ComicKit: Knowledge Acquisition of Story Scripts

by

Ryan Duane Williams

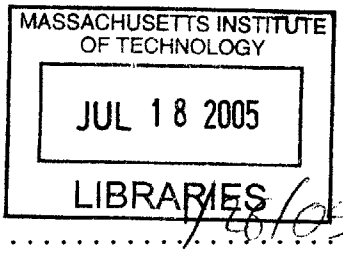
Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2005



Author .....
Department of Electrical Engineering and Computer Science
Jan 28, 2005

Certified by .....
Glorianna Davenport
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

ComicKit: Knowledge Acquisition of Story Scripts

by

Ryan Duane Williams

Submitted to the Department of Electrical Engineering and Computer Science
on Jan 28, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

The field of Artificial Intelligence stands poised to make great leaps in emulating human intelligence. The development of a way to acquire and use common sense is the key to this advancement. This thesis describes the design and construction of ComicKit, a tool for acquiring story scripts for a common sense knowledge base. ComicKit's core mechanism is dragging and dropping icons into the panels of a comic. As the user creates a comic, ComicKit makes common sense suggestions for the story. ComicKit follows on the successful model of Open Mind Common Sense, a web-based activity that thousands of people around the world contributed bits of common sense knowledge to. It is hoped that many people will contribute a few hundred thousand stories through ComicKit, and build a large corpus of common sense story knowledge.

Thesis Supervisor: Glorianna Davenport
Title: Principal Research Scientist

Acknowledgments

Many thanks are due to Glorianna Davenport for providing advice and support where it was needed. It is much appreciated that Chuck Dages at Warner Brothers Interactive Entertainment supported this work. Thanks to Barbara Barry for giving me lots of indispensable advice, and helping me stay on track. Push Singh helped me to better understand common sense and human intelligence, which came in handy.

Contents

List of Figures

Chapter 1

Introduction

The fifty-year-old promise of machines as intelligent as human beings has not yet come to pass. A wide variety of factors have contributed to this situation. Initially computers were not nearly powerful enough to solve the problems that human beings can. The difficulty terrain of Artificial Intelligence (AI) problems has also not been kind. Tasks that are simple and automatic for human beings, such as distinguishing between two objects visually, still haven't been solved satisfactorily.

Current progress in AI has led to a huge variety of techniques, each of which tackles its own particular problem. There is little commonality between these techniques, or a way to cobble them together into a being that thinks like a human[8]. In part, this situation stems from the natural tendency of AI reserachers to limit the scope of the problem they they are looking at. It appears that human-level AI will not be achieved without some approach that tackles all of the problem at once.

Marvin Minsky, among others, champions common sense computing as the approach that will lead to human-level AI[10]. Common sense computing is based on the idea that meaning and understanding both come from the ability to provide complex associations to other ideas. A common sense based solution relies on building a large number of such associations.

This thesis describes the building and design of ComicKit, a knowledge-acquisition tool for common-sense story scripts. Story scripts are a very specific type of narrative, one whose purpose is to teach a machine a small piece of common sense. ComicKit is

an application that users interact with over the Web to create stories. These stories become the basis for a corpus of common sense knowledge called StoryNet.

ComicKit uses a distributed knowledge capture method that was shown to be effective in the Open Mind Common Sense project (OMCS) [1]. OMCS succeeded at collecting about 700,000 simple commonsense facts from 15,000 contributors around the world. ComicKit takes the same approach – casual, non-expert internet users contribute commonsense knowledge via simple and engaging knowledge acquisition activities on a public website.

To engage these users, ComicKit uses a drag and drop comic format that is easy to learn and very expressive. In the base activity, users drag panels into a sequence, and organize icons into scenes and events. Each story thus created is communicated to a server which maintains StoryNet.

An innovative aspect of ComicKit is its use of existing common sense knowledge bases to help the user in creating the story. Such common sense feedback makes the story creation process more interactive[19].

1.1 Common Sense

Every person has some idea of what common sense is. It's what your mother told you that you didn't have. It's the collection of knowledge that you've acquired throughout your life that allows you to interact meaningfully with the world. Common sense is the knowledge you get through experience. Common sense allows you to know what to do when you're out of food (go to the store) or when you drop a glass (vacuum it up), and where to find an extra roll of toilet paper at a stranger's house (under the sink).

More formally, John McCarthy defines common sense thusly:

> Common-sense knowledge includes the basic facts about events (including actions) and their effects, facts about knowledge and how it is obtained,

[1] `http://openmind.media.mit.edu/`

facts about beliefs and desires. It also includes the basic facts about material objects and their properties.[7]

1.1.1 Artificial Common Sense

Artificial common sense is the attempt to teach computers the large corpus of knowledge that all humans acquire through the process of growing up. The common sense corpus is enormous, and takes many years for each person to learn (and some people, it seems, never learn common sense).

Part of the difficulty of acquiring common sense knowledge for computers is that human beings do not have a very good grasp of the scope of their own common sense knowledge. Describing it as a series of assertions, e.g., "the floor is beneath your feet," "buildings protect us from the wind and rain," "beer and water are liquids," shows how closely common sense knowledge is tied to our internal models of the world.

The Common Sense Computing project [2] at the Media Lab is an effort to give machines the ability to think about the world similarly to the way people do. One aspect of this project is the creation of broad knowledge databases. The databases are coupled with inference tools that allow AI to make sophisticated inferences about the state of the world, problem solving, and the meaning of various inputs. There are a huge variety of ways that common sense knowledge can advance the state of the art in a variety of fields – a few are listed here.

1.1.2 Applications

A wide variety of present and future problems can be addressed with common sense. This list attempts to show the breadth of potential applications of common sense story knowledge, and is not exhaustive.

- Detecting terrorists: A large corpus of common-sense knowledge could help to discover terrorist-exploitable weaknesses in an infrastructure or analyze a pattern of behavior to detect a potential attack. It would help answer questions such

[2] `http://csc.media.mit.edu`

as "if a terrorist gains access to a water purifying facility, what items could he use in a damaging way?"

- Search (Web, data mining, research): One of the problems with current search techniques is that they require the user to know something about the knowledge they are seeking.
- Computer Vision: One approach to computer vision involves working forwards from an internal representation of an object, attempting to locate the object on the scene via pattern-matching. Currently the approach is not feasible for general object recognition because of the enormous number of objects that it would have to try and discard. Common sense ties related objects together, so once a knife is identified, the vision system will quickly look for knives, plates, spoons, and glasses, instead of wasting its time trying to match computers, cars, or trains.
- Game Characters: A robust common sense system would enable computer game developers to quickly create smart non-scripted AI characters. Such characters could respond to novel circumstances in a reasonable way. Suppose your hunting party is running out of arrows, but is far from a town – the common sense characters may decide to cut off some branches and make some more.
- Step towards true AI: Common sense is a step towards developing programs that can think like human beings. Many special-purpose AIs have been developed to emulate human beings at any one particular task. Emulating humans at a variety of tasks requires the sort of large corpus of general knowledge that common sense can grant. In addition to the corpus of knowledge, which is useless on its own, common sense techniques can provide inference methods to draw conclusions and make decisions based on the knowledge.

1.1.3 Difficulty of Artificial Common Sense

Common sense is difficult even for human beings. Two of the problems of developing common sense are acquisition and utilization.

Children take years to learn a usable body of common sense knowledge. A median estimate of the amount of common sense required for a human being to go about their daily business hovers around ten million elements of knowledge[11]. This is a lot of knowledge, and worse, this counts only the useful knowledge. Knowledge that is useless, or redundant, or wrong, or inconsistent, or insufficient, must be discarded.

When you start acquiring knowledge, you suddenly start to face the problem of how to store it, and reference it. Common sense knowledge takes a huge variety of forms, from simple facts to entire naive physics models. Common sense knowledge also includes meta-knowledge about the quality of other knowledge, and about good sources of knowledge, and knowledge about how to acquire knowledge. Keeping all of these types of knowledge in a commonly-accessible format is very challenging. The Cyc project developed an extremely complex and thorough knowledge representation system, perhaps the most complete ever[5]. The Common Sense group intends to use standard English fragments for their projects, but even then we've built various different structures for sorting and coordinating these fragments.

Once you've acquired and stored a ton of knowledge, the problem becomes one of using that knowledge effectively. It's not impossible to develop a program that will answer a certain type of question or solve a certain type of problem effectively. However, it remains an open problem to develop systems that can identify the problem at hand, and choose a strategy to solve it.

1.2 Open Mind

Open Mind Commonsense[3] is a website developed to address the acquisition of common sense knowledge[14]. Its approach is to get a large number of people to provide semi-structured knowledge. The site contains a variety of activities based around filling in the blanks in templates such as "A ____ is often used for ____". The input text is processed later and entered into a database of common sense. This

Because participants on the site are volunteers, essentially dabblers, the site has

[3] `http://openmind.media.mit.edu`

been made in such a way as to encourage people to return often and enter more information. The variety of ways users can input knowledge encourages them to return to play with more. Often the activities allow the user to see what other people have entered previously, providing a small reward for their participation. Though there is not a strong emphasis on making the site fun, it has been successful at pulling in knowledge. Roughly 15,000 users have entered some information at the OMCS site, and the top contributors have entered tens of thousands of bits of knowledge each.

1.3 The X-Nets

The Common Sense group at the Media Lab has already produced several knowledge bases and inference methods. The "-Net" terminology to describe the pairing of a knowledge base with tools to use the information usefully is based on the name of the WordNet project at Princeton [4]. Thus far the Common Sense group and its collaborators has developed ConceptNet, LifeNet, ShapeNet, GoalNet, and StoryNet. Each Net addresses a different aspect of common sense knowledge.

1.3.1 ConceptNet

The common sense contained in ConceptNet is a graph-like series of associations between textual nodes. Each node is a 'concept' such as an object or an action, or a state of being. Each pair of nodes can be linked by any of a set of link types. The inference methods used on ConceptNet's knowledge traverse this graph structure, using spreading activation.[6]

Among others, there are edges that relate actions to the objects they can be performed on, edges that generalize from the specific, and edges that mean mere closeness. There are around twenty different edge types.

ConceptNet is quite useful for generating synopses of text blocks, since the graph structure lends itself to conceptual locality. The ConceptNet knowledge is essentially

[4] http://www.cogsci.princeton.edu/~wn/

static in time. It captures relationships such as "toothpaste is found in the bathroom," that convey an immediate, timeless bit of information.

ConceptNet's knowledge base was extracted from the OpenMind dataset. Approximately 1.6 million predicates were mined out of the hundreds of thousands of OpenMind assertions.

1.3.2 LifeNet

LifeNet addresses transitions between states of being. It takes first-person propositions, such as "I am at the airport", and connects them to other propositions, e.g. "I am boarding a plane". Each connection represents a probability of one proposition happening after the other[17]. The links do not necessarily represent causality, or even imply a strict ordering. For example, the propositions "I am outside" and "I can see the sky" are linked with high probability, and don't have any direct causal or temporal relationship.

LifeNet is intended to be used for context-related reasoning. Given some information about a situation, LifeNet can use its knowledge base to suggest what other things might be true, or what sorts of events might happen next.

1.4 StoryNet

StoryNet can be understood as a knowledge base of story scripts. The story scripts serve as a representation for narrative-based knowledge. This knowledge tracks temporal order, goals, and dependencies for resolution. By knowing the sequence of events, StoryNet can begin to understand causality. The goals of the stories tell StoryNet about what states might be desirable, and the way that goals are broken down into subgoals tells it about methods of solving a problem. There is currently no large-scale database of such story knowledge[15].

Stories are useful for case-based reasoning. Given a problem to solve, a case-based reasoner searches through a database of stories to find the ones that solved the most similar problem, and follows along. An important part of following along is

determining the differences between the example story and the current problem that one is trying to solve. Solving a problem by referring to a similar story that solved a similar problem is reasoning by analogy.

Using stories for reasoning does not involve much generalization. For each situation that StoryNet will attempt to reason about, it would need at least one story that is very similar to the situation. Fortunately, StoryNet can rely on the common sense knowledge captured in the other Nets to expand the meaningfulness of a single story. For example, given the story about making a sandwich in the next section, it could generate derivative stories where cheese is replaced by cold cuts, or egg salad.

1.5 Story Scripts

Jerome Bruner makes the case that narratives[5] make up most of human memory and experience[2]. An important aspect of narrative is that it is "irreducibly durative", that the events happening in it must take place over time. The study of narrative is particularly concerned with the construction of a narrative from a story – what elements are included or left out and for what purposes.

A story script is a very particular form of narrative. They are to be distinguished from Schank-Abelson scripts[13], and should be considered a more restricted form of both narrative and script. The purpose of story scripts is to aid common sense representation, in a more tightly constrained way.

Schank and Abelson describe scripts as the fundamental unit of memory. His scripts are sequences of primitive actions, converted to a formalized representation. Like Schank-Abelson scripts, story scripts also consist of a sequence of primitive actions, but there are additional constraints:

- Story scripts presume the existence of a goal or set of goals, and the purpose of the story script is to describe the accomplishment of these goals.
- Story scripts are focused and small.

[5] A *story* is a sequence of events. A *narrative* is the packaging of events for an audience

- Story scripts are didactic.
- Common sense is a major component of story scripts.

Here is a simple example story script:

- Eric took out a loaf of bread
- He saw that his fridge had no cheese in it
- Eric bought some cheese at the corner store
- He then cut slices off of the cheese and put them between bread
- Eric toasted the sandwiches, and ate them.

This simple story contains a lot of information about sandwich-making and the purpose of things like stores. There were two obstacles to the achievement of the goal – the fact that the bread and cheese were not combined or toasted, and the fact that the cheese was still at the store. The steps taken to overcome these obstacles could serve as a template for other sorts of sandwich-making problems, such as not having any lettuce when making a BLT.

1.6 Why comics?

Examine McCloud's definition of comics: "juxtaposed pictorial and other images in deliberate sequence"[9]. This definition emphasizes their sequential nature as well as the deliberation required to make them. These characteristics fit in with the definition of a story script.

Comics as a format have much to lend to the acquisition of stories. Comics are a storytelling medium that most people are extremely comfortable with. A sequential series of images, coupled with text, can convey things that no other medium can. Comics can be very efficient in communicating their ideas.

The comic medium is useful for collecting story scripts because of its sequential nature. Composing a comic is the art of arranging images in a sequence, each of which

conveys an idea. This format exactly matches the definition of story scripts. Casting the story script creation task as a comic endeavour immediately sets the tone.

Visual images are very expressive, and can convey ideas that no words can hope to. By using comics for our knowledge capture tool, we grant the users the expressiveness of images in creating their stories. The major problem with this approach is that computer interpretation of images is far behind computer understanding of words. The ComicKit attempts to get around this problem by ignoring the visual aspect of its user's creations. Words are an essential part of the ComicKit creation process, and thus can serve in lieu of the images.

Chapter 2

Goals of ComicKit

ComicKit is quite ambitious in its goals. It is intended to be the primary knowledge acquisition tool for a new and unprecedented knowledge base. ComicKit should make it easy for non-experts to contribute sophisticated and clean story knowledge to this knowledge base. It needs to be easy for a user to casually try it out, and plausible that a user will spend hours of his or her time working on stories with it. Needless to say, it is supposed to be fun to use.

2.1 Natural Language

Natural language is very difficult to work with. Common sense researchers would be much happier if it were easy, because there are literally tons of pieces of common sense knowledge distributed over the internet, books, and audio conversations. Indeed, some are making attempts to tackle these resources with the limited natural language tools that we have at our disposal now.

The very expressiveness of natural language makes it difficult to convert to a knowledge representation. Natural language is context-heavy – most words change meaning in difference circumstances, and pronoun references are sometimes difficult even for human beings to resolve.

This problem with natural language means that researchers wishing to acquire common sense knowledge must use some domain that is more limited. The Cyc team

developed an elaborate logical language that precisely specifies meaning[5]. The Open Mind Common Sense project got most of its useful data from very rigid templates[16].

One of ComicKit's goals is to constrain the input of its users very tightly, so that they do not stray into complex natural language. It is difficult to do this effectively, since natural language is extremely well suited for telling stories.

The approach we are taking is to break the input we receive into categorizable components. These components alleviate some of the burden of discovering the relationships between words in the story.

2.2 Breadth

Competing with the desire to constrain the user's input is the need to acquire a large breadth of story knowledge. Like all common sense, story knowledge will be most useful if it covers all domains of human experience. One of the concerns we had when designing the ComicKit was that the drag-and-drop interface would unnecessarily limit the scope of the stories people would contribute.

Early user testing suggested that the ComicKit strikes the right balance in this regard. While testers definitely felt limited by ComicKit, they were also inspired to overcome those limitations. Sometimes the limitations of the medium forced a user to improve his or her storytelling, e.g., by breaking one very complex panel into two or more simpler panels that showed the same actions in a clearer way.

2.3 Building Comics

Drag-and-drop comics seem to be a natural match with the goals of a StoryNet acquisition tool. Comics are naturally structured much like the story scripts that we wish to acquire. Building them by compositing them out of pieces takes away a great deal of the complexity that is associated with creating visual images. Complexity issues related to the structure of the story remain, and there is more emphasis placed on the story as a result.

Figure 2-1: Abstract versus representational objects

2.3.1 Visual

One of the debates I had with others in the Common Sense group going into the project was whether or not to use representational images in ther interface. The advantage to using representational images is that they are much more "real" and immediately lend a great deal of visual power to a created comic. Almost all artist-created comics use representational graphics – they are pretty much the *raison d'etre* of comics as a medium.

There are two huge downsides to using representational images for the components of the stories. It would take a very great deal of energy to create or find the thousands of graphics that would be required to represent the wide variety of common-sense items that we wanted people to be able to use to construct stories. Additional effort would also be required to associate common sense terms with the graphics, in order to make the suggestions mechanism work properly (if the suggestion's label was "hammer" and the graphic appeared to be a whale, it would be useless as either). The other problem was that no matter how many graphics we acquired, we would still leave enormous gaps in our coverage of common-sense items. It seemed unlikely to us that users would go too far beyond the concepts contained in a set of graphics.

We elected to ambiguate the appearance of the graphics because we wanted to keep users' imaginations running. If everything looked like a blob, the user would be forced to rely more heavily on the text label to imagine how the story "really happened". There is still a great deal of visual power in the arrangement of the abstract shapes in a panel. Relative position plays a large role, as does the addition of flavor in the text of captions and thought bubbles.

2.4 The user experience

The last challenge is how to make the experience rewarding enough that people continue to enter stories. The OpenMind project has had considerable participation despite lacking explicit incentives. Perhaps users feel altruistic pleasure from furthering the progress of science, or have a desire to make it onto the top 10 list. Whatever the reason, stories are a slightly tougher nut to crack, since creating a story is likely to be more involved than anything currently on OpenMind. Several engagement strategies are suggested by the richer nature of stories:

1. Make stories as easy to create as possible by letting the existing knowledge in StoryNet and ConceptNet provide suggestions and guide the player.

2. Provide a social element to the activity, such as the ability to share comics, or allow people to embed the comics in their web journals, or encourage cooperation/competition by placing comics created by different people in opposition.

3. Create a challenge for the player. The Kit could provide a goal and challenge the player to create a story that achieves it from a given starting condition. On the creation of a new comic, the Kit could declare a subject area for the story to evolve in ("create a story about finding your keys"). Human judges might preside over a tournament of story creation.

At the current state of the program, only Item 1 is being implemented. In order to implement it, the Kit relies heavily on the ConceptNet knowledge to suggest components that are likely to be useful. The intent is to provide the objects and actions the player might need without typing them. If the player uses a dog in a panel, the Kit will suggest a dog. If the panel is set in the park, the Kit will make it easy to use a frisbee or a bench.

This suggestion system also works to keep the player from entering too-complex information for the nascent StoryNet. We've discovered from OpenMind that given an empty text field, people will enter unparseably complicated thoughts. It is understandable that they do so – the level of knowledge we wish to capture with ConceptNet

and StoryNet is below the level of abstraction people normally operate at. In order to reduce the incidence of unparseable input, the Comic Kit should be designed so that it is difficult to create complex stories.

Another way to reduce the amount of effort required to create a story is to have players only author part of the story. OMCS already has a similar concept, allowing players to add a line to an existing story. Players are likely to experience a spirit of cooperation when engaging in this sort of story creation. The OMCS story creation encourages players to 'split' stories by suggesting alternative actions at each step, which tends to prevent stories from getting too long, but also reduces the amount of meaning associated with them. A strategy that make more of an effort to meaningfully involve the player with the currently-unfolding story may yield better retention.

Chapter 3

ComicKit interface

The ComicKit interface has two purposes: for creating stories, and for viewing the stories of others. It has a user management system, which tracks username/password combinations, and associates each story with a username. Once logged in, a user can create a story using the editing interface, the primary thrust of development.

As much as possible, the interface of the Comic Kit is designed so that the user types as little as possible. The primary activity is dragging objects around on a workspace, and it is hoped that sophisticated enough prediction on the part of the program will obviate the need for the user to do anything beyond simply selecting the proper component.

I've broken down the interface into four 'screens' each of which performs a specific function. The login screen handles both new user registration and the authentication of existing users. The gallery screen is a place where users can select stories to edit or view. The viewer scree is for looking at but not editing a story. Most importantly, the editing screen creates comics.

ComicKit is a two-part system: a simple client that handles the user interaction and the graphics, and a server that stores persistent data and computes suggestions for displaying in the client. An interaction with ComicKit is really an interaction with the server mediated by the client. This chapter deals with the interface, also referred to as the client. [insert userloop diagram here]

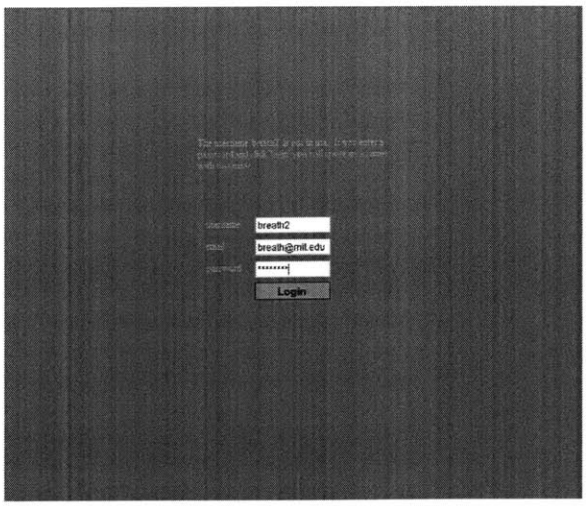


Figure 3-1: Login screen

3.1 Login screen

The login window allows existing users to authenticate themselves, and allows new users to register a new account. Both of these functions are handled in essentially the same way, since they differ very little in terms of the information required from the user.

I decided that it was best to give each account a password to prevent users from vandalizing each other's stories. An email address is also associated with each account, so that if a user forgets her password it can be emailed to her. This is decidedly a low-security affair, and no encryption is used in any stage of the process. I believe there is little incentive for users to attack the security of the system, as there is nothing of inherent value contained therein. The only plausible goal for an attacker is to vandalize or delete a story that someone else made. If the ComicKit explodes in popularity and people start using it as a service to keep personally important information, this issue may need to be revisited.

The login process for an existing user is very simple: fill in the username and password fields appropriately and click "Login". At the moment the interface does not support pressing enter after entering the password – this seems to be a Flash bug.

The login process for a new user (i.e., a "registration") is equally simple: the user types in the desired username, an email address, and a password, and clicks "Login".

ComicKit distinguishes the login of an existing user from the registration of a new user by checking the server to see if the specified username already exists. It does this with each user keystroke, providing a continually updated message alongside the username box, saying "this username is in use", or "this username has not been registered yet". The email address field appears and disappears based on whether or not the username is determined to be new.

Once the user has clicked the 'login' button the Flash client checks with the server to see if the login/password combination is valid. If it is not, the client displays an error message in red. If it is valid, the client takes one of two actions. If the user is new and has just registered, the client shows the editing screen with a new comic opened up. If the user has already registered, they are taken to their gallery.

In anticipation of people who wish to simply create a story without the hassle of registering a username, the login screen provides a "login anonymously" button. Each screen, if evoked on its own without a login process, assumes that the user is logged in as `anonymous`. Currently anonymous users can edit each other's stories, making it less useful for common sense story script due to the possibliilty of vandalism or aborted story creation attempts.

The anonymous username is thus a sort of sandbox for trying out the interface. We may find that this is not advantageous, as the burden of registering (admittedly a small burden) encourages users to be responsible with the stories they create.

3.2 Gallery screen

The gallery displays the stories created by each user. There are two main features of the gallery display: a username selection interface on the left, and a story display on the right.

If the user has logged in, the default value of the username selection box is the user's own, so that she can view her own stories. The box is editable so that the viewer can change its contents to view the stories of another person.

As the user types in the selection box, the client performs an incremental find

Figure 3-2: Gallery screen

on the set of usernames. An incremental find is a search that is performed on an incomplete input from the user. Incremental find is often useful when searching for a word in a large set of words where the user may not be certain of the spelling. As they type more and more letters into the search box, the list of matches grows shorter and shorter. At any time the user can click on a username in the list to have that username fill the selection box and the stories associated with it displayed.

Add references to incremental search.

The right side of the screen is a column of rendered stories by each user. Only the first few frames of each story are shown – the exact number of shown frames depends on the pixel size of the user's browser – enough to give the viewer an idea of the contents of the story. The user can click on a story to edit it (if the user owns the story), or for viewing (if the story was created by someone other than the user). In the current interface the stories are shown at full size. A future consideration may be to display the stories at a reduced size to fit more onto the page.

3.3 Editing screen

The editing screen is the primary tool of the ComicKit. Most of the development effort has been spent on making this one screen work most effectively. I developed a mockup version of the editing screen in Java over the summer of 2004. Lessons

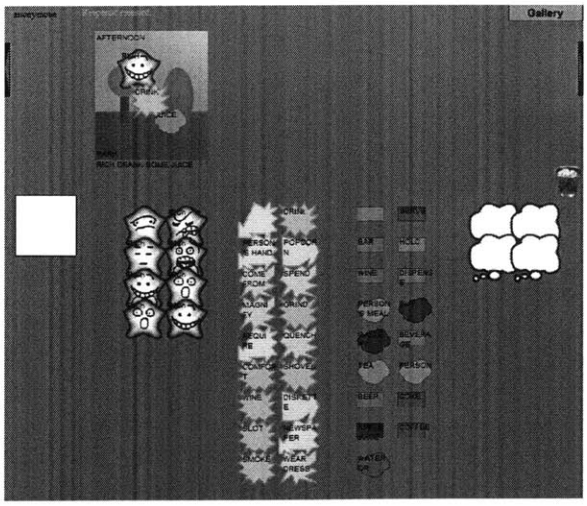


Figure 3-3: Editing screen

learned from that interface were applied to newer Flash interface developed over the fall of 2004.

The editing screen addresses the biggest goals of the ComicKit. By manipulating the interface provided by the editing screen, the user should be able to easily create as many stories of whatever type she wants. At the same time, the user should be gently guided into creating stories that are common-sense oriented, and mesh well with our existing knowledge bases, while also providing new information not provided by existing stories.

The editing screen is divided into two main components: an upper workspace where the comic is created, and a lower palette region where suggestions appear. There is a clear if invisible line dividing the two sections, and the behavior of draggable items is different depending on which section they've been dropped in.

The workspace is scrollable via the two arrow buttons at its left and right. Clicking and holding one of the buttons causes the panels to move at a fixed rate away from the button (the paradigm is similar to that of a regular scrollbar in that the arrows are perceived as moving the camera and not the panels). The workspace scroll buttons are always visible, even if all of the story's panels are visible onscreen.

Each palette in the lower region is vertically scrollable. The scroll buttons bear a visual resemblance to the horizontal scroll buttons of the workspace. They only

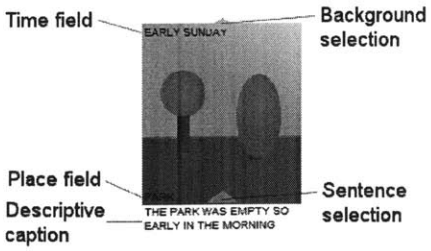


Figure 3-4: Annotated panel

appear if the palette has more contents than it can currently display. The scrolling mechanism of the palettes is difference from the panel scrolling in that it is punctuated. Each click on a palette scroll button moves the contents of the palette by one row. Continued holding has no effect.

The editing interface does not require any explicit action on the part of the user to preserve their story. Each time the user makes a change to the story, the entire story is sent to the server and saved. This process happens at the same time as suggestions are being generated for the palettes.

3.3.1 Panels

Comics are built as a sequence of panels. Each panel provides the setting for a scene. ComicKit panels have the following elements: a background image, a time-of-event text field, a place text field, and a caption. The user creates stories by filling in the contents of those text fields, and dragging various labeled icons over the background image to create a scene (not necessarily in that order).

- Place: is the name of the physical location where the scene happens. It is loosely correlated with the background image, insofar as the background image is supposed to visually represent the place.
- Time: is the comic creator's specification for the time at which the scene in the panel takes place.
- Caption: contains the textual description of the scene. It could be completely free-form text, or it could be a literal transcription of the labeled icons in the

panel.

ComicKit arranges the panels so that they line up adjoining each other in a line. By dragging the panels, it is easy to rearrange their order.

3.3.2 Drag and drop mechanism

The editing screen creates stories through direct-manipulation interaction. The primary activity in the story creation process is dragging and dropping labeled icons into panels to compose a scene. The user can drag as many icons onto a panel as are necessary to show the action, and label them with any text.

A labeled icon is an image overlaid with a text field. The user can change the label of any labeled icon in ComicKit simply by hovering her mouse over the icon. At the moment that her mouse rolls over the icon's boundaries, the text inside the label is selected, and is editable using standard techniques. Note that the text is not contained in a recessed, white-backgrounded text box, but instead is simply overlaid on the graphic. The same strategies that would work with a text box will work with editing the labeled icons (the `HOME` and `END` keys work as expected, it is possible to use `SHIFT` and the arrow keys to select a portion of the text, etc.). As long as the mouse pointer remains over the icon, the text will remain editable. Once the user moves the mouse poitner away from an icon, that icon's text ceases to be editable.

Each labeled icon is draggable to anywhere on the screen. If dropped on the panel portion of the screen, the labeled icon becomes associated with the nearest panel, and moves to stay completely in the bounds of said panel. If it is dropped in a specific position on the panel, the labeled icon always remains affixed to that spot on the panel, even if the panel is moved, until the user drags that particular icon to a new spot. When dropped on the lower, paletted, part of the screen, the labeled icon is added to the correct palette for its type. If dropped on top of a special 'trash' icon, the labeled icon deletes itself.

3.3.3 Selection mechanism for appearance

I discovered from the mockup that users wanted to be able to select the visual appearance of everything that they dragged onto the workspace. They wanted to select the appearance of the labeled icons, the facial expression of the persons, and the background images of the panels. I developed a mechanism that supports quick and easy changing of the image of any labeled icon.

The image selection method I developed is a form of pie menu. To change the image of a labeled icon, the user first clicks the yellow arrow. This yellow arrow appears only when the user's mouse is over the labeled icon. Once the arrow is clicked, the available image selections appear arranged in a full or semi circle centered around the existing image. They are spaced far enough apart that there are no overlaps of images. This is a sort of pie menu in the sense that this circle is now also nearly centered on the mouse due to the location of the just-clicked arrow. To select a new image for the icon, the user merely has to click on the desired image in the "pie". The user can dismiss the pie without making a selection simply by clicking on the arrow again.

3.3.4 Division into types

There are four primary icon types: objects, actions, persons, and thoughts/speech. Categorizing them in this way allows each type to have its own set of images, so that an action is always visually distinguishable from an object, for example. Person icons have the additional distinction of expressing one of the seven Ekman emotions[4].

The objects/actions/persons type trio is intended to represent nouns and verbs in simple declarative sentences. Accordingly, the graphics for each type are designed to appear like their part of speech: persons look vaguely face-like, objects look thing-like, and actions look active.

The simplest sort of sentence to parse is a declarative one. ComicKit facilitates the construction declarative sentences through its limitations. It is difficult to express complex tenses, interesting conditionals, and other sentence constructions using

Figure 3-5: Appearance of objects

Figure 3-6: Appearance of actions

simple nouns and verbs.

3.3.5 Suggestions

A major component of the ComicKit is its ability to common-sensically provide suggestions of labeled icons for use in the developing story. These suggestions are based on the parts of the story that the user has already created. Their intention is twofold.

First, they sometimes anticipate the user's need. The user may be writing a story about selling a car, and based on the words "sell" and "car", ComicKit would suggest things like "windshield" and "engine", anticipating the user's intention of creating a panel about how what shape the car's engine was in.

Second, they provide a feedback channel to the user. This channel delivers information about the types of things that our common sense knowledge already knows something about, and about what areas we would like people to create stories in. The suggestions also form a sort of conceptual template for the "common sense mindset". Comics are a hihgly symbolic medium, and it is common for one thing to represent another (e.g. rain representing depression). By providing only suggestions that are common-sensical (e.g. suggesting "water", "drink", "snow" when the user adds rain to the story), the ComicKit shows the sort of relationship that it considers common sense.

ComicKit always provides copies of the labeled icons that are being used in the story, since it is likely that the user will want to use the more than once. In anticipation of users wanting to define their own labels fairly often, a labeled icon with no text is always the first item in every palette.

Persons do not have a complicated suggestion method. We can presume that the user is very likely to want to reuse the same character's icon in many panels. Therefore the suggestions are limited only to multiple copies of each character in the story. If the user creates a character "Bob", the persons suggestion palette becomes filled with several "Bob" icons, plus the obligatory blank icon. If she adds a second character, "Alice", the persons suggestion panel becomes filled with equal numbers of Bobs and Alices.

Thoughts are never suggested. To some degree, the thought and speech bubbles form an outlet for creative impulses that go beyond simple action sentences. ComicKit merely provides a palette with sufficient thought and speech bubbles.

Action and object suggestions are complex, and based on the entire current contents of the story. The actions palette is filled with ConceptNet words/phrases that are either similar to the actions already in the story or can be performed on some of the objects in the story. Similarly, object suggestions are either related to the story objects or derived from the story actions.

Panel suggestions has not been incorporated into the ComicKit at the time of writing, but I hope to be able to add it soon.

Sentences

ComicKit suggests captions for each panel, based on its contents. This suggestion mechanism has much the same goals as suggesting labeled icons, but is handled through a different interface. Instead of dragging and dropping sentences, users select sentences by choosing from a list that flanks the panel, echoing the method for image selection described in section 3.3.3.

When the server makes sentence suggestions for a panel (which is not all the time), the panel subsequently gains an additional lighted arrow whose position is directly

above the caption (to be distinguished from the background-selection arrow, which is located at the top of the panel). When this arrow is clicked, the panel displays suggested sentences in two columns, one to either side. The sentences can be hidden by either selecting one, or by clicking on the arrow again.

The caption can still be edited normally by hovering the mouse over it, regardless of the suggestions.

3.3.6 Java mockup

Over the summer I developed a mockup in Java, my programming language of choice. I used the graphical toolkit Piccolo[1] to provide support for draggable icons and event handling. For the most part the Java version was very similar to the final Flash version, but the differences are listed here.

- The mockup's interface stacks the growing comics vertically on the left half, with palettes full of suggestions on the right half. The space available for each panel's caption was thus as tall as the panel itself and somewhat wider, and much roomier than even the most loquacious caption needed.

- Even though the user could edit labels simply by hovering the mouse over them, input is limited only to the alphanumeric characters. Punctuation is not possible.

- Sentence suggestions are displayed in a palette all of their own, right next to the thoughts/speech palette, and the user would drag a sugggested sentence over a panel.

- The mockup provided no mechanism for changing the image of a labeled icon, instead relying on random chance to provide the desired image in one of the suggestion palettes. This became an enormous problem, especially for the person labeled icons, where often the user wanted many copies of the same expression.

- User management is new to the Flash interface.

[1] `http://www.cs.umd.edu/hcil/jazz/index.shtml`

Chapter 4

ComicKit server

The ComicKit server provides an XML-RPC interface to the individual Flash clients. The server keeps everything centralized and offloads some computation from the client (though this creates a computational bottleneck at the server end). The server is responsible for saving stories to disk, producing the suggestions, maintaining a table of users, and returning stored stories for display.

I'll discuss the server's duties and interfaces by purpose. The first purpose is to manage users, by validating logins, creating new accounts, and checking for the username conflicts. The second purpose is to store and retrieve stories. The third purpose is to generate suggestions from the common-sense knowledge in ConceptNet.

4.1 User management

User management is the most visible of the server's duties. The Flash client queries the server for the existence of the username as the user is typing it in. The server validates the username/password combination before allowing the user to enter the other parts of the program. When the user wishes to register a new username/password, the server does that as well.

The server's interface to the client consists of four XML-RPC functions, delineated here:

check_username

Arguments: String username

Return value: "1" if the username specified has been registered before, "0" if not.

This function is called by the Flash client with each new character typed in the username box on the login screen, thus allowing the client to remain up-to-date as to whether the username is already registered.

get_incr_usernames

Arguments: String partialUsername *The first part of a username*

Return value: A list of usernames that start with the specified string.

This function is used on the gallery screen, to help the user find the username they want quicker.

new_user

Arguments: String username, String password, String email

Return value: The string "done", regardless of status.

check_login

Arguments: String username, String password

Return value: "1" if the username and password match one of the registered username/password sets in the database, "0" if they do not (either because of a nonexisting username or because of the wrong password).

These four functions handle all of the requirements of the login page and of the gallery's incremental username find feature.

The data storage that these functions manipulate is a table in a MySQL database.

The functions perform error checking and format an appropriate database query.

The table was created by the following SQL command:

```
create table user (
 uid INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY COMMENT 'User
ID field',
 name CHAR(64) NOT NULL COMMENT 'Textual username',
 password CHAR(64) NOT NULL COMMENT 'Plaintext password',
 email CHAR(200) COMMENT 'Email address',
 addedTime DATETIME NOT NULL DEFAULT 'NOW()' COMMENT 'Date/time that
user was added',
 loginCount INT UNSIGNED NOT NULL DEFAULT 0 COMMENT 'Number of times
user has logged in',
 UNIQUE (name),
 INDEX nameIdx(name)
);
```

4.2 Story management

The story management aspect of the server maintains a list of stories and the user who created each story. There are a few functions exported through XML-RPC.

create_new_story

Arguments: String username

Return value: An integer representing the story ID for the newly created story. This ID must be used when referring to the story in the future.

This function creates a new story in the database.

get_user_stories

Arguments: String username

Return value: A list of story IDs.

This function returns the story IDs for all the stories created by the specified user.

get_story

Arguments: int sid

Return value: A story in the representation discussed in the next section.

If you know the ID for a story, you can use this function to get the story's contents.

delete_story

Arguments: int sid

Return value: The value "1".

This function deletes the specified story. There is a real danger from vandalism or mistakes, so in the future it should probably merely hide a story, not delete it completely.

These methods operate on a MySQL story table, which was generated with the following command:

```
create table story (
 sid INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY COMMENT 'Story
ID field',
 uid INT UNSIGNED NOT NULL COMMENT 'ID of the user who owns the story',
 numupdates INT UNSIGNED NOT NULL DEFAULT 0 COMMENT 'Number of
updates to the story',
 created DATETIME NOT NULL DEFAULT 'NOW()' COMMENT 'Date/time that the
story was created',
 modified DATETIME NOT NULL DEFAULT 'NOW()' COMMENT 'Date/time that
the story was last modified',
 contents MEDIUMTEXT COMMENT 'The story representation.',
```

```
 INDEX userIdx (uid)
);
```

4.2.1 Story representation

In being passed from the client to the server for storage and parsing, and for being loaded by the client from storage, the the stories had to be formatted in a consistent way. The storage format I've developed is perhaps not the most readable, but it overcomes some of the limitations of the ActionScript XML-RPC implementation.

The representation is a list of mixed data types, a data structure native to both ActionScript and Python. The following notation is a list is indicated by enclosing brackets, and a list of variable length is indicated by an ellipsis. A variable's type is indicated after the colon, e.g. <username:String> indicates a variable called username which is a string.

<Representation> := [<Story title> [<Panel>, <Panel>, ...]]
[<selected:'0' or '1'>, [<Place:String>,
<image:String>, <x:float>, <y:float>],

<Panel> := <Caption:String>, <Time:String>, [<Person>,
<Person>, ...], [<Action>, <Action>, ...], [<Object>,
<Object>, ...], [<Thought>, <Thought>, ...]]

<Person> := [<Name:String>, <image:String>, <x:float>,
<y:float>]

<Action> := [<Verb:String>, <image:String>, <x:float>, <y:float>]

<Object> := [<Noun:String>, <image:String>, <x:float>,
<y:float>]

<Thought> := [<Text:String>, <image:String>, <x:float>, <y:float>]

This representation contains exactly the information required to restore a story to the editing interface and little more. The Place, Time, and Caption components of a Panel match the text fields in the client interface. Each Person, Action, Object, and Thought sub-list represents the coordinates, text, and the icon used to represent that particular component. The boolean value of the selected variable is used only by the sentence suggestion part of the backend. It would be too computationally expensive

to generate sentences for every panel each iteration, so the sentences are computed only for the most-recently-modified panel, which is indicated by the presence of a '1' in its selected field.

4.3 Parsing a story for suggestions

Frequently, the client will request a set of suggestions from the server. It does so by using the `crunch` method (which is labeled crunch3 in the XML-RPC interface for historical reasons).

crunch3

Arguments: int sid, <story representation>

Return value: [<person suggestions>, <action suggestions>, <object suggestions>, <sentence suggestions>]

This function returns a list of lists of icon representations. Each icon representation is a list containing the label of the icon, and either an image for the icon or the word "random". The sentence suggestions list only contains strings.

This method also saves the story in the MySQL database. The server expects `crunch` to be called as frequently as the user makes changes in the editing window.

To generate the person suggestions, the server constructs a list of the text labels (i.e., names) of all of the persons from the story, and creates six copies of each name.

In generating the action suggestions, the server looks at both the actions and objects in the story. It finds the analogous words to the actions in the story, and compiles them into a list. If finds the actions that can be performed on each object in the story, by searching the graph structure for "CapableOfReceivingAction" and "CapableOf" links to and from those objects. The two lists (analogous actions and actions performed by objects) are interleaved into a final list of action words, which is then converted into a list of icon representations.

The server suggests objects by also examining both the actions and objects in the

story. It construst two lists. The first list is generated by following "CapableOf" and "CapableOfReceivingAction" links from each of the actions in the story, resulting in a list of objects that those actions can be performed on. The second list comes from finding all the concepts that are analogous to each object in the story, and growing the list still further by finding all the concepts that are analogous to those concepts, i.e. concepts two degrees seperated from an object in the story.

The ComicKit server generates sentence suggestions based on the contents of one panel. Sentence suggestions are generated more or less combinatorially. It treats the actions in the panel as verbs, and the objects and persons as nouns. Using the MontyLingua natural language toolkit to convert verbs into a standard tense (simple past), the ComicKit server generates simple sentences in the following forms:

- (The/A) <Noun> <verbed> (the/a) <noun>
- (The/A) <Noun> <verbed> (the/a) <noun> (near the/on the/with the) <noun>
- (The/A) <Noun> was with (the/a) <noun>
- (The/A) <Noun> <verbed>
- <any of the previous sentence structures>, thinking <thought>

The server selects the proper pattern for the number of nouns and verbs in the panel, and generates one instance of the pattern for each possible way to arrange the nouns and verbs within it. For example, a panel containing the nouns "man" and "dog", and the verb "bite", uses the first pattern, and generates the sentences, "The man bit the dog", "The dog bit the man". This method often generates too many possible sentences, especially when many variations simply change out "a" for "the".

Chapter 5

Related work

ComicKit is superficially similar to other story-related systems, but departs significantly in its purpose. The ComicKit can be thought of as a graphical knowledge representation and generation tool, whereas other projects have focused on the learning of the user [18], fostering communication skills [3], and story authoring[1].

5.1 StoryWriter

ComicKit as a story creation tool is similar to StoryWriter because users construct a story graphically using a palette of story elements and captions. ComicKit is distinguished by its use of story suggestions and abstract appearance of story elements. ComicKit does not have representational objects like StoryWriter[18] does – a ComicKit pear does not bear any visual resemblance to a real pear, whereas in StoryWriter pears are identified and used based on their appearance. By deliberately dissociating the appearance of the object from its meaning, we allow the scope of the stories to be limited only by the user's imagination.

5.2 KidPad

KidPad is a collaborative tool for kids that allows them to draw stories on a zoomable canvas. KidPad appears similar to programs such as Kid Pix[1] in its focus on kid-friendly computer art. KidPad's uniqueness comes from its use of a zooming interface and the ability to set up hyperlinks to tell a story. Emphasis is placed on honing motor and drawing abilities. Though the use of hyperlinks in KidPad can create sequential stories, they also could be used to create nonlinear experiences such as branching storylines, or static scenes that can be explored in more detail. KidPad's interface is designed to be as expressive as possible, in contrast with ComicKit's deliberate limitations.

5.3 Daydreamer

Erik Mueller's Daydreamer project[12] is like an artificial story generator – almost the opposite of ComicKit. Daydreamer's intent is to explore the creation of subgoals to solve a "control goal", and the sequence of events that lead to resolution of those goals. Goals are structured in a tree, and their relative importances can vary over time. The emotions of the daydreamer program influence which goals are important to it as it goes through a daydream. In effect, the daydreamer program simulates the experience of a story.

StoryNet could use Mueller's techniques to generate derivative stories off of stories in its corpus. Daydreamer's stories are heavy with metadata such as the identification of which actions are taken in service of which goals, whereas stories entered via ComicKit are bereft of such information.

[1] `http://www.riverdeep.net/products/kid_pix/kpd4.jhtml`

Chapter 6

Evaluation

I performed two evaluations to test the effectiveness of the ComicKit interface, to see whether it was enjoyable to use at all, and whether it generated higher-quality stories than other interface attempts.

6.1 Comparison evaluation

We performed an initial evaluation of the mockup ComicKit interface in September 2004. The results of the evaluation were used for the subsequent IUI paper[19]. This evaluation attempted to see whether the comic interface was more effective than a plain text interface, and if the intelligent suggestions were helpful. We ran an experiment that pitted three story-entry applications against each other.

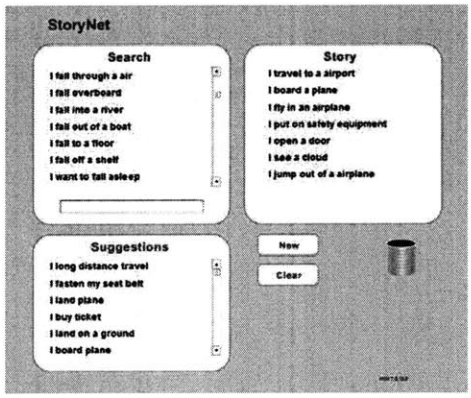


Figure 6-1: StoryNet Text interface

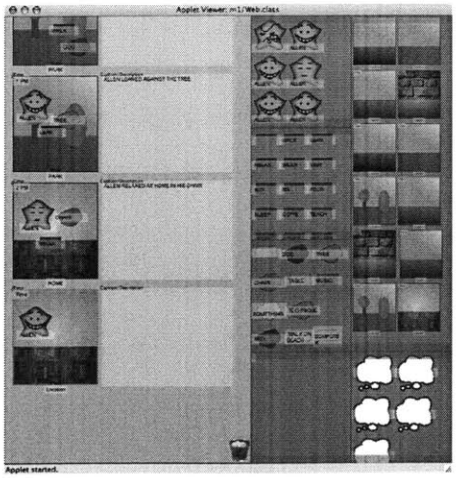


Figure 6-2: ComicKit mockup interface

1. **SN-Text** The first was a Flash-based story generator that had users dragging sentences into order. The sentences came from the LifeNet corpus[17], and were not modifiable. In order to tell their story, users had to repeatedly search the LifeNet database for the existing sentence that most closely matched their intent. A screenshot of this interface in action is seen in Figure 6.1.

2. **ComicKit-1** The second tested application was the Comic Kit's interface as described, but with no suggestions of elements from the server side.

3. **ComicKit-2** The last tested application was the fully intelligent ComicKit with suggestions for elements and captions.

For each application, subjects were asked to create three stories, then rate the application on three scales: entertainment value, helpfulness, and intelligence. We collected data from five grad students/professors at the MIT Media Lab for the preliminary study, and their average subjective responses are charted in Figure 6.1. We believe that for this test at least one user was not able to connect to the server and thus her experience for parts 2 and 3 were identical.

The evaluators found the ComicKit interface to be more rewarding than ordering sentences into a list to tell a story. They also found that the ComicKit with suggestions was more intelligent, and more helpful.

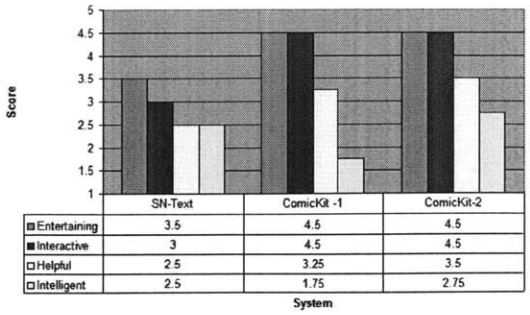


Figure 6-3: User Evaluation Subjective Results

It was fairly clear even during the experiment that the evaluators were using the three applications in different ways. They spent about ten minutes total creating stories on the text interface, and about two hours on the two versions of ComicKit, much more than we expected or asked. Some of them needed to be reminded of the time before they stopped. It is possible that the comic interface is very inefficient, and thus it took the users much longer to create an equivalent story. The other possibility is that the users found the comic interface more engrossing, and created longer and more involved stories with it.

It is hard to compare the experience of creating a story with the ComicKit with the text story creation utility. Users commented that the comic medium was richer, allowing them to tell the story both graphically in the panels, and in the text captions. They felt limited by the suggestions provided, and responded to this limitation in two ways. Some let the suggestion engine inspire them when their own ideas were running low, and sometimes they made choices they wouldn't have made on their own. Others felt challenged by the limitations, and went out of their way to overcome them. Because each label was completely editable, the only real limitation was in the amount of typing the user was willing to do in order to tell their story.

Most people used the thought and speech bubbles much more than we had anticipated – most people had at least one thought bubble in every panel. They were the primary means for people to get around the simplistic subject-object-verb model created by the objects in the interface. Thought bubbles provide an important insight into the mental state of story script characters, but unfortunately they can take on

difficult-to-understand forms due to their freeform nature.

We tracked the number of scene elements that each person composed into their comic strip, as well as the number of steps they took to generate each strip. The average values of these are presented below for comparison.

	ComicKit-1	ComicKit-2
Elements Used	13	17
Modifications	50	76

These values show that the users created comics with more elements when the interface included suggestions, and they spent more time trying out combinations. The interface afforded "playing", since virtually everything in it was interactive.

We believe that this study showed that ComicKit is likely to achieve our goals as a knowledge acquisition interface for StoryNet. As we acquire story script knowledge, we gain the ability to feed this knowledge back into the ComicKit through suggestions. This feedback loop will improve the sophistication of the system, and encourage users to play with it more. While we have yet to test the collected knowledge in an actual case-based commonsense reasoning system, ComicKit gets us closer to our goal of acquiring the very-large corpus of story-scripts that such a system would need to reason robustly about a wide range of commonsense scenarios.

6.2 Formative evaluation

I performed a simple formative evaluation in January 2005 to test the subjective feel of the Flash version of ComicKit. A primary concern was to find out whether users would want to continue entering stories after the initial encounter. If fundamental design issues kept the ComicKit from being fun to play with, this evaluation was intended to find out.

I chose to carry out the formative evaluation via a web form, which I asked several fellow MIT students to fill out. The subjects were asked via email to create a story with ComicKit, and fill out the evaluation form afterwards.

This evaluation revealed that even though Flash was chosen for its commonality,

about half of the people asked to participate did not have a recent enough version of Flash to use the ComicKit. Unfortunately, if the user does not have a recent enough version of Flash, the ComicKit fails by behaving erratically and corrupting data, rather than informing the user about the version mismatch. This issue should be addressed by including version-detection code in either the ComicKit itself or on the navigation page that leads to it.

The other major flaw revealed by this evaluation was that ComicKit does not degrade gracefully if the server is not available. Currently, ComicKit relies on the server to keep the palettes stocked with labeled icons, and to save the state of the comic as the user builds it. If the server is not available for some reason, the user will find that the palettes run out of labeled icons, and that the server is not saving the user's work. Perhaps the best way to address this issue is to completely halt the user's experience if the server is not available or unresponsive. Care should be taken to ensure that there are no false positives and that the server remains robust enough to serve for long periods of time.

The last issue revealed by this evaluation was that most people did not understand the interface right away, and required some explanation and demonstration before they understood its purpose. Despite the presence of a demonstration video on the website, this hurdle was too great for some people, and they spent their time idly dragging icons around without any direction. Visual hints, a simplified interface with clearer visuals, and a smoother way of presenting suggestions could help address this problem.

Chapter 7

Future directions

The existing interface should be thought of as a platform rather than an activity of its own. The platform allows the user to create stories by dragging icons into panels and occasionally typing new labels. Around this platform can be built a variety of activities. These activities could restrict the input from the user in a certain way, much like the templates in OMCS[16]. By creating puzzle-like activities, we could make comic creation more fun for participants, as well.

Other directions include expanding the social reach of the ComicKit. The gallery screen's ability to look at the comics created by other people hints at this ability. There is a lot of room to explore collaborative storytelling or collaborative teaching.

The richness of the existing program could use some deeper features. More expressive ways of associate the labeled icons with each other would help the creation of complex stories.

7.1 Templated Input

Some activities based on ComicKit could involve feeding existing StoryNet knowledge back to the player. The Kit could provide story facets as suggestions for the currently unfolding story, much in the manner of the labeled icon suggestions already extant in the program. These suggestions need not be limited to single panels – with a robust corpus of story knowledge, StoryNet would have the ability to produce several panels

worth of story.

More game-like activities would challenge the player to combine story fragments in a coherent way. Some example activites are listed following.

- Connect the dots: This activity would start the player off with a beginning panel (or set of panels), and an ending set of panels, and the player would coherently join them.
- Missing steps: Much like the templates of OMCS[16], this activity would ask the player to fill in gaps in a already-mostly-generated story.

As noted, these activities could be made to be very template-like.

For entertainment purposes, ComicKit could have a feature which creates a modified version of a player's story. Modifications would consist of making a minor change to one panel and attempting to propagate the results of that change through the rest of the story, e.g. changing a story about looking for keys into a story about looking for bananas. Already the suggestions from the ConceptNet backend provide entertainment, a sort of 'guess what the computer knows' exercise.

7.2 Social

Another approach is to cast the Comic Kit as a communications medium – players would have access to each other's story components and be able to send whole groups of stories to each other and would be able to follow how a story changes as it's passed around.

The server could help in the development of characters that participate in many stories. Already the server knows about the player's characters enough to provide them as suggestions. To extend the concept further, the server could infer personality characteristics from the actions that a particular character tends to take over several stories. The player could also specify character traits, perhaps in a special character-editing interface.

ComicKit could extend its reach beyond the confines of its own website. It would

be very simple, for example, to build a small Flash application that merely displays a stored comic, which players could download and use on their own websites to display comics they have created. The community would then have more control over the ways they interacted with the comics, and new and interesting interactions could result.

7.3 Better storytelling

One weakness of the existing ComicKit is that much of the story is left implicit. Subject-object relationships within a panel, overall identification of protagonists, and goals must be inferred from the captions. It would be very powerful to give the user the ability to explicitly state goals and subgoals, since that would increase the user's didactic power.

It should also be possible to tie ShapeNet, a corpus associating concepts with two-dimensional contours, into the framework, by providing contours to objects that look like the actual object. To some degree, doing so would violate some of the intentions behind leaving the icons abstract. ShapeNet does provide a large corpus of shapes, indexed by common-sense concept, saving a lot of work that would otherwise be necessary to use representational icons.

7.4 Conclusion

ComicKit has great potential to serve as the acquisition front-end for a large StoryNet corpus. Selecting a comic-based interface has made it possible to acquire simple, parseable stories from people who may not know very much about common sense or story scripts.

Bibliography

[1] Kevin M. Brooks. Do story agents use rocking chairs? the theory and implementation of one model for computational narrative. In *Proceedings of the fourth ACM international conference on Multimedia*, pages 317–328. ACM Press, 1996.

[2] Jerome Bruner. The narrative construction of reality. 18, 1991.

[3] Allison Druin, Jason Stewart, David Proft, Ben Bederson, and Jim Hollan. Kidpad: a design collaboration between children, technologists, and educators. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 463–470. ACM Press, 1997.

[4] P. Ekman, W. V. Friesen, and S. S. Tomkins. Facial affect scoring technique: A first validity study. 3:37–58, 1971.

[5] Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[6] H. Liu and P. Singh. Conceptnet: a practical commonsense reasoning toolkit. *BT Technology Journal*, 22(4):201–210, 2004.

[7] John McCarthy. *Formalization of common sense, papers by John McCarthy edited by V. Lifschitz.* Ablex, 1990.

[8] John McCarthy. From here to human-level ai. `http://www-formal.stanford.edu/jmc/human.html`, 1996.

[9] Scott McCloud. *Understanding Comics.* Kitchen Sink Press, 1993.

[10] Marvin Minsky. Commonsense-based interfaces. 43(8):67–73, 2000.

[11] Erik T. Mueller. `http://www.signiform.com/erik/pubs/cshumans.htm`, 2001.

[12] Erik T. Mueller and Michael G. Dyer. Towards a computational theory of human daydreaming. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 120–129, 1985.

[13] Roger C. Schank and Robert P. Abelson. *Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures.* L. Erlbaum, Hillsdale, NJ, 1977.

[14] P. Singh. The public acquisition of commonsense knowledge. In *AAAI Spring Symposium: Acquiring (and Using) Linguistic (and World) Knowledge for Information Access*, Palo Alto, CA, 2002. AAAI.

[15] P. Singh and B. Barry. Collecting commonsense experiences. In *Proceedings of the Second International Conference on Knowledge Capture*, 2003.

[16] P. Singh, T. Lin, E. Mueller, G. Lim, T. Perkins, and W. Zhu. Open mind common sense: Knowledge acquisition from the general public. In *Proceedings of the First International Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems*, Irvine, CA, 2002.

[17] P. Singh and W. Williams. Lifenet: a propositional model of ordinary human activity. In *Proceedings of the Workshop on Distributed and Collaborative Knowledge Capture (DC-KCAP)*, 2003.

[18] Karl E. Steiner and Thomas G. Moher. Graphic storywriter: an interactive environment for emergent storytelling. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 357–364. ACM Press, 1992.

[19] Ryan Williams, Barbara Barry, and Push Singh. Comickit: Acquiring story scripts using common sense feedback. In *Proceedings of the 2005 Internation Conference on Intelligent User Interfaces*, pages 302–304. ACM Press, 2005.

Appendix A

Interface Code

This is the Python source code for the Flash ComicKit interface.

Listing A.1: ComicIcon.as

```
class ComicIcon extends MovieClip {
        // descriptive data about this object
        var text:String;
        var imageText:String;
        var name;
        var tf_format;
        var tf;
        var tf_name;
        function onLoad() {
                // measure size of desired text
                tf_format = new TextFormat();
                tf_format.font = "Arial";
                tf_format.size = 10;
                tf_format.bold = false;
                // create movie clip to hold text
                var uid = _root.getNextHighestDepth();
                name = "comicIcon_"+text+"_"+uid;
```

```
            tf_name = "comicIcon_"+text+"_"+uid+"_tf";
            var defaultWidth = 400;
            var defaultHeight = 50;
            createTextField(tf_name, 10, 0, 0,
               defaultWidth, defaultHeight);
            this[tf_name].text = text;
            this[tf_name].textWidth = defaultWidth;
            this[tf_name].textHeight = defaultHeight;
            this[tf_name].textColor = "0x000000";
            this[tf_name].type = "dynamic";
            this[tf_name].wordWrap = true;
            this[tf_name].setTextFormat(tf_format);
            // event handler, called when object is
               released (mouse button up)
            this.onRelease = releaseMethod;
        }
        function releaseMethod() {
            // go to look at that comic
        }
        function getCenterX() {
            return _x+_width/2;
        }
        function getCenterY() {
            return _y+_height/2;
        }
}
```

Listing A.2: EditableText.as

```
class EditableText {
        static var etid:Number = 0;
```

```
var myName;
var parent;
var format:TextFormat;
var metrics:Object;
function EditableText(par:MovieClip, depth:Number,
   contents:String, xPos:Number, yPos:Number, width:
   Number, height:Number) {
        parent = par;
        myName = "editableText_"+etid;
        format = new TextFormat();
        format.font = "Arial";
        format.size = 10;
        format.bold = false;
        parent.createTextField(myName, depth, xPos,
           yPos, width, height);
        metrics = format.getTextExtent(contents);
                        parent[myName].
                           setNewTextFormat(format);
        parent[myName].text = contents;
        parent[myName].textWidth = width;
        parent[myName].textHeight = height;
        parent[myName].textColor = "0x000000";
        parent[myName].type = "input";
        parent[myName].wordWrap = true;
        parent[myName].et = this;
        parent[myName].onChanged = function(txt:
           TextField) {
              txt._parent.text = txt.text;
              txt.et.reFormat();
        };
```

```
        etid++;
    }
    function reFormat() {
        var txt = getTextField();
        // capitalize the text
        txt.text = txt.text.toUpperCase();
        // keep the text synced
        txt.setTextFormat(format);
        // keep good metrics
        metrics = format.getTextExtent(txt.text);
    }
    function getTextField():TextField {
        return parent[myName];
    }
    function getText():String{
        return parent[myName].text.toUpperCase();
    }
    function setEditable(me:EditableText) {
        if (Selection.getFocus() == ""+me.
          getTextField()) {
            return;
        } else {
            Selection.setFocus(me.getTextField())
              ;
        }
    }
    function setUneditable(me:EditableText) {
        if (Selection.getFocus() == ""+me.
          getTextField()) {
            Selection.setFocus(null);
```

```
                }
        }
        function remove() {
                getTextField().removeTextField();
        }
        function setText(newtxt:String) {
                getTextField().text = newtxt.toUpperCase();
        }
}
```

Listing A.3: Gallery.as

```
import XMLRPC.*;
class Gallery {
        var rpc:Connection;
        var myroot;
        var server;
        static var labelColor = 0x88BBFF;
        static var errorColor = 0xFF5555;
        static var storyHeight = 220;
        var originalUser:Boolean;
        var sindex:Number;
        var sids:Array;
        var storyReps:Array;
        var incrementals:Array;
        var intToken:Number;
        function Gallery(remote:String, root:MovieClip) {
                root.stop();
                myroot = root;
                server = remote;
                // types n shit
```

```
var prefix:String = "http://xnet.media.mit.
    edu/comickit/";
new Type("person", prefix+"images/faces/",
    prefix+"images/faces/", ["background1.swf"
    ], ["1.swf", "2.swf", "3.swf", "4.swf", "
    5.swf", "6.swf", "7.swf"]);
new Type("action", prefix+"images/", prefix+"
    images/", ["genericbackground.swf"], ["
    greenarrow.swf", "yellowthing.swf", "
    pinksplash.swf", "greensplash.swf"]);
new Type("object", prefix+"images/", prefix+"
    images/", ["genericbackground.swf"], ["
    gblob.swf", "gsquare.swf", "oblob.swf", "
    psquare.swf", "tsquare.swf", "rblob.swf",
    "rsquare.swf"]);
new Type("thought", prefix+"images/", prefix+
    "images/", ["genericbackground.swf"], ["
    thought.swf"]);
Panel.panelType = new Type("panel", prefix+"
    images/backgrounds/", prefix+"images/
    backgrounds/", ["ocean.swf"], ["greenplace
    .swf", "brownplace.swf", "greyplace.swf",
    "indoors.swf", "ocean.swf", "brickwall.swf
    "]);
Stage.scaleMode = "noscale";
Stage.align = "LT";
// LT - Left Top [0,0]
if (myroot.usernameStr == undefined) {
        myroot.usernameStr = "anonymous";
}
```

```
originalUser = true;
fillSidsList(myroot.usernameStr);
myroot.attachMovie("newstory", "newstory"
    , 117);
myroot.newstory.gallery = this;
myroot.newstory.onRelease = function() {
        this.gallery.myroot.loadMovie("
            editinterface.swf?usernameStr="+
            this.gallery.myroot.usernameStr, "
            POST");
};
myroot.attachMovie("TextInput", "
    usernameSearch", 208);
myroot.usernameSearch.text = myroot.
    usernameStr;
myroot.createTextField("usernameDisplay"
    , 209, 10, 2, 200, 50);
myroot.usernameDisplay.selectable = false;
myroot.usernameDisplay.text = "You are "+
    myroot.usernameStr;
myroot.createTextField("searchLabel"
    , 210, 10, 30, 200, 50);
myroot.searchLabel.selectable = false;
myroot.searchLabel.text = "Search:";
var nameListener = new Object();
nameListener.gallery = this;
nameListener.change = function(eventObject) {
        this.gallery.nameChanged();
};
```

```
myroot.usernameSearch.addEventListener("
    change", nameListener);
// the incremental list is interesting
myroot.createEmptyMovieClip("incrList", 211);
myroot.incrList.gallery = this;
myroot.incrList._quality = "HIGH";
myroot.incrList.format = new TextFormat();
myroot.incrList.format.font = "Arial";
myroot.incrList.format.size = 14;
myroot.incrList.format.bold = true;
myroot.incrList.maxHeight = myroot.incrList.
    format.getTextExtent("anonymous").height;
myroot.incrList.createTextField("content"
    , 1, 0, 0, 150, 100);
myroot.incrList.content.setNewTextFormat(
    myroot.incrList.format);
myroot.incrList.content.multiline = true;
myroot.incrList.content.selectable = false;
myroot.incrList.content.textColor =
    labelColor;
myroot.incrList.content.text = "";
myroot.incrList.onMouseMove = function() {
        if (this.hitTest(_root._xmouse, _root
            ._ymouse, false)) {
                var x:Number = _root._xmouse-
                    this._x;
                var y:Number = _root._ymouse-
                    this._y;
                var index:Number = Math.floor
                    (y/this.maxHeight);
```

```
                    if (index<this.gallery.
                        incrementals.length &&
                        index>=0) {
                            this.oldIndex = index
                                ;
                            this.drawBoxAt(index)
                                ;
                    }
            }
};
myroot.incrList.drawBoxAt = function(index:
    Number) {
            this.clear();
            this.lineStyle(0, 0x000000, 0);
            this.beginFill(0x224488);
            Drawing.drawRect(this, 0, index*this.
                maxHeight, this.format.
                getTextExtent(this.gallery.
                incrementals[index]).width+5, this
                .maxHeight, 6);
            this.endFill();
};
myroot.incrList.onRollOut = function() {
            this.clear();
};
myroot.incrList.onRelease = function() {
            var y:Number = _root._ymouse-this._y;
            var index:Number = Math.floor(y/this.
                maxHeight);
```

```
        if (index>=0 && index<this.gallery.
            incrementals.length) {
                this.gallery.selectName(this.
                    gallery.incrementals[index
                    ]);
        }
};
// now create a special storyStage clip for
    holding the stories
myroot.createEmptyMovieClip("storyStage"
    , 220);
myroot.storyStage.gallery = this;
myroot.storyStage.stories = [];
myroot.storyStage.createTextField("content"
    , -1, 0, 0, 150, 150);
myroot.storyStage.content.multiline = true;
myroot.storyStage.content.selectable = false;
myroot.storyStage.content.textColor =
    labelColor;
myroot.storyStage.content.text = "Stories:";
// resizing things to the proper size
myroot.onResize = function() {
        root.usernameSearch._x = 10;
        root.usernameSearch._y = 50;
        root.usernameDisplay._x = 10;
        root.usernameDisplay._y = 2;
        root.newstory._x = 150;
        root.newstory._y = 2;
        root.searchLabel._x = 10;
        root.searchLabel._y = 30;
```

```
                root.incrList._x = 10;
                root.incrList._y = 80;
                root.incrList.content._width = 200;
                root.incrList.content._height = Stage
                    .height-90;
                root.storyStage.content._width =
                    Stage.width-200;
                root.storyStage.content._height =
                    Stage.height;
                root.storyStage._x = 300;
                root.storyStage._y = 0;
                root.storyStage._width = Stage.width
                    -200;
                root.storyStage._height = Stage.
                    height;
        };
        // make this onResize method be called
        Stage.addListener(myroot);
        // call up stuff for the current username
        //selectName(myroot.usernameStr);
        myroot.onResize();
}
function selectName(name:String) {
        myroot.usernameSearch.text = name;
        incrementals = [];
        myroot.incrList.content.text = "";
        fillSidsList(name);
        for (var i = 0; i<myroot.storyStage.stories.
            length; i++) {
```

```
                myroot.storyStage.stories[i].
                    removeMovieClip();
        }
}
// this function assumes that there are some sids in
    the thing
function populateFirstStories() {
        var viewable:Number = Math.round(myroot.
            storyStage._height/storyHeight)+1;
        for (var i = 0; i<viewable && i<sids.length;
            i++) {
                myroot.storyStage.
                    createEmptyMovieClip("story_"+i,
                    myroot.storyStage.
                    getNextHighestDepth());
                var currentStory:MovieClip = myroot.
                    storyStage["story_"+i];
                currentStory._x = 0;
                currentStory._y = i*storyHeight+15;
                currentStory.width = myroot.
                    storyStage._width;
                currentStory.height = storyHeight;
                currentStory.sid = sids[i];
                currentStory.browser = this;
                myroot.storyStage.stories.push(
                    currentStory);
                showStory(storyReps[i], currentStory)
                    ;
                currentStory.onRollOver = function()
                    {
```

```
            this.clear();
            this.lineStyle(2, 0x224488
                , 100);
            this.beginFill(0x224488, 15);
            Drawing.drawRect(this, 2, 2,
                this.width-5, this.height
                -5, 6);
            this.endFill();
        };
        currentStory.onRollOut = function() {
            this.clear();
            this.lineStyle(2, 0x224488
                , 100);
            this.beginFill(0x224488, 5);
            Drawing.drawRect(this, 2, 2,
                this.width-5, this.height
                -5, 6);
            this.endFill();
        };
        currentStory.onRelease = function() {
            if (this.browser.originalUser
                ) {
                    trace("editing"+this.
                        sid+" "+this.
                        browser);
                    this.browser.myroot.
                        loadMovie("
                        editinterface.swf?
                        usernameStr="+this
                        .browser.myroot.
```

```
                                        usernameSearch.
                                        text+"&sid="+this.
                                        sid, "POST");
                                } else {
                                        trace("viewing"+this.
                                        sid);
                                        this.browser.myroot.
                                        loadMovie("viewer.
                                        swf?sid="+this.sid
                                        , "POST");
                                }
                        };
                }
        }
        // this function loads a story into a movieclip
        function showStory(storyRep:Array, parent:MovieClip)
            {
                trace("show story "+storyRep);
                parent.createTextField("title", 1, 5, 5,
                    parent.width, 20);
                parent.title.text = storyRep[0];
                parent.lineStyle(2, 0x224488, 100);
                parent.beginFill(0x224488, 5);
                Drawing.drawRect(parent, 2, 2, parent.width
                    -5, parent.height-5, 6);
                parent.endFill();
                var xpos = 10;
                for (var i = 1; i<storyRep.length; i++) {
                        createNewPanel(parent, storyRep[i],
                            xpos, 25);
```

```
                xpos += 180;
            }
        }
        function createNewPanel(parent:MovieClip, panelRep:
            Array, xpos:Number, ypos:Number) {
            var place:String = panelRep[1][0];
            var foreground:Number = panelRep[1][1];
            var caption:String = panelRep[2];
            var time:String = panelRep[3];
            var nextName = "panel_"+parent.
                getNextHighestDepth();
            parent.attachMovie("Panel", nextName, parent.
                getNextHighestDepth(), {type:Panel.
                panelType, fgindex:foreground, _x:xpos, _y
                :ypos, enabled:false});
            var currentPanel = parent[nextName];
            currentPanel.caption.setText(caption);
            currentPanel.time.setText(time);
            currentPanel.place.setText(place);
            //              parent.panels.push(
                currentPanel);
            // create the people
            var people:Array = panelRep[4];
            for (var i = 0; i<people.length; i++) {
                createNewTextNode(people[i],
                    currentPanel, Type.getByName("
                    person"));
            }
            // create the actions
            var actions:Array = panelRep[5];
```

```
        for (var i = 0; i<actions.length; i++) {
                createNewTextNode(actions[i],
                    currentPanel, Type.getByName("
                    action"));
        }
        // create the objects
        var objects:Array = panelRep[6];
        for (var i = 0; i<objects.length; i++) {
                createNewTextNode(objects[i],
                    currentPanel, Type.getByName("
                    object"));
        }
        // create thoughts
        var thoughts:Array = panelRep[7];
        for (var i = 0; i<thoughts.length; i++) {
                createNewTextNode(thoughts[i],
                    currentPanel, Type.getByName("
                    thought"));
        }
}
function createNewTextNode(nodeRep:Array, parent:
    Panel, type:Type) {
        var title = nodeRep[0];
        var foreground:Number = nodeRep[1];
        var x:Number = Number(nodeRep[2]);
        var y:Number = Number(nodeRep[3]);
        var canonicalName:String = title+TextNode.uid
            ;
        parent.attachMovie("Node", canonicalName,
            parent.getNextHighestDepth(), {text:title
```

```
            , type:type, fgindex:foreground, typeIndex
            :Type.getIndex(type), _x:x, _y:y, enabled:
            false});
        TextNode.uid++;
}
function isLoaded(v:MovieClip):Boolean {
        return (v.getBytesTotal()>4 && v.
            getBytesLoaded() == v.getBytesTotal());
}
// remote-call methods----------------------------------------

function nameChanged() {
        trace("changed "+myroot.usernameSearch.text);
        originalUser = (myroot.usernameSearch.text
            == myroot.usernameStr);
        var rpc = new XMLRPC.Connection();
        rpc.gallery = this;
        rpc.OnLoad = function(result:Array) {
                this.gallery.fillIncremental(result);
        };
        rpc.Server = server;
        rpc.AddParameter("string", myroot.
            usernameSearch.text);
        rpc.Call('get_incr_usernames');
}
function fillIncremental(names:Array) {
        trace("fillIncremental "+names);
        incrementals = names;
        var longString:String = "";
        for (var i = 0; i<incrementals.length; i++) {
```

```
                longString += incrementals[i]+"\n";
                var metric = myroot.incrList.format.
                    getTextExtent(incrementals[i]);
        }
        myroot.incrList.content.text = longString;
}
function fillSidsList(name:String) {
        var rpc = new XMLRPC.Connection();
        rpc.gallery = this;
        rpc.OnLoad = function(result:Array) {
                this.gallery.fillSidsListCB(result);
        };
        rpc.Server = server;
        rpc.AddParameter("string", name);
        rpc.Call('get_user_stories');
}
function fillSidsListCB(sids:Array) {
        trace("fillSidsListCB "+sids);
        this.sids = sids;
        sindex = 0;
        storyReps = new Array(sids.length);
        if (sids.length>0) {
                getStories();
        }
}
function getStories() {
        var rpc = new XMLRPC.Connection();
        rpc.gallery = this;
        rpc.OnLoad = function(result:Array) {
                this.gallery.getStoriesCB(result);
```

```
                };
                rpc.Server = server;
                rpc.AddParameter("int", sids[sindex]);
                rpc.Call('get_story');
        }
        function getStoriesCB(storyRep:Array) {
                trace("getStoriesCB "+storyRep);
                storyReps[sindex] = storyRep;
                sindex++;
                if (sindex<sids.length) {
                        getStories();
                } else {
                        populateFirstStories();
                }
        }
}
```

Listing A.4: Login.as

```
import XMLRPC.*;
class Login {
        var rpc:Connection;
        var myroot;
        var server;
        static var labelColor = 0x88BBFF;
        static var errorColor = 0xFF5555;
        function Login(remote:String, root:MovieClip) {
                createWindow(root);
                server = remote;
        }
        function createWindow(root:MovieClip) {
```

```
myroot = root;
root.stop();
Stage.scaleMode = "noScale";
Stage.align = "LT";
// LT - Left Top [0,0]
root.usernameStr = "";
root.passwordStr = "";
root.attachMovie("TextInput", "username",
    root.getNextHighestDepth(), {_x:0, _y:0,
    _visible:true});
root.attachMovie("TextInput", "password",
    root.getNextHighestDepth(), {_x:0, _y:0,
    _visible:false});
root.attachMovie("TextInput", "email", root.
    getNextHighestDepth(), {_x:0, _y:0,
    _visible:false});
root.username.tabIndex = 1;
root.password.tabIndex = 2;
root.focusManager.setFocus(root.username);
root.password.editable = false;
root.password.password = true;
root.attachMovie("login", "loginbutton", root
    .getNextHighestDepth(), {_x:231, _y:250,
    _visible:false});
root.attachMovie("anonymous", "
    anonymousbutton", root.getNextHighestDepth
    (), {_x:50, _y:250, _visible:true});
root.newUser = false;
root.createTextField("messageBox", root.
    getNextHighestDepth(), 50, 50, 100, 100);
```

```
root.messageBox.multiline = true;
root.messageBox.wordWrap = true;
root.messageBox.selectable = false;
root.messageBox.textColor = Login.labelColor;
root.createTextField("nameMessage", root.
    getNextHighestDepth(), 0, 0, 0, 100);
root.nameMessage.multiline = true;
root.nameMessage.wordWrap = true;
root.nameMessage.selectable = false;
root.nameMessage.textColor = Login.labelColor
    ;
root.nameMessage.text = "Type your existing
    username, or make up a new one!";
root.createTextField("usernameLabel", root.
    getNextHighestDepth(), 0, 0, 60, 20);
root.usernameLabel.selectable = false;
root.usernameLabel.textColor = Login.
    labelColor;
root.usernameLabel.text = "username";
root.createTextField("passwordLabel", root.
    getNextHighestDepth(), 0, 0, 60, 20);
root.passwordLabel.selectable = false;
root.passwordLabel.textColor = Login.
    labelColor;
root.passwordLabel.text = "password";
root.createTextField("emailLabel", root.
    getNextHighestDepth(), 0, 0, 60, 20);
root.emailLabel.selectable = false;
root.emailLabel._visible = false;
root.emailLabel.textColor = Login.labelColor;
```

```
root.emailLabel.text = "email";
var usernameListener = new Object();
usernameListener.change = function(
   eventObject) {
        changed(true, false);
};
root.username.addEventListener("change",
   usernameListener);
// password listener (more simple)
var passwordListener = new Object();
passwordListener.change = function(
   eventObject) {
        changed(false, true);
};
root.password.addEventListener("change",
   passwordListener);
function changed(unamChanged:Boolean,
   pwChanged:Boolean) {
        var name:String = root.username.text;
        if (name.length == 0) {
                root.nameMessage.text = "Type
                    your existing username,
                   or make up a new one!";
                root.password.editable =
                   false;
                root.password._visible =
                   false;
                root.anonymousbutton._visible
                    = true;
                root.password.tabIndex = 2;
```

```
} else {
        root.password._visible = true
            ;
        root.password.editable = true
            ;
        if (unamChanged) {
                root.login.nameExists
                    (name);
                root.nameMessage.
                    textColor = Login.
                    labelColor;
        }
}
var pass:String = root.password.text;
if (pass.length != 0 && name.length
    != 0) {
        root.loginbutton._visible =
            true;
        root.anonymousbutton._visible
             = false;
        root.focusManager.
            defaultPushButton = root.
            loginbutton;
} else {
        root.loginbutton._visible =
            false;
}
root.usernameStr = root.username.text
    ;
```

```
        root.passwordStr = root.password.text
            ;
}
root.loginbutton.onRelease = function() {
        root.nameMessage.text = "Checking
            name and password";
        if (root.newUser) {
                root.login.makeNewUser(root.
                    usernameStr, root.
                    passwordStr, root.email.
                    text);
        } else {
                root.login.checkLogin(root.
                    usernameStr, root.
                    passwordStr);
        }
};
root.anonymousbutton.onRelease = function() {
        root.username.text = "anonymous";
        root.password.text = "password";
        changed(true, true);
        root.loginbutton.onRelease();
};
root.onResize = function() {
        var leftOfInputs = Stage.width/2-50;
        var topOfInput = Stage.height/2-50;
        var inputSpacing = 7;
        root.username._x = leftOfInputs;
        root.username._y = topOfInput;
        root.email._x = leftOfInputs;
```

```
root.email._y = topOfInput+root.
    username._height+inputSpacing;
root.password._x = leftOfInputs;
if (root.email._visible) {
        root.password._y = root.email
            ._y+root.email._height+
            inputSpacing;
} else {
        root.password._y = root.
            username._y+root.username.
            _height+inputSpacing;
}
root.nameMessage._width = Stage.width
    /3;
root.nameMessage._x = Stage.width/3;
root.nameMessage._y = topOfInput-root
    .nameMessage._height-inputSpacing;
root.usernameLabel._x = leftOfInputs-
    root.usernameLabel._width-
    inputSpacing;
root.usernameLabel._y = topOfInput;
root.passwordLabel._x = leftOfInputs-
    root.passwordLabel._width-
    inputSpacing;
root.passwordLabel._y = root.password
    ._y;
root.emailLabel._x = leftOfInputs-
    root.emailLabel._width-
    inputSpacing;
root.emailLabel._y = root.email._y;
```

```
                var bottomOfInput = root.password._y+
                    root.password._height;
                root.loginbutton._x = leftOfInputs;
                root.loginbutton._y = bottomOfInput+
                    inputSpacing;
                root.anonymousbutton._x =
                    leftOfInputs;
                root.anonymousbutton._y = root.
                    loginbutton._y+root.loginbutton.
                    _height+inputSpacing;
        };
        // make this onResize method be called
        Stage.addListener(root);
        myroot.onResize();
}
function nameExists(name:String) {
        var rpc = new XMLRPC.Connection();
        rpc.login = this;
        rpc.OnLoad = function(result:Number) {
                this.login.nameExistsCB((result == 1)
                    );
        };
        rpc.Server = server;
        rpc.AddParameter("string", name);
        rpc.Call('check_username');
}
function nameExistsCB(result:Boolean) {
        if (result) {
                myroot.nameMessage.text = "'"+myroot.
                    username.text+"' has registered
```

```
                here before.";
            myroot.anonymousbutton._visible =
                false;
            myroot.newUser = false;
            myroot.email._visible = false;
            myroot.emailLabel._visible = false;
            myroot.onResize();
        } else {
            myroot.nameMessage.text = "The
                username '"+myroot.username.text+"
                ' is not in use.  If you enter a
                password and click 'login' you
                will create an account with this
                name.";
            myroot.newUser = true;
            myroot.anonymousbutton._visible =
                true;
            myroot.email._visible = true;
            myroot.emailLabel._visible = true;
            myroot.email.tabIndex = 2;
            myroot.password.tabIndex = 3;
            myroot.onResize();
        }
}
function checkLogin(name:String, password:String) {
        var rpc = new XMLRPC.Connection();
        rpc.login = this;
        rpc.OnLoad = function(result:Number) {
            this.login.checkLoginCB((result == 1)
                );
```

```
        };
        rpc.Server = server;
        rpc.AddParameter("string", name);
        rpc.AddParameter("string", password);
        rpc.Call('check_login');
}
function checkLoginCB(result:Boolean) {
        trace("checkLogin"+result);
        if (result) {
                loginSuccessful();
        } else {
                myroot.nameMessage.text = "The
                   password you typed is incorrect.
                    Try again.";
                myroot.nameMessage.textColor = Login.
                   errorColor;
                myroot.password.text = "";
                myroot.focusManager.setFocus(myroot.
                   password);
        }
}
function makeNewUser(name:String, password:String,
   email:String) {
        var rpc = new XMLRPC.Connection();
        rpc.login = this;
        rpc.OnLoad = function(result:String) {
                this.login.makeNewUserCB(result);
        };
        rpc.Server = server;
        rpc.AddParameter("string", name);
```

```
                rpc.AddParameter("string", password);
                rpc.AddParameter("string", email);
                rpc.Call('new_user');
        }
        function makeNewUserCB(result:String) {
                if (result == "done") {
                        loginSuccessful();
                } else {
                        myroot.nameMessage.text = "Problem
                            creating your username"+result;
                        myroot.nameMessage.textColor = Login.
                            errorColor;
                }
        }
        function loginSuccessful() {
                if (myroot.newUser) {
                        myroot.loadMovie("editinterface.swf?
                            newStory=true&usernameStr="+myroot
                            .usernameStr, "POST");
                } else {
                        myroot.loadMovie("browser.swf?
                            usernameStr="+myroot.usernameStr,
                            "POST");
                }
        }
}
```

Listing A.5: Palette.as

```
class Palette {
        static var spacing:Number = 1;
```

```
static var standardWidth = 54;
static var standardHeight = 40;
var x:Number;
var y:Number;
var width:Number;
var height:Number;
var type:Type;
var typeIndex:Number;
var visibleSet:Array;
var nodes:Array;
var roster:Array;
var noLayout:Boolean;
var top:Number;
var bottom:Number;
var upbutton:MovieClip;
var downbutton:MovieClip;
var root:MovieClip;
function Palette(xpos:Number, ypos:Number, w:Number,
   h:Number, mytype:Type, troot:MovieClip) {
        x = xpos;
        y = ypos;
        root = troot;
        width = w;
        height = h;
        type = mytype;
        noLayout = false;
        nodes = [];
        typeIndex = Type.getIndex(type);
        root.attachMovie("ScrollUpButton", "upbutton"
           +type.name, 10000+typeIndex, {_x:(x+33),
```

```
            _y:(y-10), typeIndex:typeIndex, _visible:
               false});
            root.attachMovie("ScrollDownButton", "
               downbutton"+type.name, 10010+typeIndex, {
               _x:(x+33), _y:(y+height-17), typeIndex:
               typeIndex, _visible:false});
            upbutton = root["upbutton"+type.name];
            upbutton.palette = this;
            upbutton.onPress = function() {
                    this.palette.scrollUp();
            };
            downbutton = root["downbutton"+type.name];
            downbutton.palette = this;
            downbutton.onPress = function() {
                    this.palette.scrollDown();
            };
}
function rescale(xpos:Number, ypos:Number, w:Number,
   h:Number) {
            x = xpos;
            y = ypos;
            width = w;
            height = h;
            //              trace("rescale " + x + " " +
               y + " " + width + " " + height);
            upbutton._x = (width-upbutton._width)/2+x;
            upbutton._y = ypos-10;
            downbutton._x = (width-downbutton._width)/2+x
               ;
            downbutton._y = ypos+height-17;
```

```
        // calculate number of objects to show
        var totalToAdd = Math.floor(width/
           standardWidth)*Math.floor(height/
           standardHeight);
        // show previously-hidden objects
        for (var j = top; j<totalToAdd+top && j<nodes
           .length; j++) {
                nodes[j]._visible = true;
        }
        // hide previously-shown objects (if this was
            a shrink)
        for (var j = totalToAdd+top; j<nodes.length;
           j++) {
                nodes[j]._visible = false;
        }
        bottom = top+totalToAdd;
        displayScrollBars();
        layoutNodes();
}
function displayScrollBars() {
        upbutton._visible = canScrollUp();
        downbutton._visible = canScrollDown();
}
function canScrollUp():Boolean {
        return top>1;
}
function scrollUp() {
        //              trace("Scroll Up");
        // remove the bottom row from the nodes, and
           move everyone down one unit
```

```
        if (not this.canScrollUp()) {
                return;
        }
        noLayout = true;
        var oneRow = Math.floor(width/standardWidth);
        // move the bottom visible row to oblivion
        for (var i = bottom-oneRow; i<bottom; i++) {
                nodes[i]._visible = false;
                nodes[i]._x = x;
                nodes[i]._y = y+height+20;
        }
        bottom -= oneRow;
        for (var i = top-1; i>=top-oneRow and i>=0; i
           --) {
                nodes[i]._visible = true;
        }
        top -= oneRow;
        noLayout = false;
        displayScrollBars();
        layoutNodes();
}
function canScrollDown():Boolean {
        return bottom<nodes.length;
}
function scrollDown() {
        //trace("Scroll down");
        // remove the top row from the nodes, and
           move everyone up one unit
        if (not this.canScrollDown()) {
                return;
```

```
        }
        noLayout = true;
        var oneRow = Math.floor(width/standardWidth);
        // move the top visible row to oblivion
        for (var i = top; i<top+oneRow; i++) {
                nodes[i]._visible = false;
                nodes[i]._x = x;
                nodes[i]._y = y+height+20;
        }
        top += oneRow;
        for (var i = bottom; i<nodes.length and i<
           bottom+oneRow; i++) {
                nodes[i]._visible = true;
        }
        bottom += oneRow;
        noLayout = false;
        displayScrollBars();
        layoutNodes();
}
function manageSet(names:Array) {
        roster = names;
        // clean out the nodes in the palette
        for (var i = 0; i<nodes.length; i++) {
                nodes[i].removeMovieClip();
        }
        unmanageAll();
        noLayout = true;
        createNewTextNode("", "random");
        for (var j = 0; j<names.length; j++) {
                if (names[j].length<2) {
```

```
                createNewTextNode(names[j
                    ][0], "random");
            } else {
                createNewTextNode(names[j
                    ][0], names[j][1]);
            }
        }
        var totalToAdd = Math.floor(width/
            standardWidth)*Math.floor(height/
            standardHeight);
        for (var j = 0; j<totalToAdd; j++) {
            nodes[j]._visible = true;
        }
        top = 0;
        bottom = totalToAdd;
        noLayout = false;
        layoutNodes();
        for (var j = 0; j<nodes.length; j++) {
            if (nodes[j]._visible) {
                nodes[j]._x = nodes[j].parX+
                    this.x;
                nodes[j]._y = nodes[j].parY+
                    this.y;
            }
        }
}
function createNewTextNode(title:String,
    representation:String) {
        // for right now, all are random
```

```
            //                  var fgIndex:Number =
                getFGIndex(representation);
            var canonicalName:String = title+TextNode.uid
                ;
            _root.attachMovie("Node", canonicalName, 100+
                TextNode.uid, {text:title, type:type,
                typeIndex:typeIndex, _x:x, _y:(y+height
                +20), imageText:representation, width:
                standardWidth, height:standardHeight,
                _visible:false});
            var currentNode = eval(canonicalName);
            _root.nodes.push(currentNode);
            this.manageNode(currentNode);
            TextNode.uid++;
    }
    function containsNode(node:TextNode):Boolean {
            for (var i in nodes) {
                    if (nodes[i] == node) {
                            return true;
                    }
            }
            return false;
    }
    function manageNode(newNode:TextNode) {
            nodes.push(newNode);
            layoutNodes();
    }
    function unmanageAll() {
            nodes = [];
    }
```

```
function unmanageNode(oldNode:TextNode) {
        // copy the old list to a new
        var newlist = [];
        for (var i = 0; i<nodes.length; i++) {
                if (nodes[i] == oldNode) {
                        continue;
                }
                newlist.push(nodes[i]);
        }
        nodes = newlist;
        layoutNodes();
}
function layoutNodes() {
        if (noLayout) {
                return;
        }
        var row:Number = 0;
        var maxheight:Number = 0;
        var currX:Number = spacing;
        var currY:Number = spacing;
        for (var i = 0; i<nodes.length; i++) {
                if (not nodes[i]._visible) {
                        continue;
                }
                var n:TextNode = nodes[i];
                //trace("Laying out "+i+", "+n);
                // set this row's maximum height
                if (maxheight<n.height) {
                        maxheight = n.height;
                }
```

```
                        // if this child will make the row
                           too wide, drop it to the next row
                        if (currX+n.width>this.width) {
                                row++;
                                currX = spacing;
                                currY += maxheight+spacing;
                                maxheight = 0;
                        }
                        n.parX = currX;
                        n.parY = currY;
                        currX += n.width+spacing;
                        //trace("Has new position "+n.parX
                           +" "+n.parY+" "+maxheight);
                }
        }
}
```

Listing A.6: Panel.as

```
class Panel extends SelectNode {
        static var panelType:Type;
        static var defaultWidth = 150;
        static var defaultHeight = 170;
        static var changeId = 0;
        static var sentHeight:Number = 32;
        var sentences:Array;
        var shownSentences:Array;
        var overTime:Boolean;
        var overPlace:Boolean;
        var overCaption:Boolean;
        var capsExpanded:Boolean;
```

```
var order:Number;
var children = [];
var name;
var tf_format;
var tf;
var tf_name;
var links;
// blinky link movie
var sentLink:MovieClip;
var intFunc2;
// for relaxation procedure
var changedTime:Number;
var arrangeWidth:Number;
var offset:Number;
var toX:Number;
var toY:Number;
// other stuff
var text;
var creationtime;
var generation;
var quality;
// text
var time:EditableText;
var place:EditableText;
var caption:EditableText;
// constructor
function Panel() {
        // these settings make things fan out
           downwards
        startAngle = -Math.PI;
```

```
        endAngle = 0;
        this.quality = 0;
        // for ranking nodes for removal
        this.dragging = false;
        overTime = overPlace=overCaption=false;
        toX = _x;
        toY = _y;
        time = new EditableText(this, 103, "TIME"
            , 0, 0, defaultWidth, 16);
        place = new EditableText(this, 104, "PLACE"
            , 0, defaultHeight-16, defaultWidth, 16);
        caption = new EditableText(this, 105, "
           CAPTION", 0, defaultHeight-2, defaultWidth
            , 32);
        markChanged();
        this.arrangeWidth = 150;
        this.offset = 0;
        // create link button
        createEmptyMovieClip("sentLink", 111);
        sentLink.loadMovie(linkUrl);
        intFunc2 = setInterval(this, "sentLinkLoaded"
            , 1000/12);
        // create event handlers
        this.onReleaseOutside = onRelease;
}
// event handler, called whenever mouse is moved
function onMouseMove() {
        if (!enabled || this.dragging) {
                return;
        }
```

```
if (this.hitTest(_root._xmouse, _root._ymouse
    , false)) {
        var x = _root._xmouse-_x;
        var y = _root._ymouse-_y;
        // show caption link
        showSentLink(true);
        // check if it's over any of the text
            fields
        if (y<time.metrics.height && x<time.
            metrics.width) {
                if (!overTime) {
                        overTime = true;
                        overTimeHandler(true)
                            ;
                }
                return;
        }
        if (y>place.getTextField()._y && y<
            place.getTextField()._y+place.
            metrics.height && x<place.metrics.
            width) {
                if (!overPlace) {
                        overPlace = true;
                        overPlaceHandler(true
                            );
                }
                return;
        }
        if (y>defaultHeight) {
                if (!overCaption) {
```

```
                                overCaption = true;
                                overCaptionHandler(
                                    true);
                            }
                            return;
                        }
                        notOverAnything();
                } else {
                        // not over the clip at all
                        showSentLink(false);
                        notOverAnything();
                }
        }
        function sentLinkLoaded() {
                if (isLoaded(sentLink)) {
                        sentLink._visible = false;
                        sentLink._x = (foreground._width-
                            sentLink._width)/2;
                        sentLink._y = defaultHeight-2-
                            sentLink._height;
                        sentLink._alpha = 50;
                        clearInterval(intFunc2);
                        // give it some functionality
                        sentLink.mypanel = this;
                }
        }
        function onLoad() {
        }
        function showSentLink(toShow:Boolean) {
                if (toShow && sentences.length>0) {
```

```
                sentLink._visible = true;
        } else {
                sentLink._visible = false;
        }
}
function notOverAnything() {
        if (overTime) {
                overTime = false;
                overTimeHandler(false);
        }
        if (overPlace) {
                overPlace = false;
                overPlaceHandler(false);
        }
        if (overCaption) {
                overCaption = false;
                overCaptionHandler(false);
        }
}
function overTimeHandler(over:Boolean) {
        if (over) {
                time.setEditable(time);
        } else {
                time.setUneditable(time);
        }
}
function overPlaceHandler(over:Boolean) {
        if (over) {
                place.setEditable(place);
        } else {
```

```
                place.setUneditable(place);
        }
}
function overCaptionHandler(over:Boolean) {
        if (over) {
                caption.setEditable(caption);
        } else {
                caption.setUneditable(caption);
        }
}
function getSentPerSide():Number {
        return Math.floor(_height/sentHeight);
}
function showSentences() {
        if (sentences.length>0) {
                markChanged();
                shownSentences = [];
                // how many can we stack on one side?
                var sideCount = getSentPerSide();
                // show left side
                for (var i = 0; i<sideCount && i<
                   sentences.length; i++) {
                        trace("showing left "+i+" "+
                           sentences[i]);
                        var curr:EditableText = new
                           EditableText(this, 25+i,
                           sentences[i], -
                           defaultWidth, i*sentHeight
                           , defaultWidth, sentHeight
                           );
```

```
                shownSentences.push(curr);
            }
            for (var i = sideCount; i< 2*
               sideCount && i<sentences.length; i
               ++) {
                trace("showing right "+i+" "+
                   sentences[i]);
                var curr:EditableText = new
                   EditableText(this, 25+i,
                   sentences[i], defaultWidth
                   , (i-sideCount)*sentHeight
                   , defaultWidth, sentHeight
                   );
                shownSentences.push(curr);
            }
            offset = defaultWidth;
            arrangeWidth = 3*defaultWidth;
            capsExpanded = true;
        }
    }
    function unshowSentences() {
        if (sentences.length>0) {
            markChanged();
            for (var i = 0; i<shownSentences.
               length; i++) {
                shownSentences[i].remove();
            }
            shownSentences = [];
            arrangeWidth = defaultWidth;
            offset = 0;
```

```
                capsExpanded = false;
        }
}
function markChanged() {
        changedTime = changeId++;
}
function startToExpand() {
        markChanged();
        arrangeWidth = defaultWidth*3;
        offset = defaultWidth;
        super.startToExpand();
}
function startClosing() {
        markChanged();
        arrangeWidth = defaultWidth;
        offset = 0;
        super.startClosing();
}
function onPress() {
        markChanged();
        if (sentLink.hitTest(_root._xmouse, _root.
           _ymouse, false)) {
                sentLink._alpha = 100;
        }
        super.onPress();
}
function onRelease() {
        markChanged();
        super.onRelease();
        if (isOverTrash()) {
```

```
                this.removeMovieClip();
            }
            if (sentLink.hitTest(_root._xmouse, _root.
                _ymouse, false)) {
                sentLink._alpha = 50;
                if (capsExpanded) {
                    unshowSentences();
                } else {
                    showSentences();
                }
            }
            if (capsExpanded) {
                // check to see if we clicked on a
                    sentence
                var x = _root._xmouse-_x;
                var y = _root._ymouse-_y;
                if (x<0) {
                    // on the left side
                    caption.setText(sentences[
                        Math.floor(y/32)]);
                    unshowSentences();
                } else if (x>defaultWidth) {
                    // on the right side
                    caption.setText(sentences[
                        getSentPerSide() + Math.
                        floor(y/32)]);
                    unshowSentences();
                }
            }
        }
```

```
        function getCenterX() {
                return _x+_width/2;
        }
        function getCenterY() {
                return _y+_height/2;
        }
}
```

Listing A.7: Relaxation.as

```
class Relaxation {
        static var BIGNUM = 1000666;
        static var running = false;
        static var spacing = 20;
        static var yheight = 35;
        static var slowness = 12;
        static var scrollSpeed:Number = 5;
        static var lastChanged:Number = 0;
        static function relaxNodes(nodes) {
                if (running) {
                        return;
                }
                running = true;
                for (var i = 0; i<nodes.length; i++) {
                        // set original center points
                        var n = nodes[i];
                        if (n.dragging || !n._visible) {
                                continue;
                        }
                        n.x = n.getCenterX();
                        n.y = n.getCenterY();
```

```
var distX:Number;
var distY:Number;
var parDragging = false;
if (n.parent != null) {
        // look at relative distance
           to final destination
        distX = n.parent._x+n.parX-n.
           _x;
        distY = n.parent._y+n.parY-n.
           _y;
        parDragging = n.parent.
           isDragging();
} else {
        // it's part of a palette if
           the parent is null
        //trace("Object using palette
            " + n.type.name + " " +
           _root.palettes[n.typeIndex
           ].x);
        distX = _root.palettes[n.
           typeIndex].x+n.parX-n._x;
        distY = _root.palettes[n.
           typeIndex].y+n.parY-n._y;
}
// set position
if (Math.abs(distX)>2 && !parDragging
   ) {
        n._x += 2*distX/slowness;
} else {
        n._x += distX;
```

```
                }
                if (Math.abs(distY)>2 && !parDragging
                    ) {
                        n._y += 2*distY/slowness;
                } else {
                        n._y += distY;
                }
        }
        running = false;
}
/*      static function relaxPanels_old(nodes, scroll
    ) {
                        if (running) {
                                return;
                        }
                        running = true;
                        var i:Number;
                        var j:Number;
                        var n:TextNode;
                        var xdist:Number;
                        var dx:Number;
                        var dy:Number;
                        var n1:Panel;
                        var dlen:Number;
                        //trace("Relaxing");
                        // set original center points
                        for (i=0; i<nodes.length; i
                            ++) {
                                n = nodes[i];
                                n.x = n.getCenterX();
```

```
            //n.mc._x;
            n.y = n.getCenterY();
            //n.mc._y;
            n.dx = 0;
            n.dy = 0;
        }
        for (i=0; i<nodes.length; i
            ++) {
            n1 = nodes[i];
            if (n1.isDragging())
                {
                    continue;
            }
            dx = 0;
            dy = 0;
            var closestDist:
                Number;
            var closest:Panel;
            var secClosestDist:
                Number;
            var secClosest:Panel;
            // find the closest
                node to this one
            closestDist =
                secClosestDist=
                BIGNUM;
            for (j=0; j<nodes.
                length; j++) {
                    // ignore
                        nodes that
```

```
        are being
        dragged
    if (i == j
        || nodes[
    j].
    isDragging
    ()) {
            continue
                ;
    }
    xdist = nodes
    [j].x-n1.x
    ;
    if (Math.abs(xdist
    )<Math.abs(
    closestDist
    )) {
            secClosestDist
                =
            closestDist
                ;
            secClosest
                =
            closest
                ;
            closestDist
                =
            xdist
                ;
```

```
                    closest
                        =
                    nodes
                    [j];

            } else if (Math
                .abs(xdist)
                <Math.abs(secClosestD
                )) {
                    secClosestDist
                        =
                    xdist
                    ;
                    secClosest
                        =
                    nodes
                    [j];

            }
        }
        // check to see if we
            're the only one
        if (closestDist ==
            BIGNUM) {
                closestDist
                    = spacing
                ;
        }
        // compute the
            distance to the
```

```
    desired location
var myspace = spacing
    +n1.arrangeWidth
    /2;
if (closestDist>0) {
        dx = 1.5*-(myspace
            +(closest.arrangeWidth
            /2)-closestDist
            )/slowness;

} else {
        dx = 1.5*(myspace
            +(closest.arrangeWidth
            /2)+closestDist
            )/slowness;

}
// if we're close
    enough to the
    destination, look
    at the next-
    closest
 if (Math.abs(dx)
    <1 &&
    secClosestDist !=
    BIGNUM) {
        if ((secClosestDist
            >0 && closestDist
            <=0) || (secClosestDis
            <0 && closestDist
```

```
>=0)) {
        if (secClosestDis
          >0)
           {
                dx
                   = -(
                myspac
                +(
                secCl
                .
                arrang
                /2)
                -
                secCl
                )
                /
                slowne
                ;

        } else
           {
                dx
                   = (
                myspac
                +(
                secCl
                .
                arrang
                /2)
                +
```

```
                    }
                }
            }
            // the y position
               needs to be
               corrected as well
            dy = (yheight-n1._y)/
               slowness;
            //trace("x: " + n1.x
                + " dx " + dx);
            if (n1.stayPut>0) {
                    n1.stayPut
                        -= .05;
            }
            if (!n1.fixed && !(n1
               .stayPut>0)) {
                    n1.x += dx;
            }
            n1.x += 3*scroll;
            n1.y += dy;
            n1._x = n1.x-n1.
               _width/2;
            n1._y = n1.y-n1.
               _height/2;
```

```
secCl
)
/
slowne
;
```

```
                    }
                    running = false;
                }*/
    static function relaxPanels(panels, scroll) {
        if (running) {
            return;
        }
        running = true;
        // see which is the latest-modifed panel
        var latest:Panel;
        var tempCh:Number = -1;
        for (var i = 0; i<panels.length; i++) {
            //trace("Looking for latest" + panels
                [i]);
            if (panels[i].changedTime>tempCh && !
                panels[i].isDragging()) {
                    latest = panels[i];
                    tempCh = latest.changedTime;
            }
        }
        // is this a new latest?
        if (latest.changedTime>lastChanged) {
            lastChanged = latest.changedTime;
            layoutPanels(panels, latest);
        }
        // move the panels to their destinations
        for (var i = 0; i<panels.length; i++) {
            driftPanel(panels[i], scroll);
        }
        running = false;
```

```
    }
    static function layoutPanels(panels, fixedPanel) {
            panels.sort(comparePanels);
            // layout the toXs relative to zero
            var currX = 0;
            for (var i = 0; i<panels.length; i++) {
                    panels[i].toX = currX;
                    currX += panels[i].arrangeWidth+
                       spacing;
            }
            // set all of these relative to the
               fixedPanel
            var correction:Number = fixedPanel._x-
               fixedPanel.toX;
            for (var i = 0; i<panels.length; i++) {
                    panels[i].toX += correction+panels[i
                       ].offset-fixedPanel.offset;
                    panels[i].toY = yheight;
            }
    }
    static function driftPanel(panel:Panel, scroll:Number
       ) {
            if (panel.isDragging()) {
                    return;
            }
            panel.toX += scroll*scrollSpeed;
            var driftAmt = 8;
            if (Math.abs(panel._x-panel.toX)<driftAmt) {
                    panel._x = panel.toX;
            } else {
```

```
                if (panel._x<panel.toX) {
                        panel._x += driftAmt;
                } else {
                        panel._x -= driftAmt;
                }
        }
        if (Math.abs(panel._y-panel.toY)<driftAmt) {
                panel._y = panel.toY;
        } else {
                if (panel._y<panel.toY) {
                        panel._y += driftAmt;
                } else {
                        panel._y -= driftAmt;
                }
        }
    }
    static function comparePanels(a, b) {
            var ax = a.getCenterX();
            var bx = b.getCenterX();
            if (ax<bx) {
                    return -1;
            } else if (ax>bx) {
                    return 1;
            } else {
                    return 0;
            }
    }
}
```

Listing A.8: SelectNode.as

```
import EnumeratedType;
class SelectNode extends MovieClip {
        static var linkUrl = "images/link.swf";
        // static definitions
        static var margin:Number = 5;
        static var defaultWidth:Number = 54;
        static var defaultHeight:Number = 40;
        public static var states = new EnumeratedType("idle"
            , "hover", "waitforload", "expansion", "selection"
            , "closing");
        // descriptive data about this object
        var type:Type;
        var parent:Panel;
        var name;
        // background image
        var background:MovieClip;
        // foreground image
        var foreground:MovieClip;
        // index into the type's array of image filenames
        var fgindex:Number;
        var bgindex:Number;
        // blinky link movie
        var link:MovieClip;
        // reference to the interval function
        var intFunction;
        // info about how to layout the exploded view
        var startAngle:Number;
        var endAngle:Number;
```

```
// these are for tracking the exploded view of all
   the movieclips
var state:Number;
var shownClips:Array;
var movedOut:Number;
var tempClip:MovieClip;
// disable everything?
var enabled:Boolean;
// for pushing to the top when hovered over
var oldDepth:Number;
// physical variables
var dragging:Boolean;
var fixed = false;
var x;
var y;
var dx;
var dy;
var width;
var height;
var parX;
var parY;
/* variables that you want to set:
```

```
function SelectNode() {
        if (enabled == undefined) {
                enabled = true;
        }
        _quality = "HIGH";
        dragging = false;
        // the default is that they spread from -PI
           /6 to 7/6PI
        if (startAngle == undefined) {
                startAngle = -Math.PI/6;
        }
        if (endAngle == undefined) {
                endAngle = 7*Math.PI/6;
        }
        // if the foreground/background indices aren'
           t specified, select them randomly
        if (fgindex == undefined or !type.
           isValidFGIndex(fgindex)) {
                fgindex = randRange(0, type.
                   foregrounds.length-1);
        }
        if (bgindex == undefined or !type.
           isValidBGIndex(bgindex)) {
                bgindex = randRange(0, type.
                   backgrounds.length-1);
        }
```

```
        // we can assume that the indices are correct
           , so let's load some movies
        createEmptyMovieClip("background", 101);
        background.loadMovie(type.backgrounds[bgindex
           ]);
        createEmptyMovieClip("foreground", 102);
        foreground.loadMovie(type.foregrounds[fgindex
           ]);
        // load that linky thing
        createEmptyMovieClip("link", 110);
        link.loadMovie(linkUrl);
        intFunction = setInterval(this, "linkLoaded"
           , 1000/12);
        // set up some shit
        this.width = defaultWidth;
        this.height = defaultHeight;
        // event handler, called when object is
           released (button up outside object)
        this.onReleaseOutside = onRelease;
        state = states.idle;
}
function showState() {
        trace("state: "+state);
}
function linkLoaded() {
        if (isLoaded(link) && isLoaded(foreground)) {
                link._visible = false;
                link._x = (foreground._width-link.
                   _width)/2;
                link._y = -link._height/2;
```

```
                clearInterval(intFunction);
        }
}
function isLoaded(v:MovieClip):Boolean {
        return (v.getBytesTotal()>4 && v.
            getBytesLoaded() == v.getBytesTotal());
}
function isOverTrash():Boolean {
        return _root.thetrash.hitTest(_root._xmouse,
            _root._ymouse, true);
}
function onLoad() {
}
public static function randRange(min:Number, max:
    Number):Number {
        var randomNum:Number = Math.round(Math.random
            ()*(max-min))+min;
        return randomNum;
}
function onRollOver() {
        if (state == states.idle) {
                link._visible = true;
                oldDepth = getDepth();
                //swapDepths(_parent.
                    getNextHighestDepth());
                if (state == states.idle && enabled)
                    {
                        state = states.hover;
                }
        }
```

```
}
function onRollOut() {
        if (state == states.hover) {
                link._visible = false;
                //swapDepths(oldDepth);
                if (state == states.hover) {
                        state = states.idle;
                }
        }
}
// event handler, called when object is released (
   mouse button up)
function onRelease() {
        if (!link.hitTest(_root._xmouse, _root.
           _ymouse, false)) {
                // stop dragging
                this.stopDrag();
                dragging = false;
        }
}
// event handler, called when object is pressed (
   mouse button down)
function onPress() {
        // begin a drag operation on the movie clip (
           handled by Flash automatically)
        if (!enabled) {
                return;
        }
        if (link.hitTest(_root._xmouse, _root._ymouse
           , false)) {
```

```
            switch (state) {
            case states.hover :
                    startToExpand();
                    break;
            case states.expansion :
                    // stop calling moveClipsOut
                    clearInterval(intFunction);
            case states.selection :
                    startClosing();
                    break;
            default :
                    // this should not happen
                    trace("unknown state at press
                        : "+state);
            }
    } else {
            this.dragging = true;
            this.startDrag();
            switch (state) {
            case states.expansion :
                    // stop calling moveClipsOut
                    clearInterval(intFunction);
            case states.selection :
                    // pick the right face
                    pickForeground(localX()-
                        foreground._width/2,
                        localY()-foreground._width
                        /2);
                    startClosing();
                    break;
```

```
                default :
                        // this is just a random
                           click -- ignore it
                }
        }
}
// this assumes that the x and y are relative to the
   center of the thingy
function pickForeground(x:Number, y:Number) {
        var dist:Number = Math.sqrt(x*x+y*y);
        if (dist<radiusOfClip(foreground)) {
                // pick nothing -- it's a cancel
                   action
        } else {
                var angle:Number = Math.atan2(-y, x);
                var wedge:Number = computeWedge(
                   shownClips.length)/2;
                if (angleBetween(angle, startAngle-
                   wedge, endAngle+wedge)) {
                        var index:Number = 0;
                        var done:Boolean = false;
                        while (!done && index<
                           shownClips.length) {
                                var testAngle:Number
                                   = computeAngle(
                                   index, shownClips.
                                   length);
                                if (angleBetween(
                                   angle, testAngle-
                                   wedge, testAngle+
```

```
                                    wedge)) {
                                        done = true;
                                        break;
                                }
                                index++;
                        }
                        if (done) {
                                fgindex = index;
                                foreground.loadMovie(
                                    type.foregrounds[
                                    fgindex]);
                        }
                }
        }
}
function startToExpand() {
        // first instantiate all the foregrounds
        shownClips = [];
        for (var i = 0; i<type.foregrounds.length; i
            ++) {
                createEmptyMovieClip("tempClip"+i
                    , 100-i);
                this["tempClip"+i].loadMovie(type.
                    foregrounds[i]);
                shownClips.push(this["tempClip"+i]);
        }
        movedOut = 0;
        state = states.waitforload;
        intFunction = setInterval(this, "waitForLoad"
            , 1000/12);
```

```
}
function waitForLoad() {
        var allDone:Boolean = true;
        for (var i = 0; i<shownClips.length; i++) {
                allDone &= isLoaded(shownClips[i]);
                if (allDone) {
                        shownClips[i]._xscale = 75;
                        shownClips[i]._yscale = 75;
                }
        }
        if (allDone) {
                clearInterval(intFunction);
                state = states.expansion;
                intFunction = setInterval(this, "
                   moveClipsOut", 1000/30);
                moveClipsOut();
        }
}
function moveClipsOut() {
        if (movedOut>1) {
                state = states.selection;
                clearInterval(intFunction);
                return;
        }
        movedOut += 1/12;
        var radius:Number = computeRadius();
        var myRad:Number = radiusOfClip(foreground);
        for (var i = 0; i<shownClips.length; i++) {
                var angle:Number = computeAngle(i,
                   shownClips.length);
```

```
                var distOut:Number = movedOut*radius;
                shownClips[i]._x = foreground._width
                    /2+distOut*Math.cos(angle)-
                    shownClips[i]._width/2;
                // flip the y for screen coordinates
                shownClips[i]._y = foreground._height
                    /2-distOut*Math.sin(angle)-
                    shownClips[i]._height/2;
            }
        }
        function startClosing() {
            state = states.closing;
            movedOut = 1;
            intFunction = setInterval(this, "moveClipsIn"
                , 1000/30);
            moveClipsIn();
        }
        function moveClipsIn() {
            if (movedOut<0) {
                // unload all of these movies
                for (var i = 0; i<shownClips.length;
                    i++) {
                        shownClips[i].unloadMovie();
                }
                clearInterval(intFunction);
                state = states.hover;
                if (this.hitTest(_root._xmouse, _root
                    ._ymouse, true)) {
                        this.onRollOver();
                } else {
```

```
                    this.onRollOut();
                }
                return;
            }
            movedOut -= 1/12;
            var radius:Number = computeRadius();
            for (var i = 0; i<shownClips.length; i++) {
                var angle:Number = computeAngle(i,
                    shownClips.length);
                var distOut:Number = movedOut*radius;
                shownClips[i]._x = foreground._width
                    /2+distOut*Math.cos(angle)-
                    shownClips[i]._width/2;
                // flip the y for screen coordinates
                shownClips[i]._y = foreground._height
                    /2-distOut*Math.sin(angle)-
                    shownClips[i]._height/2;
            }
}
// the angle for a particular foreground image
function computeAngle(i:Number, total:Number):Number
    {
        return (endAngle-startAngle)/(total-1)*i+
            startAngle;
}
// the angle of the wedge subtended by each thingy (
    same for each of them)
function computeWedge(total:Number):Number {
        return (endAngle-startAngle)/(total-1);
}
```

```
// reads the shownClips array for the largest size
   and computes the radius thereby
function computeRadius():Number {
        var maxDist:Number = 0;
        for (var i = 0; i<shownClips.length; i++) {
                var dist:Number = radiusOfClip(
                   shownClips[i]);
                if (dist>maxDist) {
                        maxDist = dist;
                }
        }
        // this strategy doesn't work so well for n
           close to 1, since we translate those sizes
            into arclengths, which only makes sense
           if there are enough that it approximates
        return (shownClips.length*maxDist)/(endAngle-
           startAngle);
}
function radiusOfClip(v:MovieClip):Number {
        var w:Number = v._width/2;
        var h:Number = v._height/2;
        return Math.sqrt(w*w+h*h);
}
function localX():Number {
        return _root._xmouse-_x;
}
function localY():Number {
        return _root._ymouse-_y;
}
function isDragging():Boolean {
```

```
        return dragging;
}
function getCenterX() {
        return _x+foreground._width/2;
}
function getCenterY() {
        return _y+foreground._height/2;
}
// brings an angle to being within +-PI
function normalizeAngle(a:Number):Number {
        while (a>Math.PI) {
                a -= 2*Math.PI;
        }
        while (a<-Math.PI) {
                a += 2*Math.PI;
        }
        return a;
}
// returns true if a is between the two angles b and
   c
// the angle is 'between' if you can rotate
   counterclockwise from b and hit a then c in that
   order
function angleBetween(a:Number, b:Number, c:Number):
   Boolean {
        //trace("abc "+a+" "+b+" "+c);
        a = a+Math.PI*2;
        b = b+Math.PI*2;
        c = c+Math.PI*2;
        if (a<b) {
```

```
                a += Math.PI*2;
            }
            if (c<b) {
                c += Math.PI*2;
            }
            //trace("abc2 "+a+" "+b+" "+c);
            return (a>b && a<c);
        }
}
```

Listing A.9: StoryRep.as

```
class StoryRep {
        public static function createStoryRep(title:String,
            nodes:Array, panels:Array, changed:Panel):Array {
//              trace("Calling story rep");
                // let's organize the panels by x value
                panels.sort(comparePanels);
                for (var i = 0; i<panels.length; i++) {
                        // mirror this order in the panel
                            itself for convenience
                        panels[i].order = i;
                }
                // now let's build up our response, panels
                    first
                var cr = new Array(panels.length+1);
                cr.isArray = true;
                // first element is experiment data
                cr[0] = title;
                for (var i = 0; i<panels.length; i++) {
                        var panel:Panel = panels[i];
```

```
                    var flag:Number = 0;
                    if (panel == changed) {
                            flag = 1;
                    }
                    cr[i+1] = [flag, [panel.place.getText
                        (), panel.fgindex], panel.caption.
                        getText(), panel.time.getText()
                        , [], [], [], []];
                    cr[i+1].isArray = cr[i+1][1].isArray=
                        cr[i+1][4].isArray=cr[i+1][5].
                        isArray=cr[i+1][6].isArray=cr[i
                        +1][7].isArray=true;
            }
            // now go through the nodes
            var typeOffset = 4;
    //      trace("Going through nodes");
            for (var i = 0; i<nodes.length; i++) {
                    var n:TextNode = nodes[i];
                    //trace("Compiling node "+n);
                    if (n.parent != null) {
                            var index:Number = n.parent.
                                order;
                            var nodeInfo:Array = [n.
                                getText(), n.fgindex, n.
                                parX, n.parY];
                            //trace("Node sending: " +
                                nodeInfo);
                            nodeInfo.isArray = true;
//                          trace("installing " +n.text
    + " in " + (typeOffset+Type.getIndex(n.type)));
```

```
                                cr[index+1][typeOffset+Type.
                                    getIndex(n.type)].push(
                                    nodeInfo);
                        }
                }
                // done!
                return cr;
        }
        static function comparePanels(a, b) {
                if (a._x<b._x) {
                        return -1;
                } else if (a._x>b._x) {
                        return 1;
                } else {
                        return 0;
                }
        }
}
```

Listing A.10: TextNode.as

```
class TextNode extends SelectNode {
        // unique ID counter
        static var uid:Number = 0;
                // static definitions
        static var margin:Number = 5;
        static var defaultWidth:Number = 54;
        static var defaultHeight:Number = 40;
        // descriptive data about this object
        var text:String;
        var typeIndex:Number;
```

```
var imageText:String;
var quality;
var parent:Panel;
var name;
var et;
// for relaxation procedure
var fixed = false;
var x;
var y;
var dx;
var dy;
var width;
var height;
var parX;
var parY;
// this compensates for a Flash bug
var middling:Boolean;
function addToAPanel() {
        if (not _visible) {
                return;
        }
        var oldparent = parent;
        parent = null;
        var dsq = Relaxation.BIGNUM;
        var closest = null;
        for (var i in _root.panels) {
                // check to see if we actually touch
                   a panel
                if (_root.panels[i].hitTest(this)) {
                        if (oldparent == null) {
```

```
                                    // remove self from
                                       the palette it
                                       belonged to
                                    _root.palettes[this.
                                       typeIndex].
                                       unmanageNode(this)
                                       ;
                            }
                            parent = _root.panels[i];
                            _root.changed = true;
                            _root.changedPanel = parent;
                    }
                    // simultaneously find the distance
                       to the center of the panel
                    var dx = _root.panels[i].getCenterX()
                      -this.getCenterX();
                    var dy = _root.panels[i].getCenterY()
                      -this.getCenterY();
                    var dist = dx*dx+dy*dy;
                    if (dist<dsq) {
                            dsq = dist;
                            closest = _root.panels[i];
                    }
            }
            // if we're outside of a panel, then we go
               either to the closest parent, or to a
               palette
            if (parent == null) {
                    if (this.getCenterY()<_root.
                       workspaceDivider) {
```

```
                    _root.palettes[this.typeIndex
                        ].unmanageNode(this);
                    parent = closest;
                    _root.changed = true;
                    _root.changedPanel = parent;
                } else {
                    // add to a palette only if
                        it's not already in it
                    if (!_root.palettes[this.
                        typeIndex].containsNode(
                        this)) {
                            _root.palettes[this.
                                typeIndex].
                                manageNode(this);
                            if (oldparent != null
                                ) {
                                    _root.changed
                                        = true;
                                    _root.changedPanel
                                        = oldparent
                                        ;
                            }
                    }
                    return;
                }
            }
            // see if we're completely in the selected
                panel
            // X first
```

```
if (this._x+this.width>parent._x+parent.
   _width) {
        // case: to the right of the parent,
           at least a little bit
        parX = parent._width-this.width;
        //trace("Out on right, parX = "+parX)
           ;
} else if (this._x<parent._x) {
        // case: to the left of the parent, a
            little bit or more
        parX = 0;
        //trace("Out on left, parX = "+parX);
} else {
        // not out at all
        parX = this._x-parent._x;
        //trace("OK X = "+parX);
}
// Y next
if (this._y+this.height>parent._y+parent.
   _height) {
        // case: below the parent, at least a
            little bit
        parY = parent._height-this.height;
        //trace("Out on bottom, parY = "+parY
           );
} else if (this._y<parent._y) {
        // case: above the parent, a little
           bit or more
        parY = 0;
        //trace("Out on left, parY = "+parY);
```

```
        } else {
                // not out at all
                parY = this._y-parent._y;
                //trace("OK Y = "+parY);
        }
}
function onLoad() {
        this.quality = 3;
        // create movie clip to hold text
        et = new EditableText(this, 103, text, margin
            , margin, defaultWidth-margin,
            defaultHeight-margin);
        x = _x;
        y = _y;
        this.width = defaultWidth;
        this.height = defaultHeight;
        // event handler, called when object is
            released (button up outside object)
        this.onReleaseOutside = onRelease;
        // final act: add to a panel or palette or
            whatever
        ///addToAPanel();
        middling = false;
}
function onRollOver() {
        super.onRollOver();
        activateText();
}
function onRollOut() {
        super.onRollOut();
```

```
        deactivateText();
    }
    function activateText() {
        if (middling == false) {
            et.setEditable(et);
            middling = true;
        } else {
            middling = false;
        }
    }
    function deactivateText() {
        if (middling == false) {
            et.setUneditable(et);
        }
    }
    function onRelease() {
        super.onRelease();
        if (isOverTrash()) {
            this.removeMovieClip();
            return;
        }
        addToAPanel();
    }
    function getCenterX() {
        return _x+_width/2;
    }
    function getCenterY() {
        return _y+_height/2;
    }
    function getText():String {
```

```
                return et.getText();
        }
}
```

Listing A.11: Type.as

```
/*
* This is to be distinguished from an enumerated type. This
   Type is for distinguishing the graphics shown
by the various draggable components in the EditInterface.
*/
class Type {
        // track all the created types in this array, sort of
            converting them into an enumerated type
        private static var typeArray:Array = [];
        public var name:String;
        public var backgrounds:Array;
        public var foregrounds:Array;
        function Type(myname:String, bgprefix:String,
           fgprefix:String, bgs:Array, fgs:Array) {
                name = myname;
                foregrounds = [];
                for (var i = 0; i<fgs.length; i++) {
                        foregrounds.push(fgprefix+fgs[i]);
                }
                backgrounds = [];
                for (var i = 0; i<bgs.length; i++) {
                        backgrounds.push(bgprefix+bgs[i]);
                }
                typeArray.push(this);
        }
```

```
        function isValidFGIndex(i:Number):Boolean {
                return (i>=0 and i<foregrounds.length);
        }
        function isValidBGIndex(i:Number):Boolean {
                return (i>=0 and i<backgrounds.length);
        }
        static function getIndex(t:Type):Number {
                for (var i = 0; i<typeArray.length; i++) {
                        if (t.name == typeArray[i].name) {
                                return i;
                        }
                }
                trace("Could not find type: "+t);
                return -1;
        }
        static function get(index:Number):Type {
                return typeArray[index];
        }
        static function getByName(name:String):Type {
                for (var i = 0; i<typeArray.length; i++) {
                        if (name == typeArray[i].name) {
                                return typeArray[i];
                        }
                }
                trace("Could not find type: "+name);
                return undefined;
        }
}
```

Listing A.12: Viewer.as

```
class Viewer {
        var myroot:MovieClip;
        var server:String;
        function Viewer(remote:String, root:MovieClip, sid:
           Number) {
                server = remote;
                myroot = root;
                // create some types
                var prefix:String = "http://xnet.media.mit.
                   edu/comickit/";
                new Type("person", prefix+"images/faces/",
                   prefix+"images/faces/", ["background1.swf"
                   ], ["1.swf", "2.swf", "3.swf", "4.swf", "
                   5.swf", "6.swf", "7.swf"]);
                new Type("action", prefix+"images/", prefix+"
                   images/", ["genericbackground.swf"], ["
                   greenarrow.swf", "yellowthing.swf", "
                   pinksplash.swf", "greensplash.swf"]);
                new Type("object", prefix+"images/", prefix+"
                   images/", ["genericbackground.swf"], ["
                   gblob.swf", "gsquare.swf", "oblob.swf", "
                   psquare.swf", "tsquare.swf", "rblob.swf",
                   "rsquare.swf"]);
                new Type("thought", prefix+"images/", prefix+
                   "images/", ["genericbackground.swf"], ["
                   thought.swf"]);
                Panel.panelType = new Type("panel", prefix+"
                   images/backgrounds/", prefix+"images/
                   backgrounds/", ["ocean.swf"], ["greenplace
```

```
            .swf", "brownplace.swf", "greyplace.swf",
            "indoors.swf", "ocean.swf", "brickwall.swf
            "]);
        getStory(sid);
}
function createNewPanel(panelRep:Array, xpos:Number,
    ypos:Number) {
        var place:String = panelRep[1][0];
        var foreground:Number = panelRep[1][1];
        var caption:String = panelRep[2];
        var time:String = panelRep[3];
        var nextName = "panel_"+myroot.
            getNextHighestDepth();
        myroot.attachMovie("Panel", nextName, myroot.
            getNextHighestDepth(), {type:Panel.
            panelType, fgindex:foreground, _x:xpos, _y
            :ypos, enabled:false});
        var currentPanel = myroot[nextName];
        currentPanel.caption.setText(caption);
        currentPanel.time.setText(time);
        currentPanel.place.setText(place);
        myroot.panels.push(myroot.currentPanel);
        // create the people
        var people:Array = panelRep[4];
        for (var i = 0; i<people.length; i++) {
                createNewTextNode(people[i],
                    currentPanel, Type.getByName("
                    person"));
        }
        // create the actions
```

```
        var actions:Array = panelRep[5];
        for (var i = 0; i<actions.length; i++) {
                createNewTextNode(actions[i],
                    currentPanel, Type.getByName("
                    action"));
        }
        // create the objects
        var objects:Array = panelRep[6];
        for (var i = 0; i<objects.length; i++) {
                createNewTextNode(objects[i],
                    currentPanel, Type.getByName("
                    object"));
        }
        // create thoughts
        var thoughts:Array = panelRep[7];
        for (var i = 0; i<thoughts.length; i++) {
                createNewTextNode(thoughts[i],
                    currentPanel, Type.getByName("
                    thought"));
        }
}
function createNewTextNode(nodeRep:Array, parent:
    Panel, type:Type) {
        var title = nodeRep[0];
        var foreground:Number = nodeRep[1];
        var x:Number = Number(nodeRep[2]);
        x += parent._x;
        var y:Number = Number(nodeRep[3]);
        y += parent._y;
```

```
        var canonicalName:String = title+TextNode.uid
            ;
        myroot.attachMovie("Node", canonicalName,
            myroot.getNextHighestDepth(), {text:title
            , type:type, fgindex:foreground, typeIndex
            :Type.getIndex(type), _x:x, _y:y, enabled:
            false});
        var currentNode = myroot[canonicalName];
        myroot.nodes.push(currentNode);
        TextNode.uid++;
}
function getStory(sid:Number) {
        var rpc = new XMLRPC.Connection();
        rpc.viewer = this;
        rpc.OnLoad = function(result:Array) {
                this.viewer.getStoryCB(result);
        };
        rpc.Server = server;
        rpc.AddParameter("int", sid);
        rpc.Call('get_story');
}
function getStoryCB(storyRep:Array) {
        trace("getStoryCB "+storyRep);
        var xpos = 10;
        for (var i = 1; i<storyRep.length; i++) {
                createNewPanel(storyRep[i], xpos, 10)
                    ;
                xpos += 200;
        }
}
```

```
        function reflect() {
                var newRep:Object = new Object();
                newRep.random = "forks";
                newRep.anum = 10;
                newRep.panels = ["wang", "schlong", 10, 25];
                var rpc = new XMLRPC.Connection();
                rpc.viewer = this;
                rpc.OnLoad = function(result:Object) {
                        this.viewer.reflectCB(result);
                };
                rpc.Server = server;
                rpc.AddParameter("struct", newRep);
                rpc.Call('dump_params');
        }
        function reflectCB(res:Object) {
                trace("got back an object: "+res.random+" "+
                   res.anum+" "+res.panels);
        }
}
```

Listing A.13: editinterface.fla

```
#include "XMLRPC/xml-rpc.as"
import XMLRPC.*;
var server:String = 'http://xnet.media.mit.edu:8055';
Stage.scaleMode = "noScale";
Stage.align = "LT";
Stage.addListener(_root);
// LT - Left Top [0,0]
// create some types
if (usernameStr == undefined) {
```

```
        usernameStr = "anonymous";
}
//var prefix:String = "http://xnet.media.mit.edu/comickit/";
var prefix:String = "";
new Type("person", prefix+"images/faces/", prefix+"images/
    faces/", ["background1.swf"], ["1.swf", "2.swf", "3.swf",
    "4.swf", "5.swf", "6.swf", "7.swf"]);
new Type("action", prefix+"images/", prefix+"images/", ["
    genericbackground.swf"], ["greenarrow.swf", "yellowthing.
    swf", "pinksplash.swf", "greensplash.swf"]);
new Type("object", prefix+"images/", prefix+"images/", ["
    genericbackground.swf"], ["gblob.swf", "gsquare.swf", "
    oblob.swf", "psquare.swf", "tsquare.swf", "rblob.swf", "
    rsquare.swf"]);
new Type("thought", prefix+"images/", prefix+"images/", ["
    genericbackground.swf"], ["thought.swf"]);
Panel.panelType = new Type("panel", prefix+"images/
    backgrounds/", prefix+"images/backgrounds/", ["ocean.swf"
    ], ["greenplace.swf", "brownplace.swf", "greyplace.swf", "
    indoors.swf", "ocean.swf", "brickwall.swf"]);
_root.createTextField("usernameDisplay", 55, 10, 2, 400, 50);
usernameDisplay.selectable = false;
usernameDisplay.text = usernameStr;
_root.createTextField("messageBox", root.getNextHighestDepth
    (), 100, 0, 100, 32);
_root.messageBox.multiline = true;
_root.messageBox.wordWrap = true;
_root.messageBox.selectable = false;
_root.messageBox.textColor = Login.labelColor;
```

```
_root.attachMovie("TextInput", "title", 56, {_x:150, _y:2,
    _width:300, _visible:true});
attachMovie("trash", "thetrash", 2, {_x:200, _y:(
    workspaceDivider-thetrash._height)});
var scroll:Number = 0;
var changed:Boolean = false;
var waitingForResponse = false;
var changedPanel:Panel;
var panels = [];
var nodes = [];
var panelID:Number = 0;
var nodeID:Number = 0;
var delay = 0;
// the y value where the workspace begins
var workspaceDivider:Number = 260;
//var currentNode;
var palettes = [];
// persons palette
palettes.push(new Palette(110, 220, 110, 200, Type.getByName(
    "person"), _root));
// actions palette
palettes.push(new Palette(220, 220, 110, 200, Type.getByName(
    "action"), _root));
// objects palette
palettes.push(new Palette(330, 220, 110, 200, Type.getByName(
    "object"), _root));
// thoughts palette
palettes.push(new Palette(440, 220, 110, 200, Type.getByName(
    "thought"), _root));
// add the left/right scrollbars
```

```
attachMovie("ScrollRightButton", "ScrollRightButton"
    , 10100, {_x:543, _y:50});
attachMovie("ScrollLeftButton", "ScrollLeftButton", 10101, {
    _x:0, _y:50});
function unscroll() {
        _root.scroll = 0;
}
var scr = eval("ScrollLeftButton");
scr.onPress = function() {
        _root.scroll = 1;
};
scr.onRelease = unscroll;
scr.onReleaseOutside = unscroll;
scr = eval("ScrollRightButton");
scr.onPress = function() {
        _root.scroll = -1;
};
scr.onRelease = unscroll;
scr.onReleaseOutside = unscroll;
// gallery button
attachMovie("gallery", "gallerybutton", 56, {_x:400, _y:0});
gallerybutton.onRelease = function() {
        loadMovie("browser.swf?usernameStr="+_root.
            usernameStr, _root, "POST");
};
function removeNode(oldNode:TextNode) {
        // copy the old list to a new
        var newlist = [];
        for (var i = 0; i<nodes.length; i++) {
                if (nodes[i] == oldNode) {
```

```
                        continue;
                }
                newlist.push(nodes[i]);
        }
        nodes = newlist;
}
function createNewPanelAtOrigin(dragging:Boolean) {
        var nextName:String = 'panel_'+panelID;
        createNewPanel(nextName, 14, 252, dragging);
}
function createNewPanel(nextName:String, xpos:Number, ypos:
   Number, dragging:Boolean) {
        attachMovie("Panel", nextName, -100-panelID, {type:
           Panel.panelType, _x:xpos, _y:ypos});
        _root.currentNode = eval(nextName);
        _root.panels.push(_root.currentNode);
        if (dragging) {
                //currentNode.onPress();
        }
        panelID++;
}
// XML-RPC shite
function serverResponse(errorCode, result) {
        _root.messageBox.text = "Response received.";
        waitingForResponse = false;
        //      trace("CALLBACK VALUE: "+result);
        var ordinary = result[0];
        // fill the palettes with stuff
        for (var i = 0; i<3; i++) {
```

```
                //                trace("Size of response: " +
                    i + " is " + ordinary[i].length);
                if (ordinary[i].length>0) {
                        palettes[i].manageSet(ordinary[i]);
                }
        }
        trace("Sentences "+ordinary[3]);
        changedPanel.sentences = ordinary[3];
}
rpc = new xmlrpc("http://xnet.media.mit.edu:8055");
rpc.AUTO_FORMAT = true;
rpc.callBack = serverResponse;
// the animation every frame
function animation() {
        Relaxation.relaxPanels(panels, scroll);
        Relaxation.relaxNodes(nodes);
        //      trace("nodes: " + nodes);
        if (delay>0) {
                delay -= 1;
        } else {
                if (changed && !waitingForResponse) {
                        changed = false;
                        delay = 75;
                        var thelist:Array = StoryRep.
                            createStoryRep(_root.title.text,
                            nodes, panels, changedPanel);
                        trace("the list "+thelist);
                        var packagedsid = rpc.setParameter(
                            sid);
```

```
                        var sending = rpc.setParameter(
                            thelist);
                        rpc.send("crunch3", [packagedsid,
                            sending]);
                        waitingForResponse = true;
                        _root.messageBox.text = "Contacting
                            server...";
                    }
                }
}
interval = setInterval(animation, 15);
if (sid != undefined) {
        trace("loading "+sid);
        loadUpAStory(sid);
        palettes[0].manageSet([[""]]);
        palettes[1].manageSet([[""]]);
        palettes[2].manageSet([[""]]);
        palettes[3].manageSet([[""], [""], [""]]);
} else {
        trace("new story for "+usernameStr);
        newStory(usernameStr);
        createNewPanelAtOrigin(false);
        panels[0].sentences = ["Sentence A", "Sentence B", "
            Sentence C", "Sentence D", "Sentence E", "Sentence
             F", "Sentence G", "Sentence H", "Sentence I", "
            Sentence J", "Sentence K", "Sentence L"];
        panels[0].markChanged();
        palettes[0].manageSet([["SAILOR"], ["CHRIS"], ["BOB"
            ], ["SALLY"], ["ALICE"]]);
```

```
        palettes[1].manageSet([["FLOAT"], ["ROCK"], ["SAIL"
            ], ["BLOW"], ["DRIVE"], ["TIE"], ["KICK"], ["
            SPLASH"], ["THROW"], ["WAVE"], ["BLOW HORN"], ["
            LOOK"], ["DRINK"], ["TASTE"], ["GET SEASICK"], ["
            OPEN"]]);
        palettes[2].manageSet([["SHIP"], ["WATER"], ["BEACH"
            ], ["SAILOR"], ["OCEAN"], ["ENGINE"], ["SAIL"], ["
            DIESEL"], ["WAVES"], ["CANOPY"], ["POOL"], ["
            CHAMPAGNE"], ["MAGICIAN"], ["PORTHOLE"], ["POSH"
            ], ["TITANIC"]]);
        palettes[3].manageSet([[""], [""], [""]]);
}
_root.onResize = function() {
//      _root.messageBox.text = "resizing " + Stage.width
    + " " + Stage.height;
        thetrash._x = Stage.width-thetrash._width;
        thetrash._y = workspaceDivider-thetrash._height;
        ScrollRightButton._x = Stage.width-ScrollRightButton.
            _width;
        ScrollLeftButton._x = 0;
        gallerybutton._x = Stage.width-gallerybutton._width;
        var palWidth = Stage.width/(palettes.length+1);
        //      trace("rescale "+Stage.width+" "+palWidth);
        for (var i = 0; i<palettes.length; i++) {
                palettes[i].rescale(palWidth*(i+1),
                    workspaceDivider, palWidth, Stage.height-
                    palettes[i].y);
        }
        _root.messageBox._x = galleryButton._x - messageBox.
            _width;
```

```
        _root.messageBox._y = galleryButton._y + 5;
};
_root.onResize();
function loadUpAStory(sid:Number) {
        var rpc = new XMLRPC.Connection();
        rpc.ef = this;
        rpc.OnLoad = function(result:Array) {
                this.ef.getStoryCB(result);
        };
        rpc.Server = _root.server;
        rpc.AddParameter("int", sid);
        rpc.Call('get_story');
}
function getStoryCB(storyRep:Array) {
        trace("getStoryCB "+storyRep);
        title.text = storyRep[0];
        var xpos = 10;
        for (var i = 1; i<storyRep.length; i++) {
                createNewPanelB(this, storyRep[i], xpos, 25);
                xpos += 180;
        }
        changed = true;
}
function createNewPanelB(parent:MovieClip, panelRep:Array,
   xpos:Number, ypos:Number) {
        var place:String = panelRep[1][0];
        var foreground:Number = panelRep[1][1];
        var caption:String = panelRep[2];
        var time:String = panelRep[3];
        var nextName = "panel_"+panelID;
```

```
trace("creating panel"+nextName+"  "+xpos+" "+Panel.
    panelType.foregrounds[foreground]);
parent.attachMovie("Panel", nextName, -100-panelID, {
    type:Panel.panelType, fgindex:foreground, _x:xpos
    , _y:ypos});
var currentPanel = parent[nextName];
currentPanel.caption.setText(caption);
currentPanel.time.setText(time);
currentPanel.place.setText(place);
parent.panels.push(currentPanel);
panelID++;
// create the people
var people:Array = panelRep[4];
for (var i = 0; i<people.length; i++) {
        createNewTextNode(people[i], parent,
            currentPanel, Type.getByName("person"));
}
// create the actions
var actions:Array = panelRep[5];
for (var i = 0; i<actions.length; i++) {
        createNewTextNode(actions[i], parent,
            currentPanel, Type.getByName("action"));
}
// create the objects
var objects:Array = panelRep[6];
for (var i = 0; i<objects.length; i++) {
        createNewTextNode(objects[i], parent,
            currentPanel, Type.getByName("object"));
}
// create thoughts
```

```
        var thoughts:Array = panelRep[7];
        for (var i = 0; i<thoughts.length; i++) {
                createNewTextNode(thoughts[i], parent,
                    currentPanel, Type.getByName("thought"));
        }
}
function createNewTextNode(nodeRep:Array, root:MovieClip,
    panel:Panel, type:Type) {
        var title = nodeRep[0];
        var foreground:Number = nodeRep[1];
        var x:Number = Number(nodeRep[2]);
        var y:Number = Number(nodeRep[3]);
        var depth = root.getNextHighestDepth();
        trace("node at "+depth);
        var canonicalName:String = title+TextNode.uid;
        root.attachMovie("Node", canonicalName, depth, {text:
            title, type:type, fgindex:foreground, typeIndex:
            Type.getIndex(type), parent:panel, _x:0, _y:0,
            parX:x, parY:y});
        root.nodes.push(root[canonicalName]);
        TextNode.uid++;
}
function newStory(username:String) {
        var rpc = new XMLRPC.Connection();
        rpc.ef = this;
        rpc.OnLoad = function(result:Number) {
                this.ef.newStoryCB(result);
        };
        rpc.Server = _root.server;
        rpc.AddParameter("string", username);
```

```
        rpc.Call('create_new_story');
}
function newStoryCB(newsid:Number) {
        trace("newStoryCB "+newsid);
        this.sid = newsid;
//      _root.usernameDisplay.text += newsid;
        changed = true;
}
```

Appendix B

Server Code

This is the Python source code for the ComicKit server.

Listing B.1: cn2server.py

```
#!/usr/bin/env python

import SocketServer
import xmlrpcserver
import xmlrpclib
import ConceptNetDB
import os
import sys
import time
import string
import daemonize
import MySQLdb;

file_dir = "/home/breath/data/"
stdoutfile = "output"
logfile = "logfile"
true = 1
```

```
false = 0
port = 8055

def interleave(lol):
    retval = []
    idx = 0
    avelen = len(lol)
    if(avelen < 1):
        return []
    while(avelen/len(lol) > idx):
        avelen = 0
        for list in lol:
            avelen += len(list)
            if(idx >= len(list)):
                continue
            retval.append(list[idx])
        idx = idx + 1
    return retval

def remove_dupes(list):
    for x in list:
        for i in range(1,list.count(x)):
            list.remove(x)
    return list

class ConceptNetXMLServer(xmlrpcserver.RequestHandler):
    allow_reuse_address = 1

    dupnum = 7
    palsize = 35
```

```
def call(self, method, params):
    limit = 0
    if(method.endswith("_limit")):
        method = method[:-6]
        limit = params[-1]
        params = params[:-1]

    # find the appropriate method
    print "Dispatching: ", method, params
    sys.stdout.flush()
    sys.stderr.flush()
    try:
        server_method = getattr(self, method)
    except:
        try:
            # If I don't have it, try the cndb
            server_method = getattr(cndb, method)
        except:
            try:
                # If the cndb doesn't have it, try the
                    nltools
                server_method = getattr(cndb.nltools,
                    method)
            except:
                try:
                    #  Try the MontyLingua beast
                    server_method = getattr(cndb.nltools.
                        m, method)
                except:
```

```
                raise AttributeError, "Server does
                    not have procedure %s" % method

    # make the call
    if(limit > 0):
        return server_method(*params)[:limit]
    else:
        return server_method(*params)

def dump_params(self, params):
    return params

#  user methods
    ------------------------------------------------

def check_username(self, username):
    print "Username:", username;
    db.execute("select uid from user where name='"+
       username+"';");
    result = db.fetchall()
    if(len(result) == 1):
        return 1
    else:
        return 0

def get_incr_usernames(self, partial):
    print "getIncrUsernames", partial
    db.execute("select name from user where name regexp
        '^"+partial+".*';")
    result = db.fetchall()
    print "matching:", result
```

```
        return self.mapfirst(result)

    def new_user(self, username, password, email):
        print "new user", username, password, email
        if(self.check_username(username)):
            return "cannot add existing user"
        db.execute("insert into user set name='"+username+"
            ', password='"+password+"', email='"+email+"',
            addedTime=NOW()")
        return "done"

    def check_login(self, username, password):
        print "checkLogin", username, password
        db.execute("select password from user where name='"+
            username+"';")
        result = db.fetchall()
        if(result[0][0] == password):
            return 1
        else:
            return 0

    def get_uid(self, username):
        db.execute("select uid from user where name='"+
            username+"';")
        result = db.fetchall();
        return result[0][0];

    # story methods
    ---------------------------------------------------------

    def create_new_story(self, username):
```

```
        print "create_new_story", username
        uid = self.get_uid(username)
        db.execute("insert into story set uid='%s';" % uid)
        db.execute("select sid from story where numupdates
            ='0';")
        sid = db.fetchall()[0][0]
        db.execute("update story set numupdates=1 where
            numupdates=0 and uid='%s';" % uid)
        return sid

    def save(self, sid, storyrep):
        print "save ", sid, "contents=%s" % MySQLdb.
            string_literal(storyrep)
        sys.stdout.flush()
        db.execute("update story set contents=%s, numupdates=
            numupdates+1 where sid='%s';" % (MySQLdb.
            string_literal(storyrep), sid))

    def get_story(self, sid):
        print "get_story",sid
        db.execute("select contents from story where sid='%s
            ';" % sid)
        results = db.fetchall()[0][0];
        print "results ", results
        sys.stdout.flush()
        return eval(results, {'__builtins__': {}})

    def get_user_stories(self, username):
        print "get_user_stories",username
        uid = self.get_uid(username)
```

```
        db.execute("select sid from story where uid='%s';" %
            uid)
        results = db.fetchall();
        print "results ", results
        sys.stdout.flush()
        return results

    def delete_story(self, sid):
        print "delete_story", sid
        db.execute("delete from story where sid='%s';" % sid)
        return true

    def save_state(self, lop):
        print >> log, lop
        log.flush()

    def crunch(self, *args):
        return self.crunch2(args)

    def crunch3(self, sid, args):
        self.save(sid,args)
        return self.crunch2(args)

    def crunch2(self, args):
        personsuggestions = []
        actionsuggestions = []
        objectsuggestions = []
        sentencesuggestions = []
        objectsused = []
        personsused = []
```

```
actionsused = []
self.save_state(args)

# correct for added experiment element
args = args[1:]

for panel in args:
    (selected, double, sentence, temporal, persons,
       actions, objects, thoughts) = panel

    objectsused.extend(objects)
    personsused.extend(persons)
    actionsused.extend(actions)

    # sentence suggestions
    print "Selected ", selected
    if(selected == 1):
        print "Have ",
        sentencesuggestions = self.make_sentences(
           self.mapfirst(persons), self.mapfirst(
           actions), self.mapfirst(objects), thoughts
           )
        print sentencesuggestions


# person suggestions
personsuggestions = interleave([self.dupe(x, self.
   dupnum) for x in self.mapfirst(personsused)])[:
   self.palsize]
```

```
personsuggestions = self.convert_to_randoms(
    personsuggestions)

# object suggestions
objectsuggestions = remove_dupes([string.upper(x) for
    x in interleave([self.get_analogous_words(string.
    lower(x)) for x in self.mapfirst(objectsused)])])
    [:self.palsize]
objsfromactions = remove_dupes([string.upper(x) for x
    in interleave([self.get_word_actionable_objects(
    string.lower(x)) for x in self.mapfirst(
    actionsused)])])
objsfromactions2 = remove_dupes([string.upper(x) for
    x in interleave([self.get_word_actions(string.
    lower(x)) for x in self.mapfirst(actionsused)])])

objectsuggestions = interleave([objectsuggestions,
    interleave([objsfromactions, objsfromactions2])])
objectsuggestions = self.convert_to_randoms(
    remove_dupes(objectsuggestions))

# action suggestions
actionsuggestions = remove_dupes([string.upper(x) for
    x in interleave([self.get_actions(string.lower(x)
    , self.palsize/len(objects)) for x in self.
    mapfirst(objectsused)])])[:self.palsize]
actionsfromactions = remove_dupes([string.upper(x)
    for x in interleave([self.get_actions(string.lower
    (x), self.palsize) for x in interleave([
    objsfromactions, objsfromactions2])])])[:self.
```

```
        palsize]
    actionsuggestions = self.convert_to_randoms(
        interleave([actionsuggestions, actionsfromactions
        ]))

    personsused.extend(personsuggestions)
    if(len(actionsuggestions) > 0):
        actionsused.extend(actionsuggestions)
    else:
        actionsused = []

    if(len(objectsuggestions) > 0):
        objectsused.extend(objectsuggestions)
    else:
        objectsused = []
    return [personsused, remove_dupes(actionsused),
        remove_dupes(objectsused), sentencesuggestions]

# utility functions
-------------------------------------------------------

def convert_to_randoms(self, list):
    return [ [x, "random"] for x in list]

def mapfirst(self, list):
    return map(lambda x: x[0], list)

def dupe(self, obj, num):
    return [obj for i in range(1,num)]
```

```
def make_sentences(self, persons, actions, objects,
    thoughts):
     result = []
     sent = cndb.nltools.m.theMontyNLGenerator.
        generate_sentence

     # lowercase the inputs
     persons = map(string.lower, persons)
     actions = map(string.lower, actions)
     objects = map(string.lower, objects)
     if(len(persons) == 0 and len(actions) != 0 and len(
        objects) != 0):
          for x in actions:
               for y in objects:
                    result.append(sent([x,y]))
     elif(len(actions) == 0 and len(persons) != 0 and len(
        objects) != 0):
          for x in persons:
               for y in objects:
                    result.append("%s IS WITH %s" % (x, y))
     elif(len(objects) == 0 and len(actions) != 0 and len(
        persons) != 0):
          for x in persons:
               for y in actions:
                    result.append(sent([y,x]))
     elif(len(objects) != 0 and len(actions) != 0 and len(
        persons) != 0):
          prepositions = ["on the", "with the", "next to
             the"]
```

```
    for x in persons:
        for y in actions:
            for z in objects:
                for a in objects:
                    if(a != z):
                        for p in prepositions:
                            result.append(sent([y,x,"
                                the " +z,p+" "+a]))
                            result.append(sent([y,x,"
                                a " +z,p+" "+a]))
                        result.append(sent([y,x,z,a])
                            )
                result.append(sent([y,x,"the " + z]))
                result.append(sent([y,"the " + z,x]))
elif(len(persons) != 0 and len(thoughts) != 0):
    result.extend([x + " was" for x in persons])
else:
    result.append(" ")
    # add ", thinking x" and ", saying x" to all of
        these

additions = []
if(len(thoughts) > 0):
    for thought in thoughts:
        # remember that each thought is a list of two
            elements
        clause = self.thought_verb(thought[1]) + " \"
            " + string.strip(thought[0]) + "\""
        additions.extend([self.add_trailing_clause(x
            , clause) for x in result])
```

```
        additions.extend(result)
        result = additions
        result = map(string.upper, result)  # uppercase it
           for the return
        return result

    def add_trailing_clause(self, sent, clause):
        sent = sent.replace(".", ",")
        return sent + " " + clause

    def thought_verb(self, image):
        if(image.count("thought") > 0):
            return "thinking"
        if(image.count("speech") > 0):
            return "saying"

    def get_context_words(self, query):
        """This method is essentially the same as get_context
            except it returns a list of strings."""
        temp = cndb.get_context(query)
        result = []
        for x in temp:
            result.append(x[0])
        return result

    def get_analogous_words(self, query):
        """This method is essentially the same as
           get_analogous except it returns a list of strings.
           """
```

```
        temp = cndb.get_analogous_concepts(query)
        result = []
        for x in temp:
            result.append(x[0])
        return result

    def get_word_actions(self, word):
        temp = self.get_edges_of_word(word)
        actions = []
        for list in temp:
            for x in list:
                if(x[0] == "CapableOf" or x[0] == "
                   CapableOfReceivingAction"):
                    actions.append(x[1])
        return actions

    def get_word_actionable_objects(self, word):
        temp = self.get_edges_of_word(word)
#       print "Edges of ", word,"----------------"
#       print temp
        objects = []
        for list in temp:
            for x in list:
                if(x[0] == "ConceptuallyRelatedTo"):
                    objects.append(x[1])
        return objects



    def get_actions(self, query, limit):
```

```
        """ Gets concepts related to the parameter via
"CapableOfReceivingAction" or "CapableOf" links.  Gets
   actions from a
'getAnalogous' with the given word."""
        searchwords = [query]
        searchwords.extend(self.get_analogous_words(query)
           [:5])
        retval = remove_dupes(interleave([self.
           get_word_actions(word)[:(limit/len(searchwords)+2)
           ] for word in searchwords]))[:limit]
        return retval


    def get_edges_of_word(self,textnode):

        """
        returns a list of tuples of outgoing edges from a
           word
        """
        decode_node,encode_node,decode_word = cndb.
           decode_node,cndb.encode_node,cndb.decode_word
        zipped2nodeuid,nodeuid2zipped = cndb.zipped2nodeuid,
           cndb.nodeuid2zipped
        zipped2edgeuid,edgeuid2zipped = cndb.zipped2edgeuid,
           cndb.edgeuid2zipped
        fw_edges,bw_edges = cndb.fw_edges,cndb.bw_edges
        textnode=textnode.strip()
        encoded_node = encode_node(textnode)
        node_uid = zipped2nodeuid(encoded_node)
```

```
        fes = map(edgeuid2zipped,fw_edges.get(node_uid,[])
            [:1000])
        fes.sort(lambda x,y:y[2]+y[3]-x[2]-x[3])
        bes = map(edgeuid2zipped,bw_edges.get(node_uid,[])
            [:1000])
        bes.sort(lambda x,y:2*(y[2]-x[2])+1*(y[3]--x[3]))
        pp_fw_edges = map(lambda x:(decode_word(x[0]),
            decode_node(nodeuid2zipped(x[1])),str(x[2:])),fes)
        pp_bw_edges = map(lambda x:(decode_word(x[0]),
            decode_node(nodeuid2zipped(x[1])),str(x[2:])),bes)
        output = (pp_fw_edges, pp_bw_edges)
        return output



# Main method, which loads the ConceptNet database
if __name__ == '__main__':
    # detach the process from this terminal, meaning it runs
        in the
    # background, forever
    print "Daemonizing"
    daemonize.createDaemon()

    # redirect standard out to a logfile
    fsock = open(file_dir + stdoutfile, 'w')
    sys.stdout = fsock
    sys.stderr = fsock
    # open other peripheral files (gotta work on my naming
        convention)
    print "Opening save files"
```

```
log = open(file_dir + logfile , "a")

pred_filename = "/home/breath/game/conceptnet2/predicates
    .txt"
print "Loading Predicates from %s..."%pred_filename
os.chdir("/home/breath/game/conceptnet2/")
cndb = ConceptNetDB.ConceptNetDB(None, pred_filename)
os.chdir("/")
sys.stdout.flush()
sys.stderr.flush()
print "Opening a connection to the database"
conn = MySQLdb.connect(user='comickit',passwd='xxxxxx',db
    ='comickit',host='localhost')
db = conn.cursor()
done = 0;
while(not done):
    try:
        server = SocketServer.TCPServer(('', port),
            ConceptNetXMLServer)
        server.allow_reuse_address = 1
        done = 1
    except:
        print "Could not open socket, sleeping 120
            seconds"
        sys.stdout.flush()
        sys.stderr.flush()
        done = 0
        time.sleep(120)
        print "Retrying"
print "Serving"
```

```
    sys.stdout.flush()
    sys.stderr.flush()
    try:
        server.serve_forever()
    except:
        print "Exiting due to exception"
        conn.close()
        sys.stdout.flush()
        sys.stderr.flush()
        log.flush()
        server.server_close()
        sys.exit(0)
```