pigs' story is shown in figure 27:

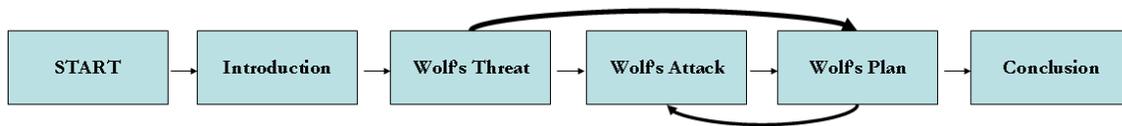


Figure 27. A high level story flow graph for the three little pigs' story

1. The **Start** stage initializes various variables and agent characters.
2. The **Introduction** stage provides all necessary story background for the reader. This guides the reader smoothly into the next stage of the story.
3. The **Wolf's Threat** stage introduces one of the main characters, namely the wolf agent, into the story. This indicates the start of the main story body.
4. The **Wolf's Attack** and **Wolf's Plan** stages depict interactions between the pigs and the wolf as they engage in a series of actions. These stages constitute the highlights of the story. As indicated by the back-pointing arrow in the **Wolf's Plan** stage, the story engages in a loop so that the wolf devises new attacks according to certain story conditions.

5. Finally, the story will reach the **Conclusion** stage when the wolf manages to eat all of the pigs or was cooked by one of the pigs, thus ending the story.

To avoid confusions that might arise from the back-pointing and branching arrows in the **Wolf's Threat**, **Wolf's Attack** and **Wolf's Plan** stages, design details of this story in these stages are provided below:

1. The story stage before **Wolf's Threat**, namely **Introduction**, should provide an entry point for the wolf agent into the story. Sentences such as “the pigs are so happy with their new homes that they are completely unaware of the dangers lurking near by ...” is used to prepare the reader for next stage of the story.
2. The **Wolf's Threat** stage serves more functionality than merely introducing the wolf agent. More variations of the story can be generated by providing story elements insertion points in various locations of the story templates for this story stage so that shortly after introducing the wolf agent into the scene, the storytelling engine may generate story components that describe how the wolf agent attacks the pigs before they can retreat into the safety of their homes. If the wolf manages to eat all of the pigs before they can run back to their homes, then the story should terminate, so the state of the story branches to the **Wolf's Plan** stage where the story termination conditions are verified, thus resulting in a branching arrow pointing from **Wolf's Threat** to **Wolf's Plan**. On the other hand, if some of the pigs managed to get back to their homes, story termination checking at the **Wolf's Plan** stage will not terminate the story, instead the state of the story moves to **Wolf's Attack** stage so that the wolf agent can plan his attacks on the remaining pigs, thus resulting in a back-pointing arrow from **Wolf's Plan** to **Wolf's Attack** story stages. Lastly, if no pig is harmed at this story stage, the wolf agent can start his attacks on the houses immediately in the **Wolf's Attack** stage.
3. In the **Wolf's Attack** stage, the wolf agent engages in various endeavors to break into a pig's house and there are only two possible outcomes from this scenario: death of the wolf agent or the house collapses and the pig agent is eaten by

the wolf. If the wolf agent is killed in action or all of the pigs are eaten by the wolf then the story will terminate after evaluating the story conditions in the **Wolf's Plan** stage and the state of the story moves into the **conclusion** stage.

The story flow graph greatly influences how the story is generated because authors are going to generate story components and templates that get recombined by the GA based on the rules and conditions derived from a story flow graph.

As described in various sections of this document, HEFTI generates a story for various story elements that are governed by rules. The underlying GA determines whether a story component is valid for a given context by using the story templates and rules provided by the author. We can view story components as LEGO™ blocks and story generation as the process of making various structures or shapes by following a set of guidelines described in the story thread. By having a high level overview of the intended story, we can incrementally build a detailed story flow graph populated with story elements and identify rules that we can incorporate into the story.

5.1 Case Study: A Simple Three Little Pigs' Story

This case study gives the reader a glimpse to a level of story authoring that is very low level for the average author, but tools (Section 5.2, Editing the Story Elements and Section 5.3, Authoring Stories with the Drag-and-Drop Interface) have been written to generate this level of detail for the author which allow them to work with a higher level language. With the story planned out, we can get our feet wet by first creating story elements for the three little pigs' story. We will generate a linear story with a few story branches. After this case study, readers can start generating more involved stories by utilizing the power of rules and agents.

We will first generate a story template for the **introduction** story time step. Since text strings in story templates are the main source of story text, we would normally populate the two templates with more descriptive text than other types of story elements. The listing below provides the story sequences embedded in the template sets:

1. Once upon a time, there was an old pig with three little pigs. {act_1} |scene_1|
2. \$set(currenthouse.value,obj_3.name) \$set(currentpig.value,ag_2.id) The {ag_2.personality} {ag_2.name} pig {pigpick}
3. \$set(currenthouse.value,obj_2.name) \$set(currentpig.value,ag_3.id) The {ag_3.personality} {ag_3.name} pig {pigpick}
4. \$set(currenthouse.value,obj_1.name) \$set(currentpig.value,ag_4.id) The {ag_4.personality} {ag_4.name} pig {pigpick}

With the various story elements referenced in the template set, we can now generate the agents, their attributes and possible actions using Table 1 (listed by order of appearance) below:

Table 1. Story elements referenced in template one

act_1	The old pig grows old and asks the little pigs to live by themselves
act_2	{currentpig.value.gender} builds a {currenthouse.value} scene_11,{currentpig.value.agentName},{currenthouse.value}
act_10	{currentpig.value.gender} buys a {currenthouse.value} scene_19,{currentpig.value.agentName},{currenthouse.value}
act_22	{currentpig.value.gender} wonders around and finally finds a {currenthouse.value} at last. scene_35,{currentpig.value.agentName},{currenthouse.value}
currentpig	<object id="currentpig" type="pointer" value="ag_2"/>
current-house	<object id="currenthouse" type="pointer" value="obj_1"/>
obj_1	<object id="obj_1" collapsable="false" material="stone" name="stoneHouse" collapsed="false"/>
obj_2	<object id="obj_2" collapsable="true" material="wood" name="woodenHouse" collapsed="false"/>

Table 1. (Continued)

obj_3	<object id="obj_3" collapsable="true" material="straw" name="strawHouse" collapsed="false"/>
ag_2	<role id="ag_2" name="Angela" personality="extremely lazy" owns="obj_3" isEaten="false" agentName="PigA" gender="she"/>
ag_3	<role id="ag_3" name="Barney" personality="not so lazy" owns="obj_2" isEaten="false" agentName="PigB" gender="he"/>
ag_4	<role id="ag_4" name="Charlene" personality="hard working" owns="obj_1" isEaten="false" agentName="PigC" gender="she"/>
pigpick	<object id="pigpick" elements="act_2,act_10,act_22" name="randompigaction" placement="noremove" type="random"/>
scene_1	\$setbackground(IMG_BACK);\$play-song(SND_INTRO);\$show(MotherPig,0,0); \$say(MotherPig,MotherPig_Greeting); \$show(PigA,300,0); \$say(PigA,PigA_Greeting); \$show(PigB,300,100); \$say(PigB,PigB_Greeting); \$show(PigC,300,200); \$say(PigC,PigC_Greeting);
scene_11	\$anim(\$getParam(1),"BuildHouse"); \$show(\$getParam(2), \$getPosX(\$getParam(1)), \$getPosY(\$getParam(1))); \$anim(\$getParam(1),"Happy");
scene_35	\$anim(\$getParam(1),"Dance"); \$show(\$getParam(2), \$getPosX(\$getParam(1)), \$getPosY(\$getParam(1))); \$relative-move(\$getParam(1),90,0); \$anim(\$getParam(1),"Happy"); \$say(\$getParam(1),Pig_Happy);

Table 1. (Continued)

scene_19	<pre> \$hide(\$getParam(1)); \$show(\$getParam(2), \$getPosX(\$get- Param(1)) + 20, \$getPosY(\$getParam(1))); \$show(\$getParam(1), \$getPosX(\$getParam(1)), \$getPosY(\$getParam(1))); \$anim(\$get- Param(1),0,"BuyHouse","yes",1);\$moveto(\$getParam(1), \$get- PosX(\$getParam(1)) + 10 , \$getPosY(\$getParam(1))); \$anim(\$getParam(1),"Happy"); \$say(\$getParam(1), Pig_Happy); \$relativemove(\$getParam(1),90,0); \$anim(\$get- Param(1),"Happy"); </pre>
----------	---

XML syntax and tags used in Table 1 are described in Section 4.4 of this document. Based on Table 1, note that we have incorporated a random variable `pick` within the template so that the GA can combine various actions associated with this random variable. This promotes randomness of the story but also allows chain invocation of rules when the action elements are associated with other actions or rules. The scene objects referenced in *tmplt_1* initializes the story environment by acts such as loading a background scene image, playing the appropriate music, displaying the active agents, animating an agent and displaying agent dialogues in the story. For more details about controlling an agent character, please refer to Appendix A.2. In this simplified three little pigs' story, we will only define one story template for the introduction story stage.

The second template, *tmplt_2*, for the **Wolf Threat** story stage, is rather simple, as it merely introduces the wolf agent and describes how the pigs try to flee from the wolf by running to their own houses. *Rule1* is referenced in *act_3* so that the pigs occasionally managed to escape from the claws of the wolf. Table 2 below lists all of the story elements that are used in this story stage.

1. One day, |scene_6| {act_4} \$set(currentpig.value,ag_2.id) {act_3} \$set(currentpig.value,ag_3.id) {act_3} \$set(currentpig.value,ag_4.id) {act_3}

Table 2. New story elements referenced in *tmplt_2*

act_3	{currentpig.value.name} runs to his house scene_12,{currentpig.value.agentName} , invoke="rule1"
act_4	The wolf shows up and frightens the pigs. scene_13 , Invoke="rule1"
act_5	wolf eats {currentpig.value.name} \$set(currentpig.value.isEaten,true) \$remove(currentpig.value,randompig), scene_14,{currentpig.value.agentName}
act_9	{currentpig.value.name} hides at home. scene_34,{currentpig.value.agentName}
rule1	precondition="\$randf() > 0.25" then="\$do(act_9)" else="\$set(currentpig.value.isEaten,true); \$remove(currentpig.value,randompig);\$do(act_5)"
scene_6	\$anim(PigA,"Dance"); \$anim(PigB,"Dance"); \$anim(PigC,"Dance");
scene_12	\$anim(\$getParam(1),"Surprised"); \$say(\$getParam(1),Pig_RunAway);
scene_13	\$show(Wolf, \$getPosX(PigA) + 250, 0); \$anim(Wolf, "ShowUp"); \$anim(Wolf,0,"MakingFun","yes",2); \$say(Wolf,Wolf_Loves_Pigs);
scene_14	\$moveto(Wolf, \$getPosX(\$getParam(1)), \$getPosY(\$getParam(1))); \$hide(\$getParam(1)); \$anim(Wolf,"EatPig"); \$anim(Wolf,"FeelsGood"); \$moveto(Wolf, \$getPosX(PigA) + 250, \$getPosY(PigA)); \$say(Wolf, Wolf_EatPig); \$say(MotherPig, MotherPig_Sigh);

Table 2. (Continued)

scene_34	\$moveto(\$getParam(1), \$getPosX(stoneHouse), \$getPosY(\$getParam(1))); \$anim(\$getParam(1),0,"scared", "yes", 1); \$anim(\$getParam(1),0,"HidesFace", "yes", 1);
----------	---

Lastly, the functions *\$set* and *\$remove* are used to inform HEFTI about the state of the pigs so that it will not potentially generate an invalid story sequence such as “the dead pig cooks the wolf” later in the story. With a certain number of pigs hiding in their houses and some, perhaps, eaten by the wolf, we will proceed to generate story template *tmplt_3* using the story elements listed in Table 3 for the **wolf attack** stage of the story:

1. {act_13} and prepares to attack |scene_8|

Table 3. Story elements referenced in *tmplt_3*

act_13	The wolf huffs and puffs. scene_22
scene_8	\$anim(Wolf,0,"Prepared", "yes", 2);
scene_22	\$say(Wolf, Wolf_Huffs); \$anim(Wolf,0,"Huffsand-puffs", "yes", 1);

Similar to the second story template, *tmplt_3* is a simple story template. For reasons that will be apparent to the reader later in this case study, this template will be incorporated in a story loop that generates the wolf attack sequence on each of the pigs.

The story template *tmplt_4* controls how the wolf attacks the pigs as they are hiding in their homes (**wolf’s plan**) using the story elements listed in Table 4.

1. \$set(currentpig.value, {randompig}), \$set(currenthouse.value, currentpig.value.owns) {act_17} and {wolfpick}

Table 4. Story elements referenced in *tmpl_4*

randompig	<object id="randompig" elements="ag_2,ag_3,ag_4" name="randompig" type="random"/>
wolfpick	<object id="wolfpick" elements="act_6,act_7,act_11" name="randomwolfaction" type="random"/>
act_6	The wolf blows at the {currenthouse.value.name} \$set(currentwolfact.value,act_6), scene_15,{currentpig.value.agentName} ,invoke="rule8"
act_7	The wolf tries to ram down the {currenthouse.value.name} \$set(currentwolfact.value,act_7), scene_16,{currentpig.value.agentName} , invoke="rule8"
act_8	The {currentpig.value.name} starts a fire at the chimney and cooks the wolf. scene_17,{currentpig.value.agentName}
act_11	The wolf climbs the chimney, scene_20,{currenthouse.value.name} , invoke="act_8"
act_17	The wolf comes to {currentpig.value.name} 's {currenthouse.value.name} scene_26,{currentpig.value.agentName}
act_18	The wolf pants, unable to break down the house. He has to think of some other forms of attack. scene_26,{currentpig.value.agentName}
act_21	the {currenthouse.value.name} collapses! \$remove(currenthouse.value,randomhouse) \$set(currenthouse.value.collapsed,true) scene_30,{currenthouse.value.name}
rule8	precondition="{currenthouse.value.collapsable} == true" then="\$do(act_21);\$do(act_5);" else="\$do(act_18);\$remove(currentwolfact.value,wolfpick)"

Table 4. (Continued)

scene_15	\$moveto(Wolf,\$getPosX(\$getParam(1)) + 60, \$get-PosY(\$getParam(1))); \$anim(Wolf,"Blow"); \$moveto(Wolf, \$getPosX(PigA) + 250, \$getPosY(PigA));
scene_16	\$moveto(Wolf,\$getPosX(\$getParam(1)) + 80, \$get-PosY(\$getParam(1))); \$anim(Wolf,"Ram"); \$moveto(Wolf, \$getPosX(PigA) + 250, \$getPosY(PigA));
scene_20	\$moveto(Wolf,\$getPosX(\$getParam(1)) + 80,\$get-PosY(\$getParam(1)) - 10); \$anim(Wolf,"Climb");
scene_26	\$moveto(Wolf,\$getPosX(\$getParam(1)) + 90,\$get-PosY(\$getParam(1)));
scene_30	\$anim(\$getParam(1),"Collapse"); \$hide(\$getParam(1)); \$say(Wolf, Wolf_Tease);

Tmplt_4 describes various ways that the wolf can attack a pig's house and some of these actions will invoke *rule8* to determine if the house can collapse in the face of the wolf's attack. If the house does not collapse, the wolf has to try other forms of attack.

Due to the setup of the story, the wolf can potentially eat all of the pigs before they can get to safety or the wolf can be cooked as he tries to attack the pigs. We will have to prepare two different set of story elements (Table 5) for the story endings: **pigs' conclusion** and **wolf's conclusion**:

Ending 1.As a conclusion, |scene_31| {act_19}

Ending 2.Lastly, |scene_10| {act_20}

Table 5. Story elements referenced in the conclusion templates

act_19	no one ever sees the wolf again. scene_28
act_20	the wolf eats the mother pig as well, and leaves this desolated place. scene_29
scene_10	\$anim(Wolf,0,"Happy","yes",2);
scene_28	\$playsong(SND_Alliluyah);\$show(Wolf, 550, 0); \$anim(Wolf,0,"GoToHeaven","yes",1); \$say(Wolf,Wolf_Revenge);\$hide(Wolf);
scene_29	\$moveto(Wolf,\$getPosX(MotherPig),\$getPosY(MotherPig)); \$hide(MotherPig); \$anim(Wolf,"EatPig"); \$anim(Wolf,"Full"); \$moveto(Wolf,\$getScreenX(), \$getPosY(Wolf)); \$hide(Wolf);
scene_31	\$anim(PigA,0,"Happy","yes",2); \$anim(PigB,0,"Happy","yes",2); \$anim(PigC,0,"Happy","yes",2);

The first ending is played when the wolf was cooked by the pigs and he vanishes from the pigs' world. On the other hand, if the wolf managed to eat all of the pigs, he will proceed to eat the defenseless mother pig and leave the stage. Either of these scenes ends the story.

The story's conclusions are rather simple for this case study, but the interested reader can expand the story by introducing more story elements into the story templates. Since these story elements span the story space for the GA to perform stochastic search, more variations in story elements can introduce very complex story behavior on behalf of the agents and significantly affect story flow.

With the story templates and elements defined, we can start putting them together with a story thread. A story thread defines the general structure of the story, i.e. how the GA should evaluate the story elements based on their occurrences in the story and associ-

ate sets of templates with different stages of the story. We will start by providing an introduction to the story:

```
<timestep order="1" name="introduction">
  <set type="introduction">
    <element name="act_10" addfitness="1"/>
  </set>
</timestep>
```

Figure 28. Story's introduction

Figure 28 indicates to HEFTI that it should start the story by choosing story templates from the set introduction. Positive values in the add fitness attribute boosts the fitness of a chromosome that references this story element, and vice versa. In listing 1, preference is given to act_10 arbitrarily. We will then associate the template set wolf terror with the story using similar XML tags:

```
<timestep order="2" name="wolf terror" condition="{ag_2.isEaten} == false OR
  {ag_3.isEaten} == false OR {ag_4.isEaten} == false">
  <set type="wolf terror"/>
</timestep>
```

Figure 29. Wolf terror

In Figure 29, conditions are used to test if the template set from this story will be executed. If the conditions are not met, HEFTI will skip this template set and execute the next template set. During this time step, we are evaluating conditions to see if the pigs are all eaten by the wolf before generating subsequent story components based on the given story template wolf terror, since if there are no pigs, no agents are there to be afraid of the wolf.

```

<timestep order="3" name="wolf attack" condition="{isPigStillAvailable.value}
  == true" ><set type="wolf attack"/>
</timestep>

```

Figure 30. Wolf attack

Similar to the previous story time step, we must check the value of the global variable *isPigStillAvailable* in Figure 30 to see if the wolf should start his attack or just conclude the story because there are no more pigs for the wolf to attack.

```

<timestep order="4" name="wolf's plan" loop="{isPigStillAvailable.value} ==
  true AND {isWolfDead.value} == false">
  <set type="wolf's plan">
    <element name="act_13" addfitness="2"/>
  </set>
</timestep>

```

Figure 31. Wolf's plan

The loop attribute of Figure 31 indicates to the engine that it should continue to generate story components for the reader until all of the pigs are eaten or the wolf is dead. Note that we have boosted the fitness of story element *act_13*, which is chosen arbitrarily in this example.

```

<timestep order="5" name="Pigsconclusion" condition="{isWolfDead.value} ==
  true AND {isPigStillAvailable.value} == true">
  <set type="Pigsconclusion" />
</timestep>

```

Figure 32. Pigsconclusion

```

<timestep order="6" name="Wolfsconclusion" condition="{isPigStillAvail-
    able.value} == false AND {isWolfDead.value} == false">
  <set type="Wolfsconclusion"/>
</timestep>

```

Figure 33. Wolfsconclusion

Lastly, Figure 32 and 33 are the two different conclusions discussed earlier. Both of these test for the value of the global variables *isPigStillAvailable* and *isWolfDead* to determine the appropriate template set to choose from.

5.2 Editing the Story Elements

The IAE provides two alternatives for authoring stories: tree views and drag-n-drop windows. Since story elements are organized as a hierarchical and structured XML collection, categorized by time steps in the story, tree view controls can succinctly display the story elements in a very compact form via the metaphor of collapsible trees, sub-trees and leaves. Thus, XML elements embedded within a given XML tag can be attached to a sub-tree that is collapsible, whereas XML attributes corresponding to that XML tag are represented as leaves within the sub-tree. An example of a collapsible subtree with attributes attached is shown below (Figure 34):

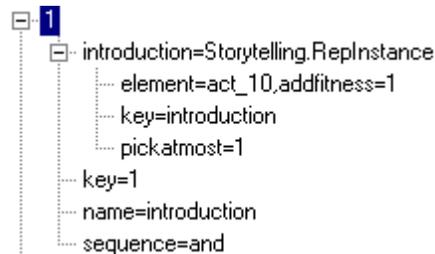


Figure 34. A collapsible sub-tree embedded within a template numbered 1

In Figure 34, the tree label at the first level corresponds to a particular time step in a story, whereas the sub-tree “*introduction=Storytelling.RepInstance*” names the set of templates that can be incorporated into the story at the given timestep. In the subtree, the definition of a single story element called *act_10* and its associated “*addfitness=1*” attribute indicates an author’s preference towards the existence of the *act_10* story element at this stage of the story. Tree views for the rest of the XML editors are simpler than what was described above (Figure 35 and Figure 36), so an author who understands the meanings behind the tree nodes in the previous example can browse around the interface easily. The storyline editor has perhaps the most complicated tree view because of the depth (four levels) of its XML document structure.

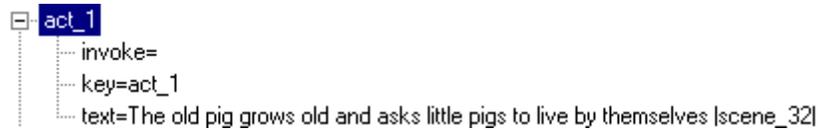


Figure 35. Tree view of the action editor

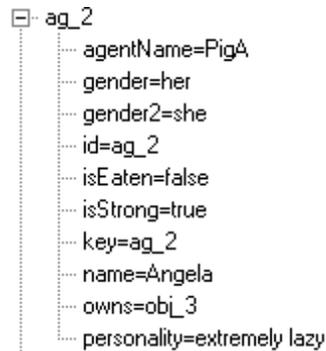


Figure 36. Tree view of the agent editor

To author story elements, authors use context menus to manipulate tree nodes in the editor windows. By right clicking on a tree node, the user can manipulate (add, delete or edit) the tree nodes and the tree leaves (Figure 37 and Figure 38).

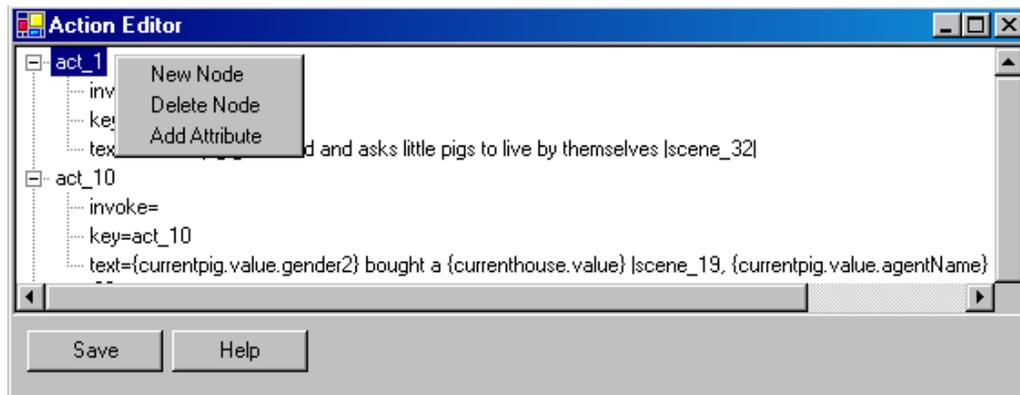


Figure 37. Manipulating a top level tree node

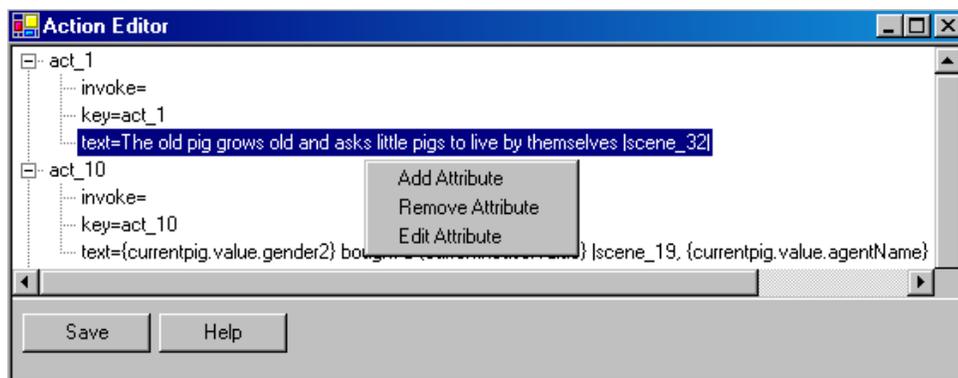


Figure 38. Manipulating a tree leaf

Additional editing support is provided to authors when they are editing certain types of story elements. For example, the menu item Edit Attribute in figure 38's context menu displays a notepad like editor for authors to edit the attribute. The get listing toolbar button supplies authors with a list of story elements that are found in the XML knowledge base when the environment is first initialized. Authors can double click on any of the list items to place that item into the editor's text area, thus referencing it within the story component (Figure 39).

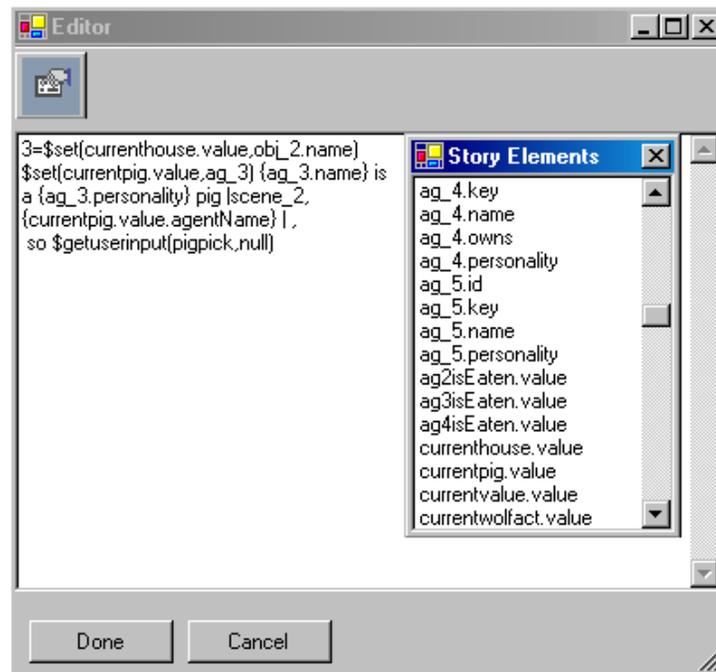


Figure 39. Attribute editor and the list of story elements

Tree view controls are acceptable in viewing the story elements as a list with collapsible tree nodes. However, tree view controls are mainly text based and visualizing story elements in tree views relies mainly on authors' understanding of the story. Therefore, a drag and drop interface is provided for story authoring as well so that authors can quickly glance at the story elements used in a story.

5.3 Authoring Stories with the Drag-and-Drop Interface

The drag and drop interface allows authors to quickly change the arrangement of the story elements, add or remove a story element from an existing story template and add/remove/edit attributes in the story templates (as seen in the Listbox control in Figure 40). The panel to the left of the form lists story elements that can be referenced in a story element/template so that authors can add a new story element by highlighting and dragging one of these listed elements to the viewing window and dropping that object near the indicated location (Figure 41):

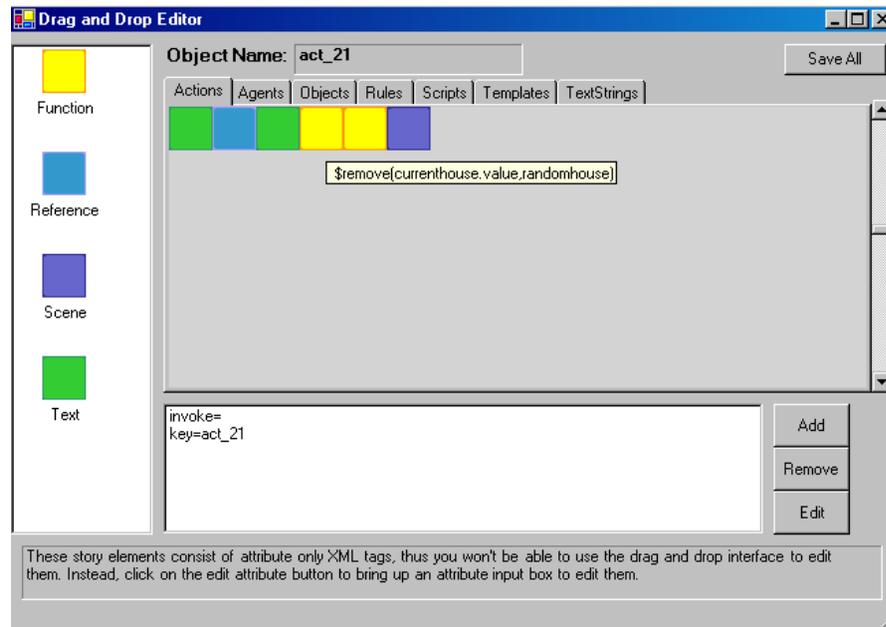


Figure 40. Visualizing a story component as a collection of story elements

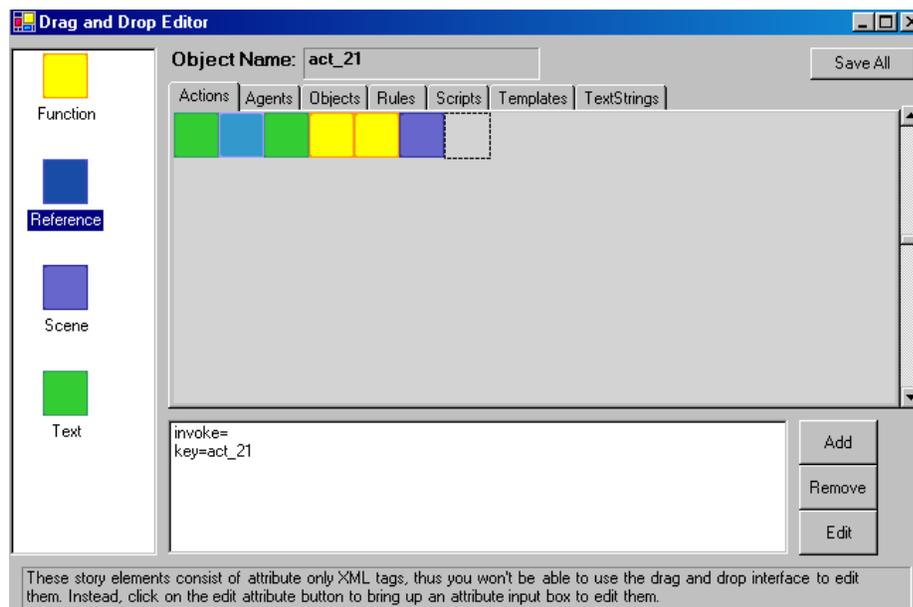


Figure 41. Placing a story element at an indicated location

To rearrange or delete a story element from the story component, authors can right click on the story element and select the “Delete Object” menu item from the context menu (Figure 42):

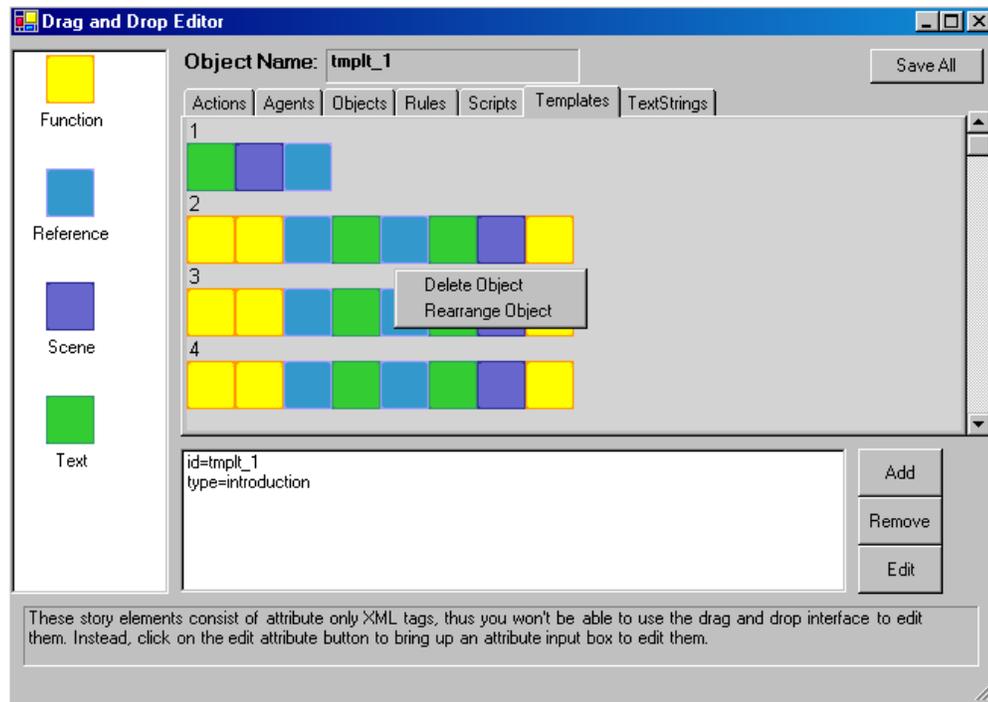


Figure 42. Deleting and rearranging a story element in the drag and drop interface

Adding/Removing/Editing an attribute in the drag and drop interface is simpler than the tree view controls because authors only have to select an attribute in the list, click on the Edit button and input the values they want to change for that attribute (Figure 43):

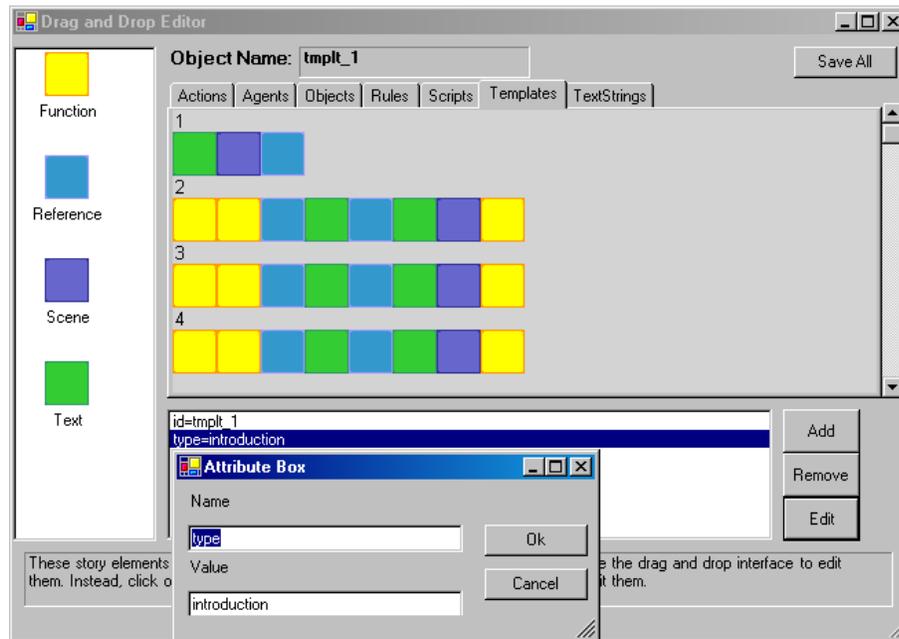


Figure 43. Attribute box that allows authors to edit a story component attribute

6. STORY STATES AND BRANCHES IN A DYNAMIC STORY

Figure 44 illustrates a small subset of the many variants of the three little pigs' story that can be constructed out of the story elements mentioned in the previous sections. As shown in the figure, the story is populated with conditional statements, loops and story elements. As the story progresses, rules jointly determine how links between story components (collection of story elements pertaining to a scene) are removed or added. These operations are dependent on the state of the story, story variables and agent attributes such that new story elements can be appended or removed from the story components as necessary. Two types of links are used in the figure: solid arrows depict ordinary links, and perforated arrows indicate conditional links that can be removed or created dynamically via story rules as the story progresses.

Figure 45 - Figure 50 illustrates an example of how a particular variant of the three little pigs' story is told. In summary, the HEFTI storytelling framework divides a story into a number of time steps that can be described by a set of story templates. The story templates combine various story elements and rules described in the knowledge base and describe how story elements can be combined by the GA. The end result is a dynamic story that is different each time it is read. After studying the variants of the story in Figure 44, we can now look at a particular run of the story with the screenshots below (Figure 45 to 50).

This version of the three little pigs' story starts off by providing background information about the story and how the pigs enjoyed their life before the wolf showed up in their neighborhood (Figure 45).

Life was fun and relaxed for the little pigs until one day the mother pig grow old and asks the little pigs to start their own lives. Each of the pigs wonders off to build/buy/find their new house. While the pigs are celebrating in front of their new houses, the wolf shows up and the pigs immediately scramble to hide in their new houses (Figure 46).

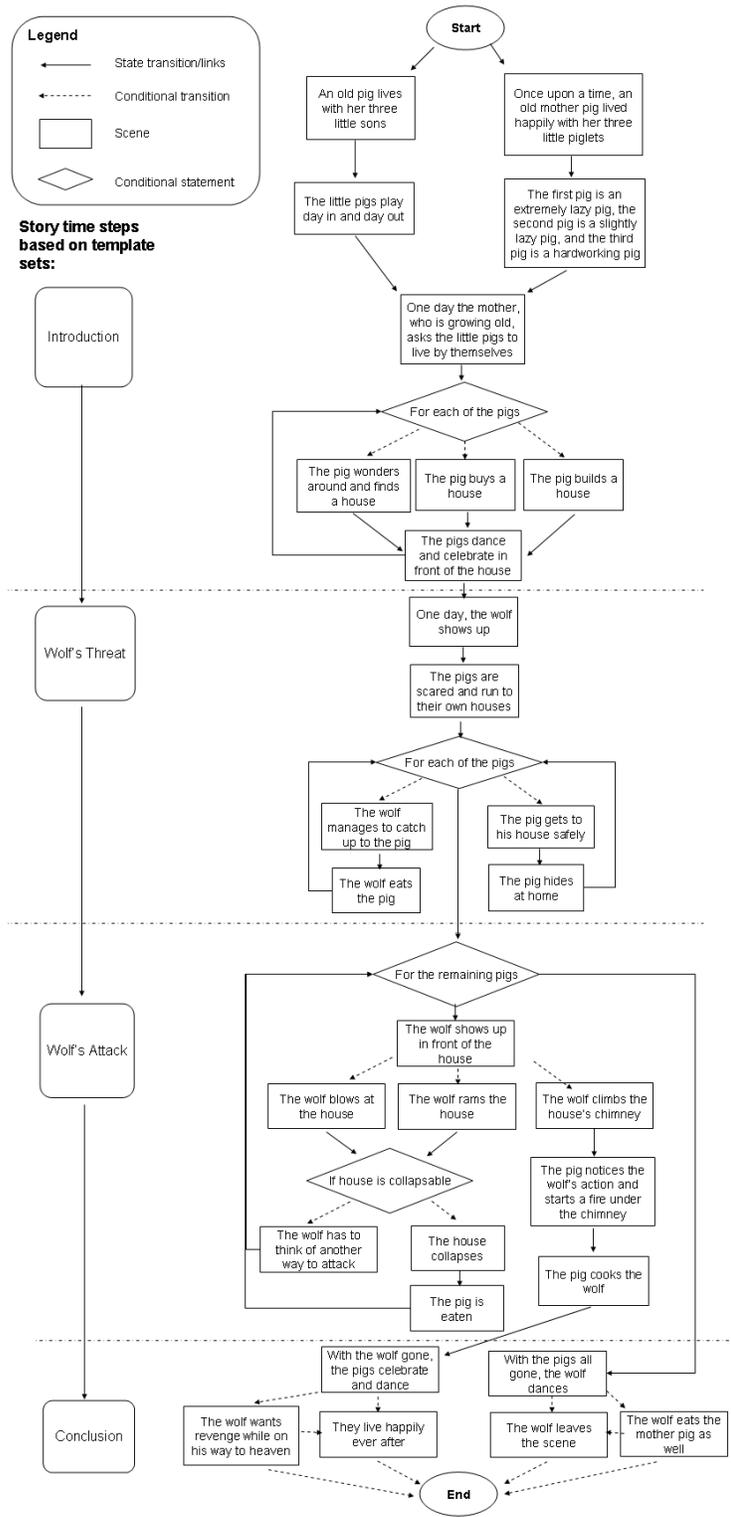


Figure 44. Several variants of the three little pigs' story

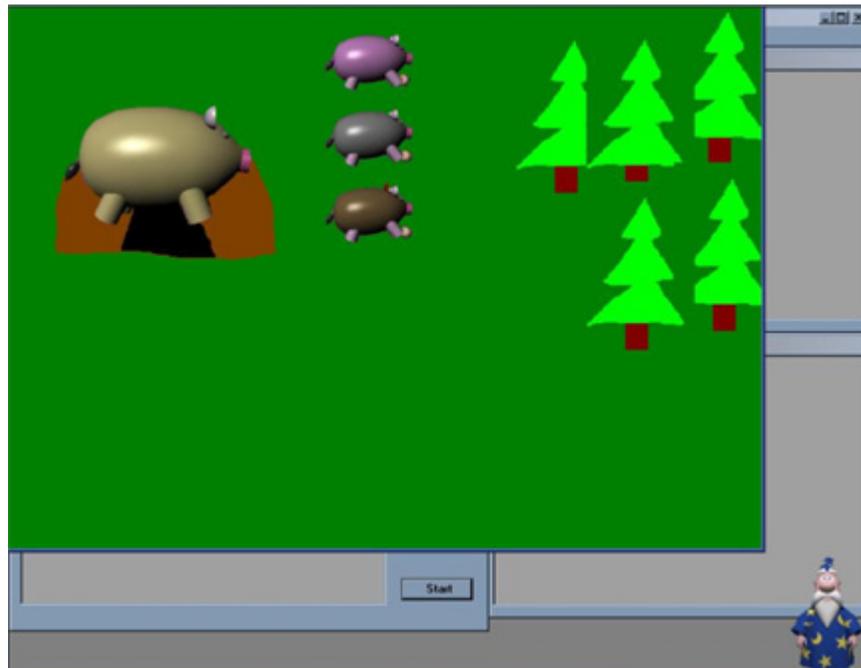


Figure 45. The pigs are having fun together with the mother pig

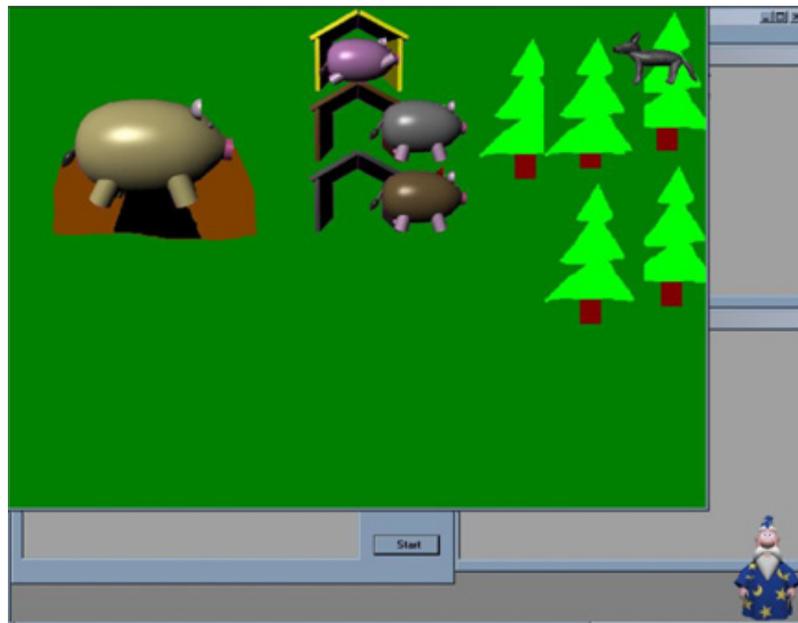


Figure 46. The wolf agent shows up in front of the pigs and the pigs scramble to safety

Seeing the pigs hiding in their houses, the wolf begins his attack. The wolf can try to blow the house away, ram down the house or climb the chimney. The actions chosen depend on the rules, the GA parameters, author's preferences to certain story elements, the story template set authored by the author and the random number generator. In this particular run of the story, the wolf agent chooses to attack the pink pig by trying to blow his house away.

Since the collapsible attribute of the straw house is set to true, the wolf can easily ram down or blow the straw house away (Figure 47). While the straw house collapses, the pink pig tries to run to the next remaining house. His rate of success is determined by a random number within a rule. However, in this example, the pink pig is unlucky because he is eaten by the wolf before he can get to the next house (Figure 48).

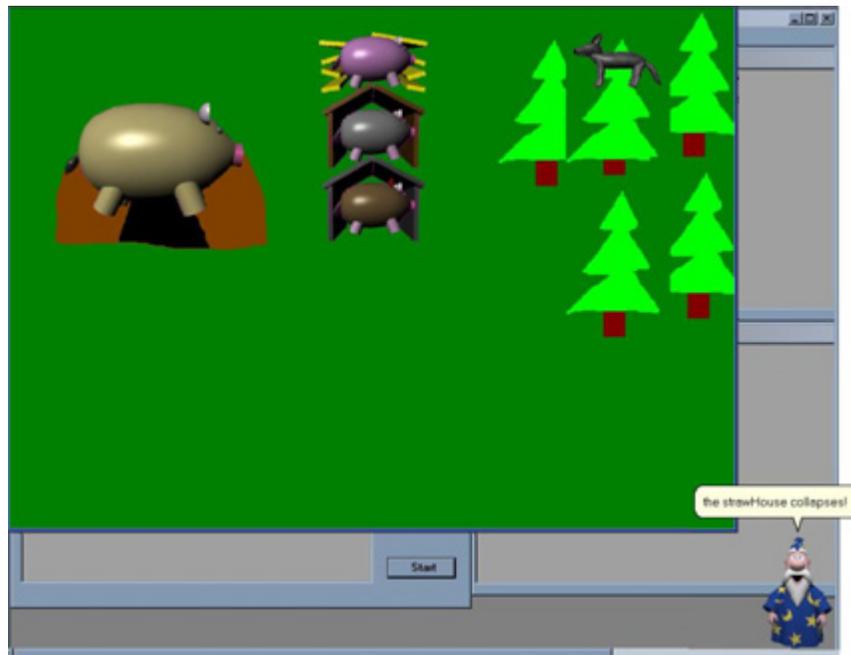


Figure 47. The straw house of the pink pig collapses!

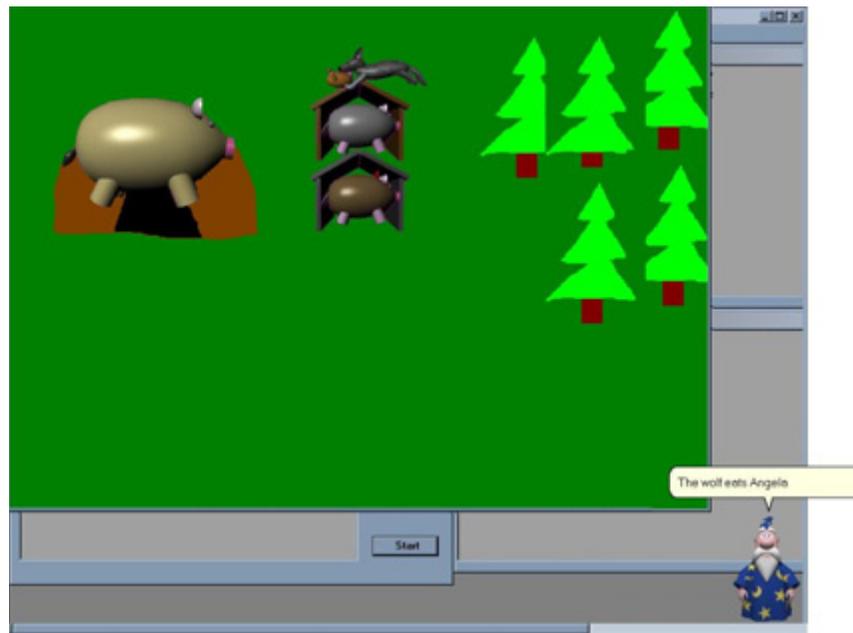


Figure 48. The pig failed to escape from the wolf and becomes wolf’s lunch

With two pigs remaining, the wolf next chooses to attack the gray pig by trying to climb the house’s chimney. Conforming to all of the variants of the three little pigs’ story, the wolf will learn his lesson whenever he chooses this method of attack. Therefore, the wolf in this story is cooked by the gray pig (Figure 49).

With the bad wolf gone, the pink pig comes back to life so they can live a happy life with their mother, and they celebrate day and night while the wolf curses them on his way to heaven (Figure 50).

While reading the story and listening to the narrative text spoken by the Merlin agent, several clips from the “Peter and the Wolf” symphonic poem composed by Rimsky Korsakov are played in the background. These visual and audio elements jointly buttress the story mood for the reader.

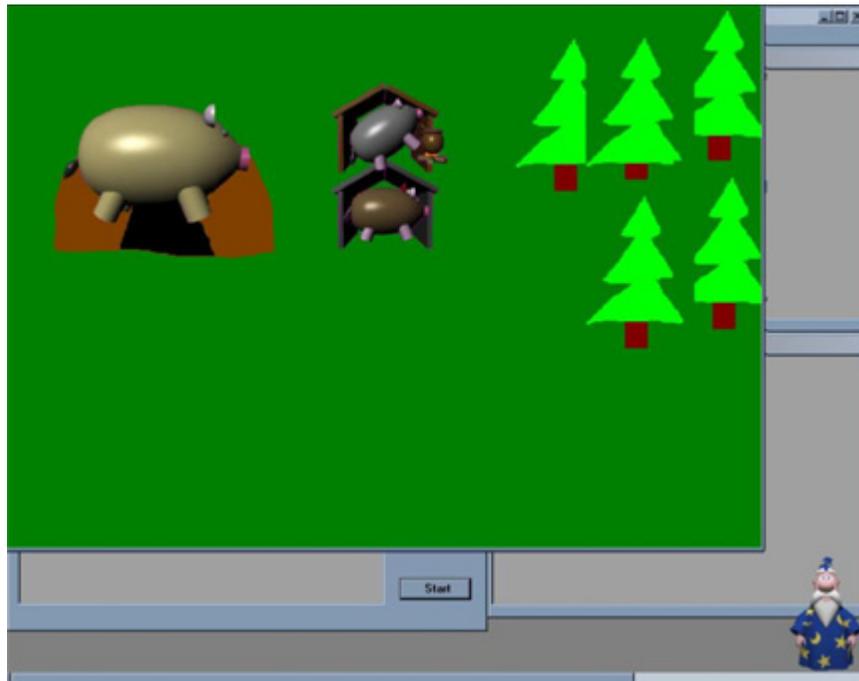


Figure 49. The gray pig cooks the wolf after he falls down the chimney of the wooden house

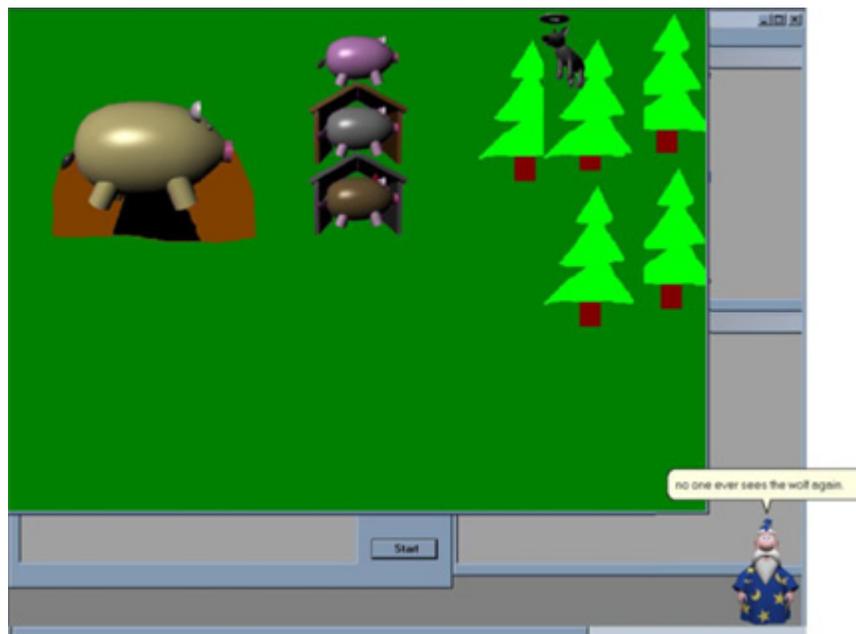


Figure 50. The pigs celebrate as the wolf rises to the heavens

7. AUTHORIZING A TEXT BASED STORY

The authoring process described in the previous sections requires authors: to be able to word the various story events appropriately, to have skills in generating 3D rendering of agent characters for their stories, and to have a certain degree of understanding of the scripting languages used by the IAE to control interaction of the agents. Since reusability of the agent characters and agent scripts specifically designed for a particular story is understandably lower than that of the story text for a different story, authors will most likely spend most of their time generating renderings of their agent characters and writing agent scripts for a new story. To save the authors time and effort, a purely text-based storytelling engine has been created based on the graphical version of IAE, so that authors can focus more on the story authoring activity rather than on some 3D modeling software.

This text-based storytelling engine adheres to the same architectural design as the graphical version of IAE with the exceptions that authors do not have to provide agent scripts and agent characters to the engine in order to tell a story. The text-based IAE maintains the same authoring elements as the graphical IAE. An overarching story thread still has to be created to control how the story templates can be used. The story templates, in turn, describe how various story elements, rules and agents can be combined over the course of the story. All in all, the graphical IAE serves as a “proof-of-concept” prototype that demonstrates how the agent characters, scripting elements and story components generated by the storytelling engine can be combined to tell a story. On the other hand, the text-based IAE focuses more on fulfilling our testing purposes and providing a better story authoring experience to the authors by providing additional facilities such as story locale setting, dynamic voice control of the narrator and the ability to view/change the state of the story variables at run time.

7.1 The Interface

The authoring interface of the text-based IAE combines and organizes story templates, elements and the story thread into a single layout called the Story View. This view is displayed in the middle panel that is populated with various colored rectangular boxes (or

enclosures) that represent the various story timesteps (Figure 51). Story timesteps are assigned their own colors so authors can quickly determine which timesteps contain particular story template. For instance, the Red rectangle in Figure 1 groups all of the story templates that belong to the timestep 0 together. The light gray story template rectangles within the timestep enclosures represent individual story templates with their names and locales written at the top, for instance `tmplt_1|USEnglish` and `tmplt_2|USEnglish`. From earlier descriptions, a story template is divided into various template sequences, these template sequences are represented by the slightly grayed out areas within a template and all of the story elements and rules are grouped as lists within these areas. From earlier descriptions (Section 4.4.8, `Templates.xml`), a story template is divided into various template sequences, these template sequences are represented by the slightly grayed out areas within a template and all of the story elements and rules are grouped as lists within these areas. For example, the list of story elements used in the first sequence of `tmplt_1|USEnglish` are represented by five colored squares (namely, brown, dark green, hot pink, light pink and white) within the template enclosure. To compensate for screen space and multiple template locales, the interface groups story template enclosures with different locales together by stacking them on top of the other, as shown by the stack with `tmplt_1|USEnglish` template at the top. The Story Timesteps panel to the right of the Authoring form provides an overview of the number of story timesteps a story has and their corresponding numbering in the story view.

Story elements and rules that are used in the stories are represented by treenodes that are grouped together based on their types. These are shown in the Story Elements tree-view located to the left of the Authoring form, so that authors can quickly inspect their attribute values or properties while authoring the story. Lastly, the Legend panel at the lower right corner of the Authoring form summarizes the color scheme used in depicting the story elements in the template sequences so that authors can quickly differentiate an agent object from a function call immediately in story view.

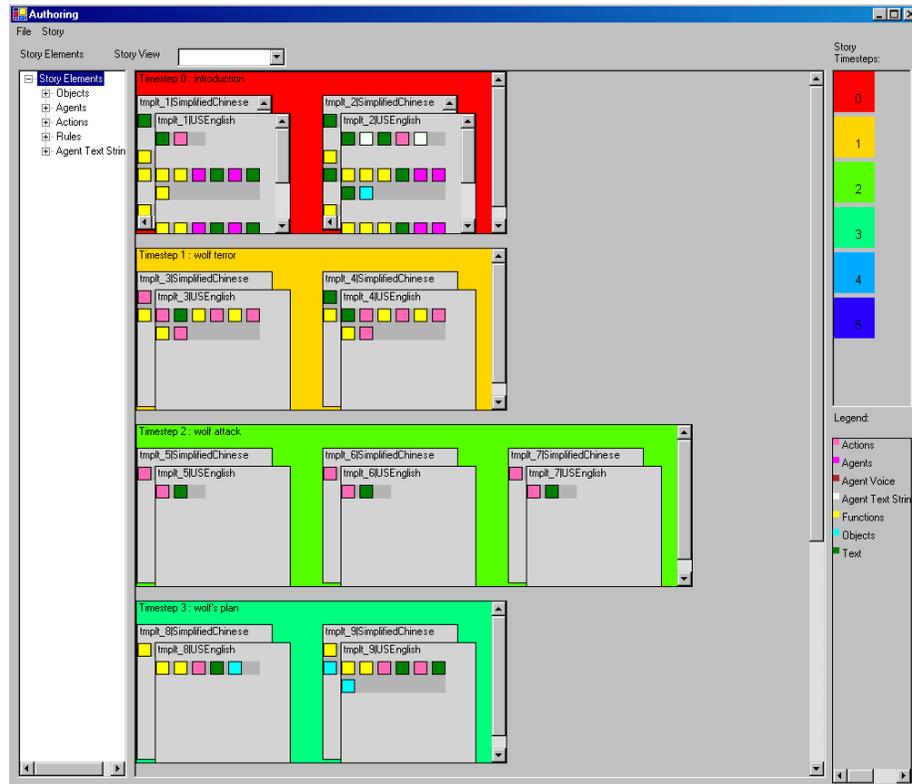


Figure 51. Authoring interface of the text-based IAE

7.1.1 Authoring a Story

All of the editing, adding and removing of story elements and story templates can be done with the context menus in story view, with the exception of creating new story elements which can be accomplished via the story elements treeview. Right clicking on different enclosures displays their corresponding context menu that provides various means for authors to modify the story. Figure 52 to Figure 54 depict the corresponding actions an author can perform on different enclosures. By right clicking on the timestep enclosure (Figure 52), an author can add, remove or edit a story timestep and add a new template into the story timestep. Similarly, right clicking on any story template brings up the template context menu (Figure 53) that allows the addition, editing or removal of story templates and cloning of the selected template with a different locale. The cloning process preserves all of the attributes and properties of the selected templates with the only excep-

tion that the template now exhibits a different locale. Lastly, the story element context menu (Figure 54) allows authors to add, remove or edit story elements that are referenced in any template sequence. Whenever the author chooses to edit the attributes or properties of a story timestep/template/element, they will be prompted by dialog boxes that are similar to Figure 55.



Figure 52. Timestep menu

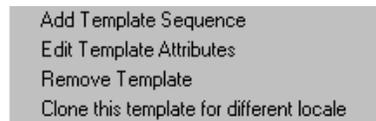


Figure 53. Template menu

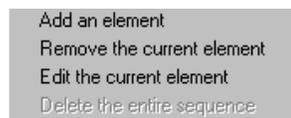


Figure 54. Element menu

This form has several text areas at the top half of the form that corresponds to different locales that are defined for this particular story timestep, template or element and the attribute list lists various attributes associated to the story element under a particular locale at the bottom left of the form. The three buttons to the right of the attribute lists allow users to Edit, Add or Remove attributes from the corresponding story timestep, template or element. Authors can click on the cancel button to undo the changes they have made to the story timestep, template or element, whereas the Done button saves the changes and brings them back to story view.

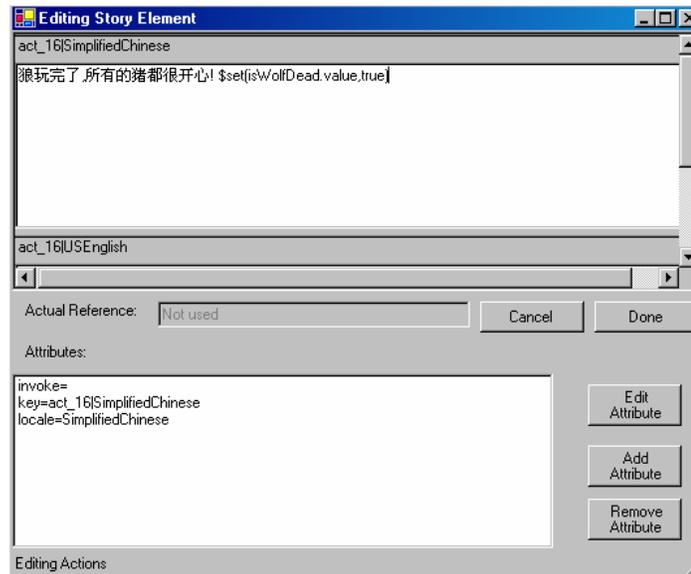


Figure 55. Editing a story element

7.1.2 *Story Element Distribution*

After the authoring process, authors might want to review their stories by running the story with different seeds repeatedly to either check for looping problems in the rules knowledge base or to acquire an overview of the likely distribution of the story elements and templates. The text-based IAE interface provides a statistical tool that generates information of interest to the authors. It automates most of the time-consuming process of choosing a story seed to start a different story and provides an overview of the likely distribution of an authored story (Figure 56). As shown in Figure 56 below, authors can specify the lower and upper bounds to which a random seed for a particular story is chosen, whereas the input box called Number of runs allows authors to generate multiple stories simultaneously using random seeds that fall within the range specified in the Seed Range field. For example, if the author inputs the value 2 in the Number of runs field and the Seed Range is set to be from 1 to 10,000, the tool is going to generate 2 different story instances, each of which is controlled by a random seed value that falls between the value 1 to 10,000. The Start button starts the generation process, the End and Exit button brings

the author back to the authoring interface and the View results button allows the author to visualize the results with various bar graphs and the affinity measure matrix.

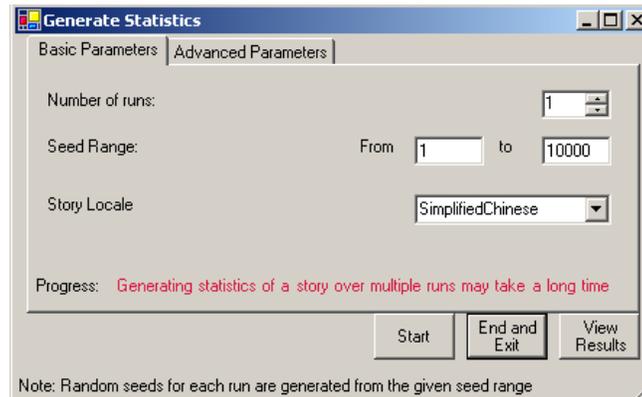


Figure 56. The statistical tool that complements the functionalities of the authoring interface

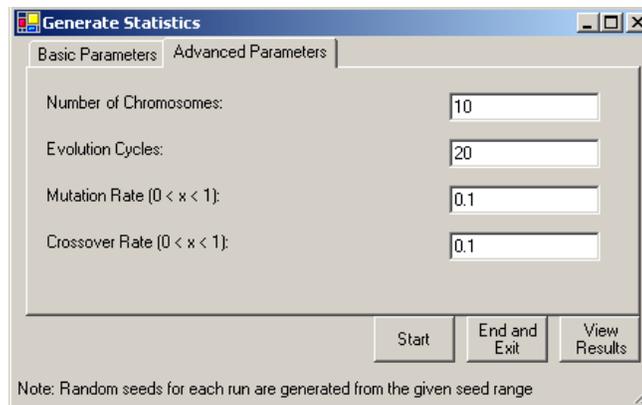


Figure 57. Advanced parameters for the generate statistics tool

Besides controlling how the random seeds are chosen for the stories, authors can also tweak the parameters of the storytelling engine to observe how these parameters influence the story generation process (Figure 57). The Number of chromosomes input box allows the author to control the number of individuals within a population, the Evolution

cycles input box refers to the parameter that determines how many times the selection process and genetic operators are applied to the chromosomes in a population, and the mutation and crossover ratios controls how likely it is that genetic materials are swapped between the genes and chromosomes.

7.1.3 *Comparing the Story Elements*

The story threads generated by the statistical tool (Section 7.1.2, Story Element Distribution) can be visualized in many different ways when the author clicks on the View Results button. All of the possible visualization interfaces are consolidated into the Story Distribution form (Figure 58). Figure 58 depicts the Ordered List view of the story distribution form that groups all of the unique story elements based on a set of categories that are shown in the legend panel located to the right of the form for each of the story timestep. The coloring scheme of the timestep enclosures follows that of the Authoring interface (Section 7.1, The Interface) so that authors can quickly determine the set of story templates that are used in these story threads according to the colors of the enclosures.

For instance, the first story thread appears to draw story elements from story templates defined in timesteps 0 to 5 (where it enters into a loop at timestep 4). On the other hand, the remaining story threads only briefly visited story templates defined in timesteps 0 to 5. This explains the obvious difference in length of these threads with the first story thread, and one should expect to see a lot of difference in the story elements the story components use.

The next tab page in the Story Distribution form is the Affinity List view. It provides a detail listing of the story elements used in the story timesteps. The timestep enclosures are color coded similarly to the timestep enclosures in the Ordered List view, but the enclosures are elongated to accommodate all of the story elements (in order) used in the corresponding timestep (Figure 59).

The Affinity List view provides visualization of the story elements at a finer granularity than the Ordered List view, for as depicted in Figure 59, it can reveal differences in

story elements that would otherwise seem similar in the Ordered List view. For instance, the last two story threads shown in Figure 58 are similar from a higher level view when their respective story elements are grouped together based on category and uniqueness. However, the affinity view in Figure 59 reveals that there are many differences between these story threads when the story elements that are used to construct the story components are laid out side by side in lists. Different coloring of the story elements further aids authors in the process of identifying even minute differences in the story elements at a glance.

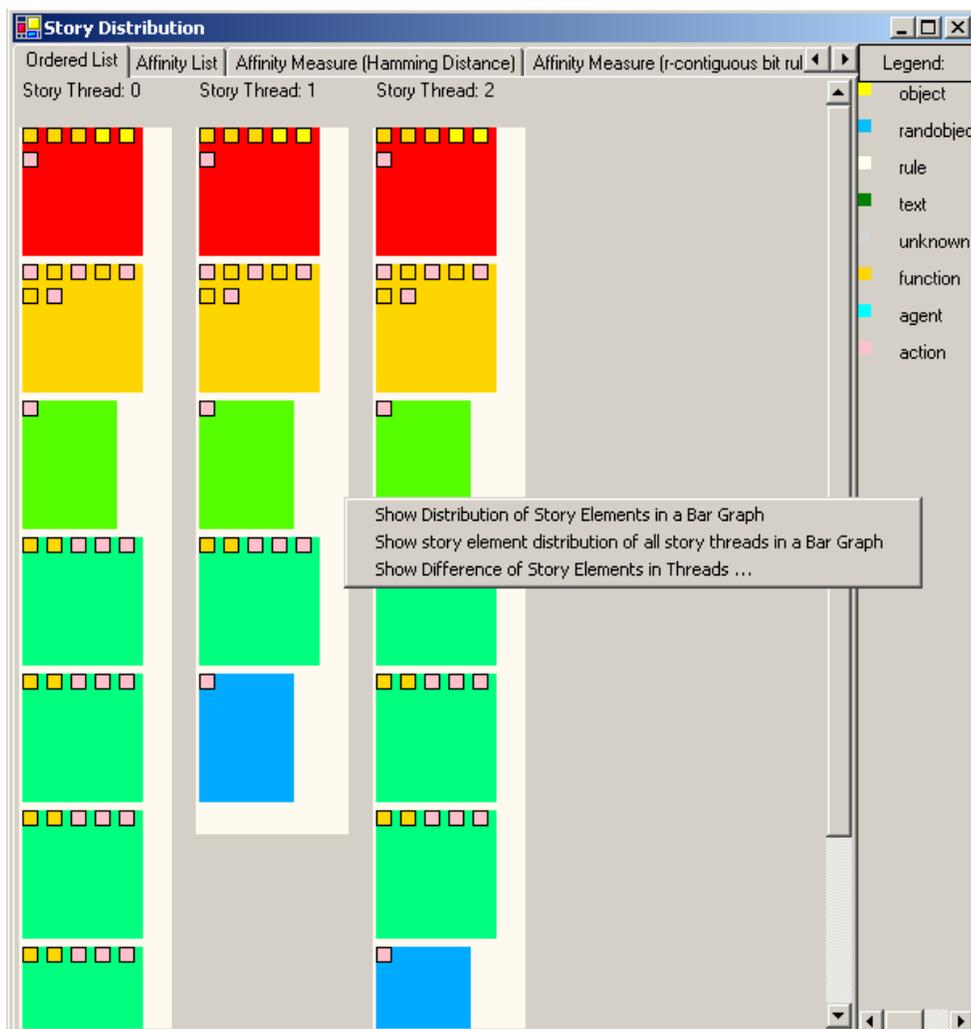


Figure 58. Story distribution form with its ordered list view

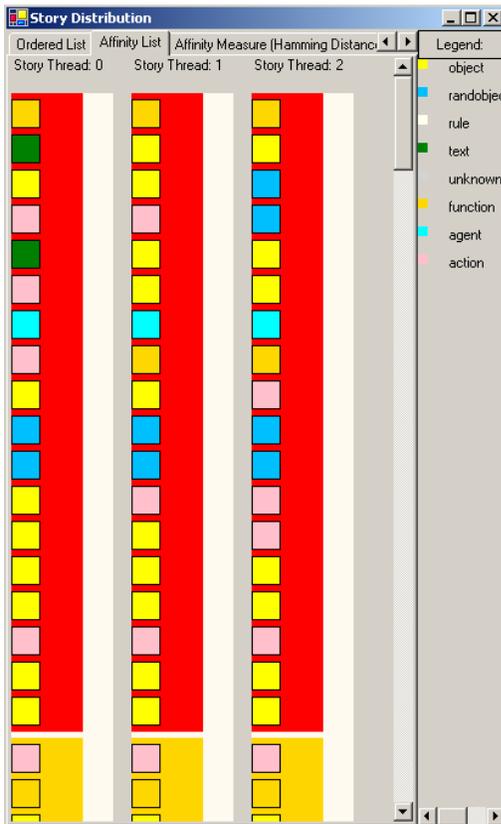


Figure 59. The story distribution form's affinity list visualization

7.1.4 Visualizing Distribution of Story Elements

The Ordered List and Affinity List views described in Section 7.1.3 allow authors to visually compare the temporal offset of the story elements in any story thread individually. An even higher level view of the distribution of these story elements is presented by bar graphs (Figure 60 and Figure 61) and a list view (Figure 62) that summarize these distributions in a concise manner. There are four ways that an author can collect such information:

1. Distribution of story elements for a particular timestep
2. Distribution of story elements for a particular story thread
3. Distribution of story elements for all of the story threads
4. Show difference of story elements between two story threads

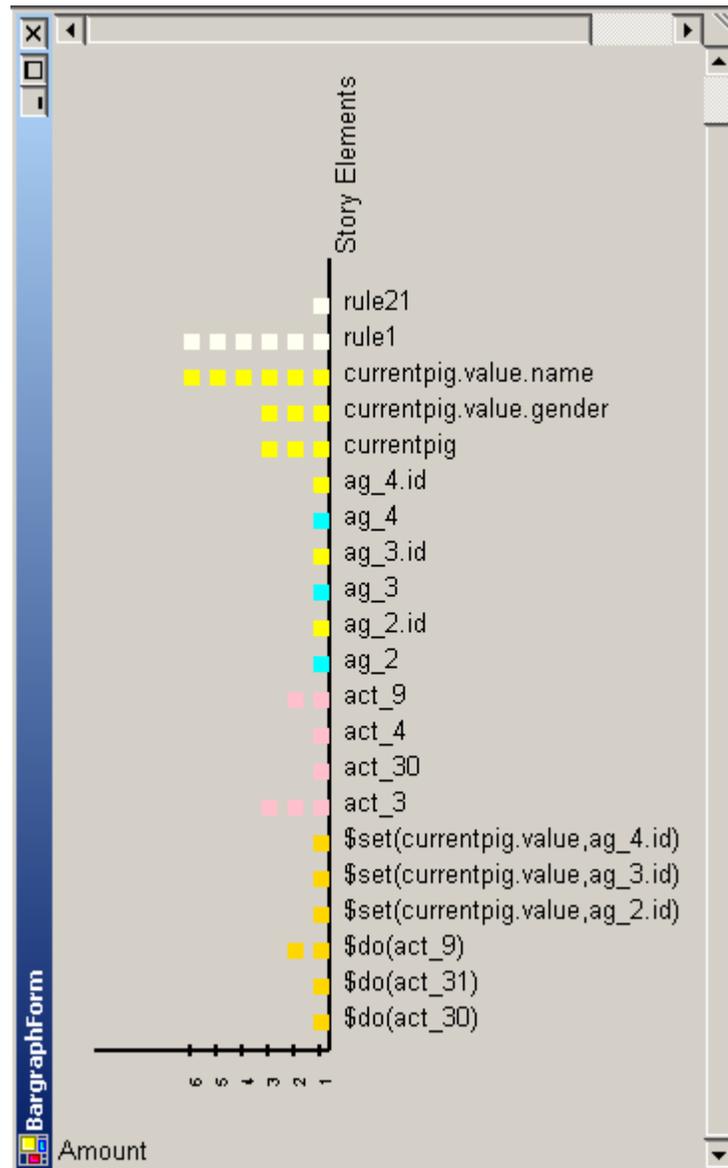


Figure 60. Story elements distribution bar graph

Displaying distribution of story elements at various story levels are highly similar, with the exception of the fourth option which displays the difference of story elements between two story threads. The operation performed by this option is the same as the logical set $A - \text{set } B$ operation that proceeds by eliminating story elements that exists in both

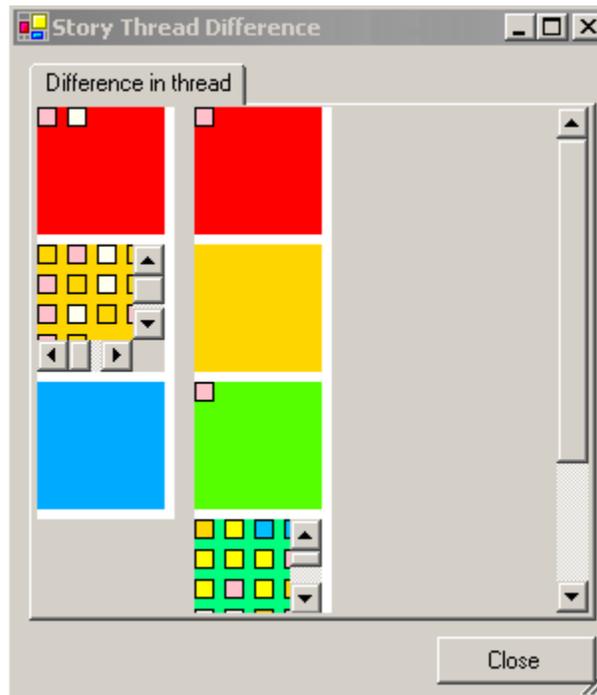


Figure 62. The story thread differences dialog box

Besides obvious information such as this, one can draw further observations about the design of the story templates set, such as insufficient story templates for a particular story timestep, if many story threads appear to generate empty enclosures across one or more timesteps. Authors can generate these visualizations by right clicking on the enclosures in the story distribution form and selecting the desired visualization.

7.1.5 Affinity Measure of Story Elements and Story Threads

Affinity measures are commonly used to measure the likeness based on relationship or causal connection between various objects, groups and sets. The differences and similarities of story threads can also be measured by affinity. A given story thread might share similar subsequences of story elements as another story thread and by measuring their affinity one can tell quickly how similar or different they are in terms of the story they tell. If a story thread “reuses” a lot of story element subsequences from another story

thread, their affinity measure should reflect this fact by a lower value in the result and vice versa. There are many different ways to compute affinity measures [33]:

1. Real-valued shape-space: the attribute strings are real-valued vectors
2. Integer shape-space: composed of attribute strings built out of a finite alphabet of length k
3. Hamming shape-space: composed of attribute strings built out of a finite alphabet of k
4. Symbolic shape-space: usually composed of different types of attribute strings where at least one of them is symbolic, such as a “name”, a “color”

Although individual genes in HEFTI are encoded as floating point numbers, individual genes can be further divided into integers that encode the corresponding offsets of a story element in the story element list pertaining to the story context. The encoding scheme makes affinity measures that are suitable for real-valued shape-space less accurate because the measure will be measuring the space represented by the genes without taking into account of the story elements therein.

The integer shape-space related affinity measures would be suitable for HEFTI if the offsets of the story elements in a gene actually convey certain information in their encoding, such as physical distance of the agents or the importance of the story branch. However, in reality, the offsets used in the encoding of the genes are arbitrary in HEFTI and they serve merely as an offset into a list of valid story elements during the story generation phase. Therefore affinity measures that are suitable for the integer shape-space will yield inaccurate information.

The symbolic shape-space related affinity measure assumes that at least one of the attribute strings (story elements in this case) in the story component is symbolic. Although one could assume that the story elements are just symbolic representations of characters, story events or scenes, decoding their exact meanings for the computation of such affinity measure requires more input and effort from the authors than necessary.

Lastly, affinity measures in the hamming shape-space allow one to evaluate the interaction between a particular subsequence with others based on more robust criteria. For example, they can be used to evaluate the hamming distance or binary matches of

attribute strings in chromosomes. The statistical tool of HEFTI generates affinity measures based on the hamming distance affinity measure and the multiple contiguous bit affinity measure.

7.1.6 Hamming Distance Affinity Measure

The hamming distance between a subsequence and another subsequence can be computed by applying the exclusive-or operator (XOR) between these subsequences. This measure has the property that if the subsequences are generated randomly the expected affinity is equal to half of their length (assuming that they are of same length). The binary matching approach is used in the statistical tool of HEFTI to measure the similarity of a given story thread with others by constructing the binary hamming shape-space based on the following conditions:

1. Nothing is added to the end result if a subsequence of story elements with length $l = x = m$ (where m is the length of the list of story elements in a particular story thread/timestep) matches another subsequence in story thread B
2. A value 1 is added to the end result if no match is found between a given subsequence in story thread A with story thread B

The conditions listed above are applied to all possible subsequences of story thread A at every possible length and to every possible subsequence of story thread B with the corresponding length. Since the measure would consider every possible alignment of the subsequences, this means that the offset in which a story element is picked wraps around to the first element in the list when we are constructing a subsequence of length greater than 1, therefore the set of subsequences of length 2 and 3 for story thread A consist of: $\{\{act1,agt2\}, \{agt2,obj1\}, \{obj1,act3\}, \{act3, act1\}\}$ and $\{\{act1,agt2,obj1\}, \{agt2,obj1,act3\}, \{obj1,act3,act1\}, \{act3,act1,agt2\}\}$. This example also conveys to us the complexity of the affinity measure and will be used in section 9 to compute its run time complexity. Figure 63 succinctly summarizes this process.

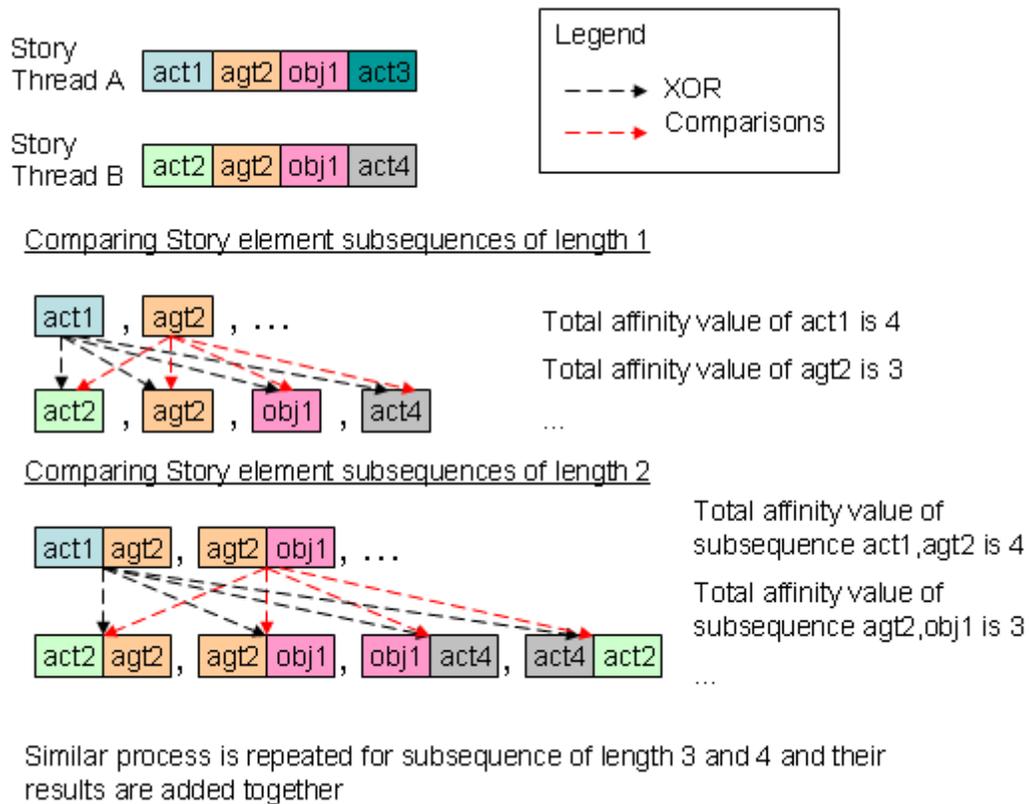


Figure 63. Process of computing the affinity measure for subsequence of length 1 and 2 between two story threads

Besides presenting the affinity measure between story threads, the hamming distance affinity measure is further computed for each of the story timesteps so that authors can determine the similarity of the story components generated from story templates in any story timesteps. They can also look at the overall affinity measure between two story threads to derive an overview of their similarities.

7.1.7 Multiple Contiguous Bit Affinity Measure

Shape-spaces that measure the number of multiple contiguous complementary symbols use the multiple contiguous bit rule [33]. The rationale of employing this measure to story components in HEFTI is that extensive complementary regions might be interesting for the detection of similar characteristics in symmetric portions of the story components, and can be useful in performing specific tasks, such as pattern recognition. This measure was proposed by Hunt and his collaborators [45] and is given by:

$$D = D_H + \sum_i 2^{L_i}$$

where D_H is the total hamming distance computed using the rules described in Section 7.1.6 and L_i is the length of each complementary region i with 2 or more consecutive complementary bits, as illustrated in Figure 64.

Several processes depicted in Figure 64 require further elaborations because their meanings might not be immediately obvious to the reader:

1. Since a story thread is divided into multiple timesteps this measure will be computed for each of the timesteps independently. In cases where the story threads do not share similar timesteps, -1 is returned from the measure to indicate such irrelevancy.
2. All of the possible alignments of the story elements in a given list will be used due to the fact that story components are constructed from lists of story elements via the combination mechanism in GA. So, the list offset for a particular subsequence in story thread A that matches another subsequence in story thread B can be different. Therefore, straightforward computing of multiple-contiguous bit affinity measure will yield less accurate results than one that tries all possible alignments of the subsequences.
3. This measure will be used mainly to assess how different the story components are across different story threads. Therefore, the higher the value the bigger the story thread differences are.

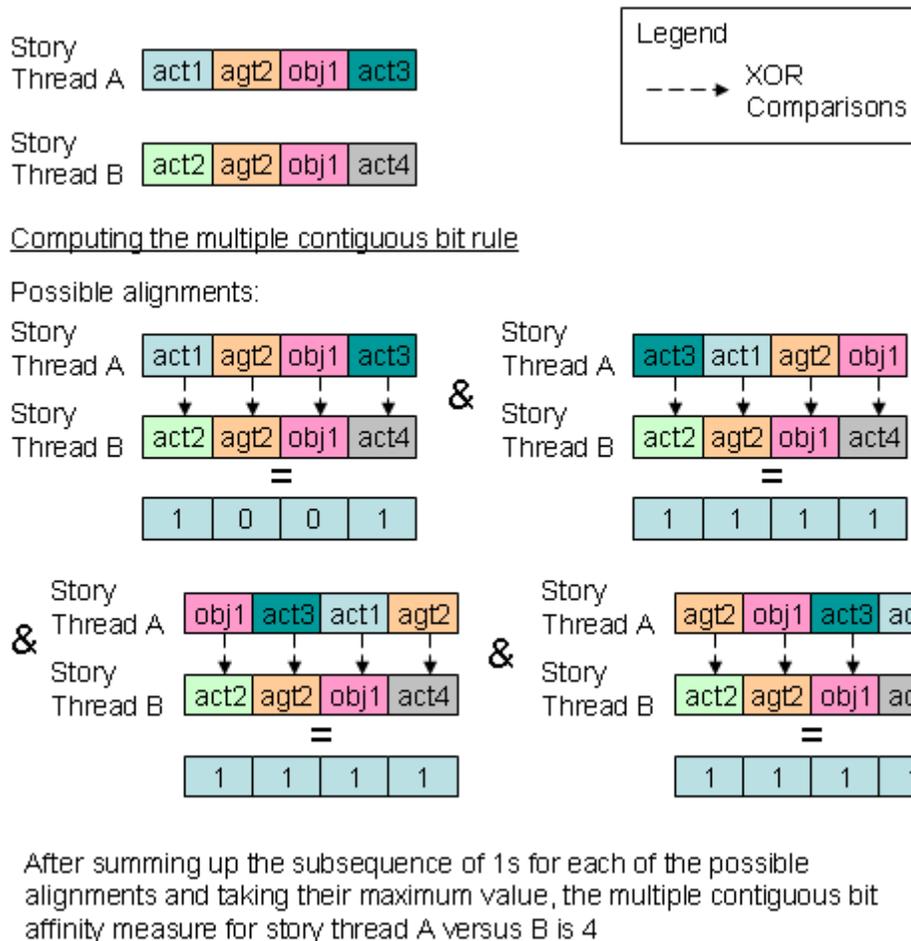


Figure 64. Process of computing the multiple contiguous bit affinity measure for story threads A and B

7.1.8 Run-time Complexity

The run-time complexity of the affinity measures are given here to illustrate the actual computations and comparisons done by the affinity measures discussed in Section 7.1.6 and Section 7.1.7. Understanding the process of computing the affinity measures will also help the reader in interpreting the evaluation results which are presented in Section 8. Before deriving the run-time complexity of the affinity measures, we note that the affinity measures are essentially comparing sublists of every possible length between two lists, so

the number of comparisons should be what we are interested in. With this in mind, we will try to generalize the operations performed by both of the measures to a list of n elements, such that given two lists A and B with 5 elements:

List A: {a, b, c, d, e}

List B: {f, g, h, i, j}

Both of the hamming distance and multiple contiguous bit affinity measures are going to compare sublists that hold list elements with length from 1 to 5, in other words, for list A these sublists are:

{a}, {b}, {c}, {d}, {e},

{a, b}, {b, c}, {c, d}, {d, e}, {e, a},

{a, b, c}, {b, c, d}, {c, d, e}, {d, e, a}, {e, a, b},

{a, b, c, d}, {b, c, d, e}, {c, d, e, f}, {d, e, a, b}, {e, a, b, c} and

{a, b, c, d, e}, {b, c, d, e, a}, {c, d, e, a, b}, {d, e, a, b, c}, {e, a, b, c, d}

The sublists are not actually generated by the program to compute the affinity measures, they are listed here merely for clarity. Similarly, the sublists for list B can be constructed by the same process.

At this point in time, we note that the number of elements in each of the sublists are n . When we are computing the hamming distance affinity measure for sublist of length of 1, each of the list elements are compared with list elements of corresponding size from list B. For instance, the sublist element {a} is compared to: {f}, {g}, {h}, {i} and {j} from List B, {b} is compared to: {f}, {g}, {h}, {i} and {j}, etc. The number of comparisons for each sublist element of length 1 is n (or 5 to be precise) and this process is repeated up to n times because the number of sublist elements is n as well.

Since we are not only comparing sublist elements of length 1 alone, we will generalize the results above by noting that the number of comparisons for the sublist elements from a sublist with length greater than 1 is proportional to m (where m represents the length of the sublist element). We will illustrate this by observing that the number of comparisons for each element from a sublist of length m that is compared to another sublist of equal length is m . In other words, we need at least 3 comparisons to determine the affinity

measure of {a, b, c} and {f, g, h} each with length 3. Since we cannot determine their affinity value without comparing each of the elements in the sublist and this operation is repeated up to n times to cover all of the sublist elements. We can generalize the complexity of the process that computes the affinity measure for a sublist of a particular length is $O(nm)$ or $O(n^2)$ to be lax. Lastly, this computation is performed for sublists of different lengths for n of them, therefore the overall complexity of the algorithm is $O(n^3)$. (The Big-O notation is just a rough approximation of the number of comparisons being made in the algorithm, one can compute the exact number of comparisons as $n \cdot \sum_i (n \cdot i)$ where i ranges from 1 to n).

Computation of the multiple-contiguous bits affinity measure can be optimized by noting that one can derive the measure directly from the process outlined above. Since a sublist of length m will yield a value of 1 if and only if each of the sublist elements matches the sublist elements with the corresponding offset from another sublist and comparisons are made across all sublists with length 1 to n , the sublist with the longest length i that yields results greater than or equal to one in the hamming distance would be the multiple-contiguous bits affinity measure for a list, since one can find no longer matches of contiguous elements that satisfy the definition of the multiple-contiguous bits affinity measure defined in Section 7.1.7. This observation allows us to compute the second affinity measure in constant time by relying on the results derived while computing the hamming distance affinity measure.

8. EVALUATION

8.1 Evaluation of the Interactive Authoring Environment (IAE)

The authoring tools in the Integrated Authoring Environment (IAE) are implemented to emulate certain aspects of a cyborg authoring environment of the future. We assume that future generations of human authors will be capable of simple programming and have software tools that assist or simplify authoring tasks. These tools will allow them to quickly visualize relationships between various modules while also providing means to quickly construct new modules for a particular story, similar to the authoring model of the IAE. As in most Integrated Authoring or Programming Environments (such as Frontpage, Dreamweaver, JBuilder and Visual Studio), the IAE is designed to shield the user from the complexity of the syntax and semantics involved in writing computer programs or authoring dynamic webpages, while also automating certain menial tasks (such as typing opening and closing XML tags in a particular story element file).

In order to evaluate how useful, effective and helpful the IAE might be to a cyborg author of the future, a usability study was designed and carried out at the Center for the Study of Digital Libraries (CSDL). The research protocol was approved by the Institutional Review Board (IRB) on April 3rd. Two test subjects, having the requisite skills, were selected from among the graduate students. Both were Ph.D. students and proficient in computer programming skills. Their involvement in various research projects in the center requires them to use various integrated development tools (such as JBuilder, Visual Studio and DreamWeaver) and command line compiling tools (such as Java Development Toolkit or JavaScript) to create applications and prototype systems. Most of the integrated development tools the test subjects have used tend to share similar features (for example, the development environment is usually an integration of programming, modeling and compiling tools) and the command line tools are generally invoked via text commands issued via a command console. These requisite experiences allow the subjects to compare and contrast features between the development tools they have used and the IAE. The test subjects were also able to provide a list of desirable or undesirable features of the IAE in

the questionnaires after completing the assigned tasks (as outlined in Section 8.1.2, Research Procedures). This study attempts to find answers to the following questions that are related to the utility of the IAE:

1. How similar is the IAE to the various integrated development tools and the command line compiling tools the test subjects have used? Why?
2. How much does the IAE help the test subjects in completing the assigned tasks?
3. What are the features most used by the test subjects and how can we improve upon them?
4. What are the features that hinder the test subjects' ability to properly carry out their authoring activities? Why?
5. What are the features missing from the IAE that may help or simplify the authoring activities?
6. How long does it take for the test subjects to get acquainted with the IAE and complete the tasks?

Having experience in the areas of user interface design, information management software and knowledge about 'modding' of game modules or Multi-User Dungeons (MUD), the two students are ideal test subjects for the IAE. In addition, they also demonstrated commitment to learn the authoring tools and syntax in the IAE. Section 8.1.1, Profile of the Test Subjects, provides details and experiences of the test subjects and how their comments and suggestions may shed light on the expectations and features of future cyborg authoring environments.

8.1.1 Profile of the Test Subjects

Test subject A has been working on projects in a software research lab for 2 years. A's responsibilities in these projects include creating new information visualization tools, web-based information retrieval applications and software that facilitates or supports the browsing of online artifacts for scholars. While conducting research, A has read extensively about how modules and story scripts of MUDs or Massively Multiplayer Online games (MMO) are 'modded' by the gaming community for personal interests or enhancements. These readings give A significant understanding of the authoring or programming features needed by the 'modders' and the processes involved in authoring new modules and scripts for the games. In addition, A's experience in customizing an information

workspace in the Visual Knowledge Builder (VKB) [69] allows A to appreciate and contrast the features of Integrated Environments. Lastly, A's research interests and experiences in the areas of hypertext and hypermedia, hypertextual novels, reading tools, and information management make A a good test subject for this study because A has a better understanding of the digital authoring process and how digital stories are authored. We believe that A will be able to provide valuable inputs to the questions that motivate this study.

Similarly, test subject B has been involved in the development of various information management and workspace projects for about 2 years. B has created various modules and add-ons for various software to allow better management and creation of personal collections in the information workspace. B has had extensive experience in creating information management tools, hypertextual collections, 'modding' applications and game engines. For example, in one of B's projects, B modified and scripted a text-based MUD engine for Linux in order to create an imaginary world. Users were able to login, explore the world, interact with the objects with their avatars, and work towards completion of the game's goal (for instance, one can fly an airplane only if one had collected a specific set of items from various non-player characters in the game). In addition, B's research interest also includes digital stories, entertainment and tools that can facilitate the reading activities of the human reader. For example, in another of B's projects, B created reviews of a book-chapter using the visual attributes, spatial layout and linking features provided by the VKB workspace. The resultant collection forms a hypertextual novel to which readers can visually interact with various sections of B's writing. B's experiences in these areas make B a good test subject for this study because B can easily compare features of the IAE with similar authoring tools and is able to provide valuable insights to the various potential authoring models supported by the IAE.

8.1.2 Research Procedures

Each test subject was asked to read and sign the informed consent form before the study. I then gave a tutorial and demonstration of the Integrated Authoring Environment

(IAE) to the test subjects to elaborate on the concepts described in the tutorial handout as well as the features of the IAE. The test subjects were able to ask questions and play with the authoring tools in the IAE to familiarize themselves with the environment during the tutorial and demonstration. After the test subjects were comfortable with the IAE, a pre-generated children's story (The Three Little Pigs) was given to the test subjects to analyze. After the subjects understood the structure of the given story, they were then asked to author three different variations into the story using the IAE. Proper understanding of the story elements, templates, rules and structure was needed in order for the test subjects to complete the assigned tasks. The steps involved are outlined in Appendix B, TASK SHEET FOR THE USABILITY STUDY. Upon completing the assigned tasks from Part I, they were asked to answer Part I of the questionnaire. Following this, they were asked to author a new short story (with variations) based on the Little Red Riding Hood story (Part II). After authoring the story, they were asked to answer Part II of the questionnaire. We collected: 1) the XML documents automatically generated by the IAE after the authoring process; 2) Questionnaires answered by the subjects; and, 3) my notes, taken during the study. A 5 minute break was scheduled at the end of each of the tasks and the test subjects could take as many breaks as they wanted while working on the tasks.

8.1.3 Observations and Analyses

Both test subjects (A and B) had no problem understanding the constructs of the story and the interfaces of the IAE covered during the tutorial and demonstration sessions. The time it took for the test subjects to complete the tasks in our usability studies are summarized in Table D.1 of Appendix D. During the sessions, one of the test subjects asked questions related to the creation of the Microsoft Agent Characters (Section 4.1, Agent Characters) as well as their possible uses in gaming environments. The test subject was disappointed to find out that although the agent characters are capable of interfacing directly to the various TTS engines (Section 4.2, Text-to-Speech (TTS) Engines) as well as performing various animations, their use in the gaming environments is rather limited because of the lack of proper synchronization and control Application Programming

Interfaces (APIs). This indicated limitation of the agent architecture used by this IAE prototype. A solution might use an implementation of an agent architecture that is written in DirectX. This would allow reuse of the story agent characters and facilitate integration of the storytelling engine within a windows gaming environment.

While working on Part I of the usability study, test subjects were at first confused by the Story View and Legend displays (Figure 51). Some of the questions asked by both test subjects were “Where do I create the random variable *randomHouse*?” and “Should I place *randomHouse* under the *Objects* or *Agents* category in the Treeview control?” Although tutorial and demonstration sessions attempted to thoroughly cover all aspects of the authoring process, users were still unclear about all of the functionality of the interfaces. However, the test subjects had no problem placing other story elements they created in subsequent tasks into the appropriate categories after the functions and roles of the various story elements were properly explained to them. All in all, the test subjects managed to complete almost all of the assigned tasks in Part I in a timely manner.

Part II of the usability study yielded more interesting results. The first test subject (A) had decided to break down the Little Red Riding Hood story into five story stages instead of the four story stages as described in the task sheet. A’s variant of the story consisted of the following stages:

1. Introduction
2. RedRidingHoodMeetsTheWolf
3. WolfVisitsGranny
4. RedRidingHoodVisitsGranny
5. Conclusion

In addition, A also adopted a top-down approach in authoring the story: 1) divided the story into five timesteps; 2) created five text only story templates in each timestep via copying and pasting the story text from the given Little Red Riding Hood story file; 3) tested the story to make sure that it ran properly; 4) made mental note of sections in the story where random variables and story elements may be introduced to generate the story variations; 5) implemented the appropriate story elements via the IAE; and, 6) repeated steps 3 to 5 until A was satisfied with the story. A finished authoring the complete Little

Red Riding Hood Story with 12 story elements and 2 random variables. However, more time was spent in authoring activities with the top-down approach because A had to repeat steps 3-5 several times.

In contrast, B adopted a somewhat bottom-up approach in authoring the story. B planned the story stages and marked in the Little Red Riding Hood story handout which sections of the story maybe modified with custom story elements and text. B also gave some thought to designing the story elements that may be used to introduce variations into the story before authoring the entire story. All in all, B authored 14 story elements, 3 random variables and a story rule to be invoked in a story template to introduce additional story text into the existing story with 30% probability.

8.1.4 Desirable Features

Both test subjects pointed out how the interface allowed them to expand, explore and visualize different parts and branches of a story. A mentioned that the story view (which was referred to as the Plot Display) tells the flow of the story, story branching points and the possible story variations (generated by story templates) in each timestep. In addition, A also pointed out that the display abstracted the programmatic nature (which A referred to as procedural connections) of dynamic stories. Both test subjects have the conception that authoring a dynamic story with the IAE is similar to authoring a homepage because the authoring process was similar to marking cards, filling out forms with preset items and moving items around. Although B also pointed out that the process of building the story elements is similar to programming because one has to populate random variables with list elements, coding the conditions and rules.

Both test subjects liked the visualization of the various story elements in the story view, as well as the breaking down of a single dynamic story into various hierarchies (such as, templates, timesteps, story elements and template sequences). A also pointed out how the authors have the ability to randomize almost all aspects of their stories since the authoring interface provides access to story object, action and template randomizations. Test subject B liked the convenience of accessing various story elements with a few

mouse clicks using the treeview control (which is located to the left of the story view panel). However, B was confused by the use of certain color codings in the Story View panel, and the lack of drag-and-drop support in Story View. For example, hot pink and magenta were used to denote action and agent objects respectively in Story View, but these colors are difficult to differentiate in certain situations. Lastly both test subjects indicated that they enjoyed authoring stories with the IAE and scored the IAE a 6 out of 7 when that question was asked in the questionnaire.

8.1.5 *Undesirable Features*

The test subjects mentioned several features that are undesirable or not supported by the IAE and storytelling engine. These features largely fall into the user interface and system categories. Some of the user interface issues that the test subjects encountered are as follows: 1) B mentioned that the creation process of story elements can be sometimes confusing, complicated or counterintuitive since the story element creation form does not provide enough information to the user. For example, while authoring the first story variation, test subject B failed to indicate to the IAE that *randomHouse* was a random variable. This crashed B's story when B was trying to execute the story. This mistake may be eliminated completely with better online documentation or explicit menu items (such as a "Create a random variable" context menu item) for the authors to create new story elements; 2) A was annoyed by the need to manually change the story seed in order to get a different variation of a story. Since the IAE prototype used in the study is designed for the authors to debug and test run their story, the assumption being made was that the authors will have to manually change the story seed to quickly make corrections to their stories so that they do not have to memorize the story seed that gave problems; and, 3) B pointed out that the authors are still required to remember some of the syntax to express conditions, functions and rule statements in the IAE. It would be very helpful if the IAE simplified the declaration of such story constructs with context menus or shortcut buttons similar to other Integrated Authoring or Programming Environments.

The system features or limitations the test subject noted are as follows: 1) there is no way to specify how the system can choose between two or more templates in the same timestep. This requirement seems to go against the original design of the IAE because only one story template is needed to generate the story text in a particular timestep based on a given story thread. Unless the author wishes to create a multi-threaded story, the IAE does not have to provide such functionality. In addition, the storytelling engine does not currently support such story form; 2) there is no support for branching within the template sequences since the IAE executes all of the template sequences before moving to the next timestep. This may be a good feature to add to the IAE although it would serve as a form of “syntactic sugar” since similar functionality is already provided in the story templates; and, 3) test subject A stated that there is no support for automatic back-stepping in time. Duplicating timesteps to mimic such functionality does not seem like a good solution. Therefore, A concluded that IAE cannot properly tell stories that have recurring episodes such as a detective story because such stories need frequent references to previous events. This comment indicates confusion on behalf of A’s understanding of the storytelling engine because the three little pigs’ story covered in the demonstration given before the study employed loop and conditions to control how the wolf plans and attacks the pigs until either party perishes in the process (Figure 27). Such confusion may result from the lack of an assigned task that asks the test subjects to author a looping (or never ending) story.

8.1.6 Future Improvements

The test subjects were also asked to provide a list of features that they think would allow the authors to better visualize and understand others’ interactive stories in the study. A history feature that logs the changes an author performed during a certain time such as the history toolbar in VKB was mentioned by test subject B. This feature would be a useful addition to the IAE because it would provide more contextual and temporal information to the original author as well as collaborating authors. It may also provide “Undo” and “Redo” functionalities to the IAE, so that authors can recover their work after com-

mitting certain mistakes. An annotation feature was also mentioned in the questionnaire to allow authors to comment about the use of particular story templates and their possible combinations. This feature is already supported by the XML attribute *comments* that can be incorporated into the story files, however, currently there is no visualization provided by the IAE to display the author's embedded comments. Lastly, one of the test subjects mentioned that it would be very helpful to link or highlight the appropriate story element in the Treeview whenever the user clicked on a particular story element displayed in the Story View since this feature allows the authors to quickly visualize and navigate between different story elements in a story.

8.1.7 Conclusions

It is clear from the discussion that many improvements can be made to the IAE to make it more user-friendly and better facilitate the authoring process of interactive, dynamic stories. Notably, features and tools that may simplify the tasks of declaring story elements, rules and conditions can help the authors greatly. Based on the inputs provided by the test subjects, we know that the IAE may help in abstracting the complexity involved in programming an interactive, dynamic story for most authors since the test subjects felt that the authoring process is similar to authoring a webpage where forms and web page elements are created and subsequently dropped into the appropriate containers. Although the study also reveals the many improvements that we can make to the authoring environment, such as full drag-and-drop, annotation and editing support throughout the Story View, as well as at the system level where generation of agent animations may be automated (a suggestion that was given by test subject B while viewing the demonstration) to reduce workload of the human authors, the design of the IAE is a step in the right direction to cyborg authoring in the future. The IAE the test subjects in authoring the dynamic stories without requiring them to edit awkward XML syntax and reduced the amount of learning required from the test subjects (as demonstrated by the time it took for them to finish the tutorial in Table D.1). With the incorporation of the features and tools

suggested by these test subjects, the authoring environment may get even closer to the cyborg authoring environment or tools used by authors of tomorrow.

8.2 Evaluation of the Storytelling Engine

In addition to conducting a usability study on the authoring interfaces, namely the IAE, we have also performed an empirical study on the behavior of the system. This portion of the study attempts to illustrate the behavior of the system with a set of controlled variables. Traditional stories narrated using contemporary paper-based media can be evaluated/analyzed based on guidelines recommended by Foster and Lodge. Foster's book [38] is divided into seven chapters that deal with the founding elements of narrative texts such as: story, people, plot, fantasy, prophecy, pattern and rhythm. Several new ideas and terms were introduced in his book, which made the text an innovation at the time it was published, and even today Foster's views form an important part of literary analysis. In contrast, Lodge's book [50] collected 50 of his essays that discuss point of view, the "narrator" and other aspects of writing in literary classics. In his work, he defines terms of the novelist's craft deftly and concisely that will serve as both a quick introduction to the beginner and a quick refresher to the more advanced writer. Since story text forms an integral part of the stories generated by HEFTI, authors/readers may still adhere to the guidelines set by Foster and Lodge for evaluation.

However, reliance on these guidelines to evaluate stories generated by HEFTI is not as relevant or appropriate because the stories are now generated by a cyborg author from story constructs provided by the human author. How closely the resultant stories adhere to the guidelines set forth by Foster and Lodge is the sole responsibility of the human author. Although the human author could still use the guidelines to evaluate the quality of their written story, the research objective of HEFTI is focused on generating large quantities of story variants from a given set of inputs and rules that govern how story variations are generated. Since stories generated by HEFTI can be broken down into story elements, affinity measures that compute similar patterns in sequences of story elements to determine to what degree stories are similar to each other can be used to evaluate the perfor-

mance of HEFTI. Affinity measures allow us to observe similarities between different stories using a set of story elements and parameters as input, so that we can compare and analyze the behavior of story generation over a period of time (or a set range of random number seeds).

8.3 Affinity Measures

Affinity measures are commonly used in evolutionary computing and immune networks for pattern recognition and feature extraction applications. For HEFTI, the expression “affinity” can be viewed as a general term that relates the quality of a sequence of story elements (story thread) in relation to its peers based on the environment (from a particular story timestep) in which it is placed. An example of a sequence of story elements is shown in Figure 65.

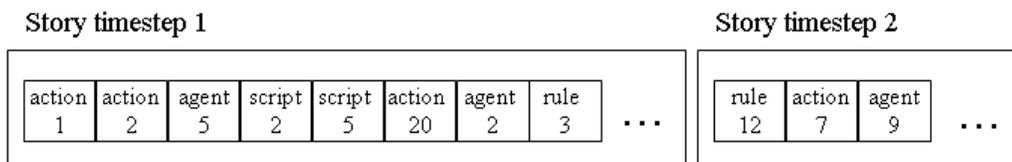


Figure 65. Sequences of story elements grouped by story timesteps

Since story templates are translated into timesteps in a story after being populated by various story elements at run time, they are not included in the computation of affinity between stories to prevent double counting. Affinity measures used in most immune networks and evolutionary computing applications are computed based on their shape similarity. Hamming, Manhattan and Euclidean shape-spaces [33] are commonly used by these applications to determine the distance between antibodies/sequences of elements such that when the distance between two sequences is minimal, the sequences exhibit maximal affinity with each other. The names of story elements (such as “action 1” and “agent 2”) do not convey distance information in any of the aforementioned shape-spaces. In other words, each of the story elements represents a unique entity to HEFTI, so it is irrelevant to derive the affinity of what story element A’s semantic meaning is to B.

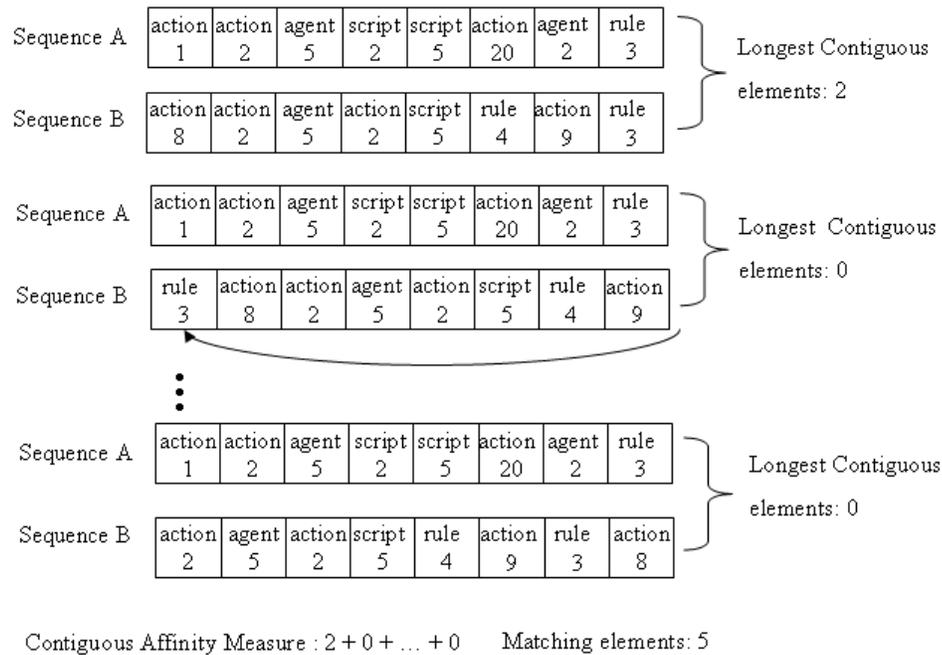
The Manhattan and Euclidean shape-spaces are not applicable in this context because they measure distance between real-valued elements using the Manhattan and Euclidean distance measures respectively (refer to [33] for more information about these measures). In contrast, elements are represented as sequences of symbols over a finite alphabet of length k in the Hamming shape-space, which makes it highly suited to compute the affinity of two sequences of story elements because the story elements are just symbols/variables representing story texts written by the human author. Equation 8.1 depicts the Hamming distance measure used to evaluate the affinity between two sequences of elements of length L in a Hamming shape-space. If the elements use binary strings to encode information, $k = \{0, 1\}$, then one has a binary Hamming shape-space. If ternary strings, $k = 3$, then one has a ternary Hamming shape-space; and so on.

$$\text{Equation 8.1} \quad D = \sum_{i=1}^L \delta_i \text{ where } \delta_i = \begin{cases} 1 & \text{if } A_{bi} \neq A_{gi} \\ 0 & \text{otherwise} \end{cases}$$

Since story elements are given a unique name by the human author, a unique symbol or integer value can be assigned to each of them, thus forming an integer Hamming shape-space. A limitation of equation 8.1 is that it assumes that there is only one possible alignment in which two story elements may react. This is an unrealistic assumption because similar subsequences of story elements may occur anywhere in a given story thread depicting similar events. The affinity measure must also take into account these various possible alignments. Thus, the total affinity between two sequences can be calculated by summing the affinity of each possible alignment, as follows:

$$D = \sum_{k=1}^L D_k$$

where D_k is given by equation 8.1 when the sequences are in a given alignment k . The process is illustrated in Figure 66 for two sequences of story elements of length $L = 8$.



Affinity for story elements considering different alignments. The top sequence is fixed and the other rotated from left to right. The hamming distance is computed for each possible alignment. This process stops when the sequence returns to its original configuration and the affinity is then taken as the sum of the match of each alignment.

Figure 66. Affinity for story elements considering different alignments

Two different measures are computed in Figure 66, namely the number of matching elements and longest contiguous measures. Each of these measures provides different information about how similar (or affine) two given sequences of story elements are to each other. Since the number of matching elements measure merely indicates the matching story elements, the measure provides information about how heavily story elements are reused between any two generated stories as well as the number of times two stories converge or branch from each other (the larger the value of the matching elements, the more often the given story threads converged thus indicating a greater degree of similarity between each other and vice versa). The matching elements measure provides a general overview of the affinity between story threads, however, it does not provide adequate information about the strength of affinity between two story threads or more localized information such as what was the greatest extent of match between two stories. Since

non-contiguous matches between story elements should be less significant than contiguous matches of story elements in two story threads, the longest contiguous measure (based on the r-contiguous bit rule [33]) is also used in conjunction with the number of matching elements measure in our evaluation to more accurately discriminate against non-contiguous matches with high matching element counts.

In summary, the larger the value of computed affinity between two story threads, the greater the number of story elements shared or the closer the story threads are to each other. Since the design goal of HEFTI is to generate as many story variations from a given set of story constructs as possible, we would like to see the matching elements measure and longest contiguous measure exhibit a downward trend as factors that influence story variations (discussed in Section 8.4.1, Linear Story and Section 8.4.2, Non-linear Story) increase during the empirical study.

8.4 Setup

Most interactive stories told via digital media can be highly non-linear. A highly non-linear story reminds one of stories told in certain role-playing games (RPG) where players (readers) are free to stray away from the main story thread at certain designated story branching points. Occasionally players' actions may influence important characters or events in the story, thus altering the course of the story. On the other hand, a linear story line forces players to unfold the story in sequential order just like flipping the pages of a book and decisions made by players (if any) carry little/no influence on the major events or characters of the main story thread. A non-linear story is characterized by multiple branching points (perforated arrows) or loops from a given story timestep as shown in Figure 67, assuming that the story starts at *Stage 1* and ends at *Stage 5*. On the other hand, a linear story has no branching points other than the solid forward arrows that allow the reader to traverse from one stage of the story to the next according to the order laid out by the author. HEFTI is capable of generating both types of stories given the appropriate set of story templates and elements. The factors that influence the number of story variants generated are the random variables (a list of story elements that are randomly picked at

run time to populate a drop point in a story template), story rules, and branch points in the story which only exist in a non-linear story setting.

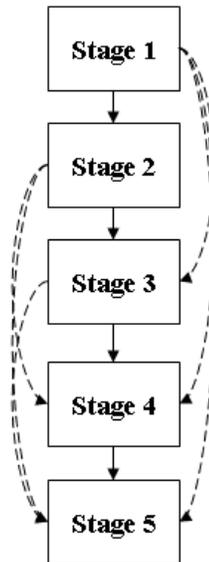


Figure 67. A non-linear story with multiple branching points

The discussion above provides the basis for evaluating the performance of HEFTI. The evaluation results reveal to the human author: 1) how many different stories can be generated from a given set of story templates - too few variations may be a result of a minimal set of story constructs (small selection pool of story templates, random variables or rules) or loopholes in the story rules, or plots and events that prevent HEFTI from populating certain story elements into the templates; 2) how unique the stories are from each other - this is measured by the two affinity measure discussed in section 8.3; and, 3) the average length of the stories given a similar setup - this information serves as a benchmark for authors to fine tune their stories to the desirable lengths.

8.4.1 Linear Story

While creating test sets, we have to take into account two factors, random variables and story rules, that influence variability of stories. In essence, a random variable is a list

that holds various story elements. These could be an agent character with its own attributes, an event with associated rules, or an agent action. Since the random variable is resolved at run time and depends on the state of the story, the end result can be very complex involving multiple rules associated with an agent action or event being invoked. This chain-reaction may propagate further to other story elements creating more randomness in the resultant story text. Similarly, a story rule can yield complex results, as well, if it is associated with a random variable that may invoke multiple rules.

In order to simplify the evaluation process, we are not going to leverage these features. Instead any random variables and story rules used in the test set have a constant branching factor of X . That is, the random variable has a list of X elements with no associated rules or attributes that may result in a chain of story elements being fired when resolved at run time. Similarly, story rules in our test set are generated to exhibit similar behavior as the random variable with the construct ‘if-elseif-elseif ... else’).

The other free parameter in our evaluation is the distribution (Y) of the random variables and story rules since the greater the number of story elements that are random or invoke a story rule, the higher the probability that a new story variant is going to be generated. Therefore, story elements that are used to populate the story templates in our test sets adhere to the probability Y of becoming a random variable or an ordinary story element. Similarly, a story element that is contained in a random variable or exists solely as an ordinary story element has similar probability Y for it to invoke a random rule.

Tables E.1 to E.3 displays the evaluation results from varying these free parameters in generating 65,535 (the complete range of random number seeds) 10-stage stories with 20-50 story elements per stage from a set of 1,000 (Table E.1), 2,000 (Table E.2) and 5,000 (Table E.3) randomly generated story elements. Treating the different distribution factors as separate series, we can generate an XY graph that allows us to observe the behavior of HEFTI given different sets of story elements. The number of matching elements measure from Figures 68 to 70 indicates an inverse relationship between the number of story elements being reused and the distribution ratio of random variables, story rules and the number of story elements being used to generate the stories. The decrease in

similarities of the dynamic stories generated are especially observable from the steep drop in the measure in Figure 68 for the values 10% to 40% on the x-axis. At the lower distribution range and smaller set of story elements to choose from, stories tend to be highly similar to each other as indicated by the value of 1,146,862 for a distribution ratio of 10% and a branching factor of 3. This value tells us that out of the 65,535 random 10-stage stories with an average composition of 350 story elements generated by HEFTI, the average number of story elements reused (convergence point in the story) is 17. In other words, any particular randomly generated story has an average of approximately 18 convergence points with at least one of the other 65,534 random stories. The average number of story elements reused can be computed with the equation:

$$A = \frac{V}{N}$$

Where V is the matching element measure and N is the number of random stories generated. Drops in story convergence points are less significant with a larger pool of story elements to generate the stories as is evident in Figure 69 and 70. Since the story generation mechanism has already benefited from a larger selection of candidates, increasing the distribution ratio and branching factors contributed less to the variability of the stories. Similar observations can be made about the series in Figure 70 where most of the series slowly settles at a particular value as indicated by the almost flat slopes at the 70% to 90% ranges.

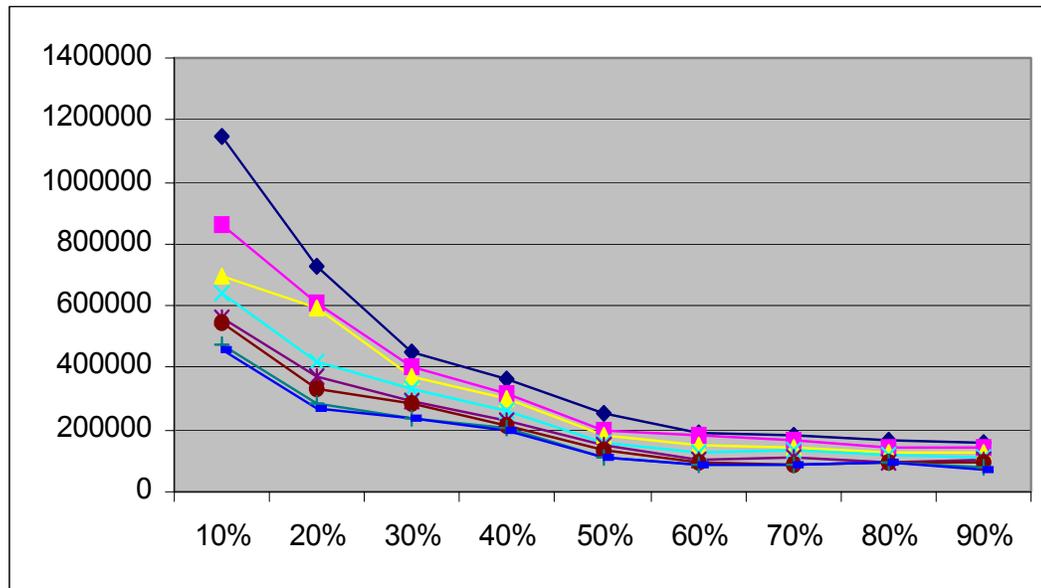


Figure 68. Graphing the series in Table E.1

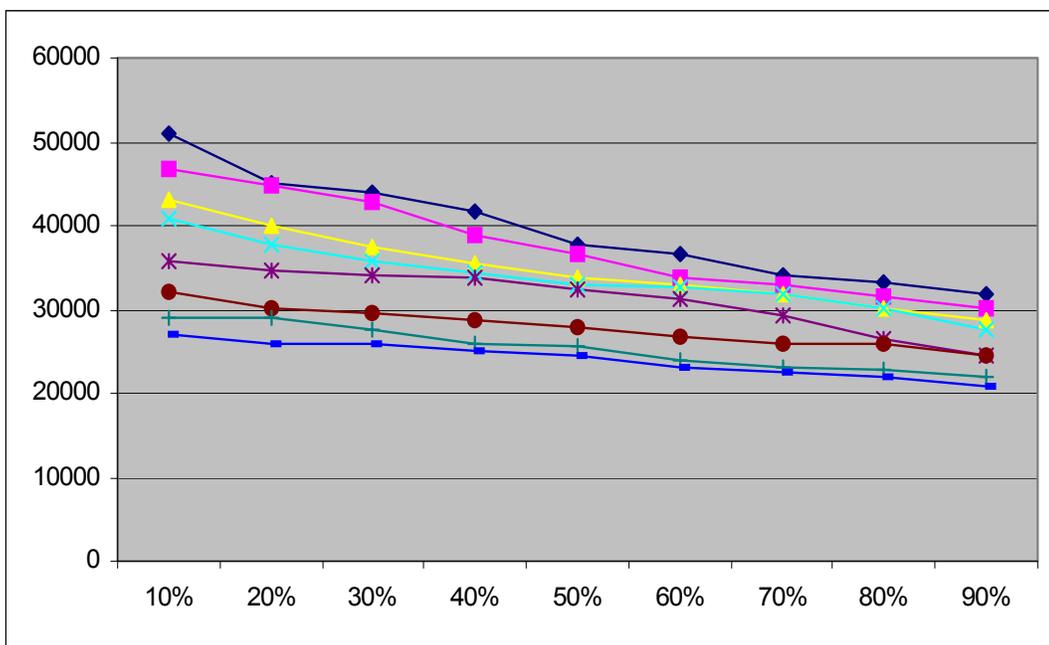


Figure 69. Graphing the series in Table E.2

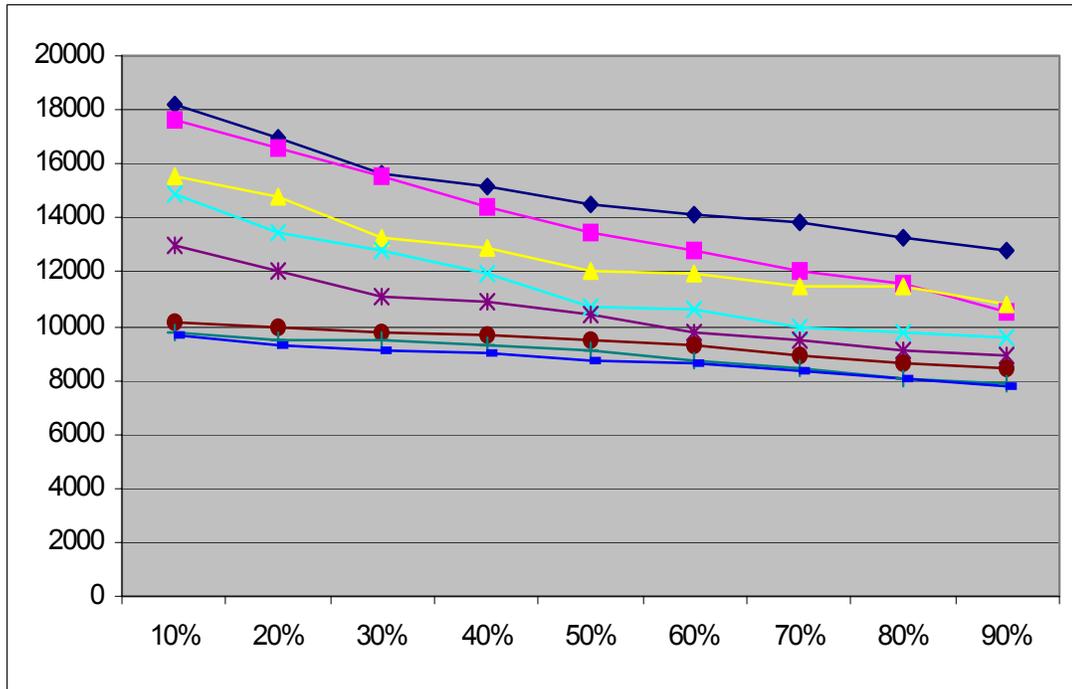


Figure 70. Graphing the series in Table E.3

The longest contiguous measure (Figure 71 to 73) is the longest sub-sequence of story elements that matches another sub-sequence in a different story thread. Since sequences of story elements that form a story are compared to other generated stories with different alignments considered, this measure along with the average contiguous measure shows to what extent the stories are similar to each other.

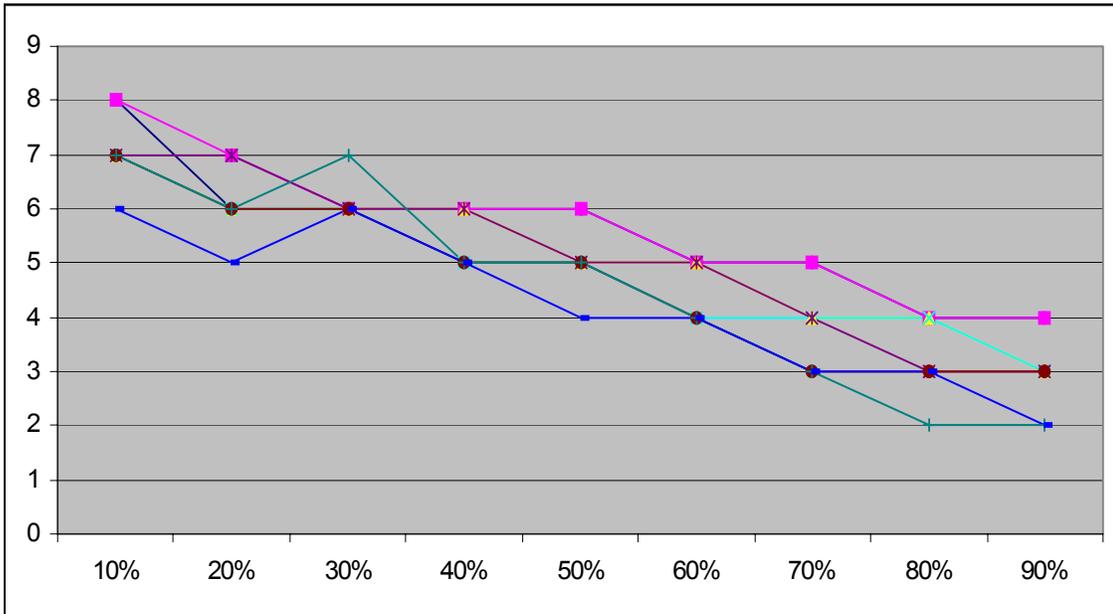


Figure 71. Graphing the longest contiguous element measure in Table E.4

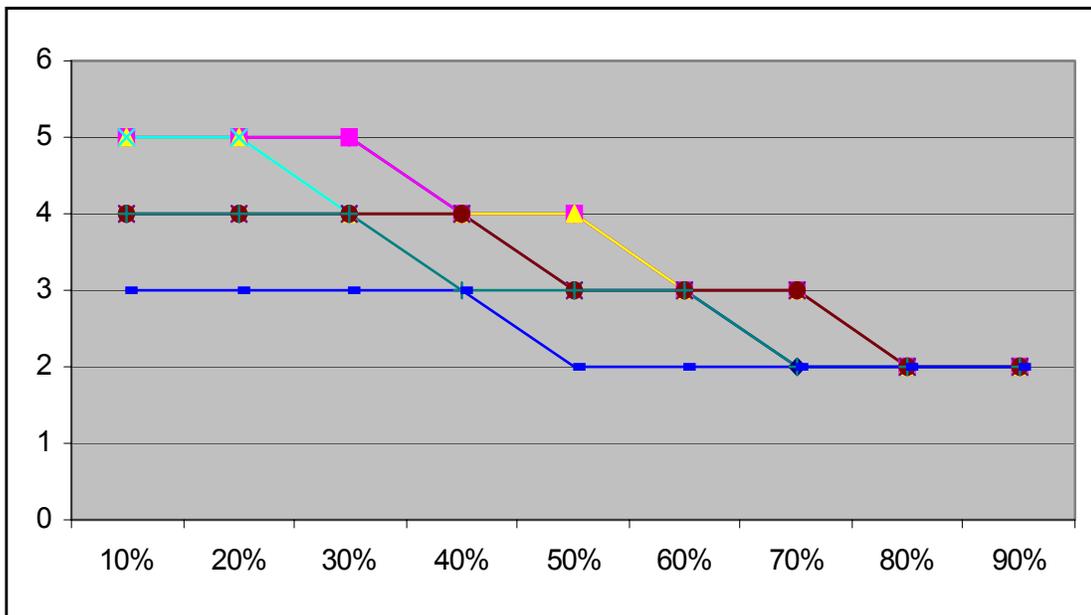


Figure 72. Graphing the longest contiguous element measure in Table E.5

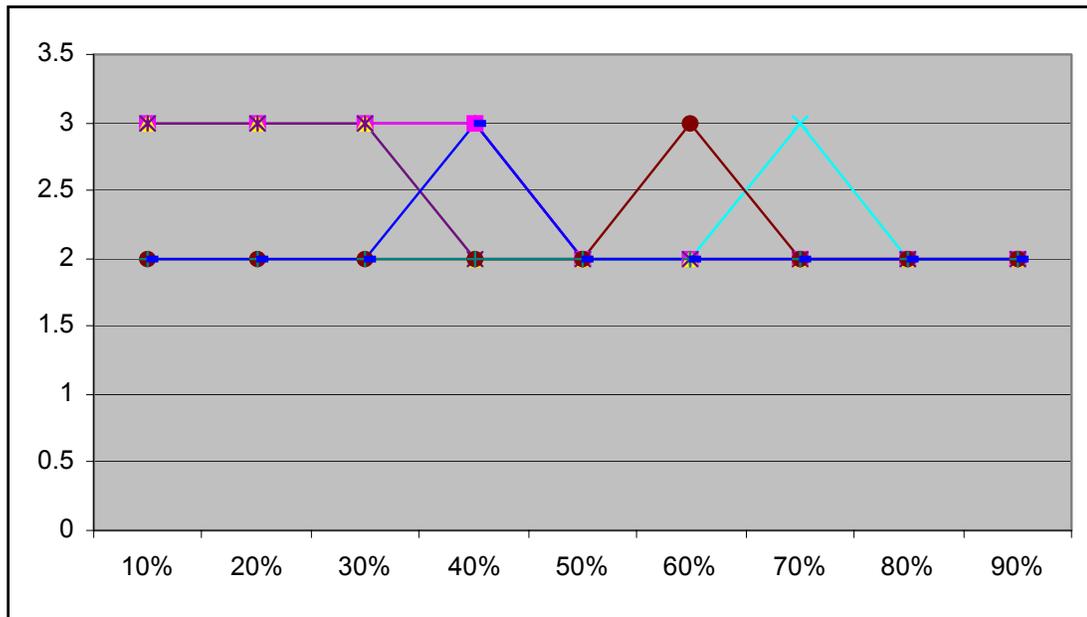


Figure 73. Graphing the longest contiguous element measure in Table E.6

The upper and lower bounds of a series' contiguous measure can be estimated based on the average (Figure 74 to 76) and longest contiguous measure (Figure 71 to 73) with the following equations:

$$l_i = \begin{cases} \varepsilon_i - |u_i - \varepsilon_i| \\ 0 \text{ if } \varepsilon_i - |u_i - \varepsilon_i| < 0 \end{cases}$$

$$u_i = \alpha_i$$

Where l_i represents the lower bound of point i in a series, ε_i being the average contiguous measure, and u_i being the upperbound for that point which is equals to the longest contiguous measure α_i

The upper and lower bounds of the series tell how significantly the generated stories differ from each other. The value of the lower bound is set to 0 whenever results from the equations above are smaller than 0, because the minimum value that the contiguous measure may yield is 0, so values smaller than 0 are meaningless.

Tables E.10 to E.12 list the lower and upper bounds of the series with different numbers of story elements (1,000, 2,500 and 5,000), distribution and branching parameters. As evident from the tables, an increase in the number of story elements to choose from coupled by distribution and branches in the story, introduced variations in the generated stories. Such results may be observed from the sharp decrease in lower and upper bounds of the series in Table E.10, where, as the branching and distribution parameters are slowly increased from 10% and 3 to 90% and 10 respectively, we see a drop of 82.79% and 75% in the lower and upper bounds of the contiguous measure. Similar results can be observed when we increase the pool of story elements in Table E.11, although the benefits of increasing the distribution and branching factor of the random variables or story rules decreases when the values of the parameters reach 70% and 8. Lastly, Table E.12 indicates to us that by further increasing the number of story elements in the selection pool, distribution and branching factors in the stories do not play such an important role in variability of the stories generated. This observation is supported by the low values of the upper and lower bounds of the series as well as knowledge that a larger pool of selection means it is less probable for the same entity to be chosen when elements are drawn randomly from the pool.

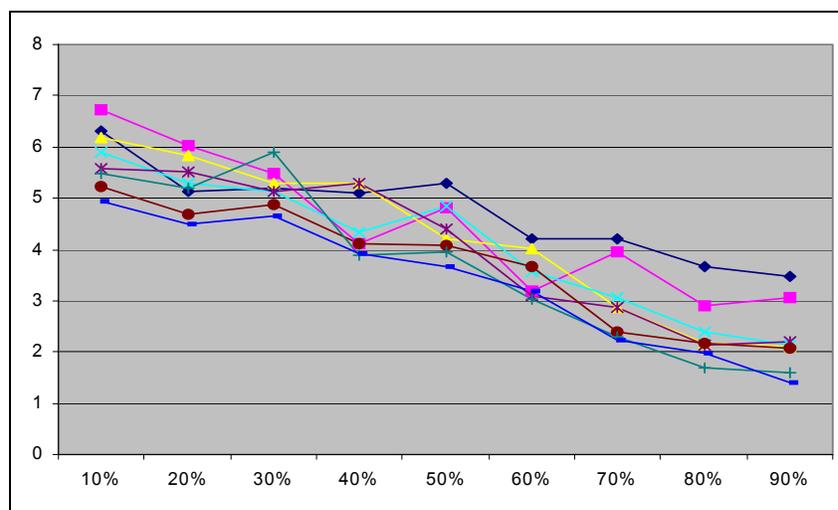


Figure 74. Graphing the average contiguous measure in Table E.7

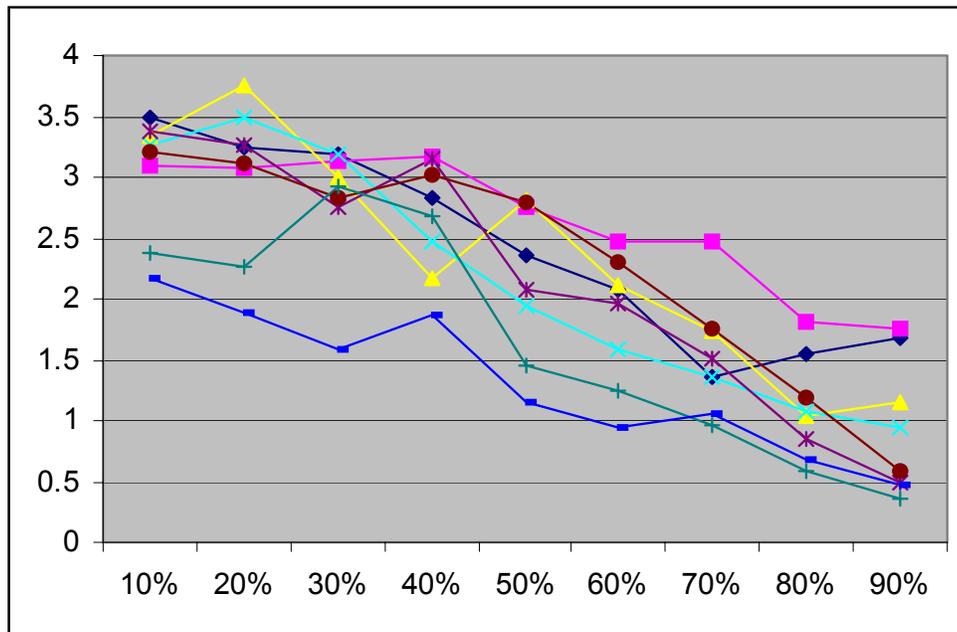


Figure 75. Graphing the average contiguous measure in Table E.8

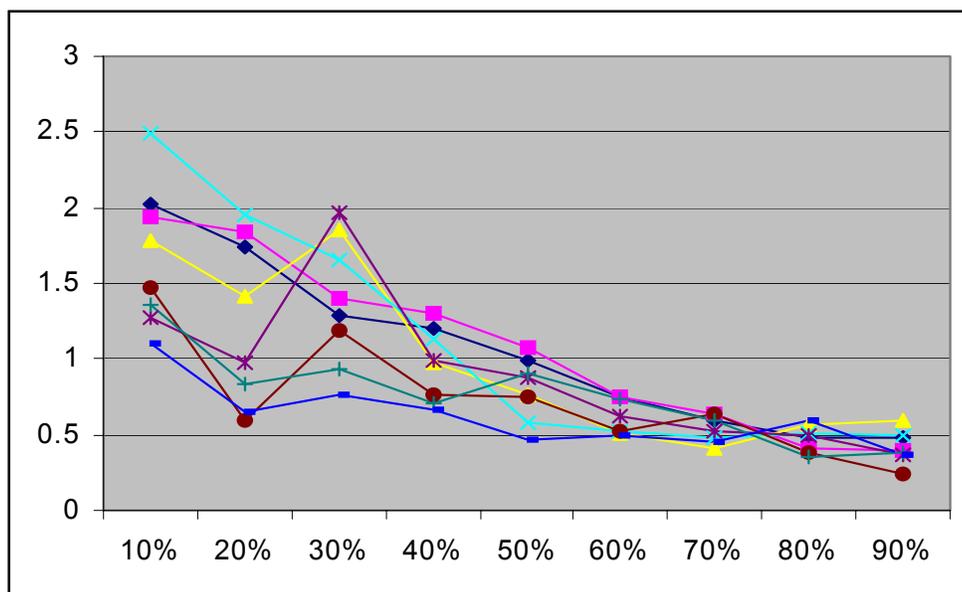


Figure 76. Graphing the average contiguous measure in Table E.9

8.4.2 Non-linear Story

Since a non-linear story has multiple branching points and possibly loops in the story thread, story branches that allow readers to leap through the story threads have to also be taken into account. Using similar constraints to the linear-story scenario, the random variables and story rules are randomly created to exhibit a branching factor of X and possible story branches from story stage Y is always $N-Y-1$ (where N is the total number of story stages in a given set of story constructs). These constraints give us all of the forward branches in a story (Figure 77). Loops are not considered in this evaluation because they can potentially generate stories of infinite length, thus infinite variants of a story.

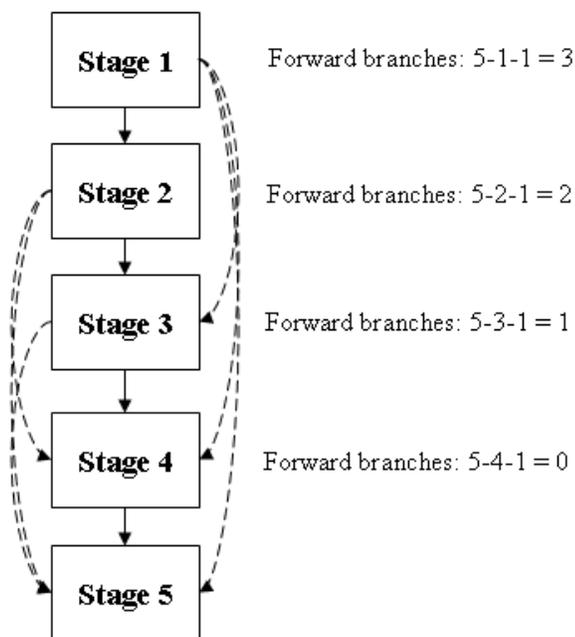


Figure 77. Number of forward branches for different story stages.

Similar affinity measures are used to evaluate performance of the engine while generating non-linear stories. However, an additional free parameter, forward branching in story thread, is introduced into the evaluation process since stories may make forward

leaps from one story stage to another. Evaluation results in this section are obtained by setting the probability of forward branching in stories at 25%, 50% and 75%.

Besides functioning as guidelines to story generation for HEFTI, story templates implicitly divide stories into various stages. Such characteristics of story templates indicate that by allowing a certain probability of forward branching in story generation, the number of contiguous story elements and matching elements measure should be inversely related to the probability of forward branching. Since HEFTI generates stories that may leap across multiple story stages, the higher the probability of forward branching for story stage n , the less likely story templates in story stage $n+1$ are going to be used since story templates from story stage $n+m$ are now in the selection pool as well. Therefore, we should expect drops in both measures because they are dependent upon how the story elements are constructed from a given set of story templates.

Figures 78 to 80 illustrate the matching element measures of the stories generated with a 1,000 story element set with 25%, 50%, and 75% forward branching probabilities respectively. Similarly, Figures 81 to 83 and Figures 84 to 86 depict the matching element measures of the stories generated with 2,500 and 5,000 story element sets with the same set of forward branching probabilities. Conforming to our earlier statement that the matching elements measure should be inversely related to the probability of forward branching, we observe that all of the dynamically generated non-linear stories have lower matching element measures between one another than linear stories. The advantage of allowing the construction of non-linear stories is particularly apparent for the smaller selection pools, namely the 1,000 story element set, for we saw approximately 30%-38% drop in matching elements measures between the values in Table E.1 and Tables E.13 to E.15. However, the advantage quickly levels off due to increase in selection pool as evident by the approximately 8%-10% drop in matching elements measures between the values in Table E.2 and Tables E.16 to E.18 and the approximately 6%-9% drop in matching elements measures between the values in Table E.3 and Tables E.19 to E.21. Uniqueness and higher variation between the stories (as indicated by lower matching element measures and longest contiguous measures as in Tables E.22 to E.30) are also reflected by the

slope of the series in Figures 78 to 86 in which initial increase in branching and distribution parameters of the story elements resulted in large increases in story uniqueness and variation.

However, further increase in these parameters brought out improvements at a decreasing rate. In confirmation to our earlier analysis on story templates, most of the observed longest contiguous measure of non-linear stories and the average contiguous measure are smaller than those for linear stories, since HEFTI is now free to use story templates from different story stages resulting in a larger selection pool. In contrast to linear stories, the estimated upper and lower bounds (Tables E.40 to E.48) of non-linear stories indicate less deviations from the average contiguous measure, most prominently from the reduced number of zero entries in the tables as well as smaller differences between most series' average contiguous measures and their corresponding upper bounds.

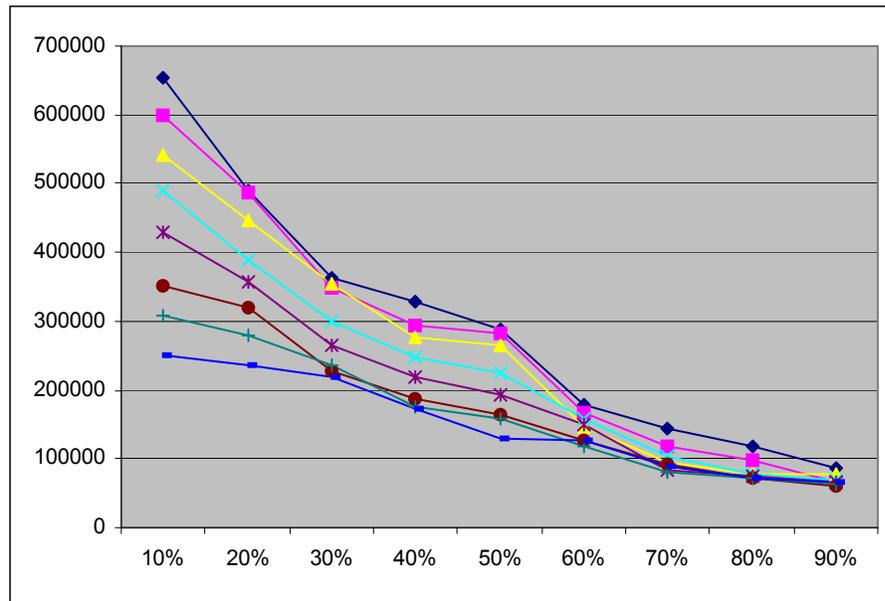


Figure 78. Graphing the series in Table E.13

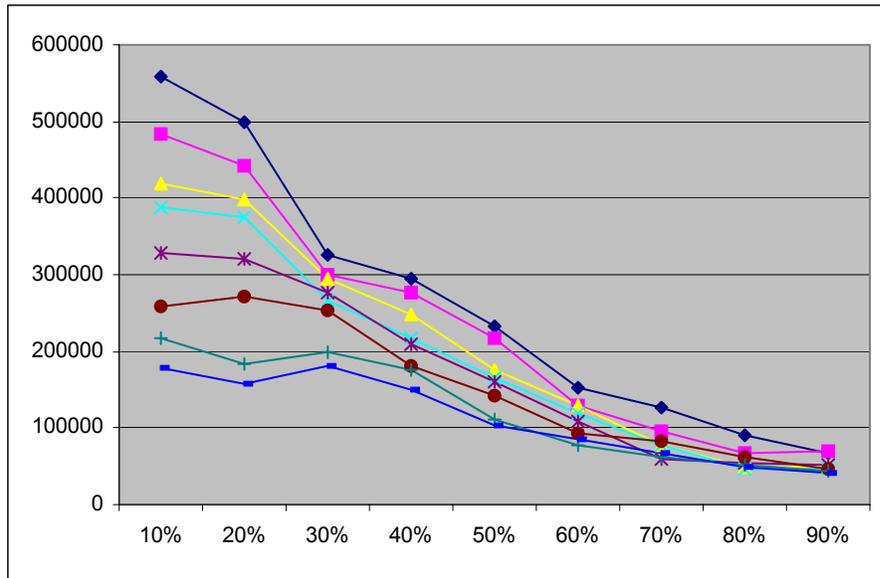


Figure 79. Graphing the series in Table E.14

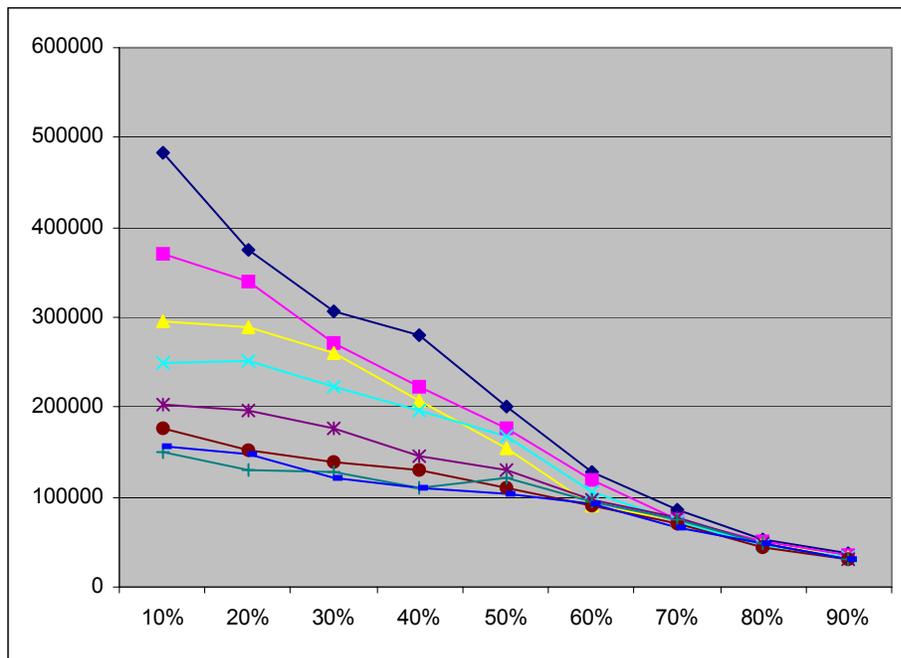


Figure 80. Graphing the series in Table E.15

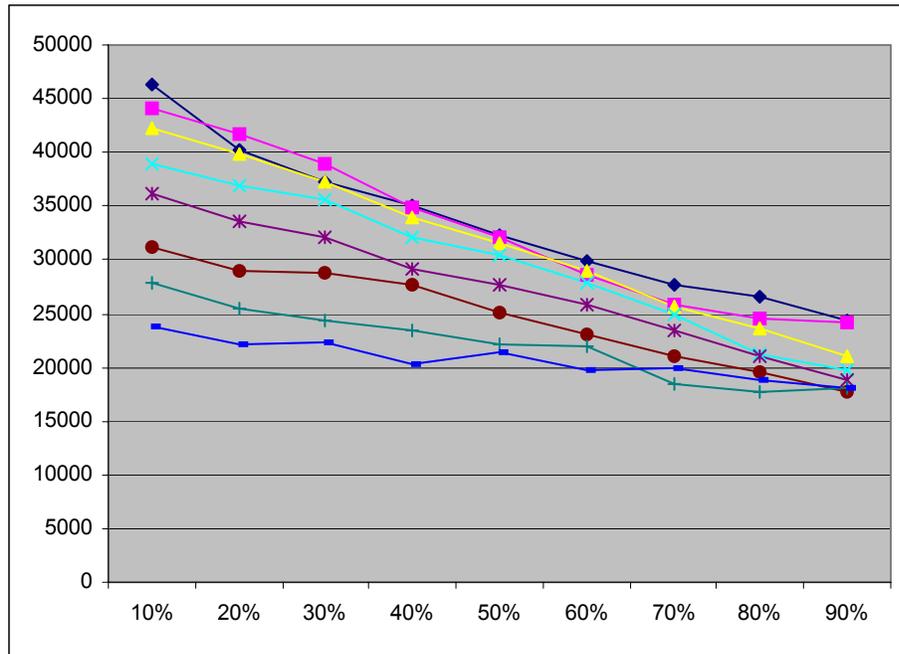


Figure 81. Graphing the series in Table E.16

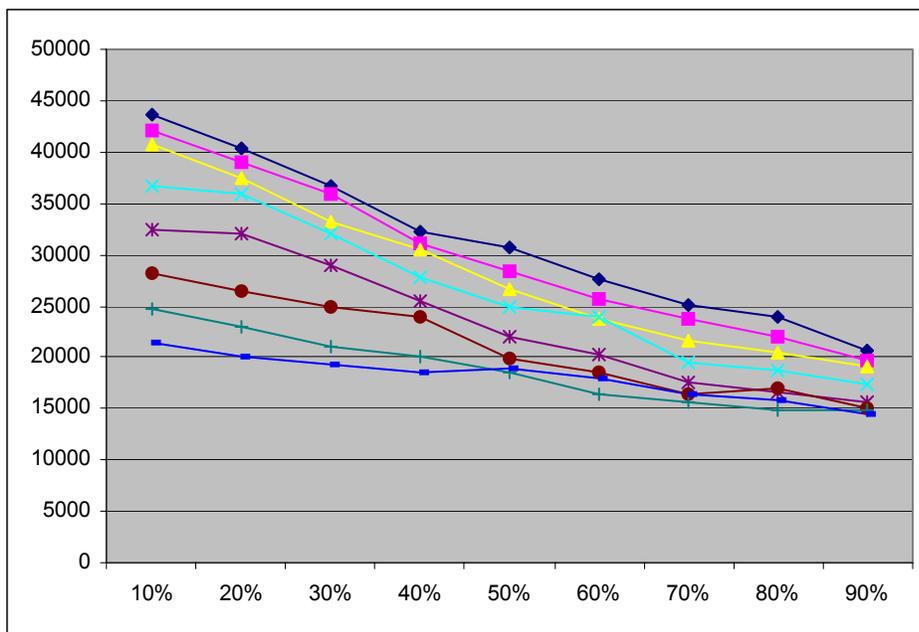


Figure 82. Graphing the series in Table E.17

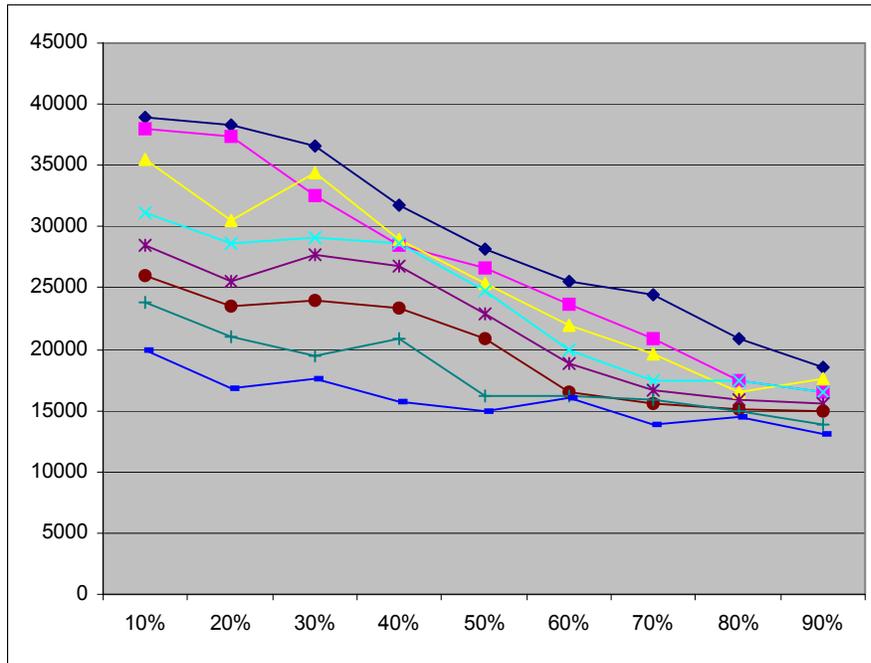


Figure 83. Graphing the series in Table E.18

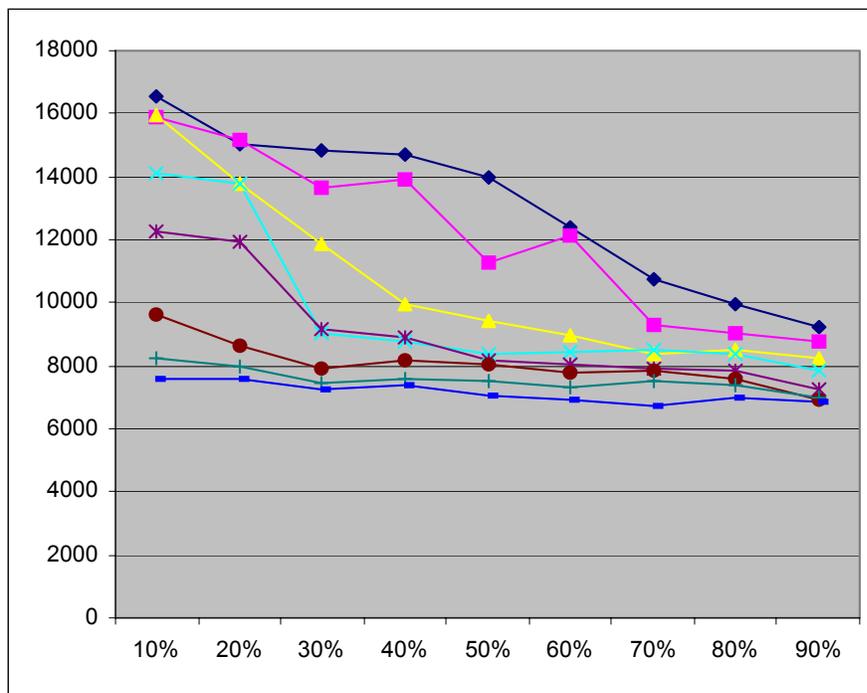


Figure 84. Graphing the series in Table E.19

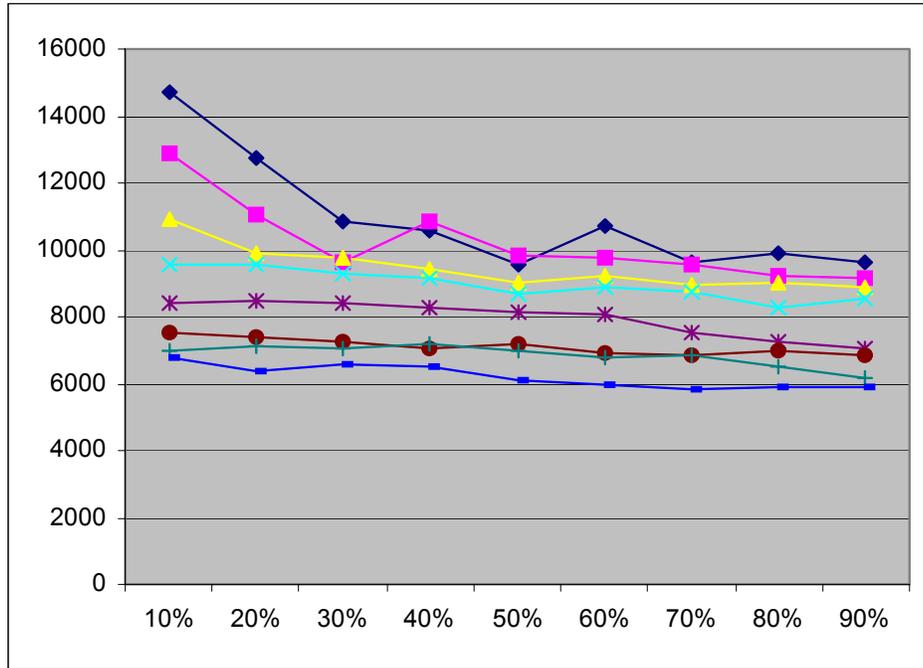


Figure 85. Graphing the series in Table E.20

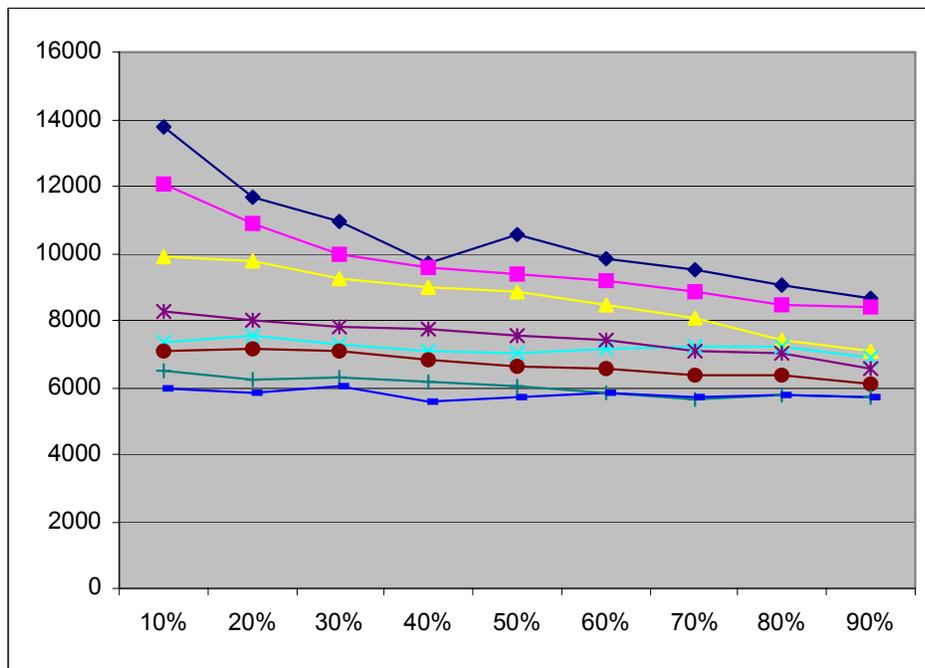


Figure 86. Graphing the series in Table E.21

9. CONCLUSION AND FUTURE WORK

In this dissertation, we have covered: 1) motivation for the emergence of storytelling engines and cyborg authors; 2) various aspects of the HEFTI storytelling engine; 3) steps involved in generating a story; 4) functions that can be used to control the state of HEFTI; 5) a rather detailed case study on generating a simple story; and, 6) a usability study and empirical evaluations of linear and non-linear stories generated from HEFTI. The reader can always expand the story space by introducing more rules and story elements, however, the story complexity can increase rather quickly as more agents, rules, events and actions are included in the story space. Therefore, it is advisable for the reader to carefully think of the general structure of the story before authoring the story elements. Since there are no visualization tools to assist authors in authoring the story elements, experimentation is needed to properly place the story elements in the story environment. As in most tasks, experience can greatly shorten the time and effort required to put a dynamic story together. The only authoring process not covered is the agent animation frame creation process, which can be easily accomplished by setting the batch rendering option on the author's favorite modeling software to generate a series of images for the agents.

Observations made on the test subjects and the suggestions we obtained from the usability study pointed out that the IAE abstracted the complexity of authoring (or programming) interactive, dynamic stories from the authors with the use of various window controls to help them visualize various aspects of a story. This reduces the amount of learning required in order to start having the pleasure of authoring stories that change based on a given story seed as well as a set of rules and conditions. Although the IAE is by no means an accurate depiction of a cyborg authoring environment of the future, we know that it reflects certain aspects of such an environment.

Results and analysis from the empirical study and evaluation indicate that a larger selection pool of story elements made available to random variables, story rules and story templates can significantly boost uniqueness and variation in the stories generated even if the author chose not to create too many branching or convergence points in his/her story. Since creating a larger selection pool of story elements can be very time consuming for a

human author (unlike the sets of story elements that are automatically generated in the evaluation section), the author could still improve variations or uniqueness of his/her stories as told by HEFTI by increasing use of random variables, number of story elements referenced by a random variable, number of branching points (in the form of *if-elseif-elseif-...-else*) in story rules, and non-linearity of the story stages. However, the author may only improve re-readability (assuming that re-readability of a story stems from uniqueness and variations in the generated stories) of his/her story to a certain extent by using such alternatives because ultimately a small selection pool limits the story space spanned by the story elements as indicated by the matching elements and average contiguous measures in Figures 68 to 70, Figures 71 to 76 and Figures 78 to 86 in linear and non-linear story settings.

Future extension of this architecture could include the introduction of template learning, possibly accomplished through neural networks that classify and construct templates out of carefully selected story samples written by human authors, effectively eliminating the bounded story context restriction. Animations of the agent characters could be automatically generated by using key frames and interpolations from 3D graphics packages such as Maya 3D or 3D studio max, thus simplifying the task of the human author. A tree visualization of the story thread could be incorporated into the authoring environment so that authors may visually alter the story components and assign probabilities to different story branches in the story thread.

The inclusion of these automations, and user interface improvements and features may make the resultant system closer to a cyborg authoring environment of the future. Various pieces of the story may be automatically generated by the system with minimal user intervention when their relationships are established by the author, so that the human authors will be able to spend more time in thinking about a story instead of 'writing' a story.

REFERENCES

- [1] E. J. Aarseth, *Cybertext: Perspectives on Ergodic Literature*, Johns Hopkins University Press, Baltimore, MD, 1997.
- [2] Animal Blocks. <http://gn.www.media.mit.edu/groups/gn/projects/animalblocks/>
[viewed on May 14, 2006]
- [3] S. Arnone, M. Dell'Orto and A. Tettamanzi, Toward a fuzzy government of genetic populations, in: *Proceedings of the 6th IEEE Conference on Tools with Artificial Intelligence (TAI-1994)*, New Orleans, LA, 1994, pp. 585-591.
- [4] R. Aylett, Narrative in Virtual Environments - Towards Emergent Narrative, in: *Proceedings of the American Association for Artificial Intelligence Fall Symposium on Narrative Intelligence (AAAI-1999)*, Menlo Park, CA, 1999, pp. 83-86.
- [5] T. Bäck , D. B. Fogel and Z. Michalewicz, *Evolutionary Computation 1, Basic Algorithms and Operators*, Institute of Physics Publishing, London, United Kingdom, 2000.
- [6] B. Barry, *Story Beads: A wearable for distributed and mobile storytelling*, MIT Masters Thesis, 2000. <http://ic.media.mit.edu/icSite/icpublications/Thesis/barbaraMS.html> [viewed on May 14, 2006]
- [7] W. Bender, *The Electronic Publishing project*, 2002. <http://ep.media.mit.edu/>
[viewed on May 14, 2006]

- [8] A. Bergmann, W. Burgard and A. Hemker, Adjusting parameters of genetic algorithms by fuzzy control rules, in: *New Computing Techniques in Physics Research III*, K.H. Becks and D. Perret-Gallix (Eds.), World Scientific Singapore, Singapore, 1993.
- [9] M. Bernstein, Patterns of hypertext, in: *Proceedings of the ACM conference on Hypertext and Hypermedia (SIGWEB-1998)*, Pittsburgh, PA, 1998, p. 21-29.
- [10] M. Bernstein, M. Joyce and D. B. Levine, Contours of Constructive Hypertext, in: *Proceedings of the ACM conference on Hypertext and Hypermedia (SIGWEB-1992)*, Milan, Italy, 1992, pp. 161-170.
- [11] M.U. Bers and J. Cassell, Interactive Storytelling Systems for Children: Using Technology to Explore Language and Identity, *Journal of Interactive Learning Research* 9 (2) (1998) 183-215.
- [12] T. Bickmore and J. Cassell, Small Talk and Conversational Storytelling in Embodied Interface Agents, in: *Proceedings of the AAAI Fall Symposium on Narrative Intelligence (AAAI-1999)*, Cape Cod, MA, 1999, pp. 87-92.
- [13] J. D. Bolter, *Writing Space: Computers, Hypertext, and the Remediation of Print*, Lawrence Erlbaum Association, Mahwah, NJ, 2001.

- [14] K. M. Brooks, *Metalinear Cinematic Narrative: Theory, Process, and Tool*, MIT Ph.D. Dissertation, 1999. <http://ic.media.mit.edu/icSite/icpublications/Thesis/brooksPHD.html> [viewed on May 14, 2006]
- [15] J. Cassell, The Gesture and Narrative Language Group. <http://gn.www.media.mit.edu/groups/gn/> [viewed on May 14, 2006]
- [16] J. Cassell and T. Bickmore: *Negotiated Collusion: Modeling Social Language and Its Relationship Effects in Intelligent Agents, User Modeling and Adaptive Interfaces* 12 (2002) 1-44.
- [17] J. Cassell, Y. Nakano, T. Bickmore, C. Sidner and C. Rich, *Annotating and Generating Posture from Discourse Structure in Embodied Conversational Agents*, in: *Workshop on Representating, Annotating, and Evaluating Non-Verbal and Verbal Communicative Acts to Achieve Contextual Embodied Agents, Autonomous Agents Conference, Montreal, Canada, 2001*.
- [18] J. Cassell, H. Vilhjalmsson and T. Bickmore, *BEAT: the Behavior Expression Animation Toolkit*, in: *Proceedings of the Conference in computer graphics and interactive techniques (SIGGRAPH-2001)*, Los Angeles, CA, 2001, pp. 477-486.
- [19] M. Cavazza, R. Aylett, K. Dautenhahn and C. Fencott, *Interactive Storytelling in Virtual Environments: Building the Holodeck*, *Workshop on Intelligent Virtual*

- Environments, the 6th International Conference on Virtual Systems and Multimedia (VSMM2000), Gifu, Japan, 2000, pp. 4-6.
- [20] M. Cavazza, F. Charles and S. J. Mead, Intelligent Virtual Agents, in: Lecture Notes in Artificial Intelligence vol. 2190, De Antonio, Aylett, & Ballin (Eds.), Springer-Verlag, Berlin, Germany, 2001.
- [21] M. Cavazza, F. Charles and S. J. Mead, AI-based Animation for Interactive Storytelling, in: Proceedings of Computer Animation, IEEE Computer Society Press, Bologna, Italy, 2001, pp. 318-325.
- [22] M. Cavazza, F. Charles and S. J. Mead, Characters in Search of an Author: AI-Based Virtual Storytelling, in: Proceedings of the International Conference on Virtual Storytelling, Avignon, France, 2001, pp. 145-154.
- [23] M. Cavazza, F. Charles and S. J. Mead, Interacting with virtual characters in interactive storytelling, in: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, 2002, pp. 318-325.
- [24] S. Coles and K. Dautenhahn, A robotic story-teller, in: 8th International Symposium on Intelligent Robotic Systems (SIRS-2000), Reading, United Kingdom, 2000, pp. 18-20.
- [25] J. Dakss, S. Agamanolis, E. Chalom and V.M. Bove, Jr., HyperSoap, 2002. <http://www.media.mit.edu/hypersoap/> [viewed on May 14, 2006]

- [26] J. Dakss and V. M. Bove, Jr., ISIS, a programming language for responsive media, 2002. <http://web.media.mit.edu/~stefan/isis/> [viewed on May 14, 2006]
- [27] C. Danis, L. Comerford, E. Janke, K. Davies, J. DeVries and A. Bertrand, Story-writer: A Speech Oriented Editor, in: Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI-1994), Boston, MA, 1994, pp. 277-278.
- [28] C. Darwin, The Origin of Species, Modern Library Paperback, New York, NY, 1998.
- [29] K. Dautenhahn, Story-Telling in Virtual Environments, in: Intelligent Virtual Environments workshop, European Conference on Artificial Intelligence (ECAI-1998), Brighton, UK, 1998, pp. 39-48.
- [30] K. Dautenhahn, Embodiment in Animals and Artifacts, Papers from the AAAI Fall Symposium, Technical report FS-96-02, AAAI Press, Cambridge, MA, 1996, pp. 27-32.
- [31] G. Davenport, Interactive Cinema, 2002. <http://ic.media.mit.edu/> [viewed on May 14, 2006]
- [32] G. Davenport and M. Murtaugh, Automatist storyteller systems and the shifting sands of story, IBM Systems Journal, 36 (3) (1997) 446 - 456.

- [33] L. N. De Castro and J. I. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer-Verlag, Berlin, Germany, 2002.
- [34] *DirectX Documentations*, 2006.
<http://msdn.microsoft.com/directx> [viewed on May 14, 2006]
- [35] E. Edmonds, G. Fischer, J. Mountford, F. Nake, D. Riecken and R. Spence, *Creativity: Interacting with Computers*, in: *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI-1995)*, Denver, CO, 1995, pp. 185-186.
- [36] D. C. Engelbart, *Augmenting Human Intellect: A Conceptual Framework*, Summary Report AFOSR-3223 under Contract AF 49(638)-1024, SRI Project 3578 for Air Force Office of Scientific Research, Stanford Research Institute, October 1962.
- [37] J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes, *Computer Graphics: Principles and Practice in C - 2nd Edition*, Addison-Wesley Publishing Co., Boston, MA, 1995.
- [38] E. M. Forster, *Aspects of the Novel*, Harvest Books, Fort Washington, PA, 1985.
- [39] A. S. Glassner, *Active Storytelling*, in: *Computer Graphics International (CGI-1999)*, Canmore, Alberta, Canada, 1999, pp. 2-9.
- [40] A. S. Glassner, *Interactive Storytelling: People, Stories, and Games*, in: *International Conference on Virtual Storytelling*, Avignon, France, 2001, pp. 51-60.

- [41] D. Grabson and N. Braun, A Morphological Approach to Interactive Storytelling, in: Proceedings of on Artificial Intelligence and Interactive Entertainment, Living in Mixed Realities, Sankt Augustin, Germany, 1996, pp. 337-340.
- [42] B. Hayes-Roth, R. van Gent and D. Huber, Acting in Character, in: Creating Personalities for Synthetic Actors, R. Trappl and P. Petta (Eds.), Springer-Verlag, New York, NY, 1997.
- [43] J. Horowitz-Murray, Hamlet on the Holodeck : The Future of Narrative in Cyberspace, Free Press, New York, NY, 1998.
- [44] Humanities Informatics Project, TAMU University and TAMU Libraries.
<http://hi.tamu.edu/index.html> [viewed on May 14, 2006]
- [45] J. E. Hunt and D. E. Cooke, Learning using an artificial Immune System, Journal of Network and Computer Applications: Special Issue on Intelligent Systems: Design and Application 19 (1996) 189-212.
- [46] B. Laurel, Computers as Theatre. Addison-Wesley Publishing Co., Boston, MA, 1993.
- [47] B. Laurel, T. Oren and A. Don, Issues in Multimedia Interface Design: Media Integration and Interface Agents, in: Proceedings of ACM Conference on Human Factors in Computing Systems (CHI-1990), Seattle, WA, 1990, pp. 133-139.

- [48] M. Lee and H. Takagi, Dynamic control of genetic algorithms using fuzzy control techniques, in: Proceedings of the Fifth International Conference on Genetic Algorithms, S. Forrest (Ed.), Morgan Kaufmann, San Mateo, CA, pp. 76-83, 1993.
- [49] J. C. R. Licklider, Man-Computer Symbiosis, I.R.E. Transactions on Human Factors in Electronics, HFE-1 (1960) 4-10.
- [50] D. Lodge, The Art of Fiction, Viking/Penguin Books, New York, NY, 1993.
- [51] I. Machado, A. Paiva, and P. Brna, Real Characters in Virtual Stories (Promoting Interactive Story-Creation Activities), in: Proceedings of the First International Conference on Virtual Storytelling (ICVS 2001), Avignon, France, 2001, pp. 127-134.
- [52] E. Maffre, J. Tisseau and M. Parenthoën, Virtual Agents' Self-Perception in Story Telling, in: Proceedings of the First International Conference on Virtual Storytelling (ICVS 2001), Avignon, France, pp. 155-160.
- [53] E. Mallen, Online Picasso Project
<http://csdll.cs.tamu.edu:8080/picasso/> [viewed on May 14, 2006]
- [54] M. Mateas and P. Sengers, Narrative Intelligence, in: Proceedings of the American Association for Artificial Intelligence Fall Symposium on Narrative Intelligence (AAAI-1999), Menlo Park, CA, 1999, pp. 1-10.

- [55] M. Mateas and A. Stern, A Behavior Language for Story-Based Believable Agents, in: Working notes of Artificial Intelligence and Interactive Entertainment, Ken Forbus and Magy El-Nasr Seif (Eds.), AAAI Press. Menlo Park, CA, 2002.
- [56] M. Michell, An Introduction to Genetic Algorithms, The MIT Press, Cambridge, MA, 1999.
- [57] Microsoft Agent home page.
<http://www.microsoft.com/msagent/default.asp> [viewed on May 14, 2006]
- [58] R. Nakatsu and N. Tosa, Interactive Movies, Handbook of Internet and Multimedia – Systems and applications, B. Furht (Ed), CRC Press and IEEE Press, Sound Parkway, NJ, 1999.
- [59] T. Nelson, Literary Machine, Mindful Press, Sausalito, CA, 1982.
- [60] J. Roschelle, J. J. Kaput, SimCalc MathWorlds for the mathematics of change, Communications of the ACM, 39 (8) (1996) 97-99.
- [61] D. Rutkowska, Neuro-Fuzzy Architectures and Hybrid Learning, Physica-Verlag, Heidelberg, Germany, 2002.
- [62] K. Ryokai, C. Vaucelle and J. Cassell, Literacy Learning by Storytelling with a Virtual Peer, in: Proceedings of Computer Support for Collaborative Learning Conference, Boulder, CO, 2002, pp. 352-360.

- [63] E. Sanchez and P. Pierre. Fuzzy logic and genetic algorithms in information retrieval, in: 3rd International Conference on Fuzzy Logic, Neural Networks and Soft Computing, Lizuka, Japan, 1994, pp. 29-35.
- [64] P. Sengers, Narrative Intelligence, in: Human Cognition and Social Agent Technology, K. Dautenhahn (Ed.), John Benjamins Publishing Company, Amsterdam, The Netherlands, 2000, pp.1-26.
- [65] N. M. Sgouros, Supporting Audience and Player Interaction during Interactive Media Performances, in: IEEE International Conference on Multimedia and Expo (ICME-2000), New York, NY, 2000, pp. 1367-1370.
- [66] N. M. Sgouros and S. Kousidou, Authoring and execution environments for multimedia applications featuring robotic actors, in: Proceedings of the ninth ACM international conference on Multimedia, Ottawa, Canada, 2001, pp. 540-542.
- [67] N. M. Sgouros and S. Kousidou, Generation and Implementation of Mixed-Reality, Narrative Performances Involving Robotic Actors, in: Proceedings of the International Conference on Virtual Storytelling, Avignon, France, 2001, pp. 69-80.
- [68] N. M. Sgouros, G. Papakonstantinou and P. Tsanakas, A Framework for Plot Control in Interactive Story Systems, in: Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), Portland OR, AAAI/MIT Press, 1996, pp. 162-167.

- [69] F. Shipman, R. Airhart, H. Hsieh, P. Maloor, J.M. Moore and D. Shah, Visual and Spatial Communication and Task Organization in the Visual Knowledge Builder, in: Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work, Boulder, CO, 2001, pp 260-269.
- [70] R. Stuart and P. Norvig, Artificial Intelligence: A Modern Approach - 2nd Edition, Prentice Hall, Upper Saddle River, NJ, 2002.
- [71] W. Swartout, R. Hill, J. Gratch, W. L. Johnson, C. Kyriakakis, C. LaBore, R. Lindheim, S. Marsella, D. Miraglia, B. Moore, J. Morie, J. Rickel, M. Thieboux, L. Tuch, R. Whitney and J. Douglas, Toward the Holodeck: Integrating Graphics, Sound, Character and Story, in: Proceedings of the Autonomous Agents Conference, 2001, Montreal, Canada, page 409-416.
- [72] A. Tettamanzi, Evolutionary algorithms and fuzzy logic: A two-way integration, in: Proceedings of the 2nd Annual Joint Conference on Information Sciences, Wrightsville Beach, NC, 1995, pp. 464-467.
- [73] A. Tettamanzi and M. Tomassini, Soft computing: Integrating Evolutionary, Neural and Fuzzy Systems. Springer-Verlag, Berlin, Germany, 2001.
- [74] E. Urbina and R. Furuta, Proyecto Cervantes 2001.
<http://www.csd.tamu.edu/cervantes/spanish/> [viewed on May 14, 2006]

- [75] WatchPoint Media. <http://www.watchpointmedia.com> [viewed on May 14, 2006]
- [76] J. Yen and R. Langari, *Fuzzy Logic, Intelligence, Control and Information*, Prentice Hall, Upper Saddle River, NJ, 1998.
- [77] M. R. Young, *An Overview of the Mimesis Architecture: Integrating Narrative Control into a Gaming Environment*, in: *AAAI Spring Symposium in Artificial Intelligence and Interactive Entertainment*, AAAI Press, Menlo Park, 2001.
- [78] M. R. Young, *Creating Interactive Narrative Structures: The Potential for AI Approaches*, in: *AAAI Spring Symposium in Artificial Intelligence and Interactive Entertainment*, AAAI Press, Menlo Park, 2000.
- [79] C. Zimmer, *Evolution: The Triumph of an Idea*, Harper Perennial, London, United Kingdom, 2001.

APPENDIX A STORY AND AGENT SCRIPTING FUNCTIONS

A.1 Functions

In the IAE, function calls are differentiated from other commands by the preceding '\$' symbol attached to the name of functions. With the help of these functions, authors can request random values, initialize variable values, request input from the user and use other functionality from HEFTI that is difficult to implement via scripts alone. HEFTI supports two types of functions: normal functions and agent control functions. The listing below gives all of the functions that can be used in the engine:

1. set(<variable>, <value>) – Assigns <value> to <variable>
2. getuserinput(<random variable>, <value>) – Provides an input box for user to select an element from the random variable.
3. do(<action instance>) – Executes a particular action instance
4. remove(<variable>, <random variable>) – Removes <variable> from the list in <random variable>
5. randf() – Generates a random floating point number between 0 and 1
6. rand() – Generates a random integer value
7. add(<variable>, <random variable>) – Appends <variable> into the list in <random variable>

Note: Although authors can invoke these functions anywhere in the XML files, the results of these functions might be redundant in certain cases, for instance, invoking the “do” function in story templates provides the same function as a reference call “{<action instance>}.”

A.2 Agent Control Functions

Agent control functions are used throughout a story to control agent animations and dialogues. A small set of functions are supported by the engine to: 1) replay an animation selected from the agent’s animation repertoire; 2) speak the corresponding dialogues in the story; 3) control visibility of an agent; 4) move to a particular location on the screen;

5) alter the background scene; and, 6) associate background music to a particular scene.

Definitions of the scripting functions and their corresponding parameters are as follows:

1. \$anim(<agent_name>, 0, <animation>,<synchronization>,<numbers>) or \$anim(<agent_name>,<animation>) – Agent animation functions that can be used to play a predefined animation of the agent.
2. \$moveto(<agent_name>,x,y) – Moves an agent from its current location to x, y.
3. \$relative_move(x,y) – Moves an agent by x pixels and y pixels from its current position
4. \$show(<agent_name>, x, y) – Makes the <agent_name> visible on the screen at the screen coordinate x, y
5. \$delay(<agent_name>, m) – Pauses <agent_name> by m milliseconds before executing the next action script
6. \$say(<agent_name>,<text>) – Makes <agent_name> speaks <text>
7. \$hide(<agent_name>) – Hides the <agent_name>
8. \$playsong(<filename or string reference>) – Plays the given song as the background music for this scene.
9. \$setbackground(<filename or string reference>) – Sets the image that is to be used as background scene.
10. \$getParam(<offset>) – Allows author to retrieval scene parameters that are referenced in various story elements.
11. \$getPosX(<agent_name>), \$getPosY(<agent_name>) – Retrieves the x and y coordinates of an agent.

To prevent confusion between scripting functions and other functions in story files, scripting functions can only be invoked from the scripts.xml file. This configuration allows authors to invoke agent scripts by referencing the scripts' id within the story files, for instance “|scene_1|.” Parameters can be passed to a particular script in the “|” closures, such as “|scene_1, {currentPig.Name} |”, and the \$getParam function can be used in the script to retrieve the value of the variable currentPig:

```
“$anim($getParam(1),”DoSomething”);”
```

The statement above resolves the name of an agent referenced by `{currentPig.Name}` and replays the animation *DoSomething* with the *\$anim* function in script *scene_1*.

APPENDIX B TASK SHEET FOR THE USABILITY STUDY

Task Sheet

Authoring Interactive Stories

Subject ID: _____

Re-reading the same story can be boring, so dynamic stories with multiple story branches and different endings are created to introduce randomness/variations. Today, you will be using the authoring tools in the Integrated Authoring Environment (IAE) to try to introduce three story variations into the Three Little Pigs children's story. The basic story structure, story elements and agent characters are already created to get you started. The three variations of the story are stored in three separate folders using the naming scheme: SubjectID\VariationX. Let me know if you need help or have questions. Please feel free to take as many breaks as you want to. Try to save often so your work is not lost.

Part I: Authoring Story Variations (Total time: 2 hours):

A) Learn about the IAE. Follow the tutorial provided (30 minutes).

1. Story elements
2. Story templates
3. Story thread
4. Agent characters
5. Story rules
6. The authoring interface
7. Functions
8. Executing the Three Little Pigs' story
9. Spend a few minutes exploring the authoring tools and understanding the story elements in the given story (e.g. inspecting attributes of the various story elements)

Please take a 5 minute break.

B) Authoring story variation A: Randomize the houses that the pigs (agent characters) build (15 minutes):

1. Open the story stored in folder SubjectID\VariationA
2. Create a random variable and name it randomhouse
3. Set the elements attribute of randomhouse to reference the ID of the house objects (obj_1,obj_2 and obj_3).
4. Create a new attribute owns in the pig agent objects (agt_2, agt_3 and agt_4). This will be used to associate a house object to the particular pig agent.
5. Locate story template 2 (ID: tmplt_2) in the IAE.
6. In sequence 2, 3 and 4 of tmplt_2, initialize the owns attribute of each of the pig agents with the randomhouse variable. (hint: use the \$set function to set the owns attribute of the pig agents to an element from the randomhouse variable).
7. Modify story text in sequence 2 of tmplt_2 so that the materials used to build the houses are read by the narrator. (hint: use the ‘{ & }’ operators to dereference an object or attribute).
8. Run the story and observe the story variations.
9. Save your work in the same folder: SubjectID\VariationA.

Please take a 5 minute break.

C) Authoring story variation B: Randomize how the pigs move in to their new homes (20 minutes):

1. Open the story stored in folder SubjectID\VariationB.
2. II.Create three action elements and name them act_buildHouse, act_buyHouse, and act_findHouse.
3. III.Populate the action elements (act_buildHouse, act_buyHouse, and act_findHouse) with story text and variables to describe how the current pig agent (referenced in the currentPig variable) built, bought or found a new home. (For example, your story text might read like: {currentpig.value.name} bought a {currentpig.value.owns} in plain text)

4. Create a random variable and name it buildingRandomHouse.
5. Set the elements attribute of buildingRandomHouse to reference the ID of the action elements created in step IV (namely act_buildHouse, act_buyHouse, and act_findHouse)
6. Modify Sequence 2, 3 and 4 of Story template 1 (tmplt_1):
 - i. Dereference the buildingRandomHouse variable and in each of the template sequences
 - ii. Hint: The resultant template sequence X might read like: \$set(current-pig.value,ag_X.id) {ag_X.name} left the mother pig and {building-RandomHouse} in plain text.
7. Run the story and observe the story variations.
8. Save your work in the same folder: SubjectID\VariationB.

Please take a 5 minute break.

D) Authoring story variation C: Create story variations by introducing new story template (15 minutes):

1. Open the story stored in folder SubjectID\VariationC.
2. Create a new story template in the Introduction story stage and assign tmplt_99 as its ID.
3. Set tmplt_2's type attribute as Introduction.
4. Create four template sequences in tmplt_2.
5. Copy the layout of the story elements in tmplt_1.
6. Alter the story text in tmplt_2.
7. Run the story and observe the story variations.
8. Save your work in the same folder: SubjectID\VariationC.

Please take a 5 minute break.

E) Answer questions from Part I of the questionnaire (20 minutes).

Part II: Authoring a New Short Story (2 hours):

You have just finished introducing variation into the three little pigs' story. Having better understanding of the authoring tools in the IAE, you are now ready to author a short story with the IAE from scratch. Author a short story (with some variations) based on the given Little Red Riding Hood children's story book. Try to save often so your work is not lost.

A) Plan your story (40 minutes).

1. There are four story stages (Introduction, RedRidingHoodMeetsTheWolf, WolfVisitsGranny and Conclusion). Determine the story text that goes in the story stages.
2. Name the agent characters in your story: Little red riding hood, her mother, her grandmother, the wolf, and the hunter.
3. Briefly think about the story elements for your story, for example:
 - i. Her mother made a red riding hood for little red riding hood.
 - ii. Her mother baked some bread and wine and asked little red riding hood to take them to her grandmother.
 - iii. Little red riding hood meets the wolf and the wolf asks her where she is heading.
4. Briefly think about the story rules you want to use, for example:
 - i. If little red riding hood is polite to the hunter then make the hunter show up later in the story to rescue her from the wolf.
 - ii. There is a 30% probability that little red riding hood might meet the wolf. If she meets the wolf, then he will try to eat her grandmother otherwise she will arrive at her grandmother's little cabin and have a great dinner that night.

Please take a 5 minute break.

B) Author your story using the IAE (50 minutes)

1. Load the empty story folder: SubjectID\NewStory.
2. Create the 5 agent characters (little red riding hood, her mother, her grandmother, wolf, and the hunter) and assign each of them a unique ID.
3. Create the story elements and rules you have listed in part 1.
4. Create four story templates for the four story stages and create a template sequence for each story paragraph:
 - i. Introduction - Provides information about characters in the story.
 - ii. RedRidingHoodMeetsTheWolf – Describes how the wolf interacts with the little red riding hood.
 - iii. WolfVisitsGanny – Describes how the wolf interacts with the little red riding hood’s grandmother.
 - iv. Conclusion – Ends the story by informing the readers about what happened to the wolf and moral of the story.
5. Populate the story templates with story text and story elements.
6. Run the story.
7. Save your work in the folder: SubjectID\NewStory.

Please take a 5 minute break.

C) Answer questions from Part II of the questionnaire (20 minutes).

APPENDIX C QUESTIONNAIRE FOR THE USABILITY STUDY**Pool of questions for the Questionnaire****Authoring Interactive Stories**

Subject ID: _____

Part I: Authoring Story Variations

1. Please briefly describe the steps/processes you went through in planning your story variation.

2. The authoring tools in the IAE helped me visualize the story elements.

1 2 3 4 5 6 7
Strongly disagree Neutral Strongly agree

3. It was easy for me to modify the given story after receiving sufficient training about the authoring tools.

1 2 3 4 5 6 7
Strongly disagree Neutral Strongly agree

4. It was easy to locate the attributes/properties of a certain story element referenced in the story with the authoring tools.

1 2 3 4 5 6 7
Strongly disagree Neutral Strongly agree

5. It was easy for me to visualize and understand the given story structure (plots, events and character interactions) with the IAE.

1 2 3 4 5 6 7
Strongly disagree Neutral Strongly agree

6. Would you use the IAE to read interactive stories that others have written? Why or why not?

Part II: Authoring a New Short Story

1. Please briefly describe the steps/processes you go through in planning your story.

2. The display of story templates in panels of different color allows me to plan my story in terms of various story stages.

1 2 3 4 5 6 7
 Strongly disagree Neutral Strongly agree

3. Organization of the various types of story elements (agents, objects, scripts and rules) in the IAE allows me to locate the information I need.

1 2 3 4 5 6 7
 Strongly disagree Neutral Strongly agree

4. Was authoring your story more like computer programming or building a web page with a visual editor (such as Microsoft Frontpage)? Please explain.

5. What kind of authoring activity could you not perform with the authoring tools? Please explain.

6. Is there a particular form of story that could not be expressed using the current IAE features? Please explain.

7. What are the features that you like about the authoring tools in IAE? Please explain.

8. What are the features that you dislike about the authoring tools in IAE? Please explain.

9. I enjoyed authoring stories with the IAE.

1	2	3	4	5	6	7
Strongly disagree			Neutral		Strongly agree	

APPENDIX D USABILITY STUDY RESULTS**Table D.1 Minutes it took the test subjects to complete the assigned tasks**

Task/Minutes took to complete	Test subject A	Test subject B
Part I: A (Tutorial)	32	35
Part I: B (Variation A)	17	11
Part I: C (Variation B)	18	26
Part I: D (Variation C)	14	15
Part II: A (Planning a new story)	23	29
Part II: B (Authoring a new story)	32	21

APPENDIX E TEST RESULTS FROM EVALUATION

Table E.1 Matching elements measure for 1,000 story elements set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	1146862	728117	452764	363093	250474	192903	180967	162870	156583
4	860146	612493	401927	312810	200379	182357	162882	139910	139320
5	696718	591372	368126	297462	182714	147443	142162	128323	126816
6	643112	420721	330192	259687	159821	125326	131920	117214	114134
7	564063	368123	290982	229874	147423	100268	112938	96431	106145
8	548938	329470	281811	210289	132812	91223	90871	96017	92346
9	477423	287168	240187	201727	111234	88202	87315	94764	78494
10	461966	271644	237349	198722	109584	86215	84293	93098	74569

Table E.2 Matching elements measure for 2,500 story elements set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	50978	45098	43927	41748	37711	36581	34012	33119	31816
4	46764	44755	42810	38761	36530	33869	32867	31513	30124
5	43215	39894	37456	35542	33762	32894	31715	30007	28761
6	40772	37654	35907	34321	32975	32566	31853	30176	27731
7	35819	34688	34075	33851	32447	31248	29166	26412	24529
8	32159	30161	29490	28650	27865	26730	25997	25873	24397
9	29071	28993	27535	26021	25758	23992	23015	22904	21878
10	27162	26009	25858	25086	24571	23043	22666	21839	20959

Table E.3 Matching elements measure for 5,000 story elements set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	18186	16945	15617	15133	14495	14157	13878	13252	12760
4	17652	16555	15538	14378	13435	12826	12072	11587	10564
5	15513	14794	13269	12869	12055	11951	11503	11438	10763
6	14880	13481	12816	11901	10714	10634	9995	9722	9552
7	12951	12066	11070	10934	10458	9779	9517	9120	8908
8	10187	9956	9782	9698	9507	9250	8931	8658	8440
9	9785	9522	9490	9290	9067	8716	8407	8034	7832
10	9624	9320	9145	8982	8726	8599	8318	8012	7751

Table E.4 Longest contiguous element measure for 1,000 story elements set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	8	6	6	6	6	5	5	4	4
4	8	7	6	6	6	5	5	4	4
5	7	6	6	6	5	5	4	4	3
6	7	6	6	5	5	4	4	4	3
7	7	7	6	6	5	5	4	3	3
8	7	6	6	5	5	4	3	3	3
9	7	6	7	5	5	4	3	2	2
10	6	5	6	5	4	4	3	3	2

Table E.5 Longest contiguous element measure for 2,500 story elements set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	5	5	5	4	4	3	2	2	2
4	5	5	5	4	4	3	3	2	2
5	5	5	4	4	4	3	3	2	2
6	5	5	4	4	3	3	3	2	2
7	4	4	4	4	3	3	3	2	2
8	4	4	4	4	3	3	3	2	2
9	4	4	4	3	3	3	2	2	2
10	3	3	3	3	2	2	2	2	2

Table E.6 Longest contiguous element measure for 5,000 story elements set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	3	3	3	3	2	2	2	2	2
4	3	3	3	3	2	2	2	2	2
5	3	3	3	2	2	2	2	2	2
6	3	3	3	2	2	2	3	2	2
7	3	3	3	2	2	2	2	2	2
8	2	2	2	2	2	3	2	2	2
9	2	2	2	2	2	2	2	2	2
10	2	2	2	3	2	2	2	2	2

Table E.7 Average contiguous measure for 1,000 story element set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	6.3244	5.1345	5.2027	5.094	5.3016	4.1985	4.2082	3.6812	3.4792
4	6.7261	6.0138	5.4838	4.1209	4.8113	3.1892	3.9549	2.9125	3.0734
5	6.1849	5.8172	5.2953	5.3018	4.2185	4.013	2.8778	2.154	2.0987
6	5.8876	5.2873	5.1276	4.3298	4.8573	3.5828	3.0718	2.3957	2.1478
7	5.5783	5.4989	5.1263	5.2987	4.3859	3.0852	2.8573	2.1275	2.1876
8	5.2348	4.6983	4.8705	4.1037	4.0765	3.6742	2.386	2.1594	2.0852
9	5.472	5.1983	5.8982	3.8854	3.9522	3.0286	2.2957	1.6885	1.5894
10	4.9458	4.5039	4.6582	3.9087	3.6749	3.1947	2.2398	1.9674	1.4

Table E.8 Average contiguous measure for 2,500 story element set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	3.4824	3.2459	3.1921	2.8253	2.3598	2.0825	1.354	1.5438	1.682
4	3.101	3.0684	3.1244	3.1761	2.7614	2.4647	2.4807	1.8068	1.7614
5	3.3426	3.7631	2.9981	2.1635	2.8102	2.1216	1.7338	1.0284	1.144
6	3.2655	3.4896	3.1875	2.475	1.9366	1.5795	1.3675	1.0727	0.9487
7	3.3756	3.258	2.7495	3.1486	2.0743	1.9573	1.5063	0.8524	0.4898
8	3.2071	3.1086	2.8208	3.0149	2.7983	2.3075	1.7466	1.1852	0.5876
9	2.3833	2.2571	2.9175	2.6723	1.4594	1.2386	0.9592	0.5832	0.3566
10	2.1759	1.8957	1.5847	1.8763	1.1493	0.9473	1.0655	0.6703	0.473

Table E.9 Average contiguous measure for 5,000 story element set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	2.0174	1.7388	1.2875	1.1985	0.9873	0.7499	0.5987	0.4792	0.4866
4	1.9376	1.8462	1.3957	1.2987	1.0776	0.7536	0.6365	0.4074	0.3898
5	1.7784	1.4083	1.8574	0.9738	0.7589	0.5112	0.4126	0.5693	0.5887
6	2.491	1.949	1.6585	1.1383	0.5786	0.5191	0.4757	0.5098	0.4927
7	1.2765	0.9742	1.9674	0.9927	0.8764	0.624	0.5276	0.4988	0.3706
8	1.4775	0.5877	1.1875	0.7679	0.7509	0.5168	0.6388	0.377	0.2376
9	1.3652	0.8345	0.9372	0.7092	0.8988	0.7392	0.5876	0.3582	0.3873
10	1.1005	0.6508	0.7699	0.6699	0.4728	0.4896	0.4512	0.5987	0.3728

Table E.10 Lower and upper bounds of the 1,000 story element set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	4.6488	4.269	4.4054	4.188	4.6032	3.397	3.4164	3.3624	2.9584
3	6.3244	5.1345	5.2027	5.094	5.3016	4.1985	4.2082	3.6812	3.4792
3U	8	6	6	6	6	5	5	4	4
4L	5.4522	5.0276	4.9676	2.2418	3.6226	1.3784	2.9098	1.825	2.1468
4	6.7261	6.0138	5.4838	4.1209	4.8113	3.1892	3.9549	2.9125	3.0734
4U	8	7	6	6	6	5	5	4	4
5L	5.36978	5.6345	4.5906	4.6037	3.4371	3.026	1.7556	0.308	1.1974
5	6.18489	5.8172	5.2953	5.3018	4.2185	4.013	2.8778	2.154	2.0987
5U	7	6	6	6	5	5	4	4	3
6L	4.7752	4.5746	4.2552	3.6596	4.7146	3.1656	2.1435	0.7914	1.2956
6	5.8876	5.2873	5.1276	4.3298	4.8573	3.5828	3.0718	2.3957	2.1478
6U	7	6	6	5	5	4	4	4	3
7L	4.1566	3.9978	4.2526	4.5974	3.7718	1.1704	1.7146	1.255	1.3752
7	5.5783	5.4989	5.1263	5.2987	4.3859	3.0852	2.8573	2.1275	2.1876
7U	7	7	6	6	5	5	4	3	3
8L	3.4696	3.3966	3.741	3.2074	3.153	3.3484	1.7719	1.3188	1.1704
8	5.2348	4.6983	4.8705	4.1037	4.0765	3.6742	2.386	2.1594	2.0852
8U	7	6	6	5	5	4	3	3	3
9L	3.944	4.3966	4.7964	2.7708	2.9044	2.0571	1.5914	1.377	1.1787
9	5.472	5.1983	5.8982	3.8854	3.9522	3.0286	2.2957	1.6885	1.5894
9U	7	6	7	5	5	4	3	2	2
10L	3.8916	4.0078	3.3164	2.8175	3.3499	2.3894	1.4796	0.9348	0.8
10	4.9458	4.5039	4.6582	3.9087	3.6749	3.1947	2.2398	1.9674	1.4
10U	6	5	6	5	4	4	3	3	2

Table E.11 Lower and upper bounds of the 2,500 story element set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	1.9648	1.4918	1.3842	1.6506	0.7196	1.165	0.708	1.0876	1.364
3	3.4824	3.2459	3.1921	2.8253	2.3598	2.0825	1.354	1.5438	1.682
3U	5	5	5	4	4	3	2	2	2
4L	1.202	1.1368	1.2488	2.3522	1.5228	1.9294	1.9614	1.6136	1.5228
4	3.101	3.0684	3.1244	3.1761	2.7614	2.4647	2.4807	1.8068	1.7614
4U	5	5	5	4	4	3	3	2	2
5L	1.6852	2.5262	1.9962	0.327	1.6204	1.2432	0.4676	0.0568	0.288
5	3.3426	3.7631	2.9981	2.1635	2.8102	2.1216	1.7338	1.0284	1.144
5U	5	5	4	4	4	3	3	2	2
6L	1.53096	1.9792	2.375	0.95	0.8733	0.159	0	0.1453	0
6	3.26548	3.4896	3.1875	2.475	1.9366	1.5795	1.3675	1.0727	0.9487
6U	5	5	4	4	3	3	3	2	2
7L	2.75118	2.516	1.499	2.2972	1.1486	0.9145	0.0125	0	0
7	3.37559	3.258	2.7495	3.1486	2.0743	1.9573	1.5063	0.8524	0.4898
7U	4	4	4	4	3	3	3	2	2
8L	2.4142	2.2171	1.6416	2.0298	2.5966	1.6151	0.4932	0.3704	0
8	3.2071	3.1086	2.8208	3.0149	2.7983	2.3075	1.7466	1.1852	0.5876
8U	4	4	4	4	3	3	3	2	2
9L	0.7666	0.5142	1.835	2.3446	0	0	0	0	0
9	2.3833	2.2571	2.9175	2.6723	1.4594	1.2386	0.9592	0.5832	0.3566
9U	4	4	4	3	3	3	2	2	2
10L	1.3518	0.7914	0.1694	0.7526	0.2985	0	0.1309	0	0
10	2.1759	1.8957	1.5847	1.8763	1.1493	0.9473	1.0655	0.6703	0.473
10U	3	3	3	3	2	2	2	2	2

Table E.12 Lower and upper bounds of the 5,000 story element set

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	1.03479	0.4775	0	0	0	0	0	0	0
3	2.0174	1.7388	1.2875	1.1985	0.9873	0.7499	0.5987	0.4792	0.4866
3U	3	3	3	3	2	2	2	2	2
4L	0.8752	0.6924	0	0	0.1553	0	0	0	0
4	1.9376	1.8462	1.3957	1.2987	1.0776	0.7536	0.6365	0.4074	0.3898
4U	3	3	3	3	2	2	2	2	2
5L	0.5568	0	0.7148	0	0	0	0	0	0
5	1.7784	1.4083	1.8574	0.9738	0.7589	0.5112	0.4126	0.5693	0.5887
5U	3	3	3	2	2	2	2	2	2
6L	1.98208	0.8979	0.3169	0.2766	0	0	0	0	0
6	2.49104	1.949	1.6585	1.1383	0.5786	0.5191	0.4757	0.5098	0.4927
6U	3	3	3	2	2	2	3	2	2
7L	0	0	0.9348	0	0	0	0	0	0
7	1.2765	0.9742	1.9674	0.9927	0.8764	0.624	0.5276	0.4988	0.3706
7U	3	3	3	2	2	2	2	2	2
8L	0.9549	0	0.375	0	0	0	0	0	0
8	1.47745	0.5877	1.1875	0.7679	0.7509	0.5168	0.6388	0.377	0.2376
8U	2	2	2	2	2	3	2	2	2
9L	0.7304	0	0	0	0	0	0	0	0
9	1.3652	0.8345	0.9372	0.7092	0.8988	0.7392	0.5876	0.3582	0.3873
9U	2	2	2	2	2	2	2	2	2
10L	0.201	0	0	0	0	0	0	0	0
10	1.1005	0.6508	0.7699	0.6699	0.4728	0.4896	0.4512	0.5987	0.3728
10U	2	2	2	3	2	2	2	2	2

Table E.13 Matching element measure for 1,000 story elements set (25% forward branching probability)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	653711	488972	362312	328091	287236	177765	142642	118762	86434
4	598273	487433	347843	293837	283464	165983	118474	98483	65553
5	541234	447370	354954	275933	265044	145949	94983	78398	78472
6	489243	388348	298931	248999	224339	158328	103375	77477	69273
7	429821	358395	264950	219831	193803	149874	83873	75763	66112
8	352875	319832	228733	187534	163593	127653	92727	72321	59742
9	309276	279439	234932	176453	157463	118872	81624	72029	62874
10	251937	234984	218723	173622	128745	127323	88234	71024	65884

Table E.14 Matching element measure for 1,000 story elements set (50% forward branching probability)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	559844	499183	325483	293834	232834	152432	126533	89433	68373
4	483838	442545	300563	276353	217645	128564	95874	67372	70658
5	418776	398741	294985	247663	176685	128749	78780	50383	49569
6	386736	375333	266512	218495	165034	118764	78223	47273	45487
7	328544	319858	277854	208212	159875	109568	59683	53291	51028
8	257404	271007	252556	182165	140987	92876	83872	61276	47493
9	217446	183876	198744	176040	110955	78765	61452	52835	44875
10	177487	156872	179856	149600	103754	85642	66342	50198	41049

Table E.15 Matching element measure for 1,000 story elements set (75% forward branching probability)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	483801	374945	305894	281221	201289	128713	85291	52389	38487
4	369873	339890	270951	222875	176450	119832	75914	50412	35142
5	295990	289813	259545	206978	154663	90466	74009	48134	33235
6	249853	250980	222985	195932	166984	105858	72198	46453	33985
7	203876	195576	176761	145941	129839	97546	77749	47983	30548
8	175988	151311	139879	130027	110098	89842	69833	45090	31953
9	149586	130958	128585	110493	120851	94872	75872	48493	30562
10	157623	146798	120985	110107	104099	93154	65878	47876	31094

Table E.16 Matching element measure for 2,500 story elements set (25% forward branching probability)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	46298	40219	37334	34997	32376	29885	27765	26609	24433
4	44095	41767	38987	34841	32048	28513	25876	24614	24119
5	42191	39865	37352	33984	31576	28908	25652	23548	20987
6	38897	36874	35587	32041	30487	27921	24987	21158	19754
7	36164	33598	32090	29187	27717	25873	23398	20958	18876
8	31216	29001	28843	27741	25116	23017	21006	19481	17743
9	27897	25471	24409	23386	22059	21995	18365	17736	17998
10	23855	22099	22294	20335	21476	19687	19874	18853	18123

Table E.17 Matching element measure for 2,500 story elements set (50% forward branching probability)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	43549	40348	36761	32216	30762	27520	25190	23951	20731
4	42085	39090	35923	31176	28472	25582	23783	21950	19747
5	40731	37487	33195	30465	26563	23763	21656	20489	19037
6	36701	35876	32013	27853	24987	23865	19583	18738	17467
7	32509	31987	28933	25476	22012	20304	17573	16693	15651
8	28175	26411	24840	23865	19927	18484	16320	16923	15018
9	24762	22989	21053	20031	18587	16473	15733	14859	14902
10	21468	20074	19234	18473	18853	17983	16368	15873	14387

Table E.18 Matching element measure for 2,500 story elements set (75% forward branching probability)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	38987	38239	36540	31764	28233	25571	24402	20853	18458
4	38065	37341	32542	28561	26674	23654	20803	17465	16539
5	35461	30487	34453	28969	25455	21982	19687	16552	17574
6	31154	28583	29068	28594	24759	19934	17463	17462	16487
7	28547	25576	27658	26852	22950	18763	16709	15863	15549
8	25986	23545	24051	23359	20905	16543	15638	15087	14876
9	23805	20954	19510	20941	16251	16257	15820	14873	13846
10	19948	16754	17609	15674	14875	15989	13865	14542	13098

Table E.19 Matching element measure for 5,000 story elements set (25% forward branching probability)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	16546	15012	14866	14732	13982	12428	10729	9965	9238
4	15874	15195	13659	13887	11295	12129	9284	9015	8744
5	15963	13811	11850	9988	9431	8965	8385	8526	8251
6	14092	13751	9051	8746	8351	8455	8481	8365	7865
7	12254	11903	9184	8882	8165	8076	7913	7814	7243
8	9643	8642	7916	8164	8047	7781	7853	7609	6916
9	8244	7951	7434	7585	7494	7293	7509	7375	7016
10	7587	7581	7265	7364	7065	6893	6716	6991	6887

Table E.20 Matching element measure for 5,000 story elements set (50% forward branching probability)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	14741	12766	10875	10573	9552	10736	9653	9870	9615
4	12865	11058	9636	10875	9814	9766	9565	9225	9184
5	10927	9892	9754	9456	9044	9221	8982	9043	8874
6	9544	9565	9307	9176	8674	8867	8724	8241	8542
7	8423	8487	8417	8296	8147	8071	7554	7221	7061
8	7534	7368	7264	7074	7154	6943	6817	6961	6873
9	7003	7149	7048	7184	7008	6812	6859	6515	6156
10	6761	6354	6606	6476	6073	5960	5835	5901	5886

Table E.21 Matching element measure for 5,000 story elements set (75% forward branching probability)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	13788	11659	10965	9712	10545	9856	9524	9038	8656
4	12063	10873	9970	9542	9351	9156	8838	8434	8362
5	9872	9771	9245	9003	8843	8456	8056	7380	7066
6	7365	7528	7276	7105	7038	7154	7197	7217	6873
7	8246	7983	7828	7714	7525	7396	7058	6997	6567
8	7060	7116	7065	6846	6641	6550	6372	6375	6121
9	6491	6228	6277	6182	6039	5823	5636	5739	5736
10	5963	5823	6056	5576	5719	5825	5681	5788	5712

Table E.22 Longest contiguous element measure for 1,000 story elements set (25% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	7	6	6	5	5	5	4	4	4
4	7	6	6	5	5	5	4	3	3
5	7	7	6	5	4	5	4	3	3
6	7	6	5	6	5	4	4	3	3
7	7	7	6	5	5	4	3	3	3
8	6	6	5	5	5	4	3	3	2
9	7	5	5	5	4	4	3	2	2
10	6	5	5	5	4	4	3	2	2

Table E.23 Longest contiguous element measure for 1,000 story elements set (50% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	7	7	6	6	5	5	4	3	3
4	7	7	6	5	5	5	4	3	3
5	7	6	6	6	5	5	4	4	3
6	6	6	6	6	5	5	4	3	3
7	6	6	6	5	5	4	3	3	2
8	6	6	5	5	5	4	3	3	2
9	6	5	5	5	5	4	4	3	2
10	5	5	5	5	4	4	3	2	2

Table E.24 Longest contiguous element measure for 1,000 story elements set (75% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	6	6	5	5	4	4	4	3	3
4	6	6	6	5	4	4	4	3	3
5	6	6	6	5	4	4	3	3	3
6	6	6	5	4	4	4	3	3	3
7	6	6	6	5	4	3	3	3	2
8	5	5	5	5	4	3	3	3	2
9	5	5	5	5	4	3	3	3	2
10	5	4	4	4	3	2	3	2	2

Table E.25 Longest contiguous element measure for 2,500 story elements set (25% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	5	5	4	4	4	3	3	2	2
4	5	4	4	4	3	2	3	2	2
5	4	4	4	3	3	3	2	2	2
6	4	4	4	3	3	3	2	2	2
7	4	4	4	4	3	3	2	2	2
8	4	3	4	3	3	3	2	2	2
9	3	3	3	2	2	2	2	2	2
10	3	3	3	3	2	2	2	2	2

Table E.26 Longest contiguous element measure for 2,500 story elements set (50% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	5	4	4	4	3	3	3	2	2
4	4	4	4	3	3	3	2	2	2
5	5	5	4	4	3	3	2	2	2
6	4	4	4	3	3	2	3	2	2
7	4	3	3	3	3	2	2	2	2
8	3	3	2	3	2	2	2	2	2
9	3	3	3	3	3	2	2	2	2
10	3	3	3	3	2	2	2	2	2

Table E.27 Longest contiguous element measure for 2,500 story elements set (75% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	5	5	4	4	3	3	3	3	2
4	4	4	4	3	3	3	2	2	2
5	4	4	4	3	3	2	2	2	2
6	4	4	3	3	2	2	2	2	1
7	5	5	4	2	3	2	2	2	2
8	3	3	4	3	2	2	2	2	2
9	3	3	2	3	2	2	2	2	1
10	3	3	3	3	2	2	2	1	1

Table E.28 Longest contiguous element measure for 5,000 story elements set (25% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	3	3	3	2	2	2	2	2	2
4	3	3	3	2	2	2	2	1	2
5	2	2	3	2	2	2	2	2	2
6	2	2	2	2	2	2	2	2	2
7	2	2	2	2	2	3	2	2	1
8	2	2	3	2	2	2	2	2	2
9	2	2	2	2	2	2	2	2	2
10	2	3	2	2	2	2	2	1	1

Table E.29 Longest contiguous element measure for 5,000 story elements set (50% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	3	3	2	2	2	2	2	2	2
4	2	3	3	2	2	2	2	2	2
5	3	2	2	3	2	2	2	2	2
6	2	2	2	2	3	2	2	2	1
7	2	2	2	2	2	2	2	2	2
8	2	2	2	2	2	1	2	2	2
9	2	2	2	2	2	2	2	2	2
10	3	2	2	3	2	2	1	2	2

Table E.30 Longest contiguous element measure for 5,000 story elements set (75% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	3	2	2	2	2	2	2	2	2
4	3	2	2	2	2	2	2	2	2
5	2	2	2	2	3	2	3	2	2
6	2	2	2	3	2	2	2	2	2
7	3	2	2	2	2	2	2	2	2
8	2	2	2	2	2	2	2	2	2
9	2	2	2	2	2	2	2	1	2
10	2	2	2	2	1	2	2	1	2

Table E.31 Average contiguous measure for 1,000 story element set (25% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	5.13004	5.0895	5.0775	4.9504	4.66	3.8213	2.8212	2.6609	2.8
4	5.1706	5.1622	5.0224	4.8163	4.2056	3.998	3.5446	2.1629	2.4349
5	5.05676	4.9241	4.9407	4.8272	3.1817	3.8685	3.169	2.754	2.2096
6	5.06744	4.938	4.2207	4.8156	4.0732	3.8169	3.3712	2.1517	2.1145
7	4.98939	4.8599	4.8164	4.3157	4.2078	2.9171	2.563	2.122	2.0726
8	4.81832	4.7628	4.3819	4.3843	3.9126	2.8127	2.6547	1.6123	1.6298
9	4.9712	4.717	4.286	4.1435	3.8723	3.024	2.4012	1.4512	1.4349
10	4.83081	4.2178	4.6501	4.377	3.2562	3.1889	2.101	1.4374	1.0962

Table E.32 Average contiguous measure for 1,000 story element set (50% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	5.10324	5.1004	4.8702	4.7337	4.3995	4.3548	3.0707	2.6949	1.9421
4	5.3457	5.1645	4.9473	4.3252	4.0695	4.1835	3.487	2.3953	2.1723
5	5.1402	5.1039	4.8294	4.5151	4.1375	3.6593	3.0445	2.8196	2.1511
6	4.9191	4.8694	4.7635	4.6253	3.9563	3.6063	3.2236	1.888	1.2509
7	4.8935	4.7868	4.862	3.9421	3.6398	2.806	2.4657	1.6939	1.2299
8	4.871	4.3389	4.3222	3.5117	3.7063	2.7931	2.1865	1.7758	1.3516
9	4.7324	4.2036	4.2159	3.2289	3.256	2.8654	2.626	1.9496	1.2087
10	4.10707	4.102	4.0939	3.8263	3.0594	2.4173	2.1632	1.2947	1.2224

Table E.33 Average contiguous measure for 1,000 story element set (75% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	4.72025	4.3637	3.8866	3.716	3.2212	3.0232	2.8472	2.6946	2.0615
4	4.5442	4.2066	4.2265	3.7891	3.0419	2.8737	2.8244	1.7101	1.9071
5	4.30441	4.2304	4.5977	3.6982	2.8659	2.9686	2.2278	2.1469	1.973
6	4.31747	4.2262	3.7168	3.0051	2.9815	2.7323	2.1313	2.2065	1.8969
7	4.10042	4.0255	4.4122	3.2214	2.7837	2.2248	1.7775	1.3133	1.2237
8	4.02176	3.8304	4.0578	3.1959	2.3228	2.0784	1.7874	1.6568	1.3406
9	3.91267	3.1646	3.6805	3.2478	2.8365	1.8643	1.4932	1.3667	1.2694
10	4.02025	3.0206	3.1304	2.99	2.6862	1.3322	2.0164	1.1247	1.371

Table E.34 Average contiguous measure for 2,500 story element set (25% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	4.1198	3.8426	3.3764	3.2804	2.8683	2.0308	1.6613	1.3815	1.2221
4	4.2758	3.2758	3.445	3.1756	2.1065	1.4106	1.5886	1.3184	1.2014
5	3.2265	3.1454	3.1054	2.497	2.3089	1.7758	1.4631	1.3159	1.3075
6	3.0568	2.7655	2.8145	2.7132	2.1894	1.861	1.318	1.2256	1.2457
7	2.8427	3.0792	3.0011	3.1273	1.9168	1.8883	1.3921	1.2882	1.3083
8	2.7585	2.2931	2.6065	2.1486	1.8459	1.5924	1.202	1.308	1.2357
9	1.9842	2.0132	2.1145	1.6104	1.3089	1.2468	1.2411	1.1853	1.2167
10	1.878	2.1166	1.9319	2.0356	1.2337	1.2918	1.3263	1.2723	1.2358

Table E.35 Average contiguous measure for 2,500 story element set (50% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	4.17521	3.2106	2.7438	3.1253	2.1778	1.9688	2.0775	1.6604	1.5753
4	3.8159	2.9728	2.8086	2.4746	2.0178	2.0175	1.7188	1.514	1.5273
5	3.97572	3.238	2.6574	3.0521	2.1968	2.1634	1.6711	1.5129	1.464
6	3.2358	3.0454	2.7281	2.6366	2.1047	1.4697	1.6321	1.3628	1.4183
7	3.1686	2.2125	1.9812	2.0394	2.1178	1.7843	1.4229	1.5329	1.5046
8	2.0752	1.8505	1.6423	2.181	1.4196	1.6065	1.3711	1.3813	1.6007
9	1.8868	1.7282	1.638	1.802	1.8803	1.7773	1.6241	1.6115	1.3273
10	1.68612	1.7632	1.5742	2.1181	1.571	1.5383	1.5348	1.4117	1.3814

Table E.36 Average contiguous measure for 2,500 story element set (75% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	4.3894	4.2138	3.4273	3.348	2.7446	2.5369	2.4144	2.1473	1.5249
4	3.7317	3.4521	3.2112	2.6978	2.5823	2.6973	1.5632	1.5529	1.6382
5	3.5974	3.1127	3.0141	2.7018	2.445	1.4852	1.4048	1.3876	1.524
6	3.446	3.1413	2.6336	2.5361	1.5823	1.3009	1.3272	1.4071	0.8906
7	4.1407	3.9655	3.7021	1.5339	2.3697	1.6092	1.6272	1.5529	1.4737
8	2.2132	2.365	2.9602	2.3319	1.382	1.446	1.4104	1.679	1.3752
9	2.2865	2.2133	1.4923	2.2283	1.4611	1.5377	1.3729	1.417	0.7375
10	2.4125	2.273	2.2323	2.1116	1.3385	1.4887	1.3892	0.8392	0.729

Table E.37 Average contiguous measure for 5,000 story element set (25% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	2.4079	2.3448	2.119	1.6289	1.5898	1.6573	1.5026	1.7502	1.7707
4	2.4443	2.2326	2.0534	1.7325	1.7138	1.6779	1.4806	0.5368	1.6255
5	1.8252	1.6332	2.1265	1.782	1.6248	1.5014	1.7172	1.6371	1.6084
6	1.7941	1.6302	1.8326	1.6782	1.6812	1.5544	1.7937	1.7172	1.6315
7	1.6843	1.5187	1.6238	1.5438	1.5502	2.5617	1.5513	1.6937	0.7183
8	1.5825	1.4109	2.2655	1.3469	1.3008	1.5736	1.4806	1.5518	1.4593
9	1.5203	1.658	1.782	1.5306	1.5173	1.524	1.5718	1.536	1.52
10	1.4079	2.4685	1.7809	1.6398	1.6798	1.4261	1.4793	0.5695	0.6842

Table E.38 Average contiguous measure for 5,000 story element set (50% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	2.4911	2.4303	1.471	1.4321	1.5642	1.6777	1.6034	1.4323	1.4401
4	1.6896	2.2317	2.5424	1.8003	1.6324	1.6434	1.4668	1.539	1.5923
5	2.758	1.7113	1.7611	2.4906	1.7386	1.4921	1.4361	1.4638	1.3823
6	1.6973	1.6508	1.5344	1.6143	2.428	1.5306	1.5668	1.6739	0.758
7	1.6367	1.5071	1.4526	1.5224	1.7867	1.601	1.6068	1.5441	1.5982
8	1.7095	1.6752	1.6091	1.4427	1.5836	0.8043	1.4819	1.6345	1.6782
9	1.6114	1.5303	1.5982	1.5932	1.6078	1.5908	1.6371	1.5737	1.5061
10	2.4754	1.6982	1.7159	1.7061	1.6432	1.6336	0.7364	1.6348	1.7017

Table E.39 Average contiguous measure for 5,000 story element set (75% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3	2.2465	1.7489	1.761	1.6102	1.7481	1.5809	1.6223	1.5092	1.4336
4	2.328	1.6771	1.5975	1.5593	1.4205	1.6147	1.4183	1.4637	1.4659
5	1.5018	1.689	1.4194	1.5006	2.6976	1.7141	2.7508	1.689	1.4423
6	1.6976	1.6285	1.7287	2.6056	1.6247	1.5482	1.5277	1.5203	1.4204
7	2.7124	1.599	1.4911	1.4654	1.4502	1.55	1.5061	1.4209	1.404
8	1.5671	1.4343	1.4871	1.5873	1.4668	1.4286	1.4537	1.5919	1.5175
9	1.6779	1.6743	1.6218	1.6371	1.5112	1.4236	1.3393	0.8133	1.366
10	1.5674	1.5109	1.5216	1.5591	0.7903	1.3849	1.4173	0.7657	1.3402

Table E.40 Lower and upper bounds of the 1,000 story element set (25% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	3.26007	4.179	4.155	4.9008	4.3199	2.6426	1.6424	1.3218	1.6
3	5.13004	5.0895	5.0775	4.9504	4.66	3.8213	2.8212	2.6609	2.8
3U	7	6	6	5	5	5	4	4	4
4L	3.3412	4.3244	4.0449	4.6326	3.4112	2.996	3.0892	1.3258	1.8698
4	5.1706	5.1622	5.0224	4.8163	4.2056	3.998	3.5446	2.1629	2.4349
4U	7	6	6	5	5	5	4	3	3
5L	3.11352	2.8481	3.8813	4.6543	2.3634	2.737	2.338	2.508	1.4192
5	5.05676	4.9241	4.9407	4.8272	3.1817	3.8685	3.169	2.754	2.2096
5U	7	7	6	5	4	5	4	3	3
6L	3.13488	3.876	3.4414	3.6312	3.1463	3.6338	2.7424	1.3034	1.229
6	5.06744	4.938	4.2207	4.8156	4.0732	3.8169	3.3712	2.1517	2.1145
6U	7	6	5	6	5	4	4	3	3
7L	2.97878	2.7198	3.6328	3.6314	3.4156	1.8342	2.126	1.244	1.1452
7	4.98939	4.8599	4.8164	4.3157	4.2078	2.9171	2.563	2.122	2.0726
7U	7	7	6	5	5	4	3	3	3
8L	3.63663	3.5256	3.7638	3.7686	2.8252	1.6254	2.3094	0.2246	1.2596
8	4.81832	4.7628	4.3819	4.3843	3.9126	2.8127	2.6547	1.6123	1.6298
8U	6	6	5	5	5	4	3	3	2
9L	2.9424	4.434	3.572	3.287	3.7447	2.048	1.8024	0.9024	0.8698
9	4.9712	4.717	4.286	4.1435	3.8723	3.024	2.4012	1.4512	1.4349
9U	7	5	5	5	4	4	3	2	2
10L	3.66162	3.4356	4.3002	3.754	2.5124	2.3778	1.202	0.8748	0.1924
10	4.83081	4.2178	4.6501	4.377	3.2562	3.1889	2.101	1.4374	1.0962
10U	6	5	5	5	4	4	3	2	2

Table E.41 Lower and upper bounds of the 1,000 story element set (50% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	3.20648	3.2008	3.7404	3.4674	3.7991	3.7095	2.1414	2.3898	0.8842
3	5.10324	5.1004	4.8702	4.7337	4.3995	4.3548	3.0707	2.6949	1.9421
3U	7	7	6	6	5	5	4	3	3
4L	3.6914	3.329	3.8947	3.6503	3.139	3.367	2.974	1.7906	1.3446
4	5.3457	5.1645	4.9473	4.3252	4.0695	4.1835	3.487	2.3953	2.1723
4U	7	7	6	5	5	5	4	3	3
5L	3.2804	4.2077	3.6588	3.0302	3.275	2.3186	2.089	1.6392	1.3022
5	5.1402	5.1039	4.8294	4.5151	4.1375	3.6593	3.0445	2.8196	2.1511
5U	7	6	6	6	5	5	4	4	3
6L	3.8382	3.7387	3.5271	3.2506	2.9126	2.2126	2.4472	0.776	-0.498
6	4.9191	4.8694	4.7635	4.6253	3.9563	3.6063	3.2236	1.888	1.2509
6U	6	6	6	6	5	5	4	3	3
7L	3.787	3.5736	3.724	2.8842	2.2796	1.612	1.9314	0.3878	0.4598
7	4.8935	4.7868	4.862	3.9421	3.6398	2.806	2.4657	1.6939	1.2299
7U	6	6	6	5	5	4	3	3	2
8L	3.742	2.6777	3.6444	2.0234	2.4125	1.5862	1.373	0.5516	0.7032
8	4.871	4.3389	4.3222	3.5117	3.7063	2.7931	2.1865	1.7758	1.3516
8U	6	6	5	5	5	4	3	3	2
9L	3.4648	3.4072	3.4318	1.4578	1.512	1.7308	1.252	0.8992	0.4174
9	4.7324	4.2036	4.2159	3.2289	3.256	2.8654	2.626	1.9496	1.2087
9U	6	5	5	5	5	4	4	3	2
10L	3.21414	3.204	3.1878	2.6526	2.1188	0.8346	1.3264	0.5894	0.4448
10	4.10707	4.102	4.0939	3.8263	3.0594	2.4173	2.1632	1.2947	1.2224
10U	5	5	5	5	4	4	3	2	2

Table E.42 Lower and upper bounds of the 1,000 story element set (75% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	3.4405	2.7273	2.7732	2.432	2.4424	2.0463	1.6944	2.3892	1.123
3	4.72025	4.3637	3.8866	3.716	3.2212	3.0232	2.8472	2.6946	2.0615
3U	6	6	5	5	4	4	4	3	3
4L	3.0884	2.4132	2.453	2.5782	2.0838	1.7473	1.6488	0.4202	0.8142
4	4.5442	4.2066	4.2265	3.7891	3.0419	2.8737	2.8244	1.7101	1.9071
4U	6	6	6	5	4	4	4	3	3
5L	2.60882	2.4609	3.1953	2.3963	1.7318	1.9372	1.4556	1.2938	0.9459
5	4.30441	4.2304	4.5977	3.6982	2.8659	2.9686	2.2278	2.1469	1.973
5U	6	6	6	5	4	4	3	3	3
6L	2.63494	2.4523	2.4336	2.0102	1.963	1.4646	1.2627	1.413	0.7938
6	4.31747	4.2262	3.7168	3.0051	2.9815	2.7323	2.1313	2.2065	1.8969
6U	6	6	5	4	4	4	3	3	3
7L	2.20084	2.051	2.8244	1.4428	1.5674	1.4496	0.555	-0.373	0.4474
7	4.10042	4.0255	4.4122	3.2214	2.7837	2.2248	1.7775	1.3133	1.2237
7U	6	6	6	5	4	3	3	3	2
8L	3.04352	2.6608	3.1156	1.3917	0.6456	1.1567	0.5748	0.3136	0.6813
8	4.02176	3.8304	4.0578	3.1959	2.3228	2.0784	1.7874	1.6568	1.3406
8U	5	5	5	5	4	3	3	3	2
9L	2.82534	1.3292	2.361	1.4956	1.673	0.7286	-0.014	-0.267	0.5389
9	3.91267	3.1646	3.6805	3.2478	2.8365	1.8643	1.4932	1.3667	1.2694
9U	5	5	5	5	4	3	3	3	2
10L	3.0405	2.0412	2.2608	1.98	2.3724	0.6644	1.0328	0.2495	0.742
10	4.02025	3.0206	3.1304	2.99	2.6862	1.3322	2.0164	1.1247	1.371
10U	5	4	4	4	3	2	3	2	2

Table E.43 Lower and upper bounds of the 2,500 story element set (25% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	3.2396	2.6852	2.7528	2.5608	1.7366	1.0616	0.3226	0.763	0.4442
3	4.1198	3.8426	3.3764	3.2804	2.8683	2.0308	1.6613	1.3815	1.2221
3U	5	5	4	4	4	3	3	2	2
4L	3.5516	2.5516	2.89	2.3512	1.213	0.8212	0.1772	0.6368	0.4028
4	4.2758	3.2758	3.445	3.1756	2.1065	1.4106	1.5886	1.3184	1.2014
4U	5	4	4	4	3	2	3	2	2
5L	2.453	2.2907	2.2108	1.994	1.6178	0.5516	0.9262	0.6318	0.615
5	3.2265	3.1454	3.1054	2.497	2.3089	1.7758	1.4631	1.3159	1.3075
5U	4	4	4	3	3	3	2	2	2
6L	2.1136	1.531	1.629	2.4264	1.3787	0.722	0.636	0.4512	0.4914
6	3.0568	2.7655	2.8145	2.7132	2.1894	1.861	1.318	1.2256	1.2457
6U	4	4	4	3	3	3	2	2	2
7L	1.6854	2.1584	2.0022	2.2546	0.8336	0.7766	0.7842	0.5764	0.6166
7	2.8427	3.0792	3.0011	3.1273	1.9168	1.8883	1.3921	1.2882	1.3083
7U	4	4	4	4	3	3	2	2	2
8L	1.517	1.5862	1.213	1.2972	0.6918	0.1848	0.404	0.616	0.4714
8	2.7585	2.2931	2.6065	2.1486	1.8459	1.5924	1.202	1.308	1.2357
8U	4	3	4	3	3	3	2	2	2
9L	0.9684	1.0264	1.2291	1.2208	0.6178	0.4936	0.4822	0.3706	0.4334
9	1.9842	2.0132	2.1145	1.6104	1.3089	1.2468	1.2411	1.1853	1.2167
9U	3	3	3	2	2	2	2	2	2
10L	0.756	1.2332	0.8638	1.0712	0.4674	0.5836	0.6526	0.5445	0.4716
10	1.878	2.1166	1.9319	2.0356	1.2337	1.2918	1.3263	1.2723	1.2358
10U	3	3	3	3	2	2	2	2	2

Table E.44 Lower and upper bounds of the 2,500 story element set (50% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	3.35042	2.4212	1.4876	2.2506	1.3556	0.9376	1.155	1.3208	1.1506
3	4.17521	3.2106	2.7438	3.1253	2.1778	1.9688	2.0775	1.6604	1.5753
3U	5	4	4	4	3	3	3	2	2
4L	3.6318	1.9456	1.6172	1.9492	1.0356	1.0349	1.4376	1.028	1.0546
4	3.8159	2.9728	2.8086	2.4746	2.0178	2.0175	1.7188	1.514	1.5273
4U	4	4	4	3	3	3	2	2	2
5L	2.95144	1.476	1.3148	2.1042	1.3936	1.3268	1.3422	1.0258	0.928
5	3.97572	3.238	2.6574	3.0521	2.1968	2.1634	1.6711	1.5129	1.464
5U	5	5	4	4	3	3	2	2	2
6L	2.4716	2.0908	1.4562	2.2732	1.2093	0.9394	0.2641	0.7256	0.8366
6	3.2358	3.0454	2.7281	2.6366	2.1047	1.4697	1.6321	1.3628	1.4183
6U	4	4	4	3	3	2	3	2	2
7L	2.3372	1.425	0.9624	1.0789	1.2357	1.5686	0.8458	1.0658	1.0092
7	3.1686	2.2125	1.9812	2.0394	2.1178	1.7843	1.4229	1.5329	1.5046
7U	4	3	3	3	3	2	2	2	2
8L	1.1504	0.7009	1.2847	1.362	0.8392	1.213	0.7422	0.7626	1.2014
8	2.0752	1.8505	1.6423	2.181	1.4196	1.6065	1.3711	1.3813	1.6007
8U	3	3	2	3	2	2	2	2	2
9L	0.7736	0.4564	0.276	0.604	0.7606	1.5546	1.2482	1.223	0.6546
9	1.8868	1.7282	1.638	1.802	1.8803	1.7773	1.6241	1.6115	1.3273
9U	3	3	3	3	3	2	2	2	2
10L	0.37224	0.5264	0.1484	1.2362	1.142	1.0765	1.0696	0.8234	0.7628
10	1.68612	1.7632	1.5742	2.1181	1.571	1.5383	1.5348	1.4117	1.3814
10U	3	3	3	3	2	2	2	2	2

Table E.45 Lower and upper bounds of the 2,500 story element set (75% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	3.7788	3.4276	2.8546	2.696	2.4892	2.0738	1.8288	1.2946	1.0498
3	4.3894	4.2138	3.4273	3.348	2.7446	2.5369	2.4144	2.1473	1.5249
3U	5	5	4	4	3	3	3	3	2
4L	3.4634	2.9042	2.4224	2.3956	2.1646	2.3946	1.1264	1.1058	1.2764
4	3.7317	3.4521	3.2112	2.6978	2.5823	2.6973	1.5632	1.5529	1.6382
4U	4	4	4	3	3	3	2	2	2
5L	3.1948	2.2254	2.0282	2.4036	1.89	0.9704	0.8096	0.7752	1.048
5	3.5974	3.1127	3.0141	2.7018	2.445	1.4852	1.4048	1.3876	1.524
5U	4	4	4	3	3	2	2	2	2
6L	2.892	2.2826	2.2672	2.0722	1.1646	0.6018	0.6544	0.8142	0.7812
6	3.446	3.1413	2.6336	2.5361	1.5823	1.3009	1.3272	1.4071	0.8906
6U	4	4	3	3	2	2	2	2	1
7L	3.2814	2.931	3.4042	1.0678	1.7394	1.2184	1.2544	1.1058	0.9474
7	4.1407	3.9655	3.7021	1.5339	2.3697	1.6092	1.6272	1.5529	1.4737
7U	5	5	4	2	3	2	2	2	2
8L	1.4264	1.73	1.9204	1.6638	0.764	0.892	0.8208	1.358	0.7504
8	2.2132	2.365	2.9602	2.3319	1.382	1.446	1.4104	1.679	1.3752
8U	3	3	4	3	2	2	2	2	2
9L	1.573	1.4266	0.9846	1.4566	0.9222	1.0754	0.7458	0.834	0.475
9	2.2865	2.2133	1.4923	2.2283	1.4611	1.5377	1.3729	1.417	0.7375
9U	3	3	2	3	2	2	2	2	1
10L	1.825	1.546	1.4646	1.2232	0.677	0.9774	0.7784	0.6784	0.458
10	2.4125	2.273	2.2323	2.1116	1.3385	1.4887	1.3892	0.8392	0.729
10U	3	3	3	3	2	2	2	1	1

Table E.46 Lower and upper bounds of the 5,000 story element set (25% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	1.8158	1.6896	1.238	1.2578	1.1796	1.3146	1.0052	1.5004	1.5414
3	2.4079	2.3448	2.119	1.6289	1.5898	1.6573	1.5026	1.7502	1.7707
3U	3	3	3	2	2	2	2	2	2
4L	1.8886	1.4652	1.1068	1.465	1.4276	1.3558	0.9612	0.0736	1.251
4	2.4443	2.2326	2.0534	1.7325	1.7138	1.6779	1.4806	0.5368	1.6255
4U	3	3	3	2	2	2	2	1	2
5L	1.6504	1.2664	1.253	1.564	1.2496	1.0028	1.4344	1.2742	1.2168
5	1.8252	1.6332	2.1265	1.782	1.6248	1.5014	1.7172	1.6371	1.6084
5U	2	2	3	2	2	2	2	2	2
6L	1.5882	1.2604	1.6652	1.3564	1.3624	1.1088	1.5874	1.4344	1.263
6	1.7941	1.6302	1.8326	1.6782	1.6812	1.5544	1.7937	1.7172	1.6315
6U	2	2	2	2	2	2	2	2	2
7L	1.3686	1.0374	1.2476	1.0876	1.1004	2.1234	1.1026	1.3874	0.4366
7	1.6843	1.5187	1.6238	1.5438	1.5502	2.5617	1.5513	1.6937	0.7183
7U	2	2	2	2	2	3	2	2	1
8L	1.165	0.8218	1.531	0.6938	0.6016	1.1472	0.9612	1.1036	0.9186
8	1.5825	1.4109	2.2655	1.3469	1.3008	1.5736	1.4806	1.5518	1.4593
8U	2	2	3	2	2	2	2	2	2
9L	1.0406	1.316	1.564	1.0612	1.0346	1.048	1.1436	1.072	1.04
9	1.5203	1.658	1.782	1.5306	1.5173	1.524	1.5718	1.536	1.52
9U	2	2	2	2	2	2	2	2	2
10L	0.8158	1.937	1.5618	1.2796	1.3596	0.8522	0.9586	0.139	0.3684
10	1.4079	2.4685	1.7809	1.6398	1.6798	1.4261	1.4793	0.5695	0.6842
10U	2	3	2	2	2	2	2	1	1

Table E.47 Lower and upper bounds of the 5,000 story element set (50% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	1.9822	1.8606	0.942	0.8642	1.1284	1.3554	1.2068	0.8646	0.8802
3	2.4911	2.4303	1.471	1.4321	1.5642	1.6777	1.6034	1.4323	1.4401
3U	3	3	2	2	2	2	2	2	2
4L	1.3792	1.4634	2.0848	1.6006	1.2648	1.2868	0.9336	1.078	1.1846
4	1.6896	2.2317	2.5424	1.8003	1.6324	1.6434	1.4668	1.539	1.5923
4U	2	3	3	2	2	2	2	2	2
5L	2.516	1.4226	1.5222	1.9812	1.4772	0.9842	0.8722	0.9276	0.7646
5	2.758	1.7113	1.7611	2.4906	1.7386	1.4921	1.4361	1.4638	1.3823
5U	3	2	2	3	2	2	2	2	2
6L	1.3946	1.3016	1.0688	1.2286	1.856	1.0612	1.1336	1.3478	0.516
6	1.6973	1.6508	1.5344	1.6143	2.428	1.5306	1.5668	1.6739	0.758
6U	2	2	2	2	3	2	2	2	1
7L	1.2734	1.0142	0.9052	1.0448	1.5734	1.202	1.2136	1.0882	1.1964
7	1.6367	1.5071	1.4526	1.5224	1.7867	1.601	1.6068	1.5441	1.5982
7U	2	2	2	2	2	2	2	2	2
8L	1.419	1.3504	1.2182	0.8854	1.1672	0.6086	0.9638	1.269	1.3564
8	1.7095	1.6752	1.6091	1.4427	1.5836	0.8043	1.4819	1.6345	1.6782
8U	2	2	2	2	2	1	2	2	2
9L	1.2228	1.0606	1.1964	1.1864	1.2156	1.1816	1.2742	1.1474	1.0122
9	1.6114	1.5303	1.5982	1.5932	1.6078	1.5908	1.6371	1.5737	1.5061
9U	2	2	2	2	2	2	2	2	2
10L	1.9508	1.3964	1.4318	0.4122	1.2864	1.2672	0.4728	1.2696	1.4034
10	2.4754	1.6982	1.7159	1.7061	1.6432	1.6336	0.7364	1.6348	1.7017
10U	3	2	2	3	2	2	1	2	2

Table E.48 Lower and upper bounds of the 5,000 story element set (75% forward branching)

distribution/ branching	10%	20%	30%	40%	50%	60%	70%	80%	90%
3L	1.493	1.4978	1.522	1.2204	1.4962	1.1618	1.2446	1.0184	0.8672
3	2.2465	1.7489	1.761	1.6102	1.7481	1.5809	1.6223	1.5092	1.4336
3U	3	2	2	2	2	2	2	2	2
4L	1.656	1.3542	1.195	1.1186	0.841	1.2294	0.8366	0.9274	0.9318
4	2.328	1.6771	1.5975	1.5593	1.4205	1.6147	1.4183	1.4637	1.4659
4U	3	2	2	2	2	2	2	2	2
5L	1.0036	1.378	0.8388	1.0012	2.3952	1.4282	2.5016	1.378	0.8846
5	1.5018	1.689	1.4194	1.5006	2.6976	1.7141	2.7508	1.689	1.4423
5U	2	2	2	2	3	2	3	2	2
6L	1.3952	1.257	1.4574	2.2112	1.2494	1.0964	1.0554	1.0406	0.8408
6	1.6976	1.6285	1.7287	2.6056	1.6247	1.5482	1.5277	1.5203	1.4204
6U	2	2	2	3	2	2	2	2	2
7L	2.4248	1.198	0.9822	0.9308	0.9004	1.1	1.0122	0.8418	0.808
7	2.7124	1.599	1.4911	1.4654	1.4502	1.55	1.5061	1.4209	1.404
7U	3	2	2	2	2	2	2	2	2
8L	1.1342	0.8686	0.9742	1.1746	0.9336	0.8572	0.9074	1.1838	1.035
8	1.5671	1.4343	1.4871	1.5873	1.4668	1.4286	1.4537	1.5919	1.5175
8U	2	2	2	2	2	2	2	2	2
9L	1.3558	1.3486	1.2436	1.2742	1.0224	0.8472	0.6786	0.6266	0.732
9	1.6779	1.6743	1.6218	1.6371	1.5112	1.4236	1.3393	0.8133	1.366
9U	2	2	2	2	2	2	2	1	2
10L	1.1348	1.0218	1.0432	1.1182	0.5805	0.7698	0.8346	0.5314	0.6804
10	1.5674	1.5109	1.5216	1.5591	0.7903	1.3849	1.4173	0.7657	1.3402
10U	2	2	2	2	1	2	2	1	2

VITA

Name: Teong Joo Ong

Address: Department of Computer Science

Texas A&M University

College Station, TX 77843-3112

Email Address: teongjoo@gmail.com

Education: B.S., Computer and Information Science, University of Oregon, 1998

M.C.S., Computer Science, Texas A&M University, 2001