Grocery Shopping Assistant for the Blind (GroZi)

UCSD TIES—Spring 2010

Faculty Advisor:
Serge Belongie

Tutor:
Yon Xiao

Community Client:
National Federation of the Blind (NFB)

Client Representative:
John Miller—NFB Representative

Visiting Scholar:
Masaaki Kokawa

Team Members:
Lance Castillo, Cankut Guven, Jeffrey Wurzbach, Saitejswi (Teja) Kondapalli,
Nil Kumar, Amir Shirkhani, Jeffrey Su, Mehran Takapoomanesh Baghaei
Shauna Thomas, Aritrick Chatterjee, Bonnie Han

Table of Contents

INTRODUCTION

There are currently 1.3 million legally blind people living in the United States who face daily obstacles with routine tasks. These individuals cannot shop independently for grocery store items without sighted assistance.

Developing assistive technologies and handheld devices allows for the possibility of increasing independence for the 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 computer vision can be advantageous in helping these blind customers, as restrictions such as the limited ability of guide dogs of white canes, 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 can 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 grocery assistant device 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 pursue the computer vision aspects of the project that allows for autonomous detection and localization in the near future. In the past quarter, 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. While this improvement awaits refinement, the focus this quarter has been shifted to the integrating CVSG to develop the software that processes images of a grocery shelf model. Additionally, a special request by John Miller has the team involved in challenging themselves to build a recording program that will facilitate communication for the blind in industrial work involving visual graphics. The following document will serve as a description of what has been accomplished thus far, what has been learned and overcome, and the processes involved in designing and implementing a usable grocery assistant device for the blind to assist future members of TIES GroZi team.

Sub Teams, this quarter…

This quarter the GroZi team was divided into Sub teams:

Hardware Team: Jeffrey W and Shauna
Demo Board Design Team: Jeffrey S. and Attrick
Protocol Team: Nil and Lance
Gameboard Team: Teja
Computer Vision Sighted Guide (CVSG) team: Cankut, , Teja
Mouse Click Recorder (MCR) : Amir, Mehran, Bonnie

Webmaster: Lance Castillo

Official TIES GroZi website: http://ties.ucsd.edu/projects/gsa/index.shtml
Official GroZi website: http://grozi.calit2.net/

Introduction

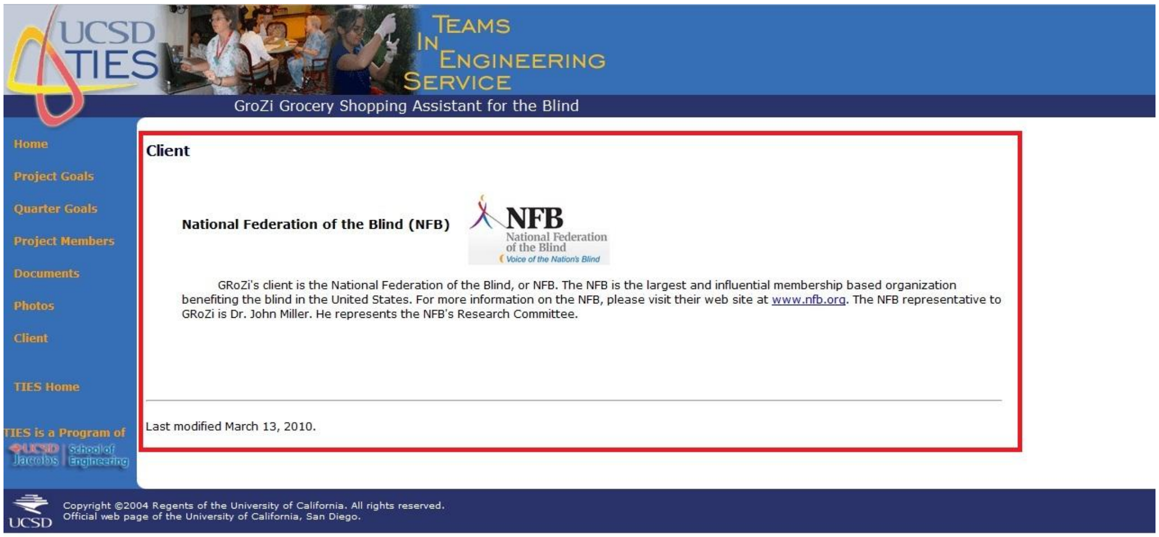
This section is written for future webmasters to gain an understanding of the setup of existing files. The task of a webmaster is to update the TIES GroZi website with the current progress of the project.

Most of the content on this website are done in SHTML. The only difference between regular HTML and SHTML is the extra letter in the extension (.shtml) and this code.

```
<!--#include file="addedfile.txt" -->
```

The S stands for 'Server Side Include' or SSI. When an SHTML webpage is sent to the web browser window, it gets assembled on the server and then sent to be viewed. The normal HTML tags all still work the same, the SHTML simply lets you INCLUDE other pieces into the HTML page.

Below is an example:


UCSD TIES
TEAMS IN ENGINEERING SERVICE
GroZi Grocery Shopping Assistant for the Blind
Home
Project Goals
Quarter Goals
Project Members
Documents
Photos
Client
TIES Home
TIES is a Program of
UCSD Jacobs School of Engineering
NFB National Federation of the Blind Voice of the Nation's Blind
UCSD

Client

National Federation of the Blind (NFB)

GRoZi's client is the National Federation of the Blind, or NFB. The NFB is the largest and influential membership based organization benefiting the blind in the United States. For more information on the NFB, please visit their web site at www.nfb.org. The NFB representative to GRoZi is Dr. John Miller. He represents the NFB's Research Committee.

Last modified March 13, 2010.

Copyright ©2004 Regents of the University of California. All rights reserved.
Official web page of the University of California, San Diego.

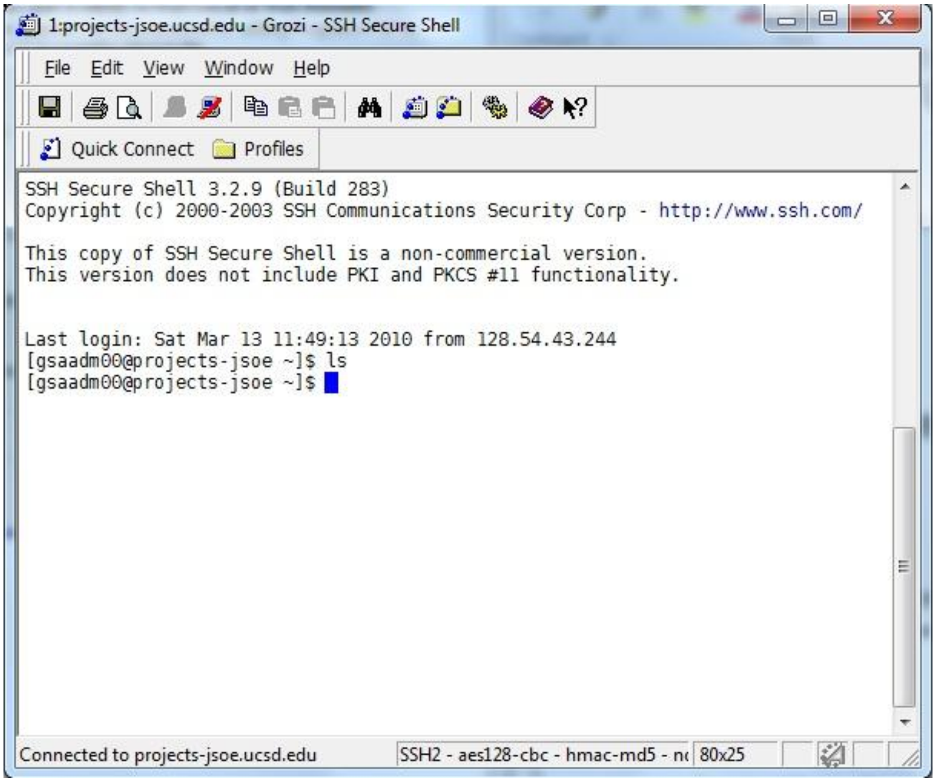
This is a screenshot of the Client page. The red box contains all the content the webmaster can edit in the "client.shtml" file.

Log-in

Username and password will be provided to the webmaster from TIES when the webmaster is designated.

Once you have received the username and password:
Host name: <server name>
Port: 22 (default)
Login: <username>
Password: <password>

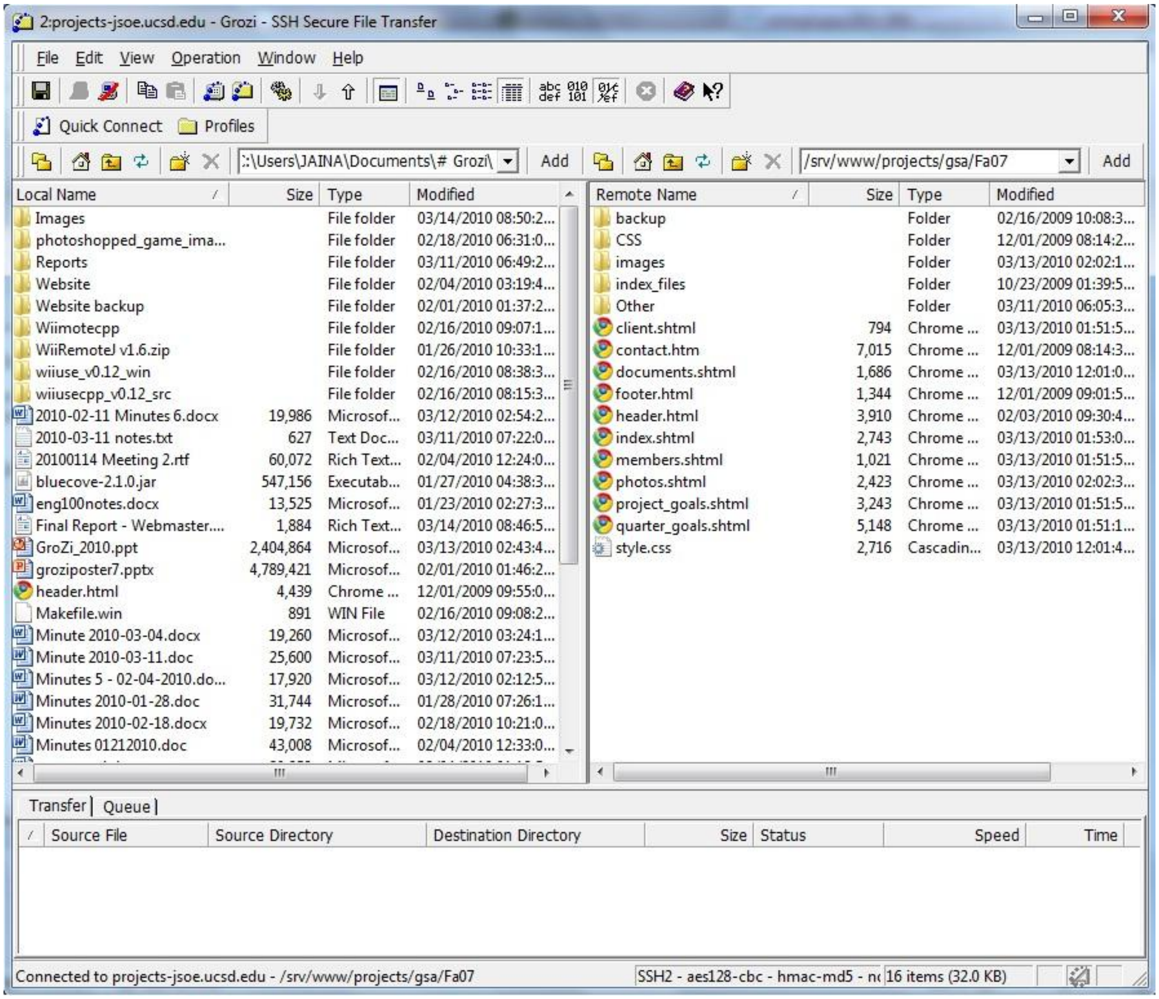
To upload files to the server in SSH, first log in:



```
SSH Secure Shell 3.2.9 (Build 283)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.


Last login: Sat Mar 13 11:49:13 2010 from 128.54.43.244
[gsaadm00@projects-jsoe ~]$ ls
[gsaadm00@projects-jsoe ~]$
```

Connected to projects-jsoe.ucsd.edu | SSH2 - aes128-cbc - hmac-md5 - n | 80x25

Open File Transfer Client:



2:projects-jsoe.ucsd.edu - Grozi - SSH Secure File Transfer

:\Users\JAINA\Documents\# Grozi\

Local Name	Size	Type	Modified
Images		File folder	03/14/2010 08:50:2...
photoshopped_game_ima...		File folder	02/18/2010 06:31:0...
Reports		File folder	03/11/2010 06:49:2...
Website		File folder	02/04/2010 03:19:4...
Website backup		File folder	02/01/2010 01:37:2...
Wiimotecpp		File folder	02/16/2010 09:07:1...
WiiRemoteJ v1.6.zip		File folder	01/26/2010 10:33:1...
wiiuse_v0.12_win		File folder	02/16/2010 08:38:3...
wiiusecpp_v0.12_src		File folder	02/16/2010 08:15:3...
2010-02-11 Minutes 6.docx	19,986	Microsof...	03/12/2010 02:54:2...
2010-03-11 notes.txt	627	Text Doc...	03/11/2010 07:22:0...
20100114 Meeting 2.rtf	60,072	Rich Text...	02/04/2010 12:24:0...
bluecove-2.1.0.jar	547,156	Executab...	01/27/2010 04:38:3...
engl100notes.docx	13,525	Microsof...	01/23/2010 02:27:3...
Final Report - Webmaster....	1,884	Rich Text...	03/14/2010 08:46:5...
GroZi_2010.ppt	2,404,864	Microsof...	03/13/2010 02:43:4...
groziposter7.pptx	4,789,421	Microsof...	02/01/2010 01:46:2...
header.html	4,439	Chrome ...	12/01/2009 09:55:0...
Makefile.win	891	WIN File	02/16/2010 09:08:2...
Minute 2010-03-04.docx	19,260	Microsof...	03/12/2010 03:24:1...
Minute 2010-03-11.doc	25,600	Microsof...	03/11/2010 07:23:5...
Minutes 5 - 02-04-2010.do...	17,920	Microsof...	03/12/2010 02:12:5...
Minutes 2010-01-28.doc	31,744	Microsof...	01/28/2010 07:26:1...
Minutes 2010-02-18.docx	19,732	Microsof...	02/18/2010 10:21:0...
Minutes 01212010.doc	43,008	Microsof...	02/04/2010 12:33:0...

/srv/www/projects/gsa/Fa07

Remote Name	Size	Type	Modified
backup		Folder	02/16/2009 10:08:3...
CSS		Folder	12/01/2009 08:14:2...
images		Folder	03/13/2010 02:02:1...
index_files		Folder	10/23/2009 01:39:5...
Other		Folder	03/11/2010 06:05:3...
client.shtml	794	Chrome ...	03/13/2010 01:51:5...
contact.htm	7,015	Chrome ...	12/01/2009 08:14:3...
documents.shtml	1,686	Chrome ...	03/13/2010 12:01:0...
footer.html	1,344	Chrome ...	12/01/2009 09:01:5...
header.html	3,910	Chrome ...	02/03/2010 09:30:4...
index.shtml	2,743	Chrome ...	03/13/2010 01:53:0...
members.shtml	1,021	Chrome ...	03/13/2010 01:51:5...
photos.shtml	2,423	Chrome ...	03/13/2010 02:02:3...
project_goals.shtml	3,243	Chrome ...	03/13/2010 01:51:5...
quarter_goals.shtml	5,148	Chrome ...	03/13/2010 01:51:1...
style.css	2,716	Cascadin...	03/13/2010 12:01:4...

Source File	Source Directory	Destination Directory	Size	Status	Speed	Time

Connected to projects-jsoe.ucsd.edu - /srv/www/projects/gsa/Fa07 | SSH2 - aes128-cbc - hmac-md5 - n(| 16 items (32.0 KB)

Drag files from your machine on the left to the directories on the server on the right to upload.

Files currently in use on the website

- index.shtml
- project_goals.shtml
- quarter_goals.shtml
- members.shtml
- documents.shtml
- photos.shtml
- client.shtml
- header.html
- footer.html
- contact.htm
- style.css
- /images – images used in “index.shtml” and “photos.shtml”
  - grozi.jpg
  - grozi_board.jpg

    - grozi_grocery.jpg
    - grozi_redrobot.jpg
    - grozi_wiimote.jpg
    - NFBlogo.jpg
- /Other – files listed in “documents.shtml”
    - grozi_slam.pdf
    - grozi_techbrief.pdf
    - TIESGroZiFa08.pdf
    - TIESGroZiFa09.pdf
    - TIESGroZiSp07.pdf
    - TIESGroZiSp09.pdf
    - TIESGroZiSu09.pdf
    - TIESGroZiWi07.pdf
    - TIESGroZiWi08.pdf
    - TIESGroZiWi09.pdf

Demo Board Design Team: Jeffrey Su & Aritrick Chatterjee

Introduction

The Demo Board design team's goals are to create demo boards that allow performance evaluation of various algorithms like those used in the Remote Sighted Guide. The demo boards need to mimic grocery store shelves and still be easily transportable. Some of the key points these demos will help us solve is variability problems likes those induced by shading/poor lighting conditions, occlusion of the image, and perspective issues. This quarter's demo boards needed to be rebuilt, and is in the process of being rebuilt. While we were building the board, we decided to redesign the board so that it is easier to use, more portable, and easy to build.

New Demo Board

Supplies used to make the new board:

Item	Quantity
White Pegboard	1
Black Felt fabric (3 yards)	1
Hinges	1
Moulding	1
2-3/4 Golf (Tees white) (200 box)	1
Fusion Spray paint Red	1
Fusion Spray paint Blue	1
Fusion spray paint Yellow	1
Fusion spray paint Green	1
Krazy Glue	1
Glue Pen (Gorilla)	1

Procedure

Procedure used to create the new demo board

1) Go to Home Depot and buy the following items:
   White peg board 27x27 inches
   Hinges
   Krazy Glue
2) Go to Wal-Mart or any thrift store and buy the following:
   Spray Paint of colors: Red, Blue, Yellow, and Green
   2 yards of black felt cloth
   3 packs of white or wood color gold tees
3) Cut out 4 strips of cloth each 1X 27 inches

4) Use the Krazy glue to glue the strips of cloth to the edges of the board.
5) Take the hinges and nail them to the back of the board (make sure they are not covering the peg holes)
6) Nail 2 long pieces of wood (about 20 inches) to the hinges (again make sure that the pieces of wood isn’t covering the hole). This is will be the moving platform
7) Take out the golf tees and line them up on the floor.
8) Take the spray paint and spray 225 of them green, 4 blue, 4 red, 8 yellow.
9) Open up 4 holes at each corner of the board through the cloth.

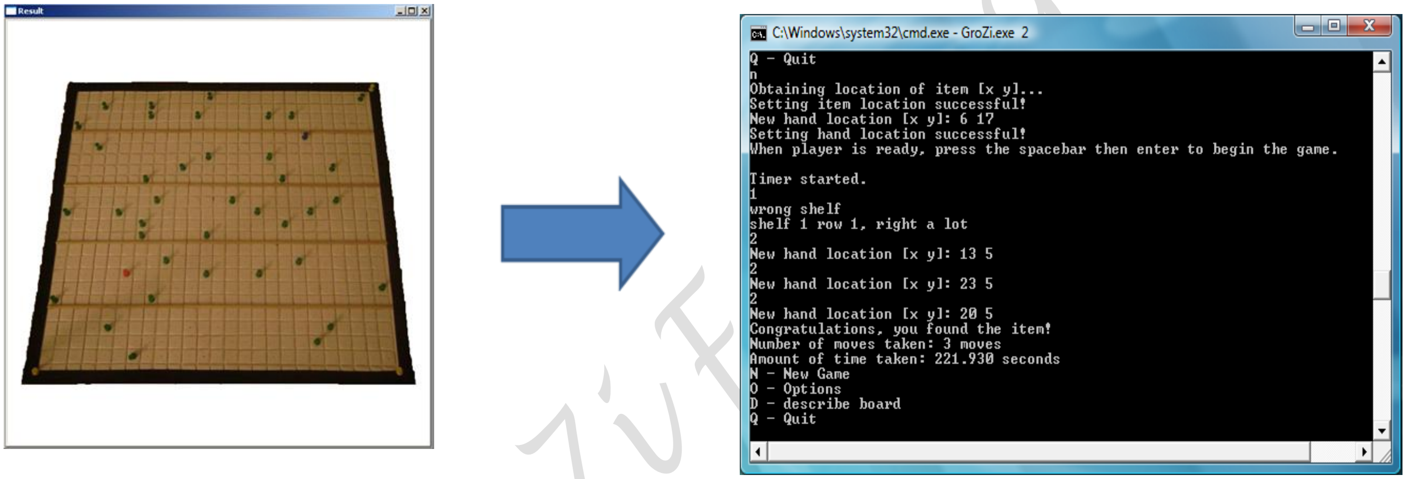
holes and can represent the pins that we use now. Issues that we are having are coming from the weight and portability of the board. One possible solution is being able to fold up the board, as well as being able to keep the pieces on.

Protocol Team: Lance Castillo & Nil Kumar

Introduction

What is the Protocol Team?

The Protocol Team is working on the GroZi Grocery Shopper Game application which is new software that is under development to work in conjunction with the images the Computer Vision Sighted Guide (CVSG) team is able to process. The purpose of this program is to be able to take the image data (i.e. hand token location, item token location) provided in a file from CVSG and process it in such a way that it will provide directions from the current hand location to the target item location. Ideally, once the implementation of this software is complete, it should help analyze how to efficiently guide a blind person to a grocery product.



```
Q - Quit
n
Obtaining location of item [x y]...
Setting item location successful!
New hand location [x y]: 6 17
Setting hand location successful!
When player is ready, press the spacebar then enter to begin the game.

Timer started.
1
wrong shelf
shelf 1 row 1, right a lot
2
New hand location [x y]: 13 5
2
New hand location [x y]: 23 5
2
New hand location [x y]: 20 5
Congratulations, you found the item!
Number of moves taken: 3 moves
Amount of time taken: 221.930 seconds
N - New Game
O - Options
D - describe board
Q - Quit
```

Requirements

1. The protocol shall not speak extra information such as long menus, a counting timer, etc.
2. The software shall support Windows 7 x64.
3. The software shall support Windows XP Professional x86.

Protocol Details

This quarter there was new output from the CVSG team that had to be parsed in order for it to work with the GroZi Grocery Shopper Game application. The team was able to successfully parse the new output from CVSG and run a full game from start to finish.

Protocol Goals

Future Considerations and Recommendations

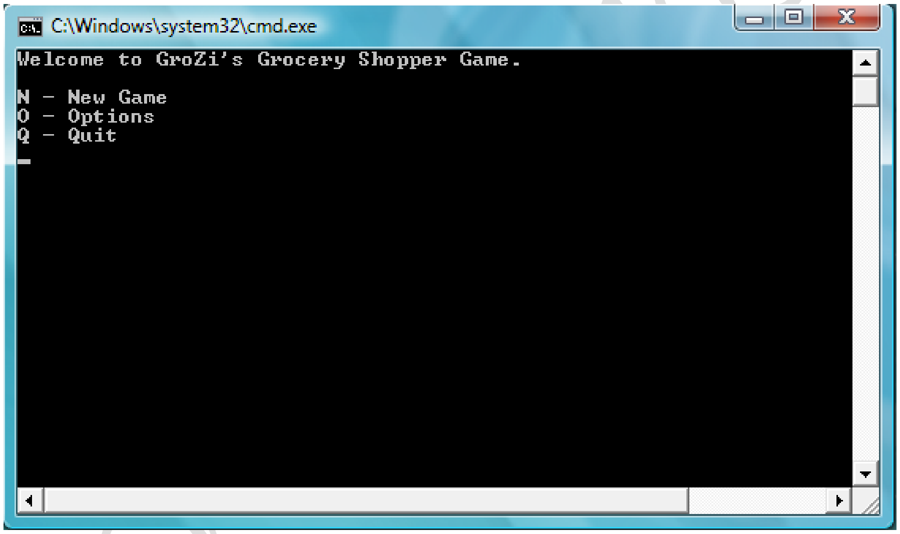
1. Integrate JAWS functionality into the software so that visually impaired users can complete experiments on their own.

What is the GroZi Grocery Shopper Game Program?

The GroZi Grocery Shopper Game application is new software that is under development to work in conjunction with the images the Computer Vision Sighted Guide (CVSG) team is able to process. The purpose of this program is to be able to take the image data (i.e. hand token location, item token location) provided in a file from CVSG and process it in such a way that it will provide directions from the current hand location to the target item location. Ideally, once the implementation of this software is complete, it should help analyze how to efficiently guide a blind person to a grocery product.

General Overview of Usage

The GroZi Grocery Shopper Game program is a Win32 Application written in C programming language. Currently, the code (located in files Game_Code.c and Game_Code.h) can be run using Visual Studio 2008 or in a Linux environment by compiling using the gcc command.
Upon starting the program, the user will be prompted with the main menu (see figure 3.1a) in which they can select to play a "new game", view "options", or "quit".

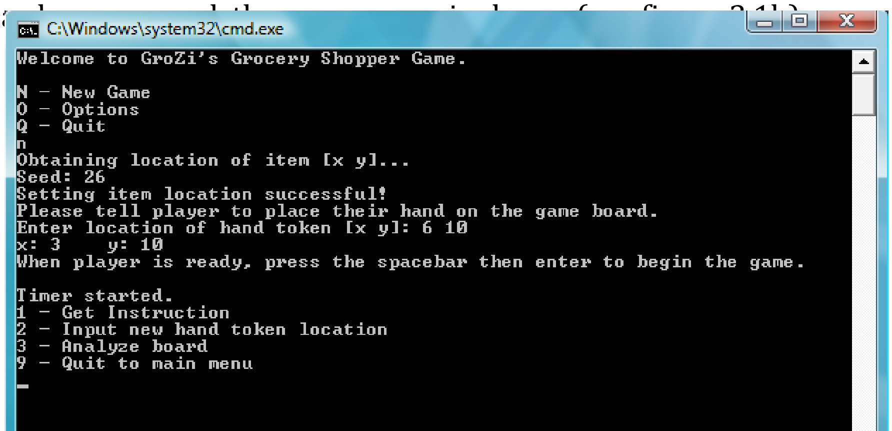


```
Welcome to GroZi's Grocery Shopper Game.

N - New Game
O - Options
Q - Quit
_
```

***Figure 3.1a :** GroZi Grocery Shopper Game main menu options*

If the user chooses to start a new game, (by entering the letter N), the software will automatically generate a random (x, y) coordinate from a supplied file which will be used for the target location. Next, the program will prompt the user to enter their initial hand location and store this into an array. The program then lets the user know to press spacebar when they would like the timer to begin. As soon as spacebar ha

with the following options:



```
Welcome to GroZi's Grocery Shopper Game.

N - New Game
O - Options
Q - Quit
n
Obtaining location of item [x y]...
Seed: 26
Setting item location successful!
Please tell player to place their hand on the game board.
Enter location of hand token [x y]: 6 10
x: 3     y: 10
When player is ready, press the spacebar then enter to begin the game.

Timer started.
1 - Get Instruction
2 - Input new hand token location
3 - Analyze board
9 - Quit to main menu
_
```

1 – Get Instruction

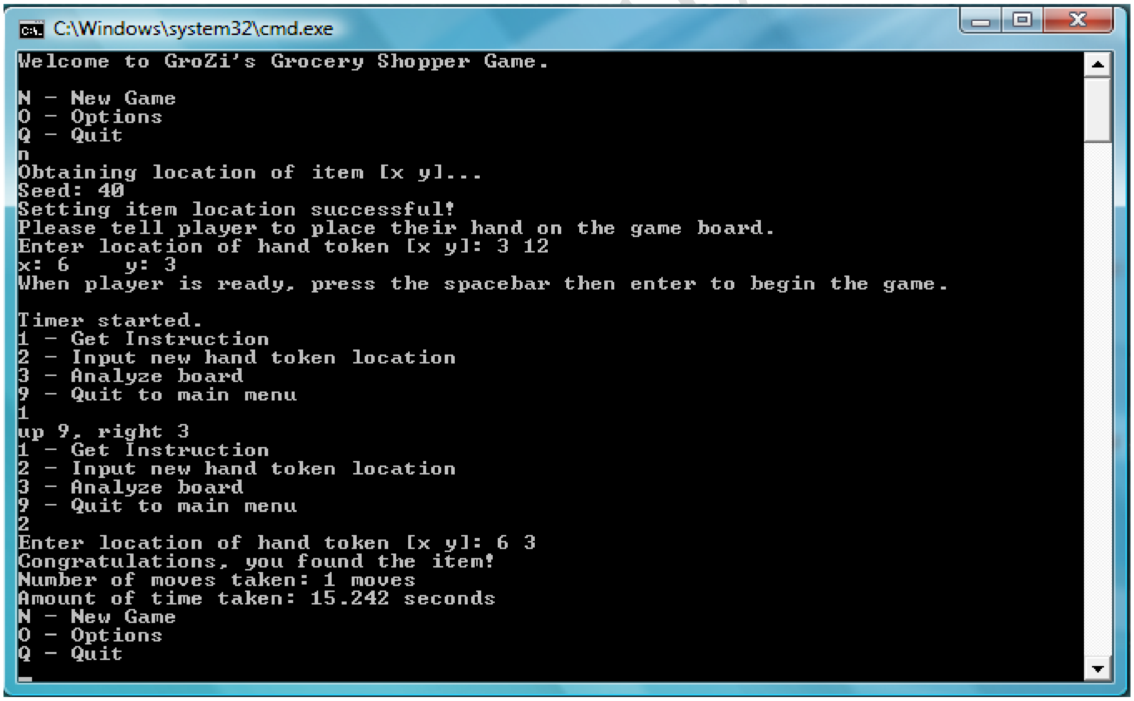
2 – Input new hand token location

3 – Analyze board

9 – Quit to main menu

Figure 3.1b: *In play menu options*

Get Instruction provides the user with the next move to get to the target location by providing an up or down and right or left instruction. The user then is able to input a new hand token location at which point the program will either congratulate you because you found the item, or wait for you to ask for a new instruction. This menu keeps looping until the item is found. At this point the final time and number of moves it took to find the item will be displayed (Figure 3.1c).



```
Welcome to GroZi's Grocery Shopper Game.

N - New Game
O - Options
Q - Quit
n
Obtaining location of item [x y]...
Seed: 40
Setting item location successful!
Please tell player to place their hand on the game board.
Enter location of hand token [x y]: 3 12
x: 6     y: 3
When player is ready, press the spacebar then enter to begin the game.

Timer started.
1 - Get Instruction
2 - Input new hand token location
3 - Analyze board
9 - Quit to main menu
1
up 9, right 3
1 - Get Instruction
2 - Input new hand token location
3 - Analyze board
9 - Quit to main menu
2
Enter location of hand token [x y]: 6 3
Congratulations, you found the item!
Number of moves taken: 1 moves
Amount of time taken: 15.242 seconds
N - New Game
O - Options
Q - Quit
_
```

Figure 3.1c: *Output from a full game, beginning to end with user finding target item*

Changes to Protocol Software:

This quarter's goal for protocol software was to be able to parse the CVSG output and use it as input. CVSG provides a plain text file with coordinates of various pins such as the hand token and desired item token. From these coordinates, the protocol software is able to parse out the target item location and set that as the initial position for target product in the protocol software. In the previous version of the protocol software, the target item location was chosen from a generated Matlab .m file of random coordinates.

This process is completely skipped now. Additionally, instead of asking the user for their current hand location, this is now automatically fed in from the CVSG output. As a result of this automated process, a full game is now able to be played without any user input except for menu options.

New Config File:

Due to the Board Team constructing a new demo board, the protocol software needed a new config file. The new dimensions used in the config file are 25x25 with 5 rows per shelf.

Computer Vision Sighted Guide (CVSG) Team: Cankut Guven, Kevin Tran, Saitejaswi Kondapalli

Introduction

The CVSG (Computer Vision Sighted Guide) team's task is to create a program to recognize various pins located on the board created by the Board Team. This information will be used to eventually lead the user to the desired item. The blind user's hand will not be present physically, but instead represented by a red pin. The idealized environment will include a “desired” token (blue pin), on a demo board, and a series of "noise" tokens (green pins). The software will determine the location of the blue and red pin and report it to the Game Square software.

This quarter, the CVSG team was able to accomplish the following tasks: bump up pin detection to 100% on the whole board; begin analysis of the new board; take images that can be used to calibrate the camera for all situations. Note that the analysis of the new board is just begun and the old software is not functional as specifications have changed (such as the edges: there are no longer perfect straight lines thus the hough transform is rendered useless). Code, documents, and images can be found at http://code.google.com/p/grozi-cvsg

Requirements for the CVSG Software

- Recognize pin locations with 99% accuracy.
- The angle between the camera face and the Demo Board shall not be more than ±5 degrees from parallel.
- The center of the image must be within the white (inner) portion of the board.
- The image must be at least 800x600 pixels.
- The input to the protocol software must be within the dimensions of the demo board.
- The software will support Windows 7 x64
- The software will support Windows XP Professional x86
- The software will make use of OpenCV
- No part of the image will be modified by the use of any photo editing tools or any other programs
- The protocol software shall work with an off-the-shelf screen reader software program for the blind such as JAWS 10.0 or higher from Freedom Scientific.

- The protocol software shall generate text output that a screen reader can turn in to the guiding instructions. No additional screen output will be present in this mode.

General Overview and Usage

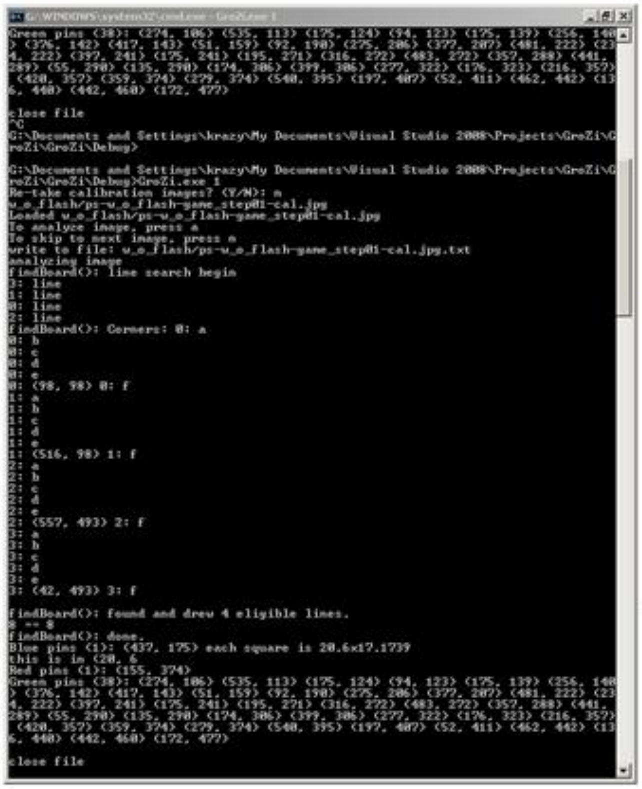
The software can be run in one of two modes: protocol or CVSG. The program must be provided with an argument of 1 or 2 in order to initialize.

Upon executing with the argument of 1, the software will proceed with the CVSG software. It will prompt the user whether or not calibration images should be retaken (this functionality is being considered for removal in future releases). This was implemented in a prior quarter when the webcam was in use, thus is a feature that is currently disregarded and thus a reply of [N]o should be given. The program will then prompt the user to make a selection of whether the software should analyze the image, and thus provide the pixel locations of the pins and the corners of the board, or to move on to the next image. In order to detect the board and the pin locations, some initial assumptions were made. Detection)of the old board) is successful under the following conditions:

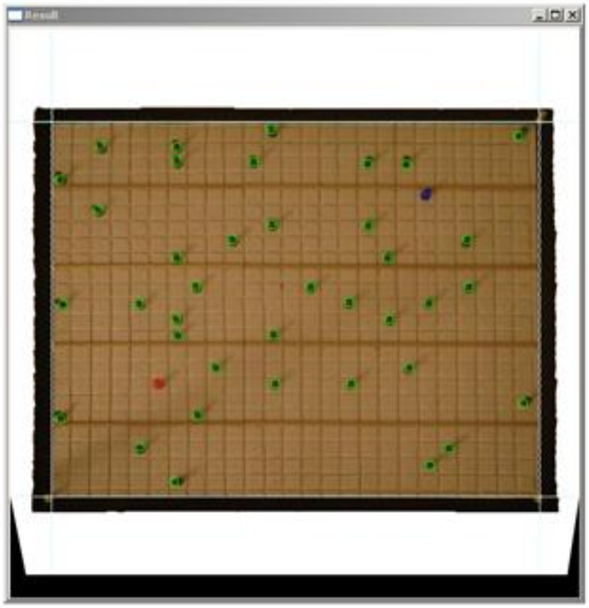
- Lighting such that glare from the tape is not too bright
- The middle of the image/photograph is on the game board (i.e. if the width x length of the board is 500x500, the point 250x250 is on the game board)
- The black border is visible in its entirety (no obstructions, such as arms, can be in the way)
- The image is photoshopped to remove any noise outside of the board (such as carpet)
- The images used are calibrated with the camera

With these restrictions in play, the CVSG software was able to detect the location of all the pins and the corners as shown in Figure 1.1a

Figure 1.1a: *On the left, the console output of the program is shown. On the right, the image of the board with the green, red, and blue pins circled with their respective colors is shown, in addition to the detection of the edges of the border*



```
Green pins (38): (274, 106) (535, 113) (175, 124) (94, 123) (175, 139) (256, 140
) (376, 142) (417, 143) (51, 159) (92, 190) (275, 206) (377, 207) (481, 222) (23
4, 222) (397, 241) (175, 241) (195, 271) (316, 272) (483, 272) (357, 288) (441,
289) (55, 290) (135, 290) (174, 306) (399, 306) (277, 322) (176, 323) (216, 357)
 (420, 357) (359, 374) (279, 374) (540, 395) (197, 407) (52, 411) (462, 442) (13
6, 440) (442, 460) (172, 477)

close file
^C
G:\Documents and Settings\krazy\My Documents\Visual Studio 2008\Projects\GroZi\G
roZi\GroZi\Debug>

G:\Documents and Settings\krazy\My Documents\Visual Studio 2008\Projects\GroZi\G
roZi\GroZi\Debug>GroZi.exe 1
Re-take calibration images? (Y/N): n
w_o_flash/ps-w_o_flash-game_step01-cal.jpg
Loaded w_o_flash/ps-w_o_flash-game_step01-cal.jpg
To analyze image, press a
To skip to next image, press n
write to file: w_o_flash/ps-w_o_flash-game_step01-cal.jpg.txt
analyzing image
findBoard(): line search begin
3: line
1: line
0: line
2: line
findBoard(): Corners: 0: a
0: b
0: c
0: d
0: e
0: (98, 98) 0: f
1: a
1: b
1: c
1: d
1: e
1: (516, 98) 1: f
2: a
2: b
2: c
2: d
2: e
2: (557, 493) 2: f
3: a
3: b
3: c
3: d
3: e
3: (42, 493) 3: f

findBoard(): found and drew 4 eligible lines.
8 == 8
findBoard(): done.
Blue pins (1): (437, 175) each square is 20.6x17.1739
this is in (20, 6
Red pins (1): (155, 374)
Green pins (38): (274, 106) (535, 113) (175, 124) (94, 123) (175, 139) (256, 140
) (376, 142) (417, 143) (51, 159) (92, 190) (275, 206) (377, 207) (481, 222) (23
4, 222) (397, 241) (175, 241) (195, 271) (316, 272) (483, 272) (357, 288) (441,
289) (55, 290) (135, 290) (174, 306) (399, 306) (277, 322) (176, 323) (216, 357)
 (420, 357) (359, 374) (279, 374) (540, 395) (197, 407) (52, 411) (462, 442) (13
6, 440) (442, 460) (172, 477)

close file
```



Camera Calibration

In order to calibrate the camera, one can follow these steps:

1. Print out the chessboard (chessboard.pdf) and tape it to a flat surface (e.g. cardboard, wooden panel) if not already done so.
2. Place Game Board on a flat surface and set up a digital camera on a tripod such that it is aimed towards the Game Board.
3. Take pictures of the Game Board.
4. In the same setting (without altering the height or location of the tripod), take several pictures of the chessboard (in experiments that were run in Winter 2010, six images were taken; this number does not have any scientific significance).
5. Run Board::calibrate with the proper settings (alter std::stringstream imagename such that it points to the correct files) and use undistortImage(image) on a non-calibrated board image to correct calibration errors. Then, have the program save the images.
6. Images should now be calibrated.

Several tests on calibrated and non-calibrated images were run and a higher success rate was discovered in running the algorithm on calibrated images. One of the algorithms in use is the Hough Transform, which searches for straight lines. In non-calibrated images, what should be straight lines become distorted due to the lens of the camera, thus masking a straight line as a curve.

Next Steps

While detection of the board and the pin locations is a positive step forward, there is a lot that remains to be accomplished. Here are the goals that would be ideal in helping the project move forward:

- Develop algorithms to detect the new board under conditions with flash (and no flash)
- Play a full game with the use of text-file outputs to other teams
- Integrate CVSG software with Game Square software to allow Game Square to identify the row and column of a pin

APPENDIX

Language: C++

IDE: Microsoft Visual Studio 2008

Operating System: Windows XP x86

Digital Camera: Sony Cybershot DSC-S650

GameSquare Team: Saitejaswi Kondapalli

Introduction:

What is the GameSquare Team?

The GroZi Grocery Shopper Game application is new software that is under development that incorporates the work three teams: CVSG, GameSquare and Protocol. The purpose of the game square program is to be able to take the image data (i.e. hand token location, item token location) provided in a file from CVSG and convert the pixel coordinates of each pin to respective coordinates on the game board in order for the Protocol team to do its calculations. This step is essential for the CVSG and the Protocol software to communicate.

Overview of Requirements

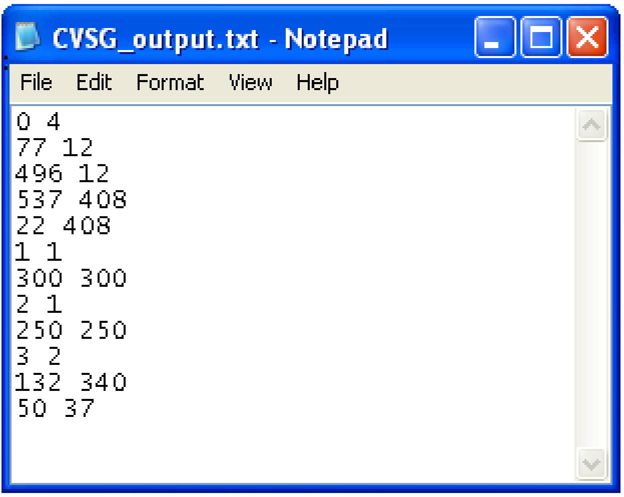
The input file into the GameSquare program will be the output file of the CVSG program. The GameSquare code will process its input and produce an output file that will be the input file to the Protocol program. These files will be plain text files and will follow strict formatting guidelines.

CVSG_output.txt

The file that is the output of CVSG will be called *CVSG_output.txt,* which will be a plain text file that should be located in the same directory as the files containing the code. In this file, the first line in each block will have two integers separated by white space, the first is an integer represents a color and the second is a integer represents the (n) number of pins that are of the color. The next n lines will each contain two integers separated by white space. The first integer is the x-coordinate pixel location of the pin and the second integer is the y-coordinate pixel location of the pin. There will be at most 4 such blocks of data, each blocks gives information about the location of a set of pins of each color.

The numbering for each color is as follows:

0 - yellow (corner)
1 - blue (item)
2 - red (hand)
3 – green (distractor)



```
0 4
77 12
496 12
537 408
22 408
1 1
300 300
2 1
250 250
3 2
132 340
50 37
```

Figure 1.a: Sample CVSG_output.txt

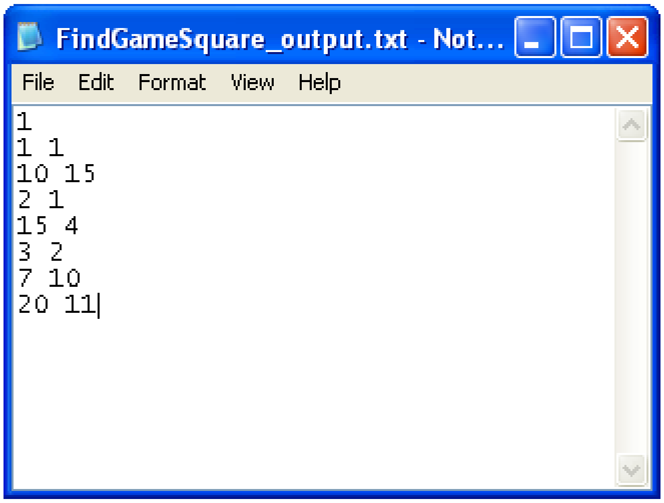
FindGameSquare Code Requirements

The GameSquare should do the following, in this order:

1. Parse through the CVSG_output.txt to get pixel locations for all pins
2. Convert each pixel location into a game board location (x, y) -> (i, j)
3. Determine the validity of the board, the follow rules must be observed:
   a. Pictures shall contain no more than 4 yellow pins
   b. Pictures shall contain exactly one red pin
   c. Pictures shall contain either zero or one blue pin
   d. Pictures shall have no two x,y coordinates for pins that map to the same i,j coordinates
4. If the gameboard is valid, write “1” as the first line of the outputfile, else write “0”
5. If the gameboard is valid, the gameboard locations (i,j coordinates) must be written to an output file called FindGameSquare_output.txt (this file is described below)

FindGameSquare_output.txt

The file that is the output of GameSquare will be called *FindGameSquare_output.txt,* which will be a plain text file that is located in the same directory as the rest of the files. In this file the first line will either contain a ‘0’ or ‘1’. A ‘1’ indicates that the gameboard is valid in the way it is set up. A ‘0’ indicates an invalid gameboard that the protocol will have to discard. This file also contains blocks of information. The first line in each block will have two integers separated by white space, the first is a integer represents a color and the second is a integer represents the (n) number of pins that are of the color. The next n lines will each contain two integers separated by white space. The first integer is the x-coordinate board square location of the pin and the second integer is the y-coordinate board square location of the pin. There will be at most 4 such blocks of data, each blocks gives information about the location of a set of pins of each color.


FindGameSquare_output.txt - Not...
File Edit Format View Help

```
1
1 1
10 15
2 1
15 4
3 2
7 10
20 11
```

Figure 1.b: Sample FindGameSquare_output.txt

The numbering for each color is as follows:

0 - yellow (corner)
1 - blue (item)
2 - red (hand)
3 – green (distractor)

FindGameSquare Code Details

This quarter the FindGameSquareCode was originally written in Java. Once the code had been tested and confirmed to be working it was translated to C++. Current the Java code executes steps 1, 2, 3 a-c, and 4 of the requirements listed above correctly. The C++ code executes steps 1 and 2 correctly.

FindGameSquare Code Goals

In following quarters the following goals must be accomplished:

1. Finish translating Java code into C++
    a. Checking validity of the board (ensuring all requirements)
    b. Writing i,j coordinates to output file
2. Implement step 3d of requirements
    a. This steps checks if there are two x,y coordinates that map to the same i,j coordinate if this is found then the board is invalid
3. Integrate GameSquare code with CVSG and Protocol

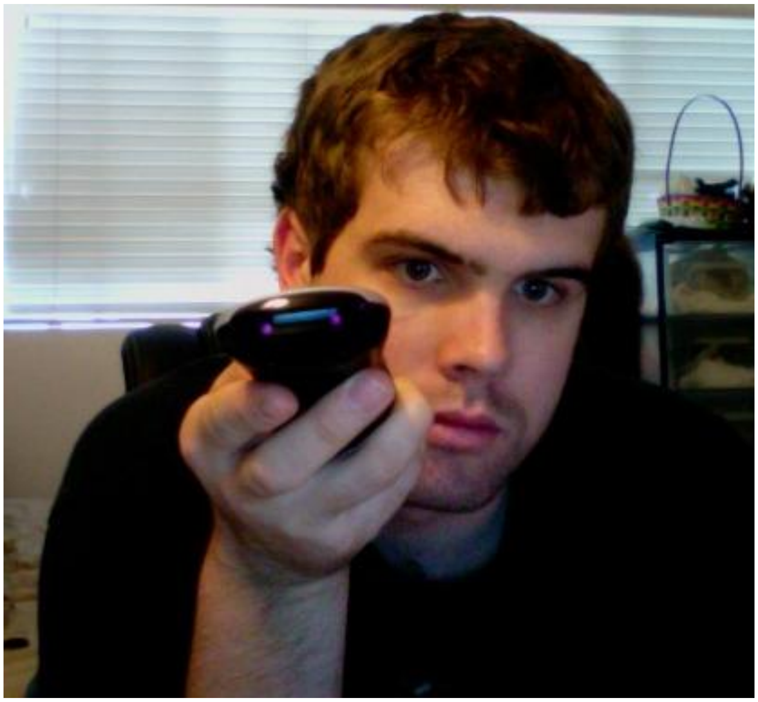
Hardware /Hand Tracker Team: Jeffrey W and Shauna

Introduction

The Hardware design team's duties are to create physical mock ups of what the finished GroZi device might look like, from an industrial design point of view. They also are tasked with the fabrication aspects of the project. They will be creating various devices to assist in the tracking of the blind user's hand as well as creating a prototype of the shoulder mounted camera system.

Objectives met this quarter…

The Hand Tracker team's goal is to create a hardware system that will read an image stream from a webcam, process them to determine the location of the user's hand and forward the image stream to the GroZi laptop. The hand tracking is achieved via a retro-reflective ring on the user's finger, a modulated IR light source and a camera. The camera and the IR light source are mounted in the same location, like the user's shoulder. The basic idea of operation is that the webcam can see near IR light, which is invisible to the majority of the human population. This is the same type mode of operation for many TV remotes. A picture with a TV remote transmitting IR light is shown in figure below.



The two purple dots in the middle of the picture are the two IR LEDs inside the remote control. To filter out the background of image, IR light source will be modulated so that it is on in every other frame and the frame where the IR light source is off, represents the background of the image. For the purposes of discussion, the IR light source is on in during even numbered frames (0,2,4,6…), and the IR light is off during odd numbered frames (1,3,5,7…). The proposed operation of the filter is to subtract and odd numbered frame from an even numbered frame, as shown in the following equation:

$$Frame[0] - Frame[1] = processedframe[0]$$

"Frame[0]" represents an even numbered frame and "Frame[1]" represents an odd numbered frame. "processedframe[0]" is the resulting frame that with minimal background noise. There are other methods to remove background noise; however, they are significantly more computationally intensive.

Once the background noise is minimized, then object detection begins. Using Canny edge detection, the edge of the retroreflector's return can be found. The center of that object is determined and reported to the GroZi laptop.

The Hand Tracker system is implemented on external hardware with respect to the GroZi laptop. The motivation for using external hardware is to reduce the workload on the GroZi laptop. The current hardware platform being pursued is the Beagle Board, which is based on a Texas Instruments OMAP3530 hybrid system on chip (SoC) processor. The OMAP3530 contains an ARM Cortex-A8 core and a DSP core. The Beagle Board was selected for two reasons; one of the team members already had one and its wide open-source support. The goal is to get the DSP core to handle most of the vision functions.

> Problems were encountered in getting Linux to acknowledge the DSP core and getting OpenCV setup on the Beagle Board. Presently, the Beagle Board has a copy of Linux installed and running on it, but it does not recognize the DSP core. There are several courses of action that can be taken from this point. The first is to continue to work with the Linux core on the Beagle Board. The second is to try and compile OpenCV from source for the ARM architecture and try to get it running on bare metal (no operation system). The third is to find another Linux distribution that runs OpenCV and installing it on the beagle Board. The Ubuntu distribution is known to run on the Beagle Board and supports OpenCV

The team also helped with board design in terms of materials selection and guidance in the early stage of the design of the new demo board. Lastly, Jeff helped John get Visual Studio and SVN repository access installed and configured on his laptop.

Goals for next quarter

- Complete the iPhone CAD Model
    - Model the front panel of the camera carrier. This sub assembly will contain the camera and IR light source.
- Create block diagram for hardware for preprocessing images and extracting the hand position

- Begin designing off board hardware for preprocessing the imagery from the camera.
  - Preprocessing will include keystone correction, image stabilization, and hand tracking.
  - USB interface that is compatible with OpenCV

Mouse-Click Recorder Team: Amir , Mehran and Bonnie

Introduction:

What is the Mouse Click Recorder Program?

The Mouse Click Recorder is a new software program that increases the independence of the visually impaired by allowing them to review a digital image. The MCR program can provide image annotation in a format that is easy for a blind person to use and is designed for a sighted assistant to run while its output is accessible for the blind. When using the mouse click recorder program, a sighted person can highlight a point of interest that a blind person can reference.

Imagine that a sighted person points to several locations on an image displayed on the screen of a computer. The person makes different comments while pointing at each location. The mouse click recorder program allows a sighted person to record coordinates clicked by a mouse on an image and save these coordinates into a generated output file that includes the description of the image, the location (x and y coordinates) of the mouse click, and a note describing the location.

General Overview of Usage:

The Mouse Click Recorder program is stored in a .jar file that runs similarly to an .exe (executable) file. It can be downloaded from the UCSD Grozi website. The user must have the most recent Java

Runtime Environment installed on their computer. (JRE is a free application that contains the minimum requirements to run a .jar file.) Once this is done, the user can simply double click on the .jar file for the program to start running.

The MCR is a simple program that writes to file the location of any mouse click on an image. It provides the user an opportunity to leave a note next to any mouse click location. Suppose the input file is “input_file.jpg.” After clicking on the image with MCR, the generated output file would appear in the same folder named “input_file.txt.” This output file would contain information of the mouse click pixel coordinates and label associated with that coordinate. The MCR program is able to open any .jpg, .bmp, .gif, or .png image as long as it is stored within a folder on his or her computer.

Example of Applicability of Program:

The Mouse Click Recorder program has applicability to blind professionals in a number of jobs. Given an image of any real-world object, a form of measurement, and several clicked coordinates of the image from a sighted assistant, a blind person can calculate and gain a better perception of the dimensions of real world objects. For example, a blind biology student can use photography as a recording tool to track a plant’s growth over time.

The student is provided a jpg image of a plant like the one shown in Figure 1, and asked to determine the height of the plant. The jpg image also contains a picture of a ruler next to the plant. The student can calculate the height of the plant, after determining the pixel locations of the bottom and top of the ruler as well as the bottom and top of the plant with the help of his sighted assistant (to click the designated locations). The blind biology student may now review an ASCII file to get the recorded pixel coordinates from the image along with related notes.

Here is an illustrative example:

Given an image **plant.jpg,** let the plant be P pixels vertically and let the tape measure be T pixels vertically.

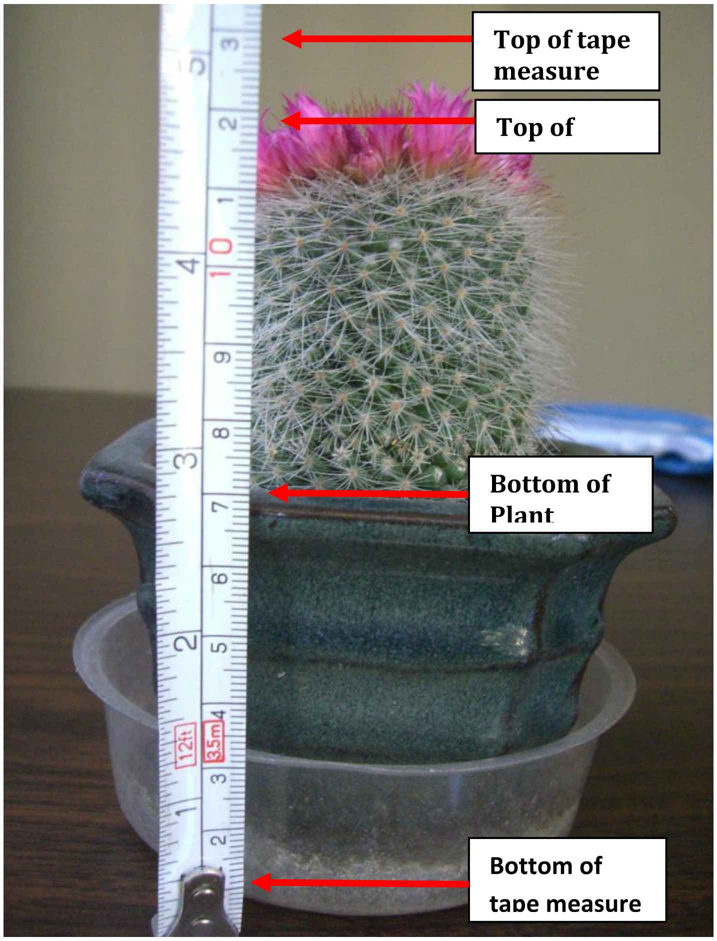


Figure 1 Plant Measurement

After clicking on the image and marking the top and bottom of the plant, and the top and bottom of the ruler, the output file **plant.txt**, should contain the annotated coordinates:

Row | Column:	Comments:
127|451	Top of Plant
137|59	Bottom of Plant
132|255	Top of ruler
136|5	Bottom of ruler

Therefore, to compute the height:

P = top of plant – bottom of plant

= 451 - 59 = 392

T = top of ruler – bottom of ruler

= 255 – 5 = 250

Suppose the tape measure is in meters, H = 1.0 meter. Then to compute the total height of the plant:

$$\text{Total} = H * P / T = 1.0 * 392 / 250 = 1.57 \text{ m}$$

Why XML?

XML (Extensible Markup Language) is a set of rules for encoding documents electronically. It's a flexible and efficient way to store, transmit, and express data over the internet. Many applications have adopted it as their data format, and the availability of comprehensive XML development libraries makes it easy to add support for XML to new and existing applications. For this reason, MCR's shift towards creating an option to generate XML code is to allow the program to become an open-source project. In this way, we can establish a large user-base the test the program and provide useful feedback, or to allow others with the same goal as Grozi to adopt MCR into their own project.

Mouse Click Recorder Installation Procedures:

Downloading Java Runtime Environment

1. Go to http://java.com/en/download/index.jsp
2. Click on **free Java download** to download the most recent JRE on your computer.

Downloading MCR

1. Go to http://grozi.calit2.net/
2. Under the '**downloads**' section, click on the '**Mouse Click Recorder Software**' link.
3. Click the most recent version. A pop up window will appear, click '**save**' to save the file. The Mouse Click Recorder program should now be on your computer in zip format.
4. Extract the **MouseClicker.zip** file.
   To extract a zip file to a specific location in Windows, right click on the file, click **Extract file**, and select the path of the location you want to extract the file to.
5. Double click on the **MouseClickRecorder.jar** file to run the program.

What is included with MCR installation package

The user will find the following files in the mouse click recorder folder:

a. The **MouseClickRecorder_v#.jar** file that runs the MCR program.

b. **Lib** folder: Contains code that allows mouse clicker to run on your computer.

c. **SampleImage** folder: contains the following images:

   a. Brain images **brain1.jpg**, **brain2.jpg**, **brain3.jpg**, **brain4.jpg**

   b. Plant image **plant.jpg,**

   c. Circuit board image **circuitboard.jpg**

   d. Text file **brain1.txt, plant.txt**
   The text file is a sample text file that contains labels with coordinates that represent locations on the image **brain1.jpg** and **plant.jpg** that have been clicked with a mouse beforehand.

   e. XML file **brain1.xml**
   This XML file is a sample XML file that contains labels with coordinates that represent locations on the image **brain1.jpg** that have been clicked with a mouse but in XML format.

d. Documentation of how to run the program.

Running the Mouse Click Recorder:

User Interface

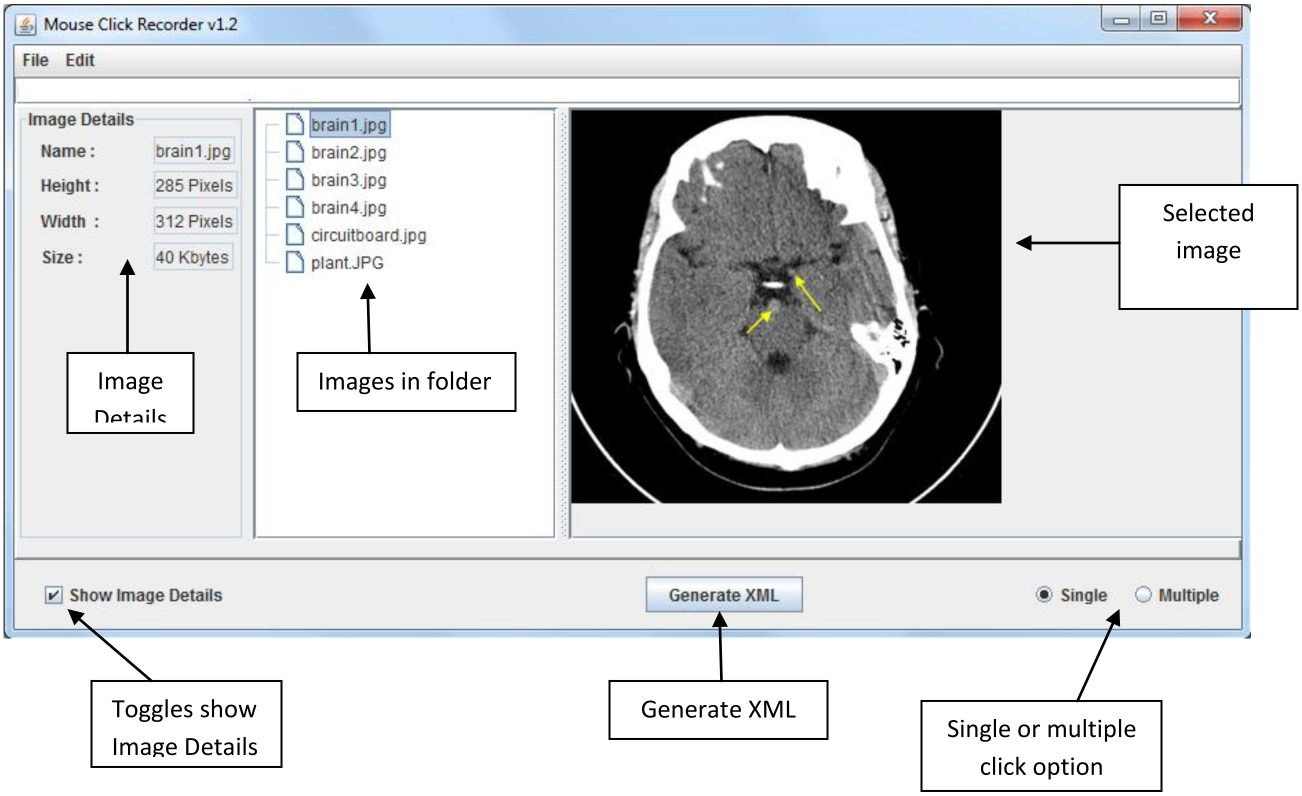


Image Details

Name : brain1.jpg

Height : 285 Pixels

Width : 312 Pixels

Size : 40 Kbytes

☑ Show Image Details

● Single ○ Multiple

- **Image details:** Display the description of the image – filename, pixel width and pixel height, and the size of the file. Image details can be toggled on and off to hide/show the panel.

- **Images in folder:** All images in the opened directory will be displayed in the file listings. MCR can only read .jpg, .gif, .bmp, .png images.

- **Single button:** Allows user to enter a label in the textbox, press and click on a point in the picture. MCR will store the coordinate under that label.

- **Multiple click button:** Allows user to enter a label in the textbox and click on multiple points in the picture. MCR will store all of these coordinates under the same label.

- **Selected Image:** The selected image will appear on the right hand panel once clicked.

- **Generate XML:** Creates an XML output file based on the coordinates clicked and the labels associated with the coordinates.

Generated Output File Format

The MCR program will generate an output file with the same name as the image that was clicked.

i.e. Given a file named **filename.jpg**, the MCR program will create an output file in the same directory named **filename.txt**.

The output file will contain the following information:

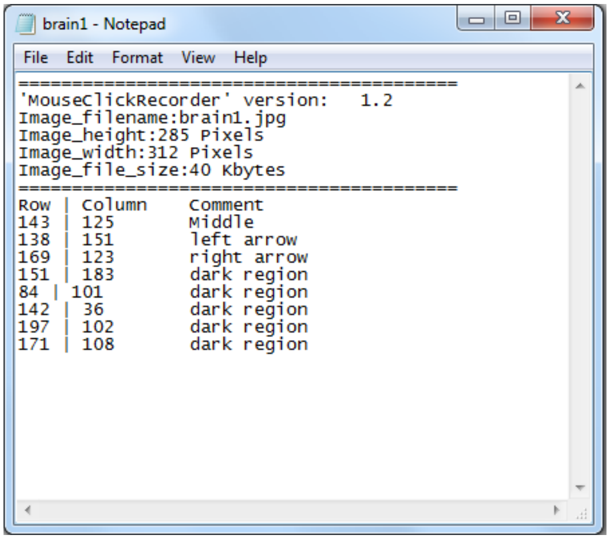
- Header with image details – version number, filename, pixel height and width, file size
- Pixel row
- Pixel column
- Label associated with clicked coordinates (Comments)

i.e. Given an image 8.5 inches wide by 11 inches tall, 4 possible corners could be:

- upper left corner (0,0)
- upper right corner (0, 849)
- lower left corner (1049,0)
- lower right corner (1049, 849)

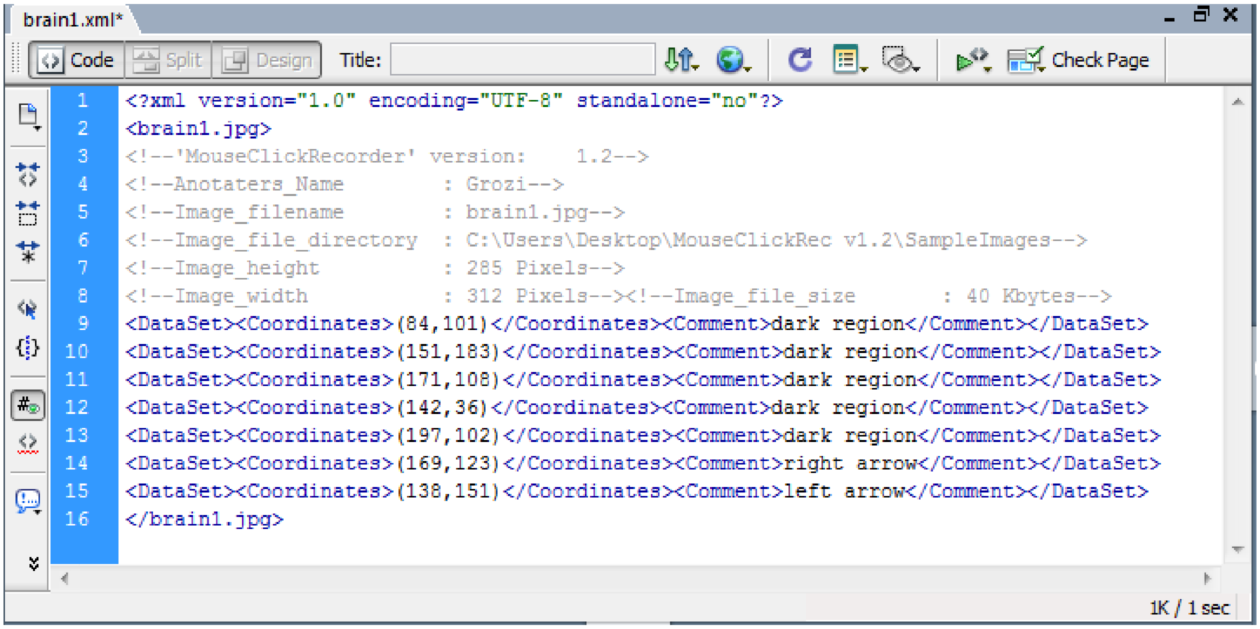
Each line of the generated txt file would contain coordinates of the mouse clicks and a label/note.

```
i.e.  output file:  "200 | 300   picture tag"
                    " ... |  ...   center of brain area"
                    " ... |  ...   dark region of brain"
                       ...     ...
```



```
=========================================
'MouseClickRecorder' version:   1.2
Image_filename:brain1.jpg
Image_height:285 Pixels
Image_width:312 Pixels
Image_file_size:40 Kbytes
=========================================
Row | Column    Comment
143 | 125       Middle
138 | 151       left arrow
169 | 123       right arrow
151 | 183       dark region
84  | 101       dark region
142 | 36        dark region
197 | 102       dark region
171 | 108       dark region
```

Generated XML File Format



```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<brain1.jpg>
<!--'MouseClickRecorder' version:    1.2-->
<!--Anotaters_Name         : Grozi-->
<!--Image_filename         : brain1.jpg-->
<!--Image_file_directory   : C:\Users\Desktop\MouseClickRec v1.2\SampleImages-->
<!--Image_height           : 285 Pixels-->
<!--Image_width            : 312 Pixels--><!--Image_file_size       : 40 Kbytes-->
<DataSet><Coordinates>(84,101)</Coordinates><Comment>dark region</Comment></DataSet>
<DataSet><Coordinates>(151,183)</Coordinates><Comment>dark region</Comment></DataSet>
<DataSet><Coordinates>(171,108)</Coordinates><Comment>dark region</Comment></DataSet>
<DataSet><Coordinates>(142,36)</Coordinates><Comment>dark region</Comment></DataSet>
<DataSet><Coordinates>(197,102)</Coordinates><Comment>dark region</Comment></DataSet>
<DataSet><Coordinates>(169,123)</Coordinates><Comment>right arrow</Comment></DataSet>
<DataSet><Coordinates>(138,151)</Coordinates><Comment>left arrow</Comment></DataSet>
</brain1.jpg>
```

The MCR program also has the option to generate an XML output file based on the coordinates clicked and the labels associated with it.
i.e. Given a file named **filename.jpg**, the MCR program will create an output file in the same directory named **filename.xml**.

The output XML file will contain the following information

- Header with image details – version number, file name, annotators names, pixel height/width, and file size.
- Coordinates (row, column)
- Comments

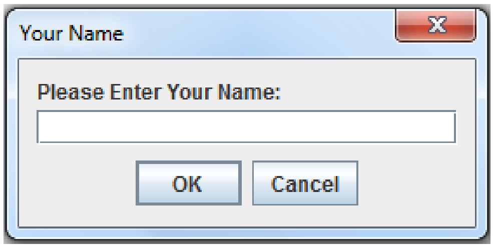
Note: Information in XML file should resemble the generated output. The only difference is the added annotator's name field in the XML file.

Mouse Click Recorder Tutorial:

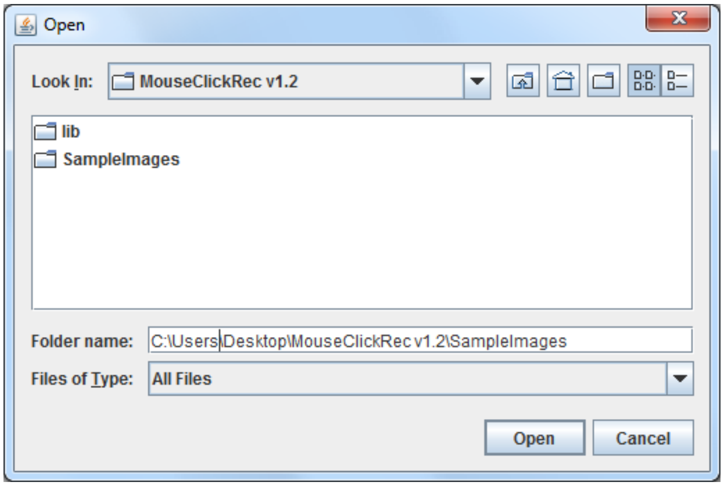
Step-by-step instructions on how to get started with the Mouse Click Recorder Program.

1. **Start the program.**
   Double click on the .jar file to start up the MCR program.

2. **Input your name**
   The MCR program will immediately bring you to an input name window like the one featured below. It will prompt for you to enter you name. This is so labels can be associated to the appropriate authors in the generated XML file.



3. **Select the folder that contains the images.**
   Afterwards, the MCR program will immediately bring you to a file open window like the one featured below. MCR can only open folders, hence if there are any image files that you wish to open, move them into a folder. Then select the folder that contains the image you wish to click on.

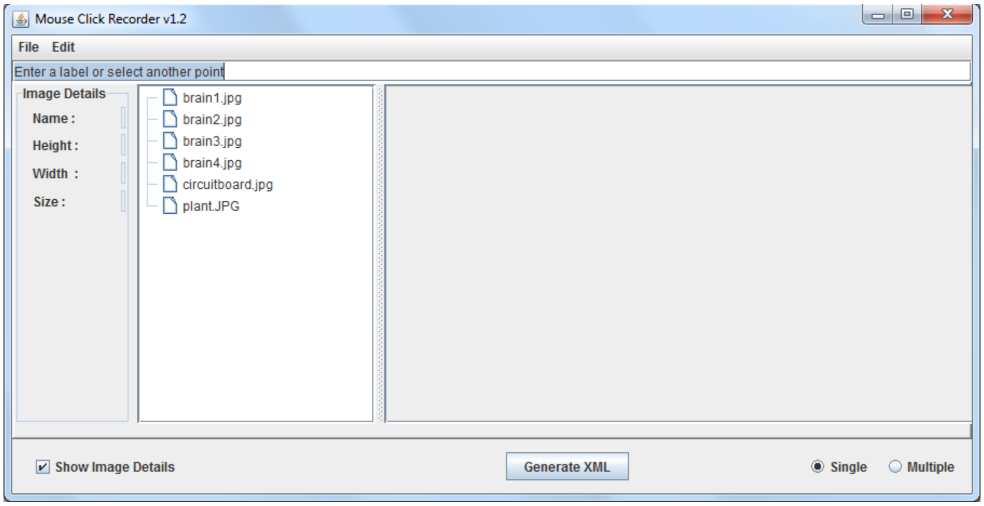


Look In: MouseClickRec v1.2

Folder name: C:\Users\Desktop\MouseClickRec v1.2\SampleImages

Files of Type: All Files

4. **Mouse Clicker Interface.**
   Once you have selected the folder with the image(s) inside, the following window should appear with a list of all the image files in that folder directory. Notice the check box and the 'Single' and 'Multiple' option at the bottom of the window.
   Show Image Details: toggles details about the selected image. The default is checked.
   Single: Stores one coordinate to one label. This is the default option.
   Multiple: Stores multiple coordinates to one label.



Enter a label or select another point

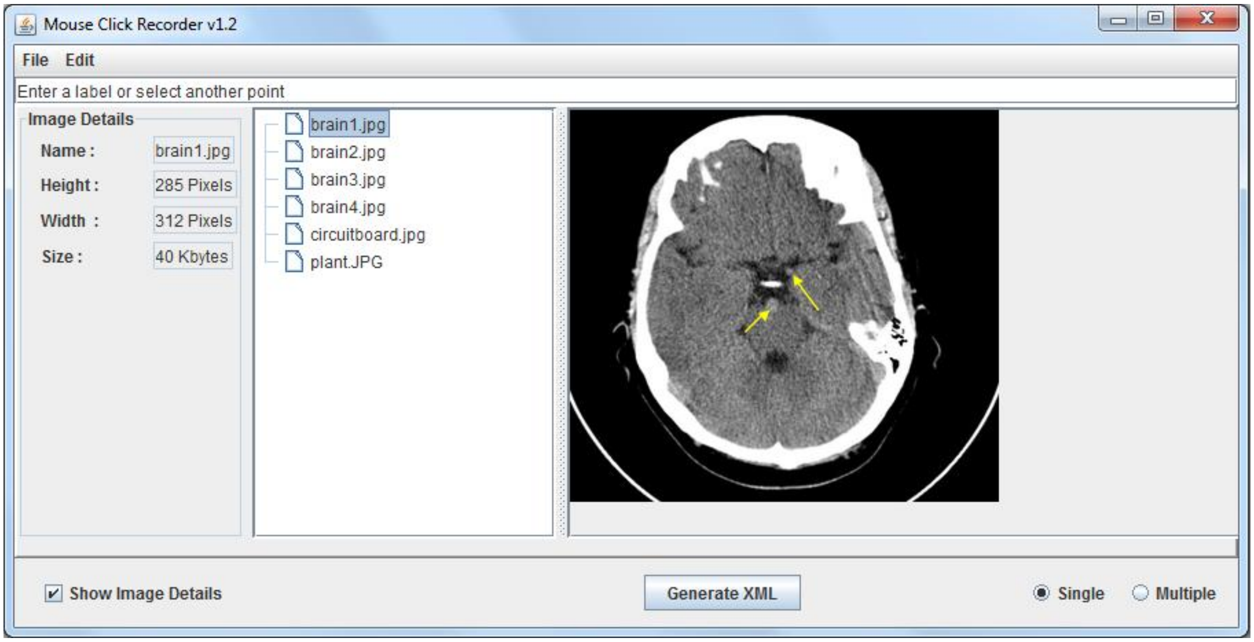
Name :
Height :
Width :
Size :

- brain1.jpg
- brain2.jpg
- brain3.jpg
- brain4.jpg
- circuitboard.jpg
- plant.JPG

☑ Show Image Details

● Single ○ Multiple

5. **Select an image.**
   Select a picture. Doing so would cause the image to show up in the right hand side of the MouseClickWindow. Let's select **brain1.jpg** as shown below:



Enter a label or select another point

Name :	brain1.jpg
Height :	285 Pixels
Width :	312 Pixels
Size :	40 Kbytes

- brain1.jpg
- brain2.jpg
- brain3.jpg
- brain4.jpg
- circuitboard.jpg
- plant.JPG

☑ Show Image Details

● Single ○ Multiple

6. **Record a point.**
   In order for MCR to save the coordinates, **click** on any region within the picture then **enter a label** in the text-box. Finally, **press enter** to save the label. The MCR will save the clicked coordinate in a txt file.
   Note: In order for MCR to save the coordinates with the label, always remember to click, enter a label, then press ENTER.

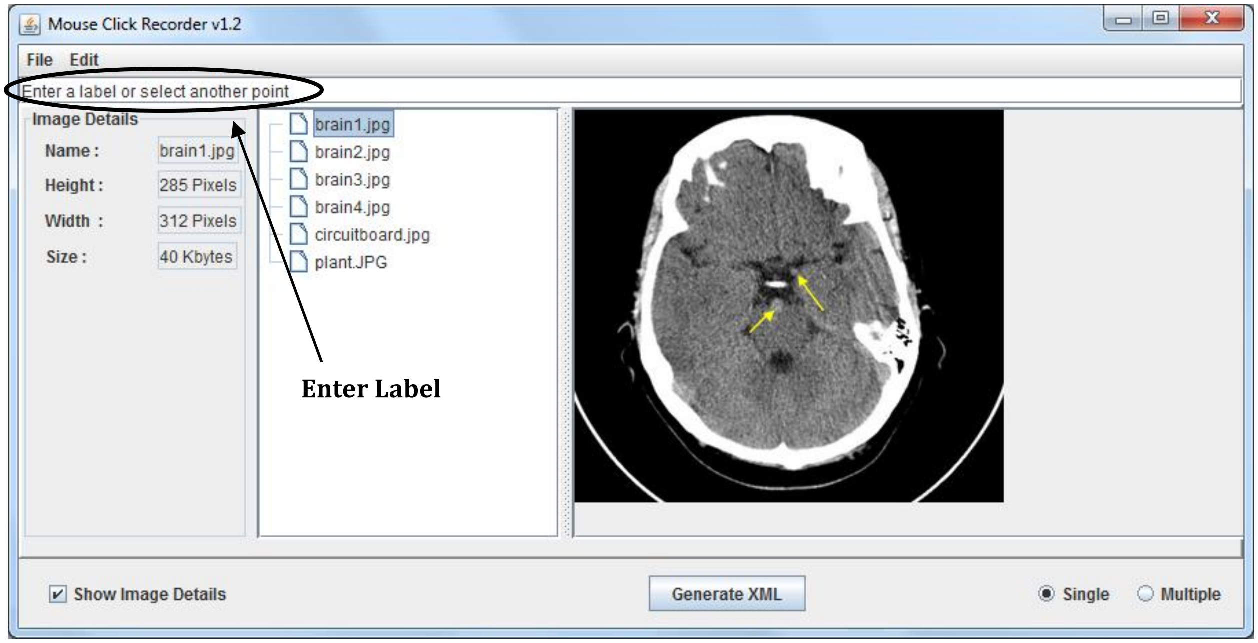


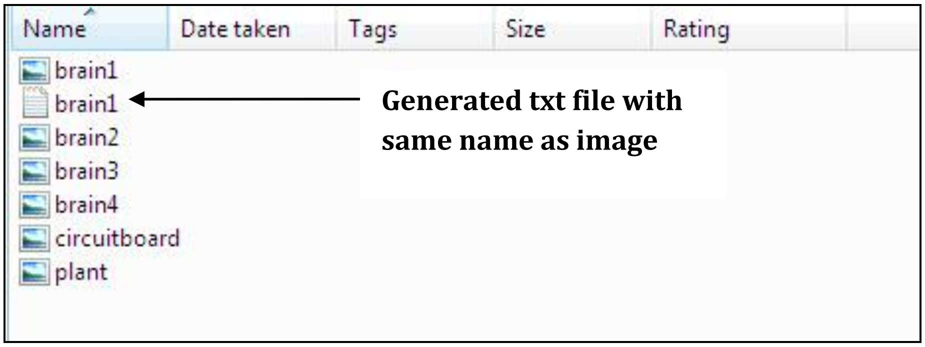
Image Details

Name :	brain1.jpg
Height :	285 Pixels
Width :	312 Pixels
Size :	40 Kbytes

brain1.jpg
brain2.jpg
brain3.jpg
brain4.jpg
circuitboard.jpg
plant.JPG

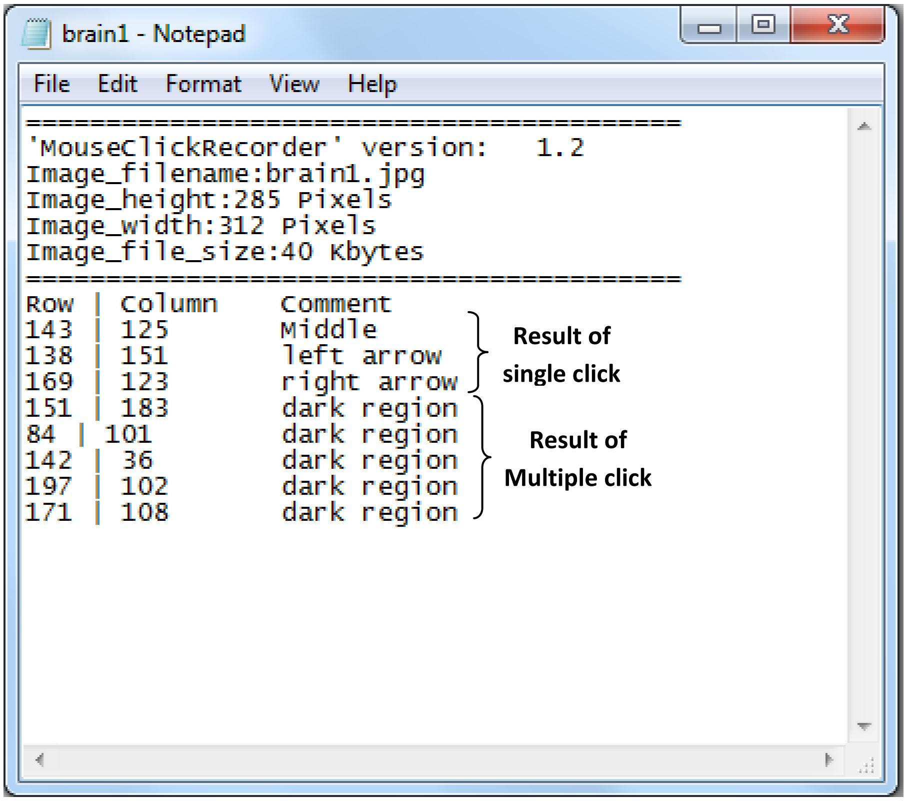
7. **Single click vs. Multiple click.**
   a. **Record a point with a label.**
   Single click: Click on the **Single** option. Click a point in the picture, enter a label in the textbox, then press enter to save the label.
   b. **Record multiple points with same label.**
   Multiple click: Click on the **Multiple** option. Enter a label in the textbox, press enter to save the label, click on multiple points in the picture. MCR will store all of these coordinates under the same label you have entered. The coordinate recording function will be disabled when another label is entered or the **Single** option is selected.

8. **Generated output file.**
   Open up the folder directory on your computer that contains the image you previously clicked on. A corresponding text file with the same name as your image should appear in the same directory.
   i.e. If the file is named ‘brain1.jpg’, an output file named ‘brain1.txt’ will be generated.



Name	Date taken	Tags	Size	Rating
brain1				
brain1				
brain2				
brain3				
brain4				
circuitboard				
plant				

9. **Output file.**
   Open the output file to see the list of coordinates and the labels associated with the clicked coordinates.

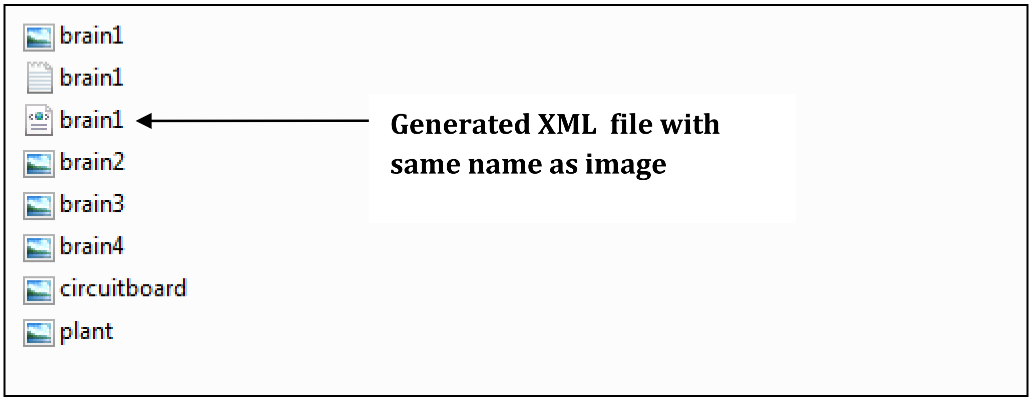


```
=========================================
'MouseClickRecorder' version:   1.2
Image_filename:brain1.jpg
Image_height:285 Pixels
Image_width:312 Pixels
Image_file_size:40 Kbytes
=========================================
Row | Column    Comment
143 | 125       Middle
138 | 151       left arrow
169 | 123       right arrow
151 | 183       dark region
84 | 101        dark region
142 | 36        dark region
197 | 102       dark region
171 | 108       dark region
```

Result of single click

Result of Multiple click

10. **Generate XML.**
    MCR also has an option where you can generate an XML output based on the file output. Just click the 'Generate XML' button at the bottom of the interface and a corresponding xml file with the same name as your image should appear in the same directory.



brain1
brain1
brain1
brain2
brain3
brain4
circuitboard
plant

Generated XML file with same name as image

Mouse Click Recorder Code:

*** The Mouse Click Recorder is written in JAVA and uses the Swing library.**

How an image is fetched and displayed in the Panel (FROM START TO FINISH):

Call Hierarchy to get an image:

MouseClickWindow() constructor → initComponents()

initComponents()→ Calls getCenterPanel()

getCenterPanel() →Calls getJSplitPane0()

getJSplitPane0()→ Calls getJScrollPane0() and getJScrollPane1()

getJScrollPane0()→ Calls getJTree0()

getJScrollPane1()→Calls getImageLabel()

getJTree0()→ invokes a SelectionListener addTreeSelectionListener()

addTreeSelectionListener()→ Calls imageLabelMouseMouseReleased()

imageLabelMouseMouseReleased()

Obtains the path of the file selected on the Tree. Creates an **ImageIcon** “i” and points the path to “i”.

Obtains the height and width of the image:

i.getImage().getHeight(null);

i.getImage().getWidth(null);

Stores the height and width for parallel use:

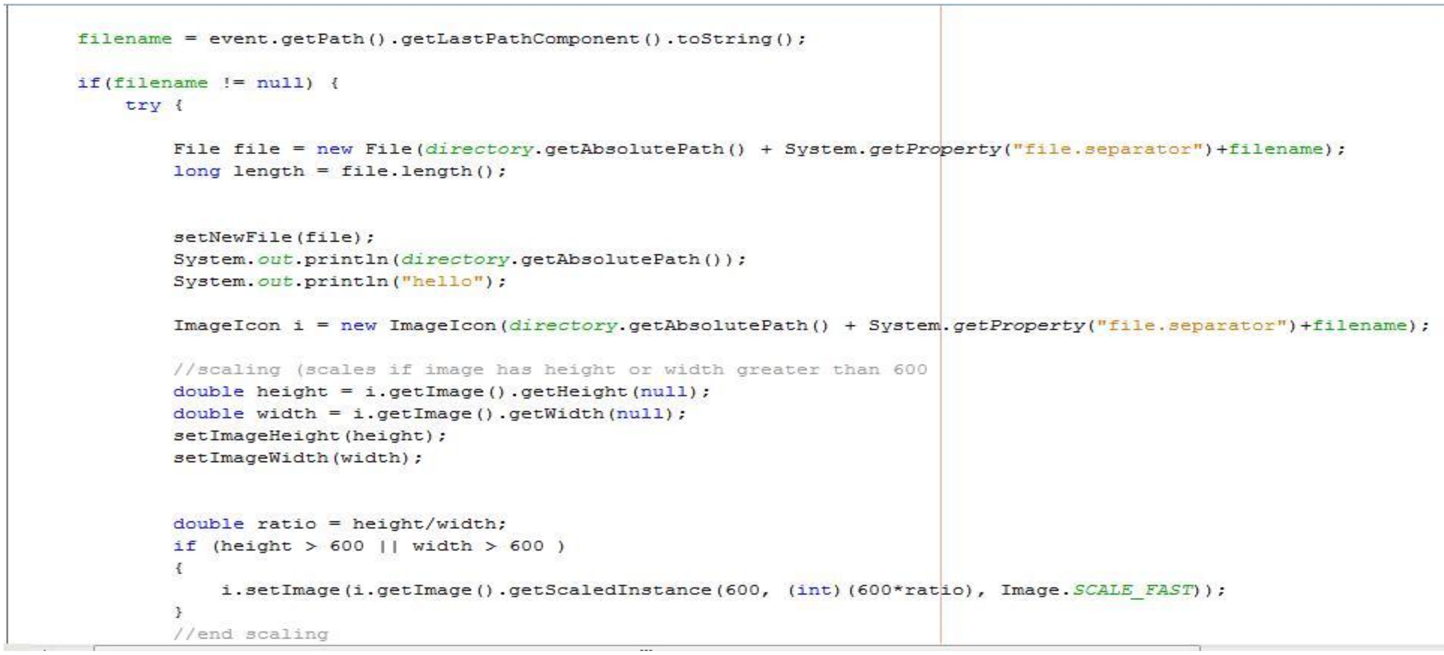
setImageHeight(height);

setImageWidth(width);

Sets the ImageIcon “i” to JLabel “ImageLabel”

imageLabel.setIcon(i);

imageLabel.repaint();



```
filename = event.getPath().getLastPathComponent().toString();

if(filename != null) {
    try {

        File file = new File(directory.getAbsolutePath() + System.getProperty("file.separator")+filename);
        long length = file.length();


        setNewFile(file);
        System.out.println(directory.getAbsolutePath());
        System.out.println("hello");

        ImageIcon i = new ImageIcon(directory.getAbsolutePath() + System.getProperty("file.separator")+filename);

        //scaling (scales if image has height or width greater than 600
        double height = i.getImage().getHeight(null);
        double width = i.getImage().getWidth(null);
        setImageHeight(height);
        setImageWidth(width);


        double ratio = height/width;
        if (height > 600 || width > 600 )
        {
            i.setImage(i.getImage().getScaledInstance(600, (int)(600*ratio), Image.SCALE_FAST));
        }
        //end scaling
```

<u>How a Mouse Click is fetched and passed on to be saved :</u>

Call Hierarchy to get a Mouse Click:

MouseClickWindow() constructor → initComponents()

initComponents()→ Calls getCenterPanel()

getCenterPanel() →Calls getJSplitPane0()

getJSplitPane0()→ Calls getJScrollPane0() and getJScrollPane1()

getJScrollPane0()→ Calls getJTree0()

getJScrollPane1()→Calls getImageLabel()

getImageLabel()→ invokes MouseListener mouseReleased()

mouseReleased() ()→ Calls imageLabelMouseMouseReleased ()

imageLabelMouseMouseReleased ():

Looks and the selection Mode:

If “multiple” is selected:

It stores the passed in point that was selected

It Calls saveAnnotation() on the point and the label in the Commentfield.

if(this.selectMode == 1)

mouseCoords = event.getPoint();

saveAnnotation(directory, filename, mouseCoords.x, mouseCoords.y, getCommentField().getText());

if “single” is selected:

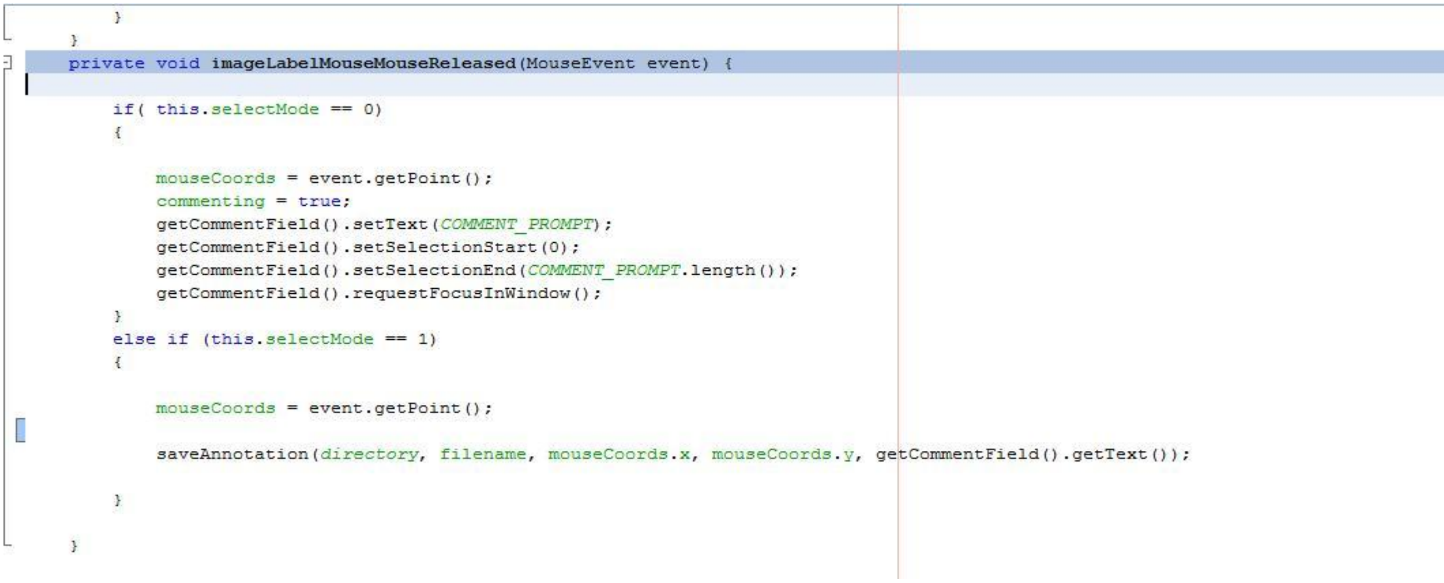
It stores the passed in point that was selected

Prompts the user to input something for the label.

if(this.selectMode == 0)

mouseCoords = event.getPoint();;

getCommentField().setText(COMMENT_PROMPT);



```
        }
    }
    private void imageLabelMouseMouseReleased(MouseEvent event) {

        if( this.selectMode == 0)
        {

            mouseCoords = event.getPoint();
            commenting = true;
            getCommentField().setText(COMMENT_PROMPT);
            getCommentField().setSelectionStart(0);
            getCommentField().setSelectionEnd(COMMENT_PROMPT.length());
            getCommentField().requestFocusInWindow();
        }
        else if (this.selectMode == 1)
        {

            mouseCoords = event.getPoint();

            saveAnnotation(directory, filename, mouseCoords.x, mouseCoords.y, getCommentField().getText());

        }
    }
```

How a label is fetched and passed on to be saved in text :

Call Hierarchy to get a Mouse Click:

MouseClickWindow() constructor → initComponents()

initComponents()→ Calls getCommentField()

getCommentField() → invokes actionPerformed ()

actionPerformed ():

Looks and the selection Mode:

If “multiple” is selected:

It highlights the text.

(label assumed to be inserted when Image was clicked)

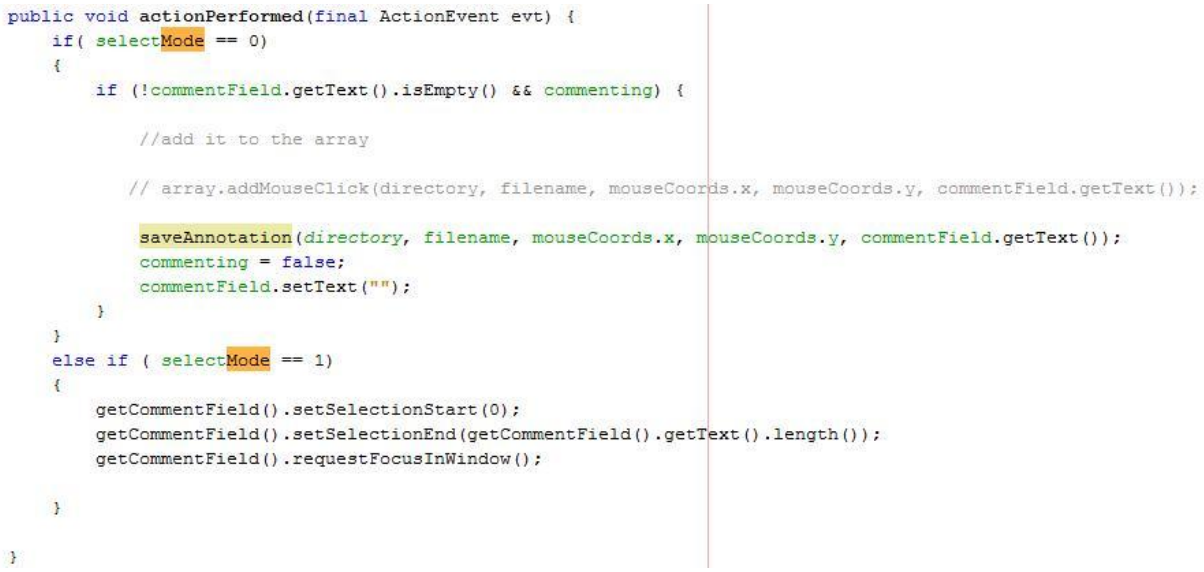
```
if ( selectMode == 1)
        setSelectionStart(0);
        setSelectionEnd(getCommentField().getText().length());
        requestFocusInWindow();
```

if “single” is selected:

It calls saveAnnotation() on the mouse click and the comment.

It sets the comment to an empty string.

```
if ( selectMode == 0)
        saveAnnotation(directory, filename, mouseCoords.x, mouseCoords.y, commentField.getText());
        commentField.setText("");
```



```
public void actionPerformed(final ActionEvent evt) {
    if( selectMode == 0)
    {
        if (!commentField.getText().isEmpty() && commenting) {

            //add it to the array

           // array.addMouseClick(directory, filename, mouseCoords.x, mouseCoords.y, commentField.getText());

            saveAnnotation(directory, filename, mouseCoords.x, mouseCoords.y, commentField.getText());
            commenting = false;
            commentField.setText("");
        }
    }
    else if ( selectMode == 1)
    {
        getCommentField().setSelectionStart(0);
        getCommentField().setSelectionEnd(getCommentField().getText().length());
        getCommentField().requestFocusInWindow();

    }

}
```

<u>How a label is passed in to a text :</u>

saveAnnotation() takes in (File directory, String fileName, int x, int y, String label, string) as its parameters.

It creates a file name with .txt as the extension.

int extPos = fileName.lastIndexOf(".");

String annotationFileName = fileName.substring(0,extPos+1) + "txt";

File annotationsFile = new File(directory.getAbsolutePath()+System.getProperty("file.separator")+annotationFileName);

It checks if the file is new:

If it’s a new file writes the Title buffer to the txt file.

If it’s an old file it continues with the process and writes the values of “x”, “y” and “label” to the txt file.

isNewFile()?

if (isNewFile())

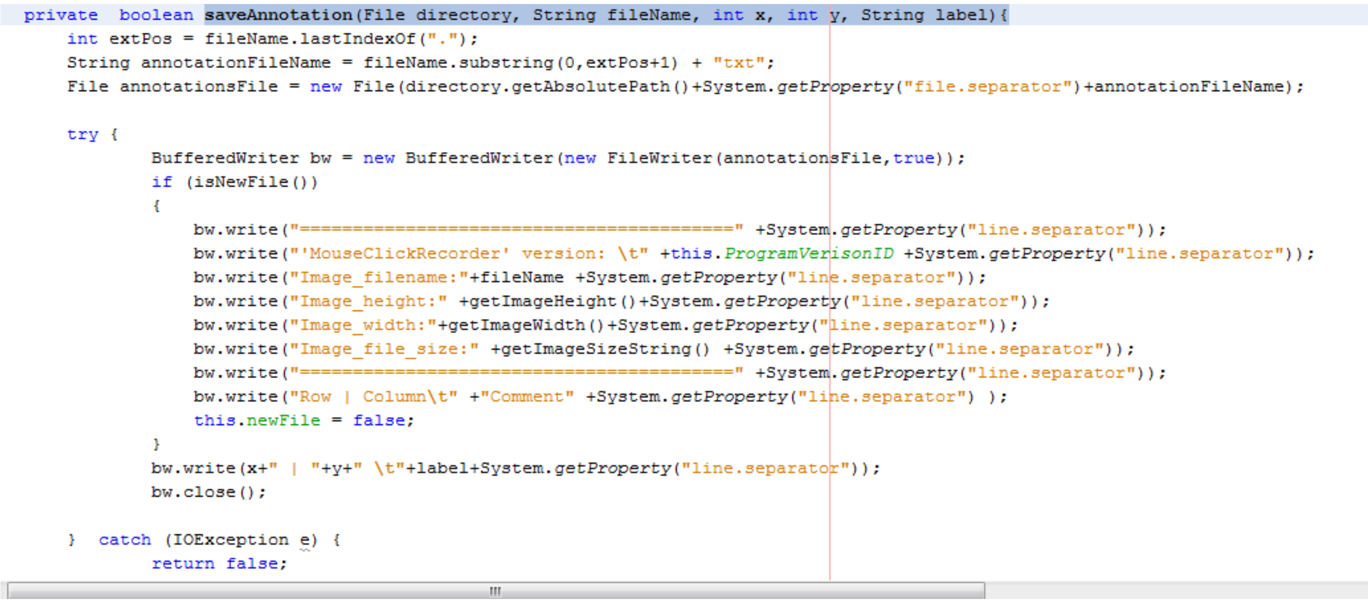
{

bw.write("==="

+System.getProperty("line.separator"));

bw.write("'MouseClickRecorder' version: \t" +this.ProgramVerisonID +System.getProperty("line.separator"));

```
            bw.write("Image_filename:"+fileName +System.getProperty("line.separator"));

            bw.write("Image_height:" +getImageHeight()+System.getProperty("line.separator"));

            bw.write("Image_width:"+getImageWidth()+System.getProperty("line.separator"));

            bw.write("Image_file_size:" +getImageSizeString() +System.getProperty("line.separator"));

             bw.write("========================================" +System.getProperty("line.separator"));

             bw.write("Row | Column\t" +"Comment" +System.getProperty("line.separator") );

            this.newFile = false;

             }

 bw.write(x+" | "+y+" \t"+label+System.getProperty("line.separator"));

  bw.close();
```



```
 private  boolean saveAnnotation(File directory, String fileName, int x, int y, String label){
     int extPos = fileName.lastIndexOf(".");
     String annotationFileName = fileName.substring(0,extPos+1) + "txt";
     File annotationsFile = new File(directory.getAbsolutePath()+System.getProperty("file.separator")+annotationFileName);

     try {
             BufferedWriter bw = new BufferedWriter(new FileWriter(annotationsFile,true));
             if (isNewFile())
             {
                 bw.write("==========================================" +System.getProperty("line.separator"));
                 bw.write("'MouseClickRecorder' version: \t" +this.ProgramVerisonID +System.getProperty("line.separator"));
                 bw.write("Image_filename:"+fileName +System.getProperty("line.separator"));
                 bw.write("Image_height:" +getImageHeight()+System.getProperty("line.separator"));
                 bw.write("Image_width:"+getImageWidth()+System.getProperty("line.separator"));
                 bw.write("Image_file_size:" +getImageSizeString() +System.getProperty("line.separator"));
                 bw.write("==========================================" +System.getProperty("line.separator"));
                 bw.write("Row | Column\t" +"Comment" +System.getProperty("line.separator") );
                 this.newFile = false;
             }
             bw.write(x+" | "+y+" \t"+label+System.getProperty("line.separator"));
             bw.close();

     }  catch (IOException e) {
             return false;
```

isNewFile(): checks if in the current folder there exists a file with the “filename” and extension “txt”. It returns “true” if file is not there.

```
 String temp = file.getPath().substring(0,file.getPath().length()-4) + ".txt";

      File tempFile = new File(temp);

     if(tempFile.exists())

           this.newFile = false;
```

How does the XML file gets Generated :

Just like the txt file the XML file gets created in the saveAnnotation() function, right now XML file can only be created if there was no annotation file present before.

In the saveAnnotation() function an XML file object gets instantiated first as follows:

```
String XMLannotationFileName = fileName.substring(0,extPos+1) + "xml";
        File annotationsFile = new
File(directory.getAbsolutePath()+System.getProperty("file.separator")+annotat
ionFileName);
        File XMLannotationsFile = new
File(directory.getAbsolutePath()+System.getProperty("file.separator")+XMLanno
tationFileName);
```

Then an instance of XML class is created:

```
XMLFile = annotationFileName;
XML XMLObj = new XML();
XMLObj.setFile(XMLannotationsFile);
XMLObj.setTitle(fileName);
XMLObj.setDirectory(directory.getAbsolutePath());
XMLObj.setHight(getImageHeight());
XMLObj.setSize(getImageSizeString());
XMLObj.setWidth(getImageWidth());
XMLObj.data("("+Integer.toString(x)+","+Integer.toString(y)+")",label);
                  XMLObj.setAuthor(authorName);
                  fileHashMap.put(annotationFileName, XMLObj);
```

What happens in the XML Class:

Inside the XML constructor, all of the member fields get initialized as follow:

```
public XML(){
      configFile = null;
      hight = null;
      width = null;
      author = null;
      title = null;
      directory = null;
      dataSets = new HashMap<String,String>();
}
```

There are getters and setters for all of the member functions in the class. The main function is the `generateXML()`, which generates the XML file with the data that has been set in the saveAnnotation() function.

`generateXML()`, uses the `DocumentBuilderFactory` to create the **Element** object and invokes the **Document** object. The code is as follow:

```
   public void generateXML(){

        if (configFile == null){
            JOptionPane.showMessageDialog(null,
                      "No file has been selected.",
                      "File Error",
                      JOptionPane.ERROR_MESSAGE);

       }else{



            try{
                   DocumentBuilderFactory documentBuilderFactory =
DocumentBuilderFactory.newInstance();
                   DocumentBuilder documentBuilder =
documentBuilderFactory.newDocumentBuilder();
                   Document document = documentBuilder.newDocument();
                   Element rootElement = document.createElement(title);
                   document.appendChild(rootElement);
                   Comment MCR = document.createComment
("'MouseClickRecorder' version:\t 1.2");
                   Comment Author = document.createComment   ("Anotaters_Name
: "+author);
                   Comment FileName = document.createComment ("Image_filename
: "+title);
                   Comment Directory =
document.createComment("Image_file_directory  : "+directory);
                   Comment FileSize = document.createComment
("Image_file_size       : "+size);
                   Comment FileHight = document.createComment("Image_height
: "+hight);
                   Comment FileWidth = document.createComment("Image_width
: "+width);
                   rootElement.appendChild(MCR);
                   rootElement.appendChild(Author);
                   rootElement.appendChild(FileName);
                   rootElement.appendChild(Directory);
                   rootElement.appendChild(FileHight);
                   rootElement.appendChild(FileWidth);
                   rootElement.appendChild(FileSize);


                   Set<String> s =dataSets.keySet();
                   Iterator<String> it = s.iterator();
                   System.out.println("Dir: "+directory);
                   while(it.hasNext()){
                     Element root = document.createElement("DataSet");
                     String x = it.next();
                     String data = dataSets.get(x);
                     System.out.println("Data = "+data);
                     String element = "Coordinates";
                     Element Cordinates = document.createElement(element);
                     Cordinates.appendChild(document.createTextNode(x));
                     Element Label = document.createElement("Comment");
                     Label.appendChild(document.createTextNode(data));
                     root.appendChild(Cordinates);
                     root.appendChild(Label);
                     rootElement.appendChild(root);
                   }
                   FileOutputStream fos;
                   DataOutputStream dos;
                   //File out = new File("D:\\MyFile.xml");
                   fos = new FileOutputStream(configFile);
                   dos = new DataOutputStream(fos);
                   //dos.write(12);
                     //BufferedWriter bf = new BufferedWriter(out);
                   TransformerFactory transformerFactory =
TransformerFactory.newInstance();
                   Transformer transformer =
transformerFactory.newTransformer();
                   DOMSource source = new DOMSource(document);
                   StreamResult result =  new StreamResult(dos);
                   transformer.transform(source, result);

                   dos.close();
            }catch(Exception e){
                  System.err.println(e.getMessage());

            }
      }
```

References

1. http://www.wiili.org/index.php/Compatible_Bluetooth_Devices

2. http://wiibrew.org/wiki/List_of_Working_Bluetooth_Devices
(Compatible Bluetooth Devices)

3. http://www.wiimoteproject.com/bluetooth-and-connectivity-knowledge-center/a-summary-of-windows-bluetooth-stacks-and-their-connection/

4. http://www.dev-toast.com/2007/01/05/uncrippling-bluetooth-in-vista-rtm/

5.http://www.rapidsharedownload.net/software/widcomm-bluetooth-software-5.1.0.1100/

6. http://www.wiili.org/forum/bluecove-210-on-bluez-tips-t6355.html

7. http://xii9190.wordpress.com/page/15/ (mainly to find out how to set vibrate)

http://www.wiili.org/forum/wiiremote-disconnection-problem-t5359.html (help on disconnection problem, but I did it slightly differently in my code)

8. http://www.wiili.org/forum/bluetooth-fails-to-initialize-without-wiiremotedisconnect()-t6805.html

http://wiki.multimedia.cx/index.php?title=PCM (Audio file needs to be a signed 8-bit PCM)

9. http://grozi.calit2.net/files/TIESGroZiSp09.pdf