Visual Tool for Searching for Japanese Characters

Aaron Reyes
2016/12/02
areyes2@uccs.edu

Project Introduction

In English, when a word is encountered that one does not know, finding that word in a dictionary is a simple process. Since there are so few letters in the English alphabet, without any further context, one can simply transcribe the letters from it into a dictionary and find the information they are looking for. For languages with non-roman writing systems, this process is often much time consuming and frustrating. This application was developed to provide a visual interface for "drawing" desired characters in Japanese, and producing a list of results that best match the character that was input. This application would best be used to compliment the features of an already existing dictionary, providing users with an easy way to take a character they do not know and quickly get information about it.

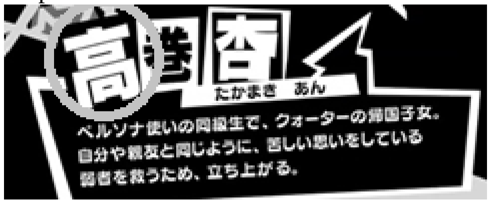
I. Application Overview and Features

This application is built to provide users with a tool to draw a Japanese character that they would like to search for into a matrix, and then get a set of results that should contain their desired character. To make the drawing process easier, users can open an image file that will be displayed behind the matrix for easy tracing. A dictionary that contains all the characters to be compared against needs to be loaded in before any searches can be completed. When a search is to be executed, the current value that is stored in the matrix is compared against all values in the dictionary, and those who are within a similarity margin are displayed to the results panel. The application was written with JavaScript, using html for a user interface.

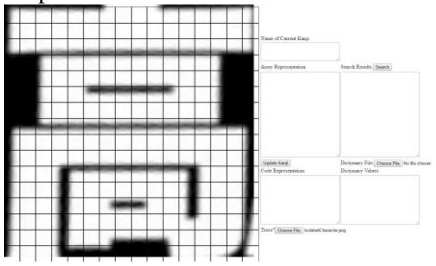
A. Use

The intended use case for this application is for finding details about Japanese characters encountered in media whose text cannot be directly accessed, such as subtitle text or text found in images, so they can look it up in a dictionary. A user would then open the application, draw the character they are interested in finding (potentially with the help of a reference image they capture), and then copy their desired character from the list of results into a separate dictionary. The character deemed most likely to be the correct answer will have the lowest number next to itself in the search results pane. To use, the html file called "KanjiSearch.html" can simply be opened in an html5 compatible browser (tests and debugging done in Google Chrome). For proper operation, the application needs access to a dictionary file that should be included in the enclosing file for the application. The application has been shown to provide correct results with a small test dictionary on a set of test values. It has yet to be tested rigorously on a *complete* dictionary with many test values. An example case follows:

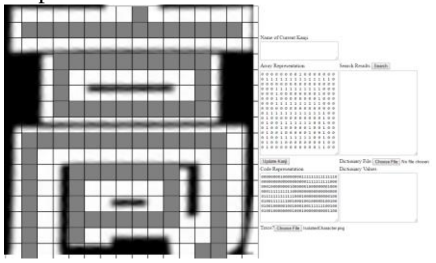
Step1: A desired character is found


高巻杏
たかまき あん
ペルソナ使いの同級生で、クォーターの帰国子女。
自分や親友と同じように、苦しい思いをしている
弱者を救うため、立ち上がる。

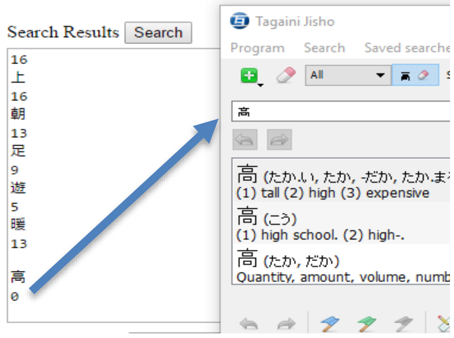
Step2: Isolate the character and load it into the application



Step3: Trace the outline



Step4: Load dictionary and perform search



```
16
上
16
朝
13
足
9
遊
5
暖
13
高
0
```

高 (たか.い, たか, -だか, たか.ま
(1) tall (2) high (3) expensive
高 (こう)
(1) high school. (2) high-.
高 (たか, だか)
Quantity, amount, volume, numb

II. Application Components and Working State

This application takes a binary square matrix representation of a Japanese character and creates a linear array from the values stored within the matrix. This array is then compared against the arrays of the characters stored in the dictionary using the concept of *dynamic time warping* which is generally used to compare time series data. This allows the array that is entered by the user and the array corresponding to the desired character to be slightly different and still return correctly. At the current time, this application is just a framework for a fully working system. Currently there is no way for the application to create a full dictionary of values to check against, and as such, all values found in the dictionary have been manually entered.

A. Application Components

The "Kanji" class is the backbone of every data item in this application. Each Kanji object has an associated *code* and *name*. When the application is instantiated, a Kanji object is generated for the form, this Kanji's data members are set by modifying the matrix on screen, and when a search is ready to be completed, the Kanji is updated with the current state of the matrix. The Kanji Dictionary is populated with Kanji objects found within the file the user selects. After a search is completed, the results pane is populated with tuples containing Kanji objects, and the distance from the Kanji it was compared against. The matrix itself is made from a square grid of flex cells that automatically maintain a size 50% of the window. Each matrix cell contains a button that has a reference to itself. When a button raises a clicked event, functions are called to update the value at that cells location within the matrix, and the visual state of the cell itself.

B. Application Working Status

Currently the application cannot be used practically as there is no way for the application to build a dictionary automatically. For the dictionary to be complete it would need at least 2,000-3,000 of the most commonly used words, and for the dictionary to be complete it would need up to 50,000 total characters. These numbers are unreasonable to input by hand, so developing a method of creating this dictionary is the next step to making the application practically useful. However, the application does work in returning the desired character while reducing the number of undesired results in test cases done on manually created dictionaries. As the number of entries into the dictionary increase, the number of false returns after a search also increase, but the desired character remains among them. As there are only 256 elements within the arrays, when characters become more complex filling more of the grid, the number of false returns from a search are also increased.

C. Testing the Application

To test the application, open the "kanjidictionary.txt" file found in the folder that contains the "KanjiSearch.html" file. Open the "KanjiSearch.html" application and for those characters found within the dictionary, attempt to draw them into the matrix. To test the tracing functionality, load in the "ExampleKanji.png" file found in the same folder, and attempt to trace it into the matrix. After the character has been drawn into the matrix and you are ready to perform a search, press the "Update Kanji" prompt to ensure that the current Kanji object has the most up to date information. Then, load in the dictionary file name "kanjidictionary.txt" using the file prompt. After these steps have been completed, pressing the "search" prompt will populate the search results box with the characters most close to that which was drawn into the matrix. If the program works correctly, the expected character should be among the results, and the expected character should have a low number corresponding to it. The lower the number the closer the returned character was to the character drawn in the matrix.

III. Modularity

The main modules of the program are the Kanji class and square-grid which forms the visual matrix. The square-grid class provides abstraction for the user's Kanji. The Kanji class is made up of the combination of an array representation of its code and its name, they are grouped together to make referencing both simpler. The Kanji's internal values can only be accessed by making changes to the matrix created by the square-grid class's update methods. Currently, the Kanji class just possesses member variables for its name and its code. As the Kanji class is modified to contain different member variables or methods, the existing modules need not be altered to support this development. The overall program is built to separate the total problem into several sub-problems. Generating a visual representation of a character, managing the internal data corresponding to that representation, calculating the results, and displaying them to the user. The interfaces of the abstractions are kept simple and few in number, with most of the work being done inside the implementation of the enclosing modules.

IV. Extensibility

The number of data and function abstractions included in the application are limited. The Kanji class acts as an abstraction of data that pertains to each Kanji object. Since the Kanji class is the main abstraction found within the program, there are no other types with which to have subtype relationships. All functions found within the program are dynamically bound functions. The square-grid class is made up of square-grid_cell elements that can be of any number. To extend the current implementation into a version that has more granularity, such as a version of the application that has a 25x25 matrix instead of the 16x16 default size, all that needs to be modified is the html body, all other details of the program will continue to work as expected. To continue to make the program easier to use, methods would be introduced for the user to fill out the visual matrix by clicking once to identify a start point, and then clicking on a different cell in the matrix to fill every block between the two. That change would only require altering the updateKanjiArray function, no other functions rely on its specific implementation.

V. Reusability

The Kanji class can be inherited from to create a class that has more functionality. The Kanji class could be extended to contain members that would aid in the creation of a working dictionary like word definitions and component details. The square-grid class that is used to form the basis of the visual matrix can be applied to any application needing some number of automatically rescaling elements. The entire program was designed to complement an existing dictionary, so its components can be extracted in part or in whole for an application's needs. The main algorithmic processing done by this program is the process of comparing two arbitrarily length arrays through the process of dynamic time warping. This process can be inherited from to compare any two sets of data. This application was designed for its components to be flexible and reusable, such that it could be incorporated into a larger system without the need for too high a degree of refactoring.