

**Grocery Shopping Assistant for the
Blind (GroZi)**

UCSD TIES Winter 2009

Advisor:

Serge Belongie

Community Client:

National Federation of the Blind (NFB)

Client Representative:

John Miller

Team Members:

Grace Sze-en Foo, Michael Tran, Jaina Chueh, Steven Matsasuka,
Raul Marino, Bonnie Han, Nikhil Joshi, Jasmine Nourblin,
Amalia Prada, Marissa Sasak, Hannah Brock, Aritrick Chatterjee

Table of Contents

INTRODUCTION	4
SOYLENT GRID	5
APPROACH AND METHODOLOGY	6
 DATABASE TEAM	
Introduction	7
MySQL	7
Database Java Class.....	10
Entity-Relationship Model	13
Experiment Images	13
User Entered Labels	14
Control Images	14
Future Work.....	15
 USER INTERFACE TEAM	
Problems and Solutions	16
Approach.....	17
How to create styles to alter the visual aspects of the UI	18
How an image is fetched and displayed in the UI.....	19
Fixing grid size and Aspect Ratio of pictures:.....	24
Future Plans for UI	26
 USABILITY TEAM	
What is Usability Team	27
Aim of Usability Team	27
Recaptcha Model	27
Problems and Solutions	28
Previous quarter UI	28
Current UI.....	29
Future Reference	31
Traffic Generation	32
Confidence Level.....	33
Help File	33
Summary	34
REFERENCES	35

List of Tables and Figures

List of Figures

Figure 1: Illustration of SoyLent Grid System	5
Figure 2: Entity-Relationship Model	13
Figure 3: Old User Interface – Output from WiW_Task.java	19
Figure 4: Distorted Images.....	25
Figure 5: UI with Aspect Ratio Fix	26
Figure 6: User Interface from Fall 08	28
Figure 7: User Interface designed in Winter 09	29
Figure 8: Appropriate Control Images	31
Figure 9: Inappropriate Control Images.....	31
Figure 10: Draft version of Future UI	32

List of Tables

Table 1: Changes made to User Interface in Winter 2009.....	17
Table 2: Creating style in html and java	18

Introduction

There are currently 1.3 million legally blind people living in the United States who face daily obstacles with routine tasks, especially in regards to their experiences within supermarkets and stores. Developing assistive technologies and handheld devices allows for the possibility of increasing independence for blind and visually impaired. Currently, many grocery stores treat those that are blind as “high cost” customers, and dramatically undersell to this market, neglecting to take their needs into consideration. The use of computational vision can be advantageous in helping these blind customers, as restrictions such as the limited ability of guide dogs, frequently changing store layouts, and existing resources do not allow for a completely independent shopping experience. Using technologies such as object recognition, sign reading, and text-to-speech notification could allow for a greater autonomous solution to the relevant problem.

In conjunction with Calit2, UCSD’s Computer Vision Lab, and TIES, the GroZi project is working to develop a portable handheld device that can help the blind to collect information and navigate more efficiently within difficult environments as well as better locate objects and locations of interest. GroZi’s primary research is focused on the development of a navigational feedback device that combines a mobile visual object recognition system with haptic feedback. Although still in its early stages of development, when complete, the GroZi system will allow a shopper to navigate the supermarket, find a specific aisle, read aisle labels, and use the handheld MoZi box to then scan the aisle for objects that look like products on the shopper’s list (compiled online and downloaded onto the handheld device prior to going into the store).

This quarter, under the supervision of our advisor, Serge Belongie, we pursued the computer vision aspects of the project that allows for autonomous detection and localization in the near future. Thereby, our team successfully customized the User Interface (UI) for new labeling tasks as well as improved the computer program that allows for inserting and storing data into the database as effortlessly as possible. However, there is still room for improvement. This means that the incoming contributors to this project should continue on improving our codes that could further improve the outcome of the project as a whole. The following document will serve as a description of what we have accomplished thus far, what we have learned and overcome, and the processes involved in designing and implementing a usable and accessible interface for the blind to assist future members of TIES GroZi team.

Soylent Grid

The Soylent Grid, a subcomponent of GroZi, combines Human Computational Tasks (HCTs) and Machine Computable Tasks (MCTs) by labeling images and solving common vision problems such as segmentation and recognition.¹ For GroZi to work efficiently, a database, containing images of grocery store products and their appropriate labels, is needed. Soylent Grid functions by sending these images to a computer which performs an MCT algorithm in order to identify any text labels. This system performs optical character recognition, also known as OCR, on product packaging in order to differentiate between certain grocery items as peanut butter and grapes. Because these items contain differing colors, fonts, and background images, recognizing the product's text is challenging. From a Soylent Grid perspective, the main goal is to obtain differing product images and to obtain such information as the product's brand name and a description of the item. If the computer algorithm cannot successfully perform this task, the image is then sent to users who identify the location and the content of the text (HCTs).

In addition to labeling grocery store products via text, Soylent Grid also functions as a way to secure websites. These Soylent Grid images can be used as a portal before accessing private information such as a bank account or an email address. To protect such personal information, these images can be combined with reCAPTCHA, in which users must enter the text displayed in a distorted image in order to obtain the desired information. In this way, Soylent Grid provides a double service, a win-win situation, by not only labeling images for the GroZi database, but also by providing security for users regarding any personal information.

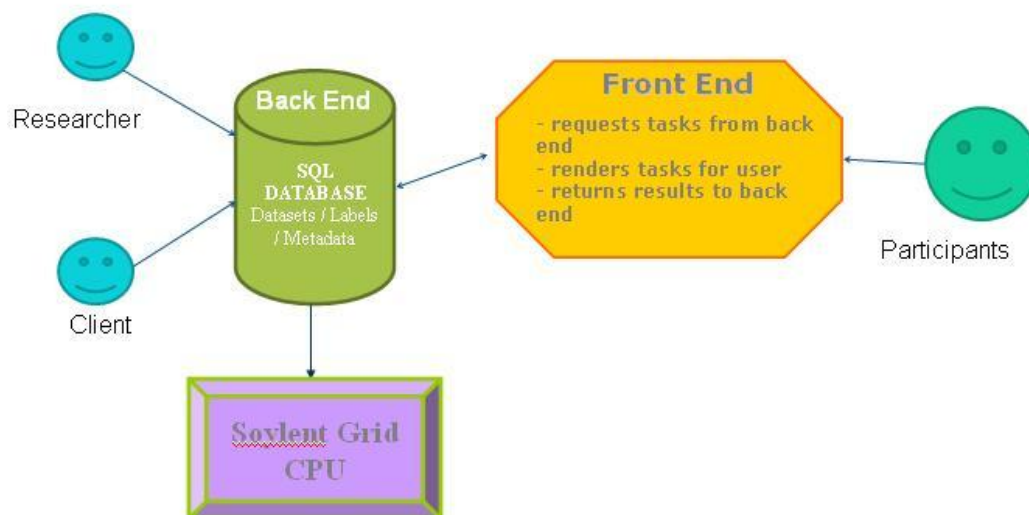


Figure 1. Illustration of Soylent Grid System

Approach and Methodology

In order to design a grocery shopping assistant for the visually impaired, we need a program that will allow us to save images of grocery store items in a real environment. To avoid the intense human labor, we exploited the concept introduced above, Soylent Grid. In order to proceed, we then split into three groups this quarter to set up the necessary preliminary steps:

- **User Interface** : Amalia Prada, Aritrick Chatterjee, Jasmine Nourblin, Bonnie Han
 - Customizing User Interface for new labeling tasks and storing results into a database for use in future computer vision systems.
- **Database**: Marissa Sasak, Nikhil Joshi, Raul Marino, Hannah Brock
 - The database team's task consisted of creating the database for storing labels and images.
- **Usability**: Michael Tran, Grace Sze-en Foo, Jaina Chueh, Steven Matsasuka
 - The Usability team ensured that the Soylent Grid interface is efficient, effective and satisfying by critiquing results from the Database and User Interface teams.

Database

Introduction

This quarter, the database team for Grozi created a MySQL database to store all of the grocery shopping product image information being used in Soylent Grid. The MySQL database consists of 3 tables; one to hold the experiment images, one to hold the labels entered by users for the images, and finally a table of control images. After the tables were created, we inserted “dummy data” into the tables for testing purposes. The “dummy data” is currently the only data in the tables.

With sample information in the database, we created MySQL code to query and update the information. The following is a list of functions that can be performed with the current code:

1. Create tables
2. Select all information from any table
3. Select specific rows or columns from any table
4. Delete all information from any table
5. Delete specific entries in a table
6. Insert an entry into any of the tables
7. Checks to see if there are any experiment images that have been shown the specified number of times. If there is, the code checks all of the labels for the image, to see if any are over the set percent confidence. If there are labels for an image that meet the percent confidence criteria, they can be inserted into the control images table.
8. Update the number of times an image has been shown in the UI
9. Update the number of times a user enters a label
10. Check to see if a control image exists
11. Get a random control or experiment image from the database

MySQL

Below are sample mysql commands to insert data into the tables and select data from the tables. Comments are in red.

.....

Database Basics

Author: Hannah Brock (hrbrock@ucsd.edu)

```
/* See all the entries in the Experiment Images table */  
SELECT * FROM Table_Name;
```

```

/* Delete all of the entries in the table */
DELETE FROM Table_Name;

/***** Generic 'Insert' Commands for the tables *****/

/* Insert into ControlImages Table */
/* 1st Param - The Image ID off the original experiment image. If
there never was an experiment image for the control label/image, a
random number can be used here. We will need to code to make sure the
random number chosen is not already used in the control OR experiment
image table.

2nd Param - Filename of the image. If the image was originally an
experiment image, it needs to match
3rd - 6th Params - TopLeftX, TopLeftY, Height, Width

7th - Confident label for the image

8th - Percent confidence of the image label when it was moved from
experiment to control table. If it was never in experiment
table, default is 100. This will need to be coded as well.

9th - This is the time stamp that holds the date and time that the
image was put in the control table. */

INSERT INTO ControlImages VALUES(100,
"file100.jpg",0,0,10,50,"twix",95,NOW());

/* Insert into ExperiemntImages Table */
/* 1st Param - The Image ID of the experiment image. This value is
ALWAYS 0 because the attribute itself is on auto-
increment.

2nd Param - Filename of the image.

3rd - 6th Params - TopLeftX, TopLeftY, Height, Width

7th - The number of times the image is shown in the UI. Default = 0
*/

INSERT INTO ExperimentImages VALUES(0, "file2.jpg",0,0,20,30,0);

/* Insert into UserEnteredLabels Table */

```



```

/* 1st Param - The Image ID of the experiment image that the label is
    being made for
    2nd Param - Label entered by user for particular experiment image

    3rd - The label's ID. This value is ALWAYS 0 because the attribute
    itself is on auto-increment.

    7th - The number of times the label has been entered. Default = 1
*/

INSERT INTO UserEnteredLabels VALUES(4, "Tide",0,1);

/***** Possible Threshold Commands *****/
/* Note that the times shown threshold is set to the generic
 * value of 100. This will need to be changed. Also, the
 * percent confidence check is set at 95%. This can also be changed
 */
/* Delete the images that have been show 100 times */
DELETE FROM ExperimentImages WHERE ShownCount = 100;

/* See all of experiemental images that have been shown 100 times */
SELECT FROM ExperimentImages WHERE ShownCount = 100;

/* Query to find all images that have been shown 100 or more times and
that have a user entered label that has been entered as the same text
95 or more of the times that the image was shown. The query prints out
the image ID number, Image filename, the number of times the image was
shown, the name that the user entered for the image file, and the
number of times that the user entered that specific name. */
SELECT ExperimentImages.ImageID,
ExperimentImages.ImageFileName,ExperimentImages.ShownCount,
    UserEnteredLabels.UserEnteredLabel,
UserEnteredLabels.TimesEntered
FROM ExperimentImages JOIN UserEnteredLabels
WHERE ExperimentImages.ImageID = UserEnteredLabels.ImageID AND
    ExperimentImages.ShownCount >= 100 AND
UserEnteredLabels.TimesEntered >= 95;

/* Insert the experiment images with a label with percent confidence
over 95 into control images. Basically, take a 'snap shot' of
experiment image as it is moved. */
INSERT INTO Control_Images (
    SELECT ExperimentImages.ImageID, ExperimentImages.ImageFileName,
        ExperimentImages.TopLeftX, ExperimentImages.TopLeftY,

```

```

        ExperimentImages.Height, ExperimentImages.Width,
        UserEnteredLabels.UserEnteredLabel, 95, NOW()
    FROM ExperimentImages JOIN UserEnteredLabels
    WHERE ExperimentImages.ImageID = UserEnteredLabels.ImageID AND
        ExperimentImages.ShownCount >= 100 AND
    UserEnteredLabels.TimesEntered >= 95 );

/* Add one to the 'TimesEntered' attribute of a specific label example
*/
UPDATE UserEnteredLabels SET TimesEntered = TimesEntered + 1 WHERE
ImageID = 4;

/* Add 1 to the 'ShownCount' for an experimental image example */
UPDATE ExperimentImages SET ShownCount = ShownCount + 1 WHERE ImageID
= 1;

/* See if a control image with a specified Label and ImageID exists
example */
SELECT * FROM ControlImages WHERE ImageLabel = "twix" AND ImageID =
100;

/* Get a random row from the ExperimentImages table */
SELECT * FROM ExperimentImages ORDER BY RAND() LIMIT 1;

```

Database Java Class

After we created the MySQL code, we integrated some of the commands into a java class. The following is a description of the functionality of the Database java class 'Database.java'. The class contains methods to:

1. Connect to the GROZI database.
2. Get a random control image or/and experiment image. If an experiment image is generated, its 'ShownCount' is increased.
3. Allow a user to see if a label is already in the 'UserEnteredLabels' database table. If it is, the label's 'TimesEntered' attribute is increased by one. If it is not in the table, it gets added.
4. The user can check if a control image with a given 'ImageID' and label exists. This is so that we can check to see if the user enters the correct control image label when working with the UI.

Below is a sample run of the Database.java class on the dummy data in the database. For the queries, dummy information is fed to the function calls from main. Comments explaining what the outputs mean are colored red.

* Execution of the program starts by telling you whether the connection to the database is successful

Database connection established

* When a random image is queried, that image's 'TimesShown' attribute is increased by 1, so that we know how many times an image has been shown in the UI.

Incrementing Random Experiment Image with ImageID = 6's 'TimesShown' attribute...

* These are the attributes of the random experiment image that is generated

RANDOM EXPERIMENT IMAGE ATTRIBUTES

ImageID: 6

ImageFileName: file2.jpg

TopLeftX: 0

TopLeftY: 0

Height: 20

Width: 30

Image Type: 1

* These are the attributes of the random control image that is generated

RANDOM CONTROL IMAGE ATTRIBUTES

ImageID: 100

ImageFileName: file100.jpg

TopLeftX: 0

TopLeftY: 0

Height: 10

Width: 50

Image Type: 0

ImageLabel: twix

PercentConfidence: 95

Time Stamp: 2009-03-13 18:28:59.0

* This was the result returned when we entered ImageID = 4 (the ImageID that links the label to the experiment image that it is the label for) and ImageLabel = 'Tide' into the function call that figures out if a Label exists in the UserEnteredLabels table. The result is valid, as there was a label in the table with these attributes. The function increases the times the label was entered for the specific experiment image also.

* In main, we also do a test run with an ImageID and ImageLabel that are NOT in the table. When this occurs, the program successfully adds

the new label to the table and initializes the times it has been entered to 1.

Label Tide Exists...

Incrementing 'TimesEntered' attribute...

* The two outputs below are tests done in main that call the function to check if there is an entry in the ControlImages table for the ImageID and ImageLabel fed to the function. The first is correct, as there was an item with ImageID = 100 and ImageLabel = 'txix' in the ControlImages table. 'True' displayed below it is the Boolean value returned to main from the function call. The second call to the function was given information for an image that was not in the table. It correctly prints that the item does not exist in the ControlImages table and Boolean False is returned to main and displayed.

Control Image with ImageID = 100 and ImageLabel = twix Exists!

true

Control Image with ImageID = 105 and ImageLabel = NotAnImage DOES NOT Exist.

False

When the java code gets a random image from the database, it returns all the image attributes. The attributes are going to be used by the UI team in the future. The image attributes will describe the image's filename and bounding box information that needs to be displayed. Once the user enters text for an image, the UI team can evaluate what the user entered to check if it exists as a label in the table. They can do this by using the Database class. The code will create a new label if the label that the user entered does not exist. The UI team will also be able to use the code to figure out if the user enters the correct label for the control image when they enter text into with the UI.

Entity-Relationship Model

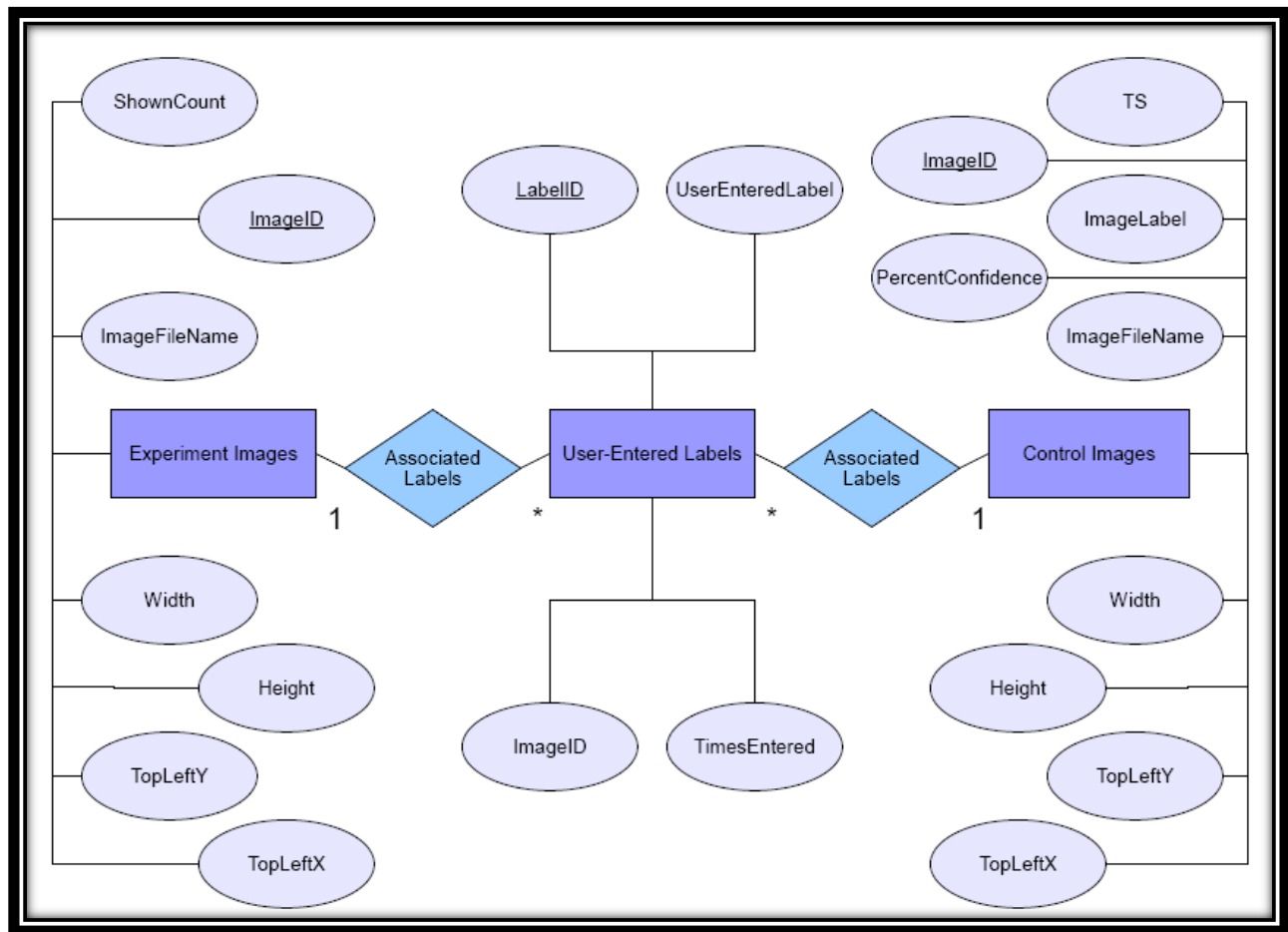


Figure 2: Entity-Relationship Model

LEGEND: The “Associates Labels” is the relationship that links the tables together. The “1” and the “*” mean that there is one experiment image linked to many User-Entered Labels, and so on. The ellipses correspond to attributes, while the rectangles represent tables.

Experiment Images

1. ImageFileName: This is a foreign key that relates the image to the Control Images table, and it describes the location of the picture (i.e. image5.jpg).
2. ShownCount : This is the number of times the image, or ImageFileName is shown in the UI for labeling.
3. Width, Height, TopLeftX, TopLeftY: These are the coordinates of the bounding box. The bounding box is given along with the ImageFileName.

4. ImageID: The primary key of this table. This ID contains the information that is created in the User-Entered Labels table. In other words, the ImageID is what relates each table. Each ImageFileName can have several ImageID's depending on how many different ways the image has been labeled by users.

User Entered Labels

1. LabelID: The primary key associated with this table. There are several LabelIDs for each ImageID. Each LabelID keeps track of the UserEnteredLabel. For example, someone could label one ImageID as "twix" and someone else could label the same ImageID as "tix", so these would both get different LabelIDs.
2. UserEnteredLabel: This is the text that the user enters to label the image.
3. ImageID: This is the table in which ImageID is first created, and then sent to Experiment Images or Control Images. It contains the UserEnteredLabel and LabelID. There are several LabelID for each ImageID.
4. TimesEntered : Each LabelID is associated with how many times a UserEnteredLabel. The TimesEntered keeps track of how many times the exact filename and text for the filename has been inputted. So if there was a twix bar and three people labeled it "twix" and 2 people labeled it "tix", each of those labels would get its own LabelID with a count of 3 or 2, respectively.

Control Images

1. ImageFileName – This is the same as the ImageFileName in Experiment Images. When an image is moved from Experiment to Control, it will still contain the history it had in Experiment.
2. ImageID: The primary key in this table. This contains the same information that Experiment Images contained except that it has a final ImageLabel, Coordinates, Perfect Confidence, ImageLabel associated with it. Each ImageFileName can have several ImageID associated with it. There may be several ImageID because we might move the same ImageFileName to Control Images if more than one ImageID has reached the threshold.
3. TS: This is a timestamp assigned to when the experiment image has been moved to Control Images.
4. ImageLabel: This is text users have entered with the highest percent confidence. In other words, it is the "UserEnteredLabel" that achieved the highest confidence.

5. PercentConfidence: We will use this value for how many times an image needs to be displayed before its labels are evaluated. Once at this threshold, we can use the number of times it was shown, ShownCount, to evaluate the PercentConfidence of the labels for the image
6. Width, Height, TopLeftX, TopLeftY: the same coordinates associated with ImageFileName in each table.

Future Work

If next quarter's focus is also on the Soylent Grid portion of the GroZi project, then the members of the database team should focus on the following:

- Replace the existing dummy data with all the GroZi 120 images.
- Integrate JAVA code with the User Interface team so that the fused system can perform the following functions: (1) call attributes to such as the experiment and control images from the database, (2) display those images, (3) receive the typed text for the images and send that back into the database
- Possibly change to a system in which user draws box around a given text.
- Test the system on a small scale at first, such as within the group on the Soylent Grid website.
- Test the system on a larger scale: WEBCT, possibly Amazon in the future?
- Label thousands of grocery store images with the help of millions of internet users.
- Integrate system with camera/device capable of image recognition for the final goal of being able to provide item information to the Blind.

User Interface

The role of the User Interface team as a subset of the GroZi team is to implement a user interface for Soylent Grid. This interface will be user-friendly, and modeled after the reCAPTCHA user interface, to make the labeling task simple and quick for the user.



To view the current user interface for Soylent Grid, see this website-development server: <http://grozi.freeclinicproject.org:8180/SoylentGrid/>

For the sake of efficiency this quarter, the team has assumed that bounding boxes will be pre-provided with each image. To program the interface, Java was used with an integrated development environment, Eclipse. Eclipse not only compiles and runs the script efficiently, but it also accesses the libraries more easily. In addition to this, Google Web Toolkit was used to build the JavaScript. Thanks to Subversion, the team was able to simultaneously work on pieces of the code and update their latest work for the others to build on.

Problems and Solutions

Soylent Grid's user interface (UI) has seen many modifications over the past ten weeks. The images below depict the user interface's appearance on January 5th, 2009 and its appearance by the end of the quarter, March 13th, 2009. Beneath the screen shots is a table outlining all the changes made to the UI by the conclusion of Winter quarter 2009.

Table 1: Changes made to User Interface in Winter 2009

BEFORE: January 5, 2009	AFTER: March 13, 2009
	
<ul style="list-style-type: none"> • White color scheme 	<ul style="list-style-type: none"> • UCSD blue and gold color scheme
<ul style="list-style-type: none"> • Widget changed sizes according to the largest image 	<ul style="list-style-type: none"> • Widget has a fixed size of 300 by 150 pixels • Images have a fixed size of 150 by 150 pixels
<ul style="list-style-type: none"> • Rectangle and polygon tools displayed, as well as a “choose tool” button 	<ul style="list-style-type: none"> • These tools have been disabled
<ul style="list-style-type: none"> • Only one image displayed 	<ul style="list-style-type: none"> • Two images now display—an experiment image and a control image
<ul style="list-style-type: none"> • Image not centered horizontally or vertically 	<ul style="list-style-type: none"> • Image centered horizontally
<ul style="list-style-type: none"> • Images were subject to distortion 	<ul style="list-style-type: none"> • Images are no longer distorted- aspect ratio is fixed
<ul style="list-style-type: none"> • Inconsistent refresh time (3-7 seconds) due to difference in image sizes 	<ul style="list-style-type: none"> • Time to refresh has been reduced (3-5 seconds) due to fixed image sizes

In addition, the Winter 2009 UI team now has an updated website:
http://ties.ucsd.edu/projects/gsa/index_files/page0007.htm

Approach

This quarter, we worked in `WiW_Task.java`, `Image.java`, and `SoylentGrid.html`.

`WiW_Task.java`: Displays a picture with a toolbox.

`Image.java`: Contains a new `Image` class that stores the necessary data information about the image: the path (the file path name where the image is located), the height, the width, and the `imgType` (distinguishes between control and experimental image).

- `SoylentGrid.html`: Uses Cascading-Styles Sheet to alter the visual aspects of the UI, such as the background color and size of the buttons.

How to create styles to alter the visual aspects of the UI

To add a background color, it is necessary to create a style definition in SoyLentGrid.html first, and then call it from the WiW_Task.java document. Several examples are shown on the table below that explains how to create a style:

Table 2: Creating style in html and java

SoylentGrid.html code. Creations of style:	WiW_Task.java code to implement the style:
<pre>.gwl-button { width: 55px; height: 45 px; text-align: center; border: 1px solid #FFFFFF; background: #191970; padding: 3px; color: rgb(255,255,255); }</pre>	<pre>refresh.setStyleName("gwl-button"); // "refresh" is the reference variable to an instance of a button class. We then call the method setStyleName on this instance.</pre>
<pre>.gwl-background { width: 197px; text-align: center; background: #FDF2CC; padding: 2px; }</pre>	<pre>RootPanel.get().setStyleName("gwl- background"); //sets the style "gwl- background" to the entire panel.</pre>
<pre>.gwl-simpleCalendar { width: 197px; text-align: center; background: #FFD700; padding: 2px; }</pre>	<pre>main.setStyleName("gwl-simpleCalendar"); //sets the style "gwl-simpleCalendar" to the grid.</pre>
<pre>img { display: block; text-align: center; margin-left: auto; margin-right: auto; margin-top: auto; margin-bottom: auto; }</pre>	<pre>//the code on the left written in SoylentGrid.html centers the images on the grid.</pre>

The main goal for the User Interface this quarter was to create a more visually appealing interface that is modeled after reCAPTCHA, alter the code so that two images now display – an experimental and a control image, and create a fixed size grid so the grid does not resize with each refresh.



Figure 3: Old User Interface -- Output from WiW_Task.java

As shown in Figure 3 above, the old user interface was only able to incorporate one image into the widget. This was because the code in WiW_Task.java was fixed in such a way that one function calls another, which calls another, until the final image is displayed. Hence, even when getRandomImg() was called again in an attempt to get the second image, it failed. This happens because after fetching the first image, when the second image is fetched, all the data of the first image is overwritten by the data in the second image.

How an image is fetched and displayed in the UI

Call Hierarchy to get an image:

WiW_Task constructor → getFiles()
 getFiles() → Calls getRandomImg()
 getRandomImg() → sets the width and height of the image, Calls changeImg()
 changeImg() → sets the width/height of image in the drawing canvas
 Return from calls into getFiles() → Call displayInterface()

To remedy this problem, a new Image class is created:

```

public class Image {
    public String path;
    public int height;
    public int width;
    public int imgType;

    public Image(String mypath, int myheight, int mywidth, int myimgType){
        path = mypath;
        height = myheight;
        width = mywidth;
        imgType = myimgType;
    }
}

```

The Image class stores the information of an image. It has one constructor which initializes all passed in parameters to the data fields of the image.

- Path: The path name of where the image is located.
- Height: The height of the image.
- Width: The width of the image.
- ImgType: A flag to distinguish between control and experimental images. If imgType = 0, the image is a control; otherwise, if imgType = 1, the image is experimental.

To accommodate the new Image class, the following code was also added or changed:

- Two new Image objects are created at the beginning of file – ‘image1’ and ‘image2’.

In getFiles(): [line 271]

```
image1 = getRandomImg(1); /* experimental image */
image2 = getRandomImg(0); /* control image */
displayInterface(image1, image2);
```

Image1 and image2 calls getRandomImg() and passes in a hardcoded number to generate, either 1 for an experimental image, or 0 for a control image. This is just temporary to distinguish between the two images. In the future, there will be a randomizer function that automatically generates either a control or experimental image for image1 and use the opposite for image2.

In getRandomImg(int): [line 308]

```
String curImages = (String)files.get(random);
```

The file is accessed from this line of code to get the path name of where the image is located. The path name is stored into a variable ‘curImages’ and then passed into the Image constructor below so it is stored into the image’s path data field.

```
Image myimage = new Image(curImages, width, height, conExpImage);
getImage(myimage);
return myimage;
```

getRandomImg() is adjusted so it can fetch multiple images and store them in different locations so that the second image can be created without overwriting the data of the first image. This is done by allowing getRandomImg to return an image object. This function creates a new image object with the default constructor and returns the image back to getFiles() after the other functions set its width, height, and so on.

In getImage(Image): [line 317]

```
int[] temp = new int[2];
temp = (int[]) result;
```

```

int curWidth = temp[0];
int curHeight = temp[1];
currentImage.width = curWidth;
currentImage.height = curHeight;

```

getImage() is also adjusted so it takes in an Image object. It gets the width and height of the image and sets the width and height data field of the Image object so it can be accessed later on in displayInterface().

In changeImg(Image): [line 354]

The Document Object Model (DOM) is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of HTML, XML and related formats. DOM is the way JavaScript sees the containing HTML page and browser state.

```

Element
    e1 = DOM.getElementById("draw"),
    e2 = DOM.getElementById("temp"),
    e3 = DOM.getElementById("Picture"),
    e4 = DOM.getElementById("draw2"),
    e5 = DOM.getElementById("Picture2");

```

changeImg() sets the width and height of the picture in the drawing canvas. Initially, changeImg() only had Elements e1 and e3, which is the draw area for the image, so it wasn't able to accommodate two images. The draw area of the first was constantly being replaced by the second. To solve this problem, a second draw area was created, Element e4 and e5, for the second picture.

```

String curImage = currentImage.path;
int curWidth = currentImage.width;
int curHeight = currentImage.height;
int expConImage = currentImage.imgType;

```

At the beginning of changeImg(), the Image object's data fields are accessed to obtain the width, height, and imgType (control/experimental). It then checks whether the image is either a control or an experiment and sets the corresponding element to that particular image's height and width.

```

DOM.setElementProperty(e2, "width", Integer.toString(curWidth));
DOM.setElementProperty(e2, "height", Integer.toString(curHeight));

//java.util.Random
if (expConImage == 0) {
    DOM.setElementProperty(e1, "width", Integer.toString(curWidth));
    DOM.setElementProperty(e1, "height",

```

```

Integer.toString(curHeight));

DOM.setElementProperty(e3, "width", Integer.toString(curWidth));
DOM.setElementProperty(e3, "height",
Integer.toString(curHeight));

DOM.setImgSrc(e3, "Pictures/WiW/" + curImage);
}
if (expConImage == 1) {
DOM.setImgSrc(e5, "Pictures/WiW/" + curImage);

DOM.setElementProperty(e4, "width", Integer.toString(curWidth));
DOM.setElementProperty(e4, "height",
Integer.toString(curHeight));

DOM.setElementProperty(e5, "width", Integer.toString(curWidth));
DOM.setElementProperty(e5, "height",
Integer.toString(curHeight));
}

```

In displayInterface(): [line 60] (Putting it All Together)

Finally, once the draw area for each image is set, the function calls return back to getFiles() and getFiles() calls displayInterface(), which sets up the final UI. Now that the data fields of both images are set, these data fields are accessed at the beginning of displayInterface():

```

String curImage = myImage1.path;
int curWidth = myImage1.width;
int curHeight = myImage1.height;

String curImage2 = myImage2.path;
int curWidth2 = myImage2.width;
int curHeight2 = myImage2.height;

```

Since we now have two draw areas, the second drawArea is also initialized for the second image (The JsGraphicPanel is made to draw on and display a picture):

```

final JsGraphicsPanel drawArea = new JsGraphicsPanel("draw");
final JsGraphicsPanel drawArea2 = new JsGraphicsPanel("draw2");

```

A new clickArea must also be created for the second image.

```

final FocusPanel clickArea2 = new FocusPanel(drawArea2);
clickArea2.setSize(Integer.toString(curWidth2) + "px",
Integer.toString(curHeight2) + "px");

```

Furthermore, the buttons were rearranged to create a more visually appealing UI:
This displays the first image in the grid:

```
VerticalPanel panel = new VerticalPanel();
panel.add(clickArea);
panel.setSize(Integer.toString(curWidth) + "px",
Integer.toString(curHeight) + "px");
```

This code was added to display the second image in the grid:

```
VerticalPanel toolPanel = new VerticalPanel();
toolPanel.add(clickArea2);
toolPanel.setSize(Integer.toString(curWidth2) + "px",
Integer.toString(curHeight2) + "px");
```

The buttons were moved to the last column and first row of the grid:

```
VerticalPanel toolPanel1 = new VerticalPanel();
toolPanel1.add(refresh);
toolPanel1.add(submit);
toolPanel1.add(help);
```

These lines set the first and second textbox in the corresponding grid:

```
HorizontalPanel buttonPanel = new HorizontalPanel();
buttonPanel.add(tb);

HorizontalPanel buttonPanel2 = new HorizontalPanel();
buttonPanel2.add(tb2);
```

The last panel currently contains nothing but in the future, we hope to move the submit button here.

```
HorizontalPanel buttonPanel3 = new HorizontalPanel();

Grid main = new Grid(2,3);
main.setBorderWidth(3);
main.setWidget(0, 0, panel);
main.setWidget(1, 0, buttonPanel);
main.setWidget(1, 1, buttonPanel2);
main.setWidget(1, 2, buttonPanel3);
main.setWidget(0, 1, toolPanel);
main.setWidget(0, 2, toolPanel1);
main.setStyleName("gwl-simpleCalendar");
```

A rough draft of how the grid is structured is shown below:

(0,0) FIRST IMAGE	(0,1) SECOND IMAGE	(0,2) REFRESH SUBMIT HELP Buttons
(1,0) TEXTBOX	(1,1) TEXTBOX	(1,2) NULL

With all of these changes in place, Soylent Grid is finally able to display the second image. However, to keep this behavior consistent even after the “Refresh” button is triggered, the following code was added to the code that triggers the refresh button on line 169:

```
image1 = getRandomImg(1); /* experimental image */
image2 = getRandomImg(0); /* control image */
displayInterface(image1, image2);
```

...and this is how both images are displayed.

Fixing grid size and Aspect Ratio of pictures:

Now that the second image appears in the UI, there is a problem with the grid size. Since the picture in the directory the UI is fetching from consists of pictures in many different sizes and shapes, every time a new image is reloaded, the large picture would resize the grid. This behavior is undesirable because it creates inconsistent user experiences and different load times depending on how large the image is.

The GroZi team agreed for the fixed-grid to be 300 by 150 pixels so the widget is small enough to be incorporated into other websites.

These changes were integrated into the WiW_Tool.java in the following ways:

New static variables were created at the beginning of this class to hardcode the values 150x150 into the grid:

```
public static final int width = 150;
public static final int height = 150;
```

In getRandomImg(), these two values are passed into the Image constructor to automatically set the width and height to the value 150:

```
Image myimage = new Image(curImages, width, height, conExpImage);
```


After this change was implemented, the Soylent Grid interface remained a fixed size no matter the size of the image. However, it created a new problem; images that did not fit the grid ratio of the draw area were stretched and distorted. Of the images in the database, the Gatorade bottle was the most distorted. See figure 4 below for an example:



Figure 4: Distorted images

Fixing the aspect ratio of the images only requires a simple check. After the image's width and height is obtained from `getImage()`, inside `changeImg()` on line 366 the following checks were made:

```
float ratio = 0;

if (curWidth > curHeight)
    ratio = ((float)width / (float)curWidth);
else
    ratio = ((float)height / (float)curHeight);

curWidth = Math.round(curWidth * ratio);
curHeight = Math.round(curHeight * ratio);
```

There is a condition to check which side of the image is greatest (either the width or the height). Then the largest side of the image is taken and scaled down to fit the 300x150 grid width. In doing so, the browser also automatically resizes the shorter side of the image. The resulting change creates a more visually appealing picture as shown in Figure 5. As shown, both the Snyder's Pretzels and (especially) the Gatorade bottle look proportionate after the change:



Figure 5: UI with Aspect Ratio fix

Although the aspect ratio has been fixed, the key thing to keep in mind is the image scaling is still done by the browser. Therefore, with each refresh, the full image still reloads. This is a problem that the future GroZi team may work on. GroZi's user interface has come a long way this quarter but there are still more changes on the horizon.

Future Plans for UI:

- Integrate LightboxJS (<http://www.huddletogether.com/projects/lightbox/>) into the UI so users can easily enlarge the images.
- Integrate the Database with the UI so the UI can fetch an image from the Database, and so the Database may retrieve a submitted input from the user.
- Add a randomizer to the UI so experimental and control images are randomized.
- Create smaller, pictorial buttons (i.e. replace the Submit button with a return arrow, replace the help button with a "?") and move the submit button to the bottom right panel.
- Add hover text. (i.e. When hovering over refresh, it would say "get a new image." When hovering over Help, it would say "need help?" When hovering over Submit, it would say "submit your answer.").
- Create and incorporate a logo for Soylent Grid.

Usability

What is Usability Team

Soylent Grid consists of a User Interface which displays grocery images and records user inputs. These user inputs are then sent to and stored in the database, where the inputs are labeled as either control or experiment. In order for the Database and the User Interface Teams to successfully accomplish these tasks, the teams put in long, difficult hours of programming work. While the teams are programming, issues such as “user friendliness” can seem unimportant and can be very easily overlooked. Although these issues seem small, they are equally as important to create a successful Soylent Grid experience for our valuable users. The Usability Team was created so that these issues do not get overlooked.

Aim of Usability Team

Soylent Grid is powered by its users. Because Soylent Grid depends on its users to be the power behind the labeling of the Grozi database, Soylent Grid must not upset or chase the user away. Thus, the Usability Team worked with both the User Interface Team and the Database Team to ensure that the Soylent Grid widget remained “friendly” to its users. The Usability Team did most of its work with the UI team. The UI team created the widget, and the Usability team gave feedback and suggestions about how to improve the design. The Usability Team basically designed the layout of the widget, selecting necessary buttons and features. The Usability Team also worked with the database team with confidence levels. These confidence levels were what decided if an image could be considered a control image. In summary, the Usability Team did study and made suggestions about any possible problem that could possibly make the user unhappy.

Recaptcha Model

Recaptcha is a parent project, very similar to Soylent Grid. In Recaptcha, two captchas are displayed for labeling. The captchas are used to block spam bots from accessing websites. However, Recaptcha displays one captcha with a known correct answer (control), and one captcha in which the correct answer is not known (experiment). The experiment captchas are actually words from books that need to be digitized. When users successfully enter the control answer correct, the experiment answer is assumed right and the responses are stored in the database. Experiment words are digitized when they reach a certain confidence in which they are considered control words. In summary, instead of simply paying people to manually label and digitize these books, Recaptcha found a way to get free labor from the mass computing power available on the net. Soylent Grid is modeled after the Recaptcha idea; we instead display grocery images to be labeled for the GroZi project.

Problems and Solutions

Previous quarter UI



Figure 6: User Interface from Fall 08

- I. Labeling

Four options were considered for product labeling. If there was infinite time and supply of support, the bounding box should outline each letter according to its individual curve. The second option is to have a rectangle around the whole word. The third option is to draw the bounding box around the product (see Figure 6), and the user will be prompted to type in the words they see in the region. Finally, the user interface can have no bounding box and users are free to type in anything they see.
- II. Bounding Box (Assumption of given bounding box)

The bounding box should hold only one word.
- III. What should users type?

The ideal situation would be to get the user to label every word on the product. This will accelerate the amount of time needed for image labeling. However, this would require too much work from the user. Therefore, while this is ideal, it is not realistic. One option would be to give the user a word, and ask them to draw a bounding box around the word provided.
- IV. Polygonal tools

The “Select Polygon Tool” seen in the old interface should be eliminated. It is too complicated for users to comprehend easily.
- V. Distortion of image/Resizing

To ensure that Soylent Grid images are associated with a fixed grid, images should be stretched to Soylent Grid margins while maintaining ratio size of height and width. This should not pose a problem with blurry pictures as the extension is not extreme.
- VI. Loading time Soylent Grid

Assuming a good connection speed, the load time should last no longer than three to five seconds. Waiting an excessive amount of time creates user frustration.

The difference in image sizes, grid sizes and loading time creates an inconsistency in user experience. If it is possible, the team should aim to quantify the longest loading time. Here,

the Usability team considered two possibilities – discard, keep or resize the image. Our team concluded that the image should be kept for statistical purposes, but also resized to reduce the average loading time. In fact, it will be excellent if Soylent Grid can someday detect connection speed in order to give users with slow connections small images and the like for fast users.

Current UI



Figure 7: User interface designed in Winter 09

I. Zooming option – Lightbox (include code)

Since the widget is modeled after the ReCAPTCHA interface, the images might sometimes be too small for the texts to be identifiable. To solve this issue, we implemented the Lightbox JS feature so the users can click on the images to get a larger view that overlays the current web page.

Lightbox JS using html:

```
<script type="text/javascript" src="lightbox.js"></script>
```

For animated text “Loading...” and “Close”, add the following lines to the top of lightbox.js

```
var loadingImage = 'loading.gif';
```

```
var closeButton = 'close.gif';
```

File needed: lightbox.js, loading.gif, close.gif

Lightbox JS using CSS:

```
#lightbox{
    background-color:#eee;
    padding: 10px;
    border-bottom: 2px solid #666;
    border-right: 2px solid #666;
}
#lightboxDetails{
```

```

        font-size: 0.8em;
        padding-top: 0.4em;
    }
    #lightboxCaption{ float: left; }
    #keyboardMsg{ float: right; }

    #lightbox img{ border: none; }
    #overlay img{ border: none; }

    #overlay{ background-image: url(overlay.png); }

    * html #overlay{
        background-color: #000;
        back\ground-color: transparent;
        background-image: url(blank.gif);
        filter:
        progid:DXImageTransform.Microsoft.AlphaImageLoader(src="overlay.png"
        , sizingMethod="scale");
    }

```

Files needed: lightbox.css, overlay.png

Files can be downloaded from <http://www.huddletogogether.com/projects/lightbox/>

II. Picture icons instead of text (Readjust second column)

More and more internet interfaces now choose picture icons over text because pictures are easier for human eyes to register. People are more willing to read pictures than texts.

III. Hover text

To reduce excessive texts on the interface and still provide enough information for users to use the widget without confusion, we added hover text for the icon buttons. The buttons now have hover texts which display the purpose of each button.

IV. Double Clicks on Image

Double clicks on the images will not have any affect because internet users are used to this effect. We do not want to introduce features that users are not familiar with because they might cause confusion.

V. Right clicking

The affect that right clicks have on the images will depend on the web browser that the user is using.

VI. Differences in control/experiment size

Because the images from the control and experiment sets are placed side-by-side, we want to keep the images the same size to avoid stretching or shrinking the interface, which might frustrates the webmasters and the users. To resolve this issue, the size of the interface will be fixed, and images of different sizes will be stretched or shrunk to the interface size while maintaining its aspect ratio.

VII. Which information is significant?

Although the brand for the image product is important, texts containing information such as flavor and quantity of sugar contained are just as important. Thus, there will be bounding boxes for different texts for the same image.

VIII. Refresh Images (2 images simultaneously)

When the “refresh” button is clicked, both images will be refreshed to decrease the chance for spam bots to determine which image is the control image and which image is the experiment image.

Future Reference

Coloured Bounding Boxes

In cases where bounding boxes concurrently contain the brand name and small words (e.g. Tic Tac brand with Wintergreen flavor), either word should be taken as correct. To eliminate the bias of results, the future User Interface team might consider having color-coded bounding boxes. For example, the bounding box surrounding “Tic Tac” is green, but the bounding box containing “Wintergreen” is blue. This way, we can instruct the user to only type in the word in the color specific box.

Control Images

Appropriate control images need to be shown so not to overly frustrate users who cannot decipher the SoyLent Grid images. Figure 8 and 9 are some examples of what the control image should and should NOT look like:



Figure 8: Appropriate Control Images

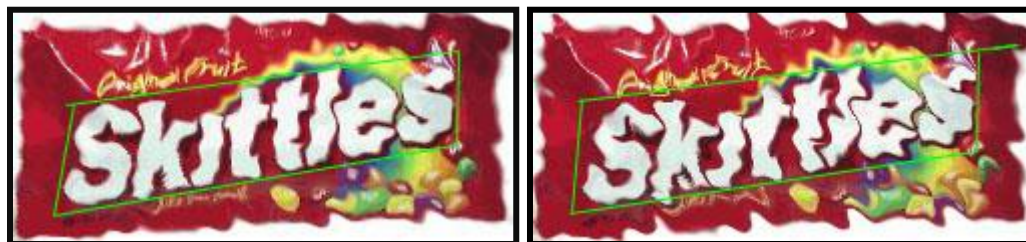


Figure 9: Inappropriate Control Images

User Interface

The usability team has come up with a draft version of what the next User Interface should look like:

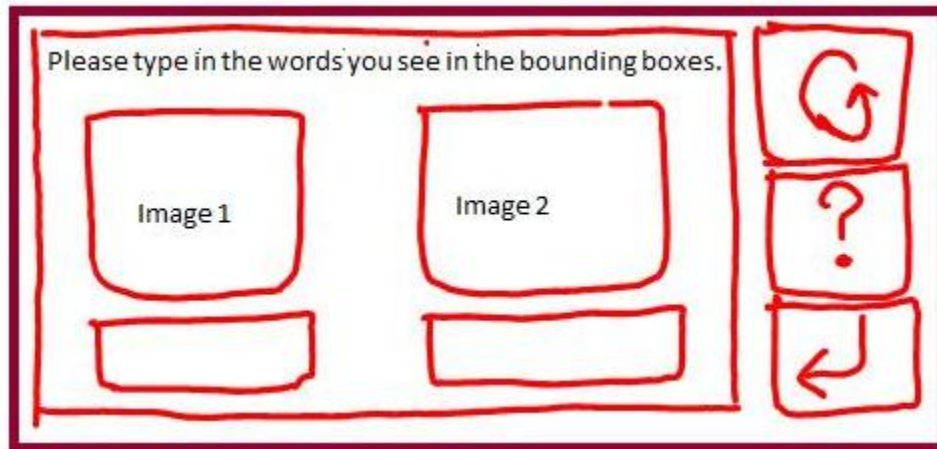


Figure 10: Draft version of Future UI

Traffic Generation

While the most important aspect of usability to focus on is the actual experience of the user as he/she encounters Soylent Grid, one can't help but question "Where exactly are users going to run into Soylent Grid?" With that in mind, the Usability team set out to find potential traffic that could be generated. Listed below are some of the mediums and locations that could heavily speed up the progress of the project.

- I. ACS – With the amount of students at UCSD checking their email every day, a simple task of labeling two product images shouldn't be too much of a hassle. However, because the rate at which students log in (that is to say, one student most likely logs in at least 5 times a day), requesting the student to go through Soylent Grid for every login is unreasonable. Therefore, while ACS is a very promising traffic location, its harms outweigh its benefits.
- II. WebCT – WebCT is a product owned by BlackBoard, with licenses purchased by UCSD, that serves as an online blackboard for UCSD courses. Essentially, it is a way to achieve electronic learning in a user-friendly interface. While it doesn't generate as much traffic as ACS email services, its statistics are favorable. Currently at UCSD, there are 60 categories of classes with each containing on average 5 courses per category. Each class accounts for about 100 students. In other words, there are about 300 courses being visited daily by about 3000 students. With its encouraging volume, and low rate of login (about one login per day), employing Soylent Grid with WebCT as a host could possibly create large traffic with little burden to UCSD students. Currently, the usability team is working to find out licensing costs and the requirements to qualify WebCT to consider Soylent Grid as a partner. While it is narrowed to just UCSD at this point, we hope in time that WebCT will employ Soylent Grid among other universities as well.
- III. GradeSource – Written by UCSD professor Gary Gillespie, it is an Instructor course management tool with student viewable web page reports. The software is currently free for UCSD professors and is widely used by professors in Computer Science and Engineering department. Students will be given secret numbers so they can check their grades any time during the quarter without

revealing their grades to other students. Although the traffic it generates can be significant, students are not requested to log in when they check their grades, thus it will not generate as much traffic as software such as WebCT.

- IV. Public computers at UCSD - The numerous computers at Geisel Library and Price Center provide access to internet for UCSD students. What sets this medium from the rest is that students only use the computers for a short amount of time. In fact since the duration of usage is so short and quick, many students line up at the library for a quick-and-go computer use. By deploying Soylent Grid on these computers, it serves as a quick and reliable source of labeling.
- V. Tritonlink—the student profile website that houses information to nearly every aspect of UCSD. With 24,000 students enrolled per year and at least one login per day by every student, tritonlink proves itself very well worthy of traffic generation. However, as ideal as it would be to employ Soylent Grid in a high volume website as Tritonlink, it is probably best to focus on smaller locations and secure a position in these positions. By introducing Soylent Grid and slowly achieving recognition for its uses across the web, then perhaps the user interface could be launched into Tritonlink without giving UCSD students a surprise to what Soylent Grid is.

Confidence Level

After deliberating over confidence level models we should employ to move experiment images to control images, the Usability team decided on three concurrent strategies:

- If experiment image is recorded with seven consecutive labels, move experiment image to control.
- If experiment image is recorded with three consecutive labels, but fails to achieve seven consecutive labels, perform a check that 65% of the labels are identical with a minimum 20 count vote. If 65% are identical, move to control; else, continue with labels until 65% confidence is achieved.
- If neither of the above criteria is fulfilled, run a minimum 50 votes. If 80% of labels are identical, pass the word; else, continue the check.

Our decision is based on the Recaptcha confidence level.

Help File

A help file is essential in ensuring that users know exactly what they are meant to do, and also, how they should accomplish it. Therefore, the Usability team has come up with a specific help file that acts as a pop up when users click on the “Need Help” icon. A draft version of the file looks like this:

Instructions

Two product images are displayed, each with a corresponding text box. Within each product image, word(s) are contained in a colored box.

Please enter these word(s) into the text box below that image. Doing so helps prevent automated spam bots from abusing this service.

If you are not sure what the words are, click on the image for a larger view. If you are still unsure, either enter your best guess or click the refresh button on the right side of the interface for a new pair of product images.

Still need help? Contact us. ([Link to troubleshooting survey](#))

Soylent Grid Vision

By entering the words into the boxes, you are also helping to label grocery product images. Blind people will use these labeled images to create a shopping list, which will be used along with GroZi's Grocery Shopping Assistant for the Blind. This project will allow blind people to shop for groceries independently. With your help, this goal can be accomplished.

[Learn more about GroZi?](#) ([Link directing user to GroZi website](#))

[Learn more about Soylent Grid?](#) ([Link directing user to Soylent Grid website](#))

Possible survey questions for the troubleshooting survey includes: (Modeled after reCAPTCHA)

- There are no words bounded by colored boxes.
- There is no colored box.
- The product image is too small or distorted.
- It says I'm wrong every time I enter the words, but I'm sure I got them right.
- The words are too hard to read.
- Something else (please describe below).

 Text box

- Other Comments?

 Text box

Summary

Graphically sophisticated designs of the Soylent Grid underpinned by leading-edge technologies are but digital art, if the users cannot navigate or use the interface. The Usability team functions to ensure that the Soylent Grid interface is efficient, effective and satisfying. In this report, we have outlined parameters that need to be considered for the deployment of Soylent Grid. These include problems and solutions to the Fall 08 and Winter 09 User Interface, traffic generation options, confidence level for the database and the Soylent Grid User Help File.

References

1. Valid Web Designs. 17 March 2009.
<<http://validwebdesigns.com/glossary/#d>>.
2. Recaptcha Website:
<http://www.recaptcha.com>
3. Fall 08 GroZi Final Report
<http://grozi.calit2.net/files/TIESGroZiFa08.pdf>