

# [ENGR2510] Software Design Final Project : Madlib Generator

Emily Wang, Nicole Rifkin, Maura Cosman

Spring 2013

# 1 Introduction

"madlibbery" is a madlib generator module coded entirely in Python. Its features include the ability to make a madlib from self-composed text or from a random Wikipedia article, processing to choose blanked words at random based on natural language toolkit processing, and functionality with either a command-line program or a GUI implementation.

## 2 Installation Guide

"madlibbery" is currently compatible with Python 2.7. The current version cannot be done with Python 3 because a full NLTK module is not yet available for Python 3, and the language-processing features of NLTK is vital for madlib generation.

Here is a list of the required modules to run "madlibbery". All of these modules are either preinstalled on Ubuntu or can be installed via "sudo pip install (insert module name here)" via the Ubuntu command line; further information can be accessed at the documentation pages for the corresponding module if needed:

1. bs4 (Beautiful Soup)
2. nltk (Natural Language Tool Kit)
3. numpy
4. random
5. re
6. sys
7. termcolor
8. threading
9. Tkinter
10. urllib2

## 3 User Guide

This program was designed to be a fun twist on a favorite game of ours. In order to start the program, run `python final_madlib_gui.py` from the command line in Ubuntu. It is important to make sure that all of the necessary modules are downloaded beforehand. From there, the game is relatively self-explanatory. A series of `input()` statements will prompt the player to go through the madlib, and features such as a "Need inspiration?" button are even included in case the user gets stuck while trying to think of words for the madlib.

In order to use the command line version, run `python final_madlib_gui.py` from the command line. The command line version is not as intuitive, and the user is required to hit enter after every input. However, it is much more robust to typos/invalid answers and much more difficult for the player to break the code.

## 4 Design Guide

The design guide provides details on the project goals, overcoming challenges to make the program more robust, and functions within the "madlibbery" module.

### 4.1 Goals

The first objective of this project was to write code for a working madlib generator command-line program. The minimum deliverable for the command-line program would take in a string of text (the story composed by the user), replace some of the original words with blanks, and be able to fill in these blanks with new words (more inputs from the user). After the command-line program was completed, we proceeded to work towards several other goals for this project as well.

One goal of this project was to learn how to implement a GUI, which was relatively successful. We chose Tkinter because of its simplicity, and because of this, we were able to get a working model with almost all of the features that we wanted in a short amount of time.

A second goal was to write a working function to scrape a URL for text which could be converted to a madlib. Initially, we attempted to write an algorithm that would work for any URL. Even after running *Beautiful Soup* (a python module created for URL scraping) we were getting undesirable text from the URLs that we used. After several attempts to write a function that compared the number of spaces, the total number of characters, the number of periods and commas, and the total number of symbols, we failed to come up with a reliable algorithm that would separate ideal madlib text from the gibberish that *Beautiful Soup* was returning. Instead, we adapted our goal to be specific to Wikipedia. By studying the source code of several random articles, we used regular expressions to reliably scrape a chunk of text. The madlib takes the first blurb about the desired subject: a paragraph characteristic of almost all Wikipedia articles. These paragraphs are an optimal length and (usually) in colloquial language to begin with, making them ideal madlib material.

### 4.2 Challenges

The biggest challenge of this project was making the code friendly to all kinds of user inputs and detect invalid answers to prompts. Since, by nature, the program is heavily controlled by the user, we discovered all kinds of ways that the player could break our code in the command line, and had to find ways to prevent the program from crashing easily. For example, we had to make sure that the text was long enough to be converted into a madlib, that the player did not request more blanks than there were words in the text, that they used an integer input when necessary, and that they didn't make typos. In addition, the URL scraping feature added another level of complexity. We discovered the rare case in which a Wikipedia article has no paragraph text (in some cases it can be only a graph or a table), then the code would break. To solve this, if the program is unable to take text from a certain article, it defaults to using the Wikipedia article on computer science.

Adapting the entire code, especially these features, to work with Tkinter proved to be especially difficult. This process involved simplifying our code (i.e. implementing the relatively newer features after the bare-bones GUI started working) substantially in the GUI version. Even so, in some aspects, the GUI made the program more elegant/user friendly. For example, rather than inserting one word at a time, the user inputs all of the words at once in individual text boxes all in the same window, eliminating the need for an "undo" button. There were also fewer opportunities to make typos, since most of our game could be controlled by premade buttons. However, we found that Tkinter was incompatible with a lot of features that made it less robust, despite these modifications. For example, we could not use try and except commands within the GUI. Therefore, the code can break if they user types a string when the program requires an integer, or if the random Wikipedia article it happens to pull has no paragraph text. Finally, one bug that we discovered during our demonstration on Wednesday that we had not anticipated was the lack of a scroll bar. If the user chooses to create a large number of blanks in their madlib, and there is enough text to allow it, the list will

go off of the screen. The user will then be unable to finish the game because the "submit" button is not within reach. Details like these made the process of designing a madlib maker very challenging.

### 4.3 Important Functions

The code was split into several functions in order to make debugging and organization easier. The first function was a welcome function that displayed the first message and then called all other functions. Next was a set of nested functions that determined where the madlib would get its text from, and then returned that text. The next function passed in the text, tagged each word with its part of speech, and randomly replaced words with blanks. It then output a list of words and blanks in addition to a dictionary where the keys were the words removed and the values were both the index of that word in the story and the part of speech of that word. The next function asked for the users input to fill in the blanks, and output the final madlib. The basic structure of these functions were similar in both the command line version and the GUI version for simplicity and consistency.

Here is a breakdown of the functions in the `madlib_maker.py` file (The script itself is also relatively well-commented). This list will have the functions in "chronological" order (assuming the beginning is when you first see the welcome message printed in the terminal).

1. **madlibbing()** takes in no parameters and is the user interface function. This function calls the other functions as the user goes through the madlib experience.
2. **blankage(inputstory\_list)** takes in one parameter, which is the input story as a list; each element of the list is a string of an individual word in the story to be made into a madlib.
3. **playage(madlib)** takes in one parameter, in which `madlib[0]` is the madlib stored as a list (each element of of the list is a string, which is either a word or a "blank") and `madlib[1]` is the dictionary of blanked-out words in which the key is the original word, and the value is a tuple in which the first element is the index of the blanked-out word in the original `inputstory_list`
4. **inspiration()** is a function nested within `playage`. It prints a randomly selected word from our "bank" of inspirational words and returns nothing. It's essential to the "Need inspiration?" feature.
5. **timing()** is a function involved in the time delay for the "Need inspiration?" line to appear - this time delay aspect is only present in the command-line version.

## 5 Self Evaluation

While we were all very happy with how the final deliverable turned out, there are a lot of things that we would have done differently if we were to do this project again. We would have most likely chosen a different GUI module to learn and implement. Although Tkinter is relatively simple, it is not particularly elegant and has very few customizable options. If we had continued to use Tkinter, we would have most definitely chosen to learn and implement a decorator rather than nesting all of the functions. Since functions called by buttons in Tkinter can't take inputs, there were many instances where functions were within several layers of functions. The code got increasingly complicated as the Madlib progressed. Using a decorator for a generic "submit" button would have condensed many lines of code and would have made the script cleaner and much more easy to follow. Some of the faults in our code are detailed above, and using a different GUI may have solved some of those problems that were not present in the command line but were present in the GUI.

One of the pieces that we were most satisfied with was the integration of random Wikipedia articles, as well as the 'Need inspiration?' button. Both of these features are ones that can't be found in tradition madlibs, and enhance the overall gameplay experience. In addition, both provided an interesting challenge from a coding perspective.

Finally, one of the original goals of this project was to learn some web development and get this code

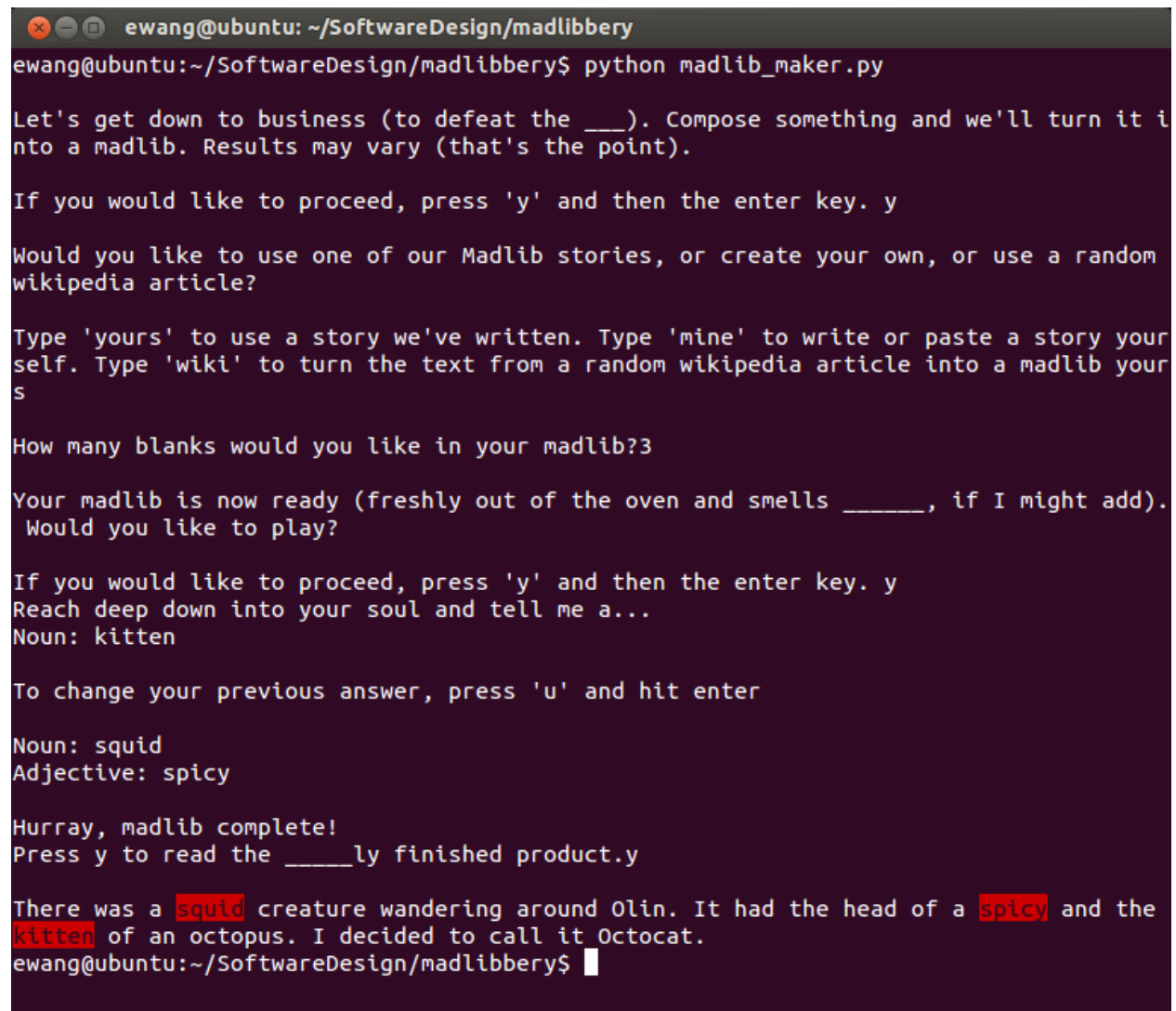
working on a website. In this case, we probably took on a little bit more than we could handle in the time given. However, we hope to continue learning web development with python/flask over the summer and plan to ultimately get it up and running on a herokuapp.

## 6 It works!

As requested, here is screenshot evidence that our madlib generator works.

### 6.1 Command-line program screenshots

Note: screenshots of the overall madlib game in the overall command line are shown for brevity. Each step (whenever the program mentions prompts such as "press 'y'" or "Type 'yours'") is driven by a user input.



```
ewang@ubuntu: ~/SoftwareDesign/madlibberrry
ewang@ubuntu:~/SoftwareDesign/madlibberrry$ python madlib_maker.py

Let's get down to business (to defeat the ____). Compose something and we'll turn it i
nto a madlib. Results may vary (that's the point).

If you would like to proceed, press 'y' and then the enter key. y

Would you like to use one of our Madlib stories, or create your own, or use a random
wikipedia article?

Type 'yours' to use a story we've written. Type 'mine' to write or paste a story your
self. Type 'wiki' to turn the text from a random wikipedia article into a madlib your
s

How many blanks would you like in your madlib?3

Your madlib is now ready (freshly out of the oven and smells _____, if I might add).
Would you like to play?

If you would like to proceed, press 'y' and then the enter key. y
Reach deep down into your soul and tell me a...
Noun: kitten

To change your previous answer, press 'u' and hit enter

Noun: squid
Adjective: spicy

Hurray, madlib complete!
Press y to read the _____ly finished product.y

There was a squid creature wandering around Olin. It had the head of a spicy and the
kitten of an octopus. I decided to call it Octocat.
ewang@ubuntu:~/SoftwareDesign/madlibberrry$
```

Figure 1: Using the madlib generator with the "Use one of your stories" option. This option uses the default story stored in the madlib generator program about Octocat. The original story is "There was a strange creature wandering around Olin. It had the head of a cat and the body of an octopus. I decided to call it Octocat."

```
ewang@ubuntu: ~/SoftwareDesign/madlibberrry
ewang@ubuntu:~/SoftwareDesign/madlibberrry$ python madlib_maker.py

Let's get down to business (to defeat the ____). Compose something and we'll turn it i
nto a madlib. Results may vary (that's the point).

If you would like to proceed, press 'y' and then the enter key. y

Would you like to use one of our Madlib stories, or create your own, or use a random
wikipedia article?

Type 'yours' to use a story we've written. Type 'mine' to write or paste a story your
self. Type 'wiki' to turn the text from a random wikipedia article into a madlib wiki

The title of your madlib is: Muhammad Shafiq Jamal

How many blanks would you like in your madlib?5
We don't think that this text will make a good madlib. Please try again.

Let's get down to business (to defeat the ____). Compose something and we'll turn it i
nto a madlib. Results may vary (that's the point).

If you would like to proceed, press 'y' and then the enter key. █
```

Figure 2: In the scenario that the input story is less than 10 words or the Wikipedia page material is not ideal, the program advises the user to restart.

```
ewang@ubuntu: ~/SoftwareDesign/madlibbery
ewang@ubuntu:~/SoftwareDesign/madlibbery$ python madlib_maker.py

Let's get down to business (to defeat the ____). Compose something and we'll turn it into a madlib. Results may vary (that's the point).

If you would like to proceed, press 'y' and then the enter key. y

Would you like to use one of our Madlib stories, or create your own, or use a random wikipedia article?

Type 'yours' to use a story we've written. Type 'mine' to write or paste a story your self. Type 'wiki' to turn the text from a random wikipedia article into a madlib mine
Type or paste your story here Olin College is a small engineering school in Needham, MA. Contrary to the typical engineering school stereotype, there is much fun and joy of learning to be had at this quirky institution.

How many blanks would you like in your madlib?3

Your madlib is now ready (freshly out of the oven and smells _____, if I might add).
Would you like to play?

If you would like to proceed, press 'y' and then the enter key. y
Reach deep down into your soul and tell me a...
Noun: phoenix

To change your previous answer, press 'u' and hit enter

Noun: u
Noun: frank-the-phoenix
Noun: tree
Noun: hand

Hurray, madlib complete!
Press y to read the _____ly finished product.y

Olin College is a small engineering frank-the-phoenix in Needham, MA. Contrary to the typical engineering school tree there is much fun and joy of hand to be had at this quirky institution.
ewang@ubuntu:~/SoftwareDesign/madlibbery$
```

Figure 3: Using the madlib generator with the "mine" (aka compose my own story) option.

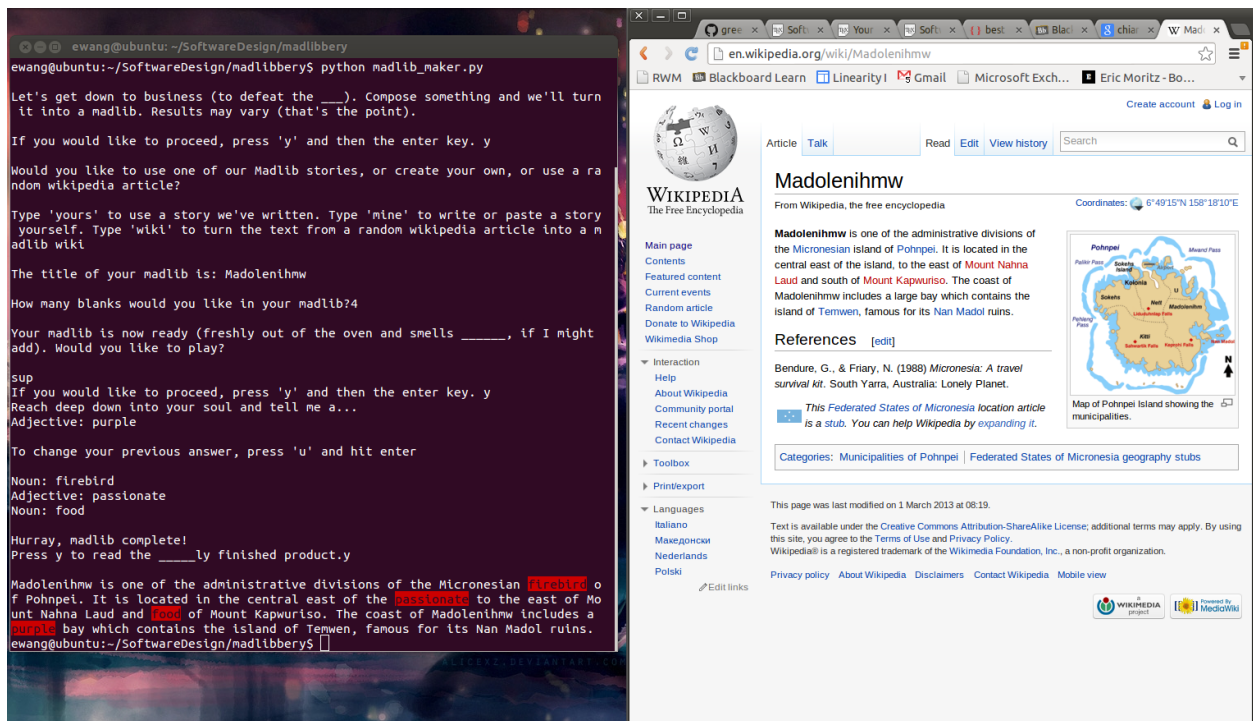


Figure 4: Using the madlib generator with the random Wikipedia article option. The screenshot includes the corresponding Wikipedia article as a reference for what the program scraped and used for the story text.

## 6.2 GUI program screenshots

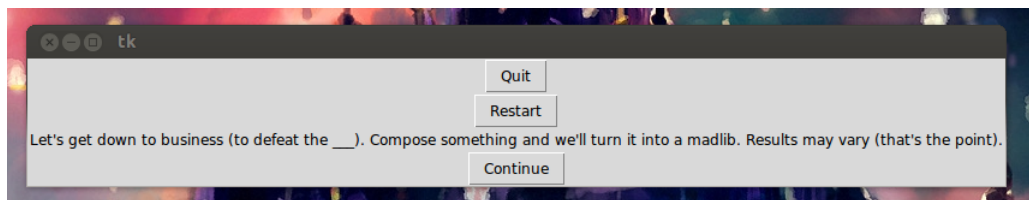


Figure 5: The first page that appears when using the madlib generator GUI.





Figure 6: After pressing "continue" on the welcome screen, the user has several options for input story sources.

### 6.3 GUI Story Source: Use one of your stories

When clicked, this option will allow the user to play through the default story already stored in the program. There is no intermediate screen; afterwards the process for filling in the blanks, etc is identical to the other options' routes.

### 6.4 GUI Story Source: Write or paste my own story

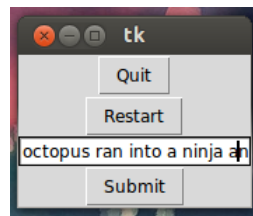


Figure 7: The page that appears when the user chooses to compose his/her own input story.

### 6.5 GUI Story Source : Take text from a random Wikipedia article

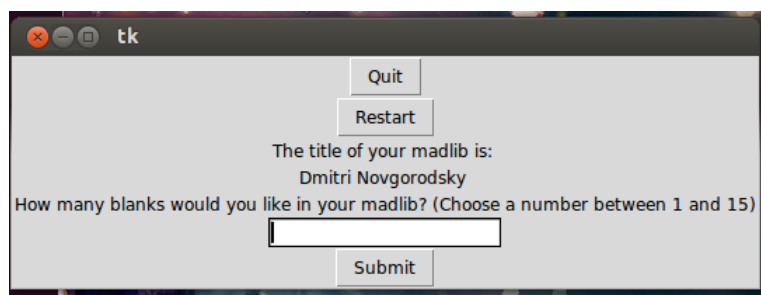
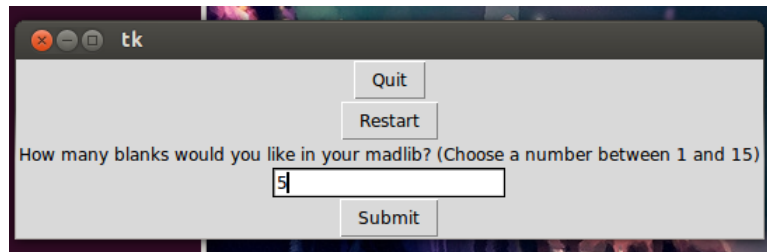


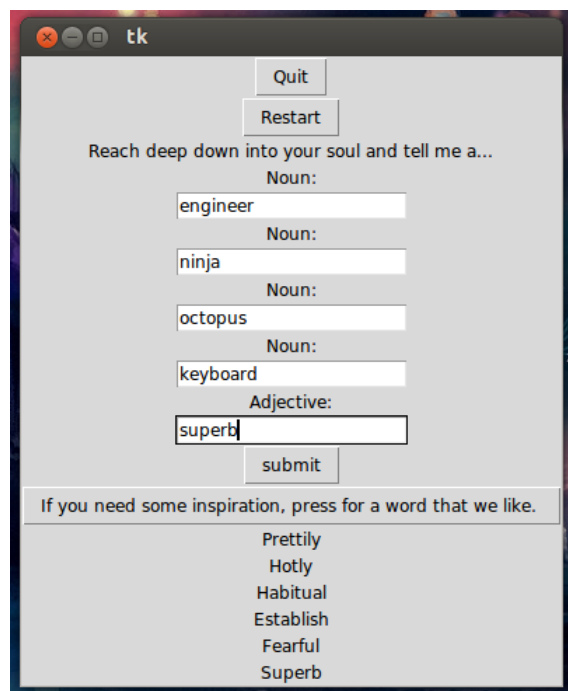
Figure 8: One slight difference with the "Take text from a random Wikipedia article" option is that the user is shown the title of the randomly selected article when the program asks him/her for the number of blanks desired.

## 6.6 Filling in the blanks and onwards



A screenshot of a Tk window titled "tk". The window has a light gray background. At the top, there are three window control buttons (close, minimize, maximize) and the title "tk". Below these, there are two buttons: "Quit" and "Restart". The main text in the window reads: "How many blanks would you like in your madlib? (Choose a number between 1 and 15)". Below this text is a text input field containing the number "5". At the bottom of the window is a "Submit" button.

Figure 9: After the user has decided on a madlib source, the program asks the user how many fill-in-the-blanks s/he would like in the madlib.



A screenshot of a Tk window titled "tk". The window has a light gray background. At the top, there are three window control buttons (close, minimize, maximize) and the title "tk". Below these, there are two buttons: "Quit" and "Restart". The main text in the window reads: "Reach deep down into your soul and tell me a...". Below this text, there are five input fields, each preceded by the label "Noun:" and followed by a "submit" button. The input fields contain the following text: "engineer", "ninja", "octopus", "keyboard", and "superb". Below the input fields, there is a label "Adjective:" followed by an input field containing the text "superb". At the bottom of the window, there is a text box containing the text: "If you need some inspiration, press for a word that we like." Below this text box, there is a list of words: "Prettily", "Hotly", "Habitual", "Establish", "Fearful", and "Superb".

Figure 10: Now it's time to fill in the blanks! The "Need inspiration" feature is present on this screen.

## 6.7 Example madlib solutions

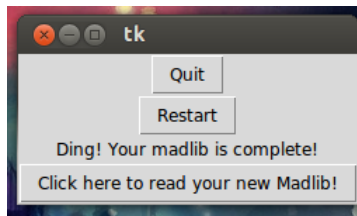


Figure 11: The blanks have been filled, now it's time to see the result!

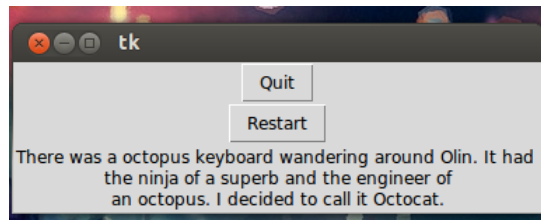


Figure 12: The madlib solution to the "Use one of yours" option in this particular example, in case you were curious.

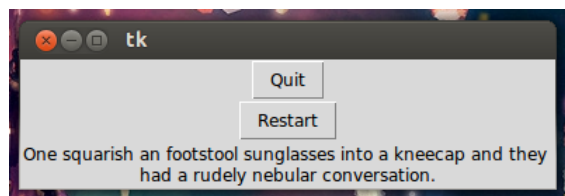


Figure 13: The madlib solution to the "compose my own" option in this particular example, in case you were curious.

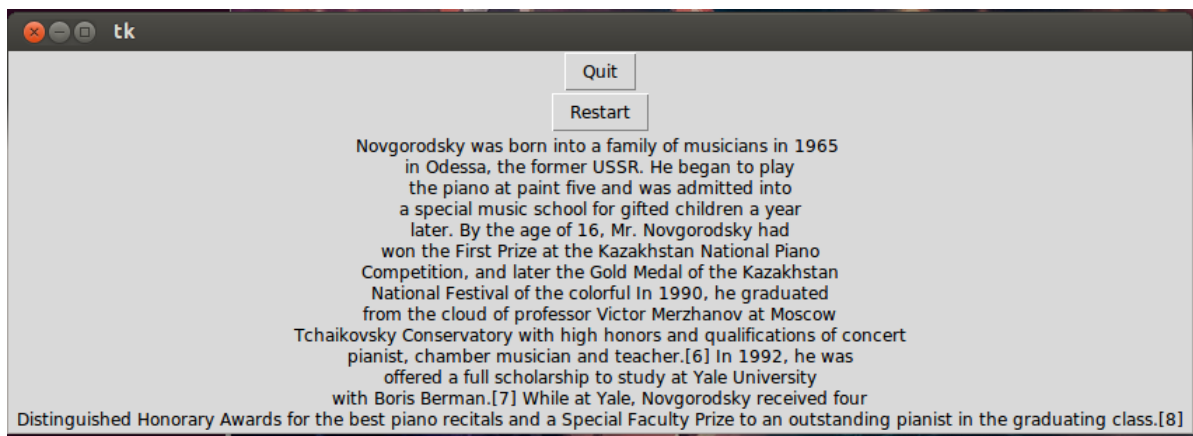


Figure 14: The madlib solution to the "Take text from a random wikipedia article" option in this particular example, in case you were curious.