GNG2101

Design Project User Manual

magnifierPLUS

Submitted by:

Accessibility8, A8

Erin Cox, 300051053

Alec Edward, 300115955

Alexander Delisle, 300116627

Isabelle Tam, 300115159

Date: December 10th, 2020

University of Ottawa

Abstract

The capability of using a computer with ease is often overlooked, however, there is a large fraction of the population who struggle with computer use due to a disability or impairment. Although there are many virtual magnifiers that currently exist to help people with visual impairment, they tend not to consider those who also struggle with dexterity or other issues. As a result, they struggle with activating and navigating the magnifier, rendering it useless or making computer usage an even more difficult task than it was before. Therefore, there is a dire need for a new solution, a solution that considers people who struggle with more than one impairment. Thus, there is magnifierPLUS, a magnifier that allows users to easily activate and use. The magnifier consists of a magnifying window that follows the cursor on the screen, where the magnification can be increased with the simple push of a button. The magnifier is easily activated and deactivated by the push of a hotkey and does not require different configurations of keys to be pressed. This design recognizes the broad range of disabilities and struggles users may have that impact their ability to use a computer.

Table of Contents

List of Tables

List of Figures

1. Introduction

In this user manual, the workings of the project are discussed. This includes how it works, and how the users can access and use the key features of the product. The other sections outline how the project was made, the operations behind the functionality, how to install and perform regular maintenance, health and safety concerns, and a troubleshooting section if the user is having trouble with certain features of the product. Lastly, the team summarizes the recommendations for future work on this project, this section includes how the product can be generally improved as well as how modifications could be made to help out people with other disabilities who have difficulty using the device.

2. Important Features, Functions and Capabilities

2.1 Software Prototype

The software component of our project is central to the project overall. The magnifier software allows users to read text, navigate websites, and use social media. The magnifier is able to zoom up to three times at preset intervals which can be changed with tweaks to the magnifier code. The software component is intended to remain active for as long as the computer is active to allow users to activate the magnifier easily through the button without interacting with an onscreen menu which is not suitable for the users with dexterity issues and visual impairments. The key listener software currently listens for the "`" key but could easily be changed to a key with the least use on the keyboard if needed. The software is designed to have the magnifier window to follow the cursor, the size of the window can be adjusted as needed within the magnifier code. By following the cursor, the magnifier is exceptionally easy to use as the magnified area will be updated as the cursor is moved so the user is not required to input additional parameters.

2.2 Hardware Prototype

The hardware component of our project was designed to serve a user base with a range of accessibility needs, as a result the features of our design were chosen to accommodate these needs. We chose a large button inside a project box to be connected to the computer to give our software an activation method without navigating through on screen menus. The external button itself has a translucent cover to allow light from an LED within the box to indicate to the user

that the button is active. When the button is depressed, the light turns off and will remain off as long as the button is pressed. The button on the interior button is pressed via the external button which then interacts with the Arduino Leonardo within the box, the interior button was selected to be particularly ‘click-y’ to give the user haptic feedback when the button is pressed. In addition, the button box is suited for a COVID-19 environment, the usb output is sealed with a gasket and the remaining holes are well sealed meaning that the box could be sprayed with a disinfectant to accommodate multiple users on one computer.

3. How The Prototype Was Made

This section will describe how the prototype was made, including the materials needed to make the hardware portion as well as the code needed for the software portion.

3.1 Software Prototype

There are two main components of the software prototype. First, is the Arduino code that needs to be compiled and uploaded to the Arduino Leonardo. Second, is the Java code that is used for the magnifier and key listener.

3.1.1 Arduino Leonardo Hot-Key Setup

1. Download Arduino from the following link https://software.intel.com/en-us/get-started-arduino-install
2. Open Arduino and change board to “Arduino Leonardo” and select the correct serial port for the Arduino.

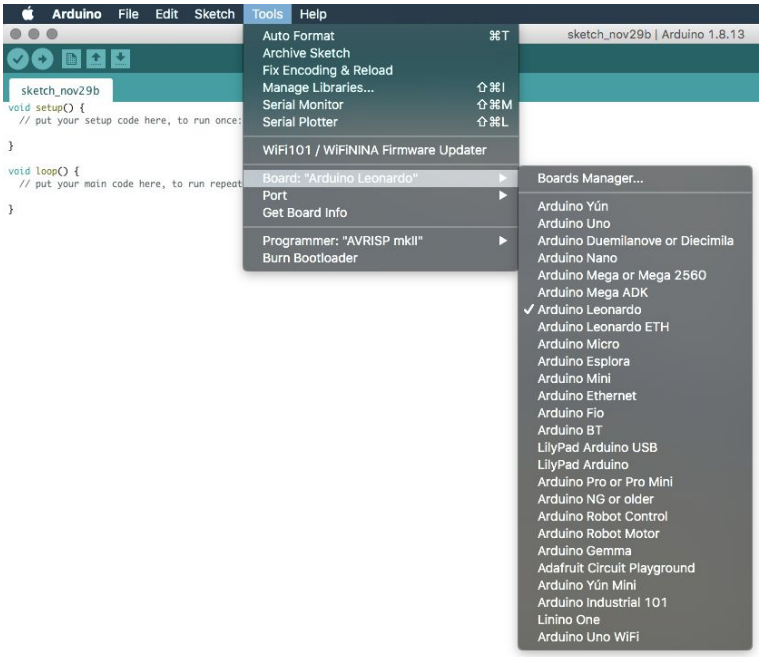

Figure 1: Board Name

3. Download button input code from https://www.arduino.cc/en/Tutorial/BuiltInExamples/KeyboardMessage and open it as a new project in Arduino.

```
*/

#include "Keyboard.h"

const int buttonPin = 4;          // input pin for pushbutton
int previousButtonState = HIGH;   // for checking the state of a pushButton
int counter = 0;                  // button push counter

void setup() {
  // make the pushButton pin an input:
  pinMode(buttonPin, INPUT);
  // initialize control over the keyboard:
  Keyboard.begin();
}

void loop() {
  // read the pushbutton:
  int buttonState = digitalRead(buttonPin);
  // if the button state has changed,
  if ((buttonState != previousButtonState)
      // and it's currently pressed:
      && (buttonState == HIGH)) {
    // increment the button counter
    counter++;
    // type out a message
    Keyboard.print("You pressed the button ");
    Keyboard.print(counter);
    Keyboard.println(" times.");
  }
  // save the current button state for comparison next time:
  previousButtonState = buttonState;
}
```

Figure 2: Default Button Input Code

4. The key listener will, by default, output “You have pressed the button n times” where n is the number of presses. Change the Keyboard.Print line “Keyboard.print(“`”) if desired, as currently configured the key listener will detect the “`” key but this can be easily changed, any character that does not have a predefined function within Arduino.
5. Verify and upload the modified code to the Arduino Leonardo, if the Arduino is equipped with a button and is plugged into the computer after the code has been uploaded, a button press will print “`” as keyboard output.

```
#include "Keyboard.h"

const int buttonPin = 4;          // input pin for pushbutton
int previousButtonState = HIGH;   // for checking the state of a pushButton
int counter = 0;                  // button push counter

void setup() {
  // make the pushButton pin an input:
  pinMode(buttonPin, INPUT);
  // initialize control over the keyboard:
  Keyboard.begin();
}

void loop() {
  // read the pushbutton:
  int buttonState = digitalRead(buttonPin);
  // if the button state has changed,
  if ((buttonState != previousButtonState)
      // and it's currently pressed:
      && (buttonState == HIGH)) {
    // increment the button counter
    counter++;
    // type out a message
    Keyboard.print("`");
    //Keyboard.print(counter);
    //Keyboard.println("");
  }
  // save the current button state for comparison next time:
  previousButtonState = buttonState;
}
```

Figure 3: Modified Button Code

3.1.2 Java Code

1. Download IntelliJ IDEA from: https://www.jetbrains.com/idea/download/#section=linux

2. Create a new project, name it “KeyListener”
3. Right click src, located on the left side of the screen, hover over new and click new class.

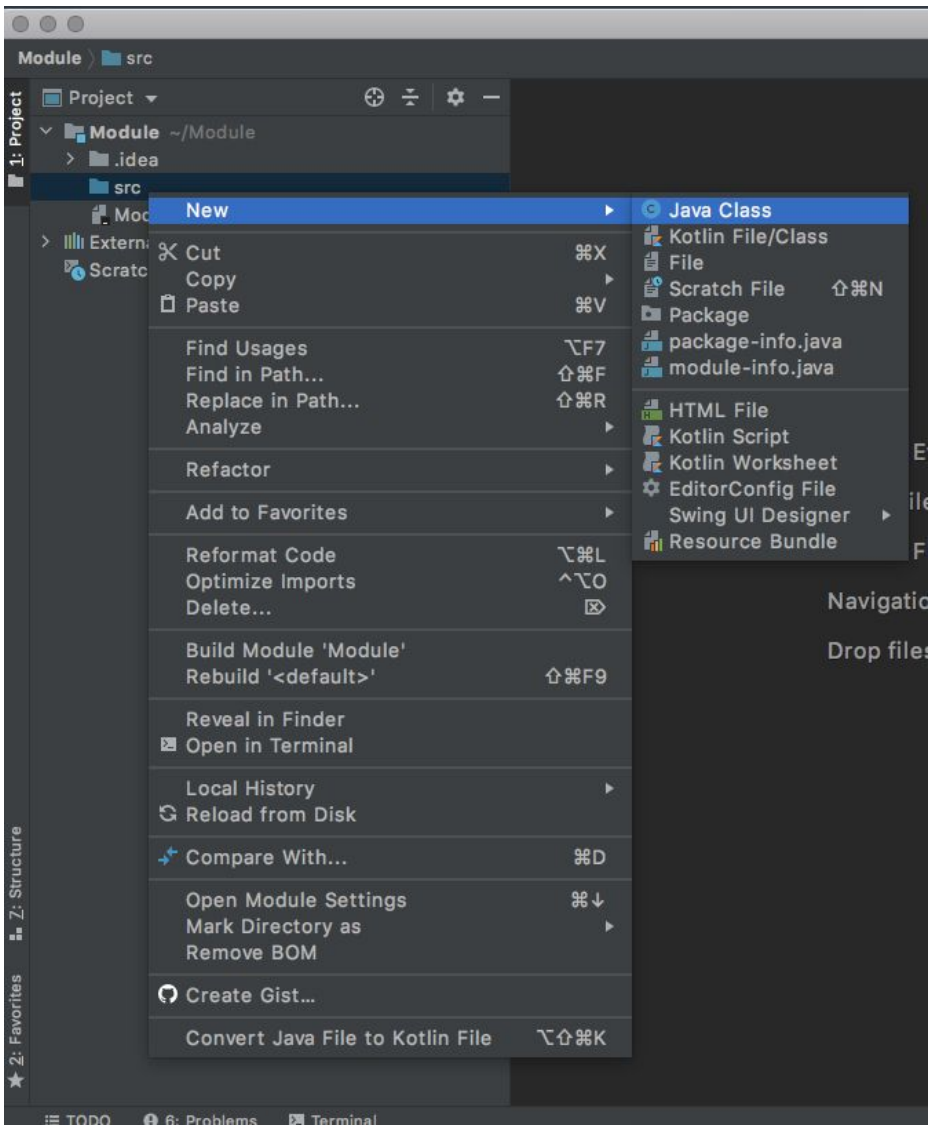

Figure 4: Create a New Class

4. Name the new class “test”
5. From File, Click Project structure
6. From Modules, click dependency and then library, as seen in figure #.

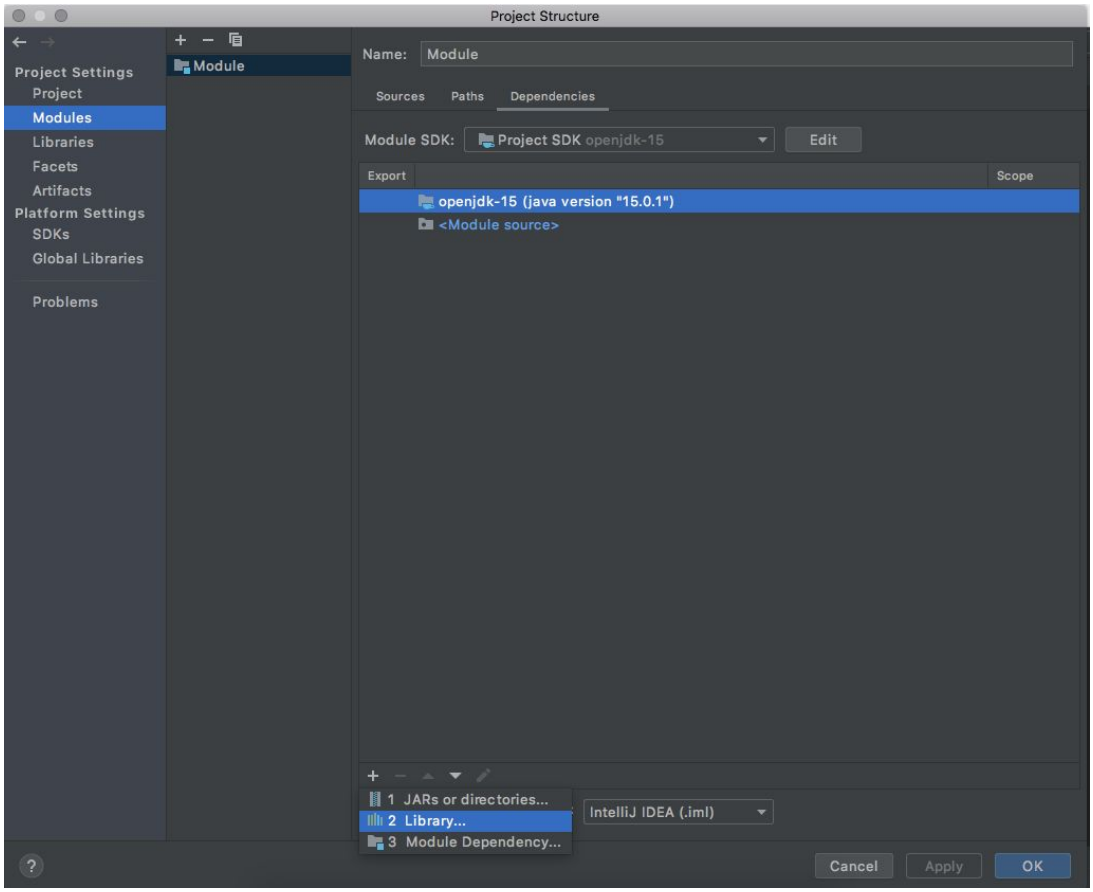

Figure 5: New Dependency

7. Once you have clicked Library, select a new library from Maven.

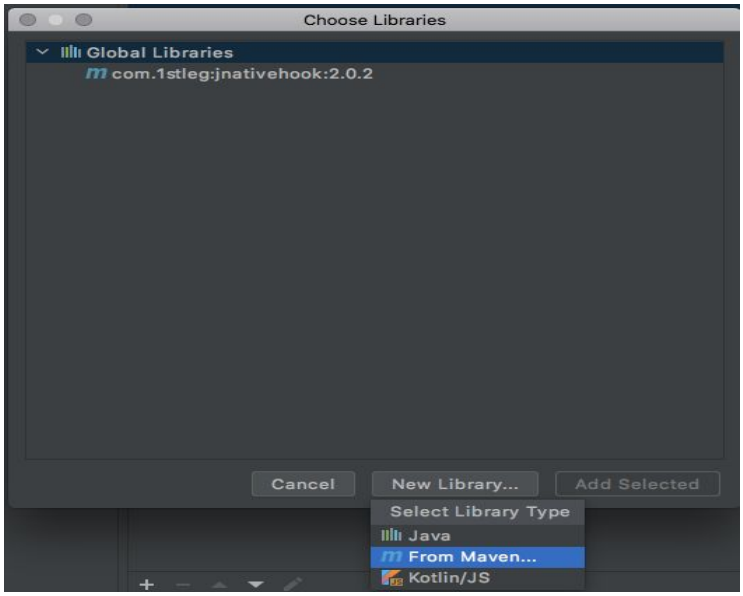

Figure 6: New Library From Maven

8. Search “jnativehook” and select the following.

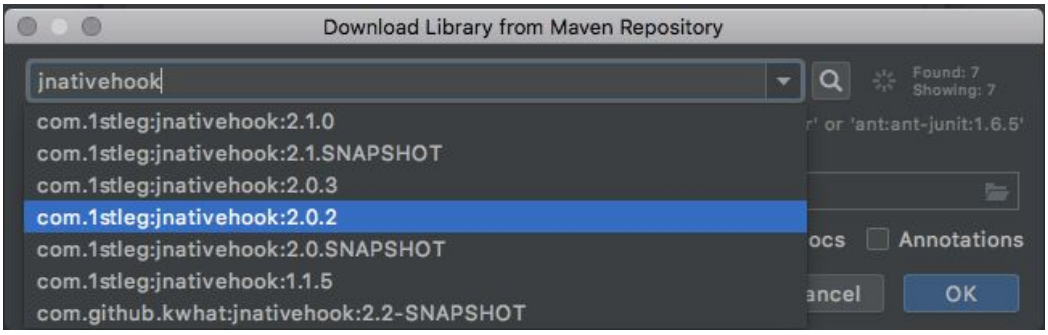

Figure 7: jnativehook 2.0.2

9. Now, import the necessary libraries by writing them above the public class.

```
1  import javax.swing.*;
2  import org.jnativehook.GlobalScreen;
3  import org.jnativehook.NativeHookException;
4  import org.jnativehook.keyboard.NativeKeyEvent;
5  import org.jnativehook.keyboard.NativeKeyListener;
6  import java.awt.*;
7  import java.awt.event.*;
8  import java.awt.image.BufferedImage;
9
```

Figure 8: Import Statements

10. Extend JFrame and implement NativeKeyListener on the same line as the “test” public class. This is seen in figure #.
11. Next, initialize variables under the public class “test” created in step 4.

```
10  public class test extends JFrame implements NativeKeyListener {
11      Robot robot;
12      JFrame frame;
13      int on = 0;
14      int zoomFactor =2;
15      PointerInfo pi;
16      JPanel gui;
17      JLabel output;
18      Timer t;
19
```

Figure 9: Variables

12. Next, create a construct, this construct will include the magnifier code.

```
19
20
21      public test() throws AWTException {
22
23          frame = new JFrame( title: "Simple GUI");
24          robot = new Robot();
25          gui = new JPanel(new BorderLayout( hgap: 2,  vgap: 2));
26          output = new JLabel( text: "point to see zoom");
27          gui.add(output, BorderLayout.PAGE_END);
28          final int size = 200;
29          final BufferedImage bi = new BufferedImage(size, size, BufferedImage.TYPE_INT_RGB);
30          final JLabel zoomLabel = new JLabel(new ImageIcon(bi));
31          gui.add(zoomLabel, BorderLayout.CENTER);
32
33
34          ActionListener zooomListener = (ActionEvent e) -> {...};
52          t = new Timer( delay: 40, zooomListener);
53          t.start();
54          GlobalScreen.addNativeKeyListener(this);
55          while (true) {
56              Point p = MouseInfo.getPointerInfo().getLocation();
57              frame.setLocation((p.x + 55), (p.y + 20));
58              updateRegion();
59          }
60
61      }
```

Figure 10: Construct

13. Now, create a public void method called “updateRegion” that holds magnifier code to update the magnification region.

```
63      public void updateRegion()
64      {
65          final int size = 200;
66          BufferedImage bi = new BufferedImage(
67                  size, size, BufferedImage.TYPE_INT_RGB);
68          JLabel zoomLabel = new JLabel(new ImageIcon(bi));
69          pi = MouseInfo.getPointerInfo();
70          Point p = pi.getLocation();
71          Rectangle r = new Rectangle(
72                  x: p.x - (size / (2 * zoomFactor)),
73                  y: p.y - (size / (2 * zoomFactor)),
74                  (size / zoomFactor),
75                  (size / zoomFactor));
76          BufferedImage temp = robot.createScreenCapture(r);
77          Graphics g = bi.getGraphics();
78          g.drawImage(temp,  x: 0,  y: 0, size, size,  observer: null);
79          g.setColor(new Color( r: 255, g: 0, b: 0, a: 128));
80          int x = (size/2)-1;
81          int y = (size/2)-1;
82          g.dispose();
83          zoomLabel.repaint();
84          showInfo();
85      }
```

Figure 11: Update Region

14. Create a public void method named “stop”, as well as another public method named “getGui()”

```
87      public void stop() {
88          t.stop();
89      }
90
91      public Component getGui() {
92          return gui;
93      }
```

Figure 12: Stop Method, getGui Method

15. Now, create a public void method named “showInfo”

```
95      public void showInfo() {
96          pi = MouseInfo.getPointerInfo();
97          output.setText("Zoom: " + zoomFactor + "  Point: " + pi.getLocation());
98      }
```

Figure 13: showInfo Method

16. Next, create the main method. This method will register the global screen listener and call the test method.

```
101     public static void main(String[] args){
102         try{
103             GlobalScreen.registerNativeHook();
104         } catch (NativeHookException ex){
105             System.exit( status: 1);
106         }
107         try {
108             new test();
109         } catch (AWTException e) {
110             e.printStackTrace();
111         }
112     }
```

Figure 14: Main Method

17. At this point, there should be an override public method named “nativeKeyPressed” that has been initialized. This method will be responsible for listening for the hotkey. Write in the following code.

```
@Override
public void nativeKeyPressed(NativeKeyEvent e) {

    if (e.getKeyCode() == NativeKeyEvent.VC_BACKQUOTE && on == 0) {
        zoomFactor = 2;
        frame.setDefaultCloseOperation(EXIT_ON_CLOSE);
        frame.setAlwaysOnTop(true);
        frame.setResizable(false);
        frame.add(getGui());
        frame.pack();
        frame.setAlwaysOnTop(true);
        frame.setVisible(true);

        WindowListener closeListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                frame.dispose();
            }
        };
        on = 1;
        frame.addWindowListener(closeListener);

    } else if (e.getKeyCode() == NativeKeyEvent.VC_BACKQUOTE && zoomFactor == 2 && on == 1) {
        zoomFactor = 4;
        on = 2;
    }else if (e.getKeyCode()==NativeKeyEvent.VC_BACKQUOTE && zoomFactor == 4 && on == 2){
        zoomFactor = 6;
        on =3;
    }
    else if (e.getKeyCode() == NativeKeyEvent.VC_BACKQUOTE && on == 3) {
        frame.setVisible(false);
        on = 0;

    }
}
```

Figure 15: nativeKeyPressed Method

18. Leave the “nativeKeyReleased” and “nativeKeyTyped” methods empty.
19. Run the code by pressing the green run button on the top right hand side of the screen.

3.2 Hardware Prototype

3.2.1 Bill Of Materials

Table 1: B.O.M

Material	**Cost**	**Where To Buy**
Arduino Leonardo	$24.91	[1]
Wire	~$0.05 (A very small amount was used)	[2]

USB Cord	$12.99	[3]
Resistor	$0.24	[4]
Button	$1.66	[5]
LED Light	$0.30	[6]
Circuit Board	$2.09	[7]
Button Casing	$5.05	[8]

3.2.2 Equipment List

Table 2: Equipment List

Equipment	**Use**
Soldering Iron	Solder Components to Circuit Board
Micro USB to USB Cable	Connect battery pack to NodeMCU
Super Glue	Glue Circuit Board to Plastic Case
Drill	Make Holes to Insert Internal Supports for Arduino, Button, USB Output
Screwdriver	Screw in Supports for Arduino Within the Box, Close Box using Machine Screws

3.2.3 Instructions To Build

The Button and LED light need to be attached to the Arduino Leonardo, in order to do this they will be placed onto a circuit board and then soldered to ensure for secure connections.

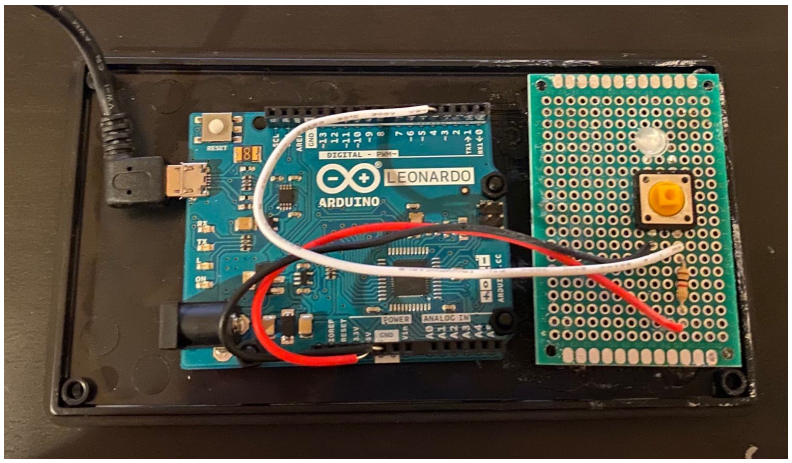

Figure 16: Circuitry

The resistor, Button and LED should be connected as seen in figure 1. Where the white wire is soldered to pin 4, the red wire is soldered to the 5V pin and the black wire is soldered to the GND pin. Once Soldered, the circuit board should be super glued to the four small stands located within the plastic cover. The micro USB should be plugged into the arduino leonardo and then fed through the button casing. The final result should resemble that of figure 2.

Figure 17: Final Button

4. How The Prototype Works

The java component of the prototype and the button component work together to create a seamless design. The Java code is programmed to open the magnification window if the backwards quote key on the keyboard is pressed. The Arduino Leonardo is coded using Arduino software to act as the backwards quote key. This means that everytime the button is pressed, the arduino leonardo outputs a backwards quote symbol. The java code is globally listening for the

backwards quote key to be pressed. If the key is pressed once, the magnification window will open and an integer within the code named “on” that was initially set to 0 will be set to 1. If the key is pressed again, the magnification factor will increase by two, and on will be set to 2. If the key is pressed again, the magnification factor is increased by two and on will be set to 3. If the key is pressed one last time, on will be set to 0 and the magnification window will be set to invisible.

5. Installation

This subsection will instruct the user on how to install the prototype. Maintenance on the prototype is not necessary.

5.1 Software Installation

The Java code for the magnifier can be created using the instructions found from section 3.2.1. However, the Java code can also be found and installed from the Accessabiltity-8 MakerRepo found at https://makerepo.com/ecox058/gng2101a8-magnifierplus.
Instruction to install the Java code are as follows;

1. Download IntelliJ IDEA from: https://www.jetbrains.com/idea/download/#section=linux
2. Go to https://makerepo.com/ecox058/gng2101a8-magnifierplus and download the zip file named “jframe.zip”
3. Open the src file from this zip file
4. Follow steps 5-8 from section 3.2.1
5. On the top right corner of the screen, click the green run arrow.

5.2 Button Installation

Once the button is made using the instructions found in section 3.2.3 and the Arduino Leonardo has been configured using instructions found in section 3.1.1, the usb simply needs to be plugged into the computer. When the USB is plugged in, the computer will recognize the button as an additional keyboard.

6. Health and Safety

The software component of magnifierPLUS in no way interacts with the mic or camera of the user, which ensures the safety and privacy of the user. Anyone who uses this product, does not have to worry about an invasion of privacy or data being recorded.

The hardware component consists of the button, the button box and its inner content. The internal wiring was safely done and follows basic rules of electrical circuits. Furthermore, the wiring was checked by a PM. There is minimal to no risk of electrocution as all the wiring is contained inside a box. An important safety feature of the box in relation to the pandemic is that it is sealed and made of material that can be easily sanitized. Therefore, should the computer lab open back up, the button and button box can be disinfected between uses to ensure no germs or viruses are spread.

7. Troubleshooting

The troubleshooting section is a table which contains some frequent or probable problems that the user may have. These problems correspond to a certain solution or course of action. The possible problems are categorized as being either: fixable by the user or contact manufacturer. If a problem can be fixed by the user the steps to do so are included. If the problem is technically complex, the problem can be fixed by contacting the manufacturer. In cases where the manufacturer must edit the software, a new file will be sent to you after the problem has been resolved.

Table 3: Troubleshooting

Potential Problem	Solution
"Application will not start when I double click it"	Do you have java installed on your computer? This is a requirement for running this application. The latest Java can be downloaded here: https://www.java.com/en/ Also assure you have followed all instructions in the installation section (5.1)

“Tactile button not lighting up”	Make sure the button’s USB cable is firmly connected to your computer. Otherwise, there could be a problem with your USB port. Try switching to another USB port
“Tactile button is lighting up but won’t launch the magnification window”	There could be an issue with the software. Please contact the manufacturer.
“The window is too small”	An adjustment to the code needs to be made. A trained software expert can change the size of the window in the java file. Please contact the manufacturer.
“The colour scheme is not right for me”	An adjustment to the code needs to be made. A trained software expert can change the colour scheme in the java file. Please contact the manufacturer.
“I need to use the hotkey for other purposes”	An adjustment to the code needs to be made. A trained software expert can change the hotkey to something more convenient for you. Please contact the manufacturer.

8. Recommendation For Future Work

While our solution, magnifierPLUS, is an effective solution to our project, it could be improved given increased budget and time allotment. For future work on the magnifier, further simplification of the installation process and fine tuning for users are the main goals. By creating a single downloadable package for our software, it could be more easily implemented as a consumer technology rather than as a bespoke solution as it currently exists. In the case where a single downloadable package was created, it could be added to the arduino box to automatically download when the button is plugged into a computer by the user. In addition, the button box

could be further refined to give improved feedback by putting the indicator LED on the outside of the box and/or adding a buzzer to give audible feedback when the button is pressed. In terms of software, the project could be refined to detect text boxes i.e. login screens requiring username and password and activate the magnifier automatically.

9. Conclusion

Our final solution offers a viable solution to the initial problem statement, the users of our solution will be able to navigate web pages, read text, and use social media with greater ease. For future improvements of our project, the magnification window size could be customizable on the fly. Additionally, the project could be further improved with an alteration to the magnifier code to view video through the magnifier as well as improve the fidelity of the image that the magnifier produces.

References

[1] [Online]. Available: https://www.digikey.ca/en/products/detail/arduino---bcmi-us-llc/A000052/6212657?utm_adgroup=Evaluation%20Boards%20-%20Embedded%20-%20MCU%2C%20DSP&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Development%20Boards%2C%20Kits%2C%20Programmer.

[2] [Online]. Available: https://www.amazon.ca/CBAZYTM-Stranded-Flexible-Silicone-Electric/dp/B075M7YZXC/ref=sr_1_6?dchild=1&keywords=electronics+wire&qid=1606843494&sr=8-6.

[3] [Online]. Available: https://www.amazon.ca/gp/product/B012S16PES/ref=ppx_yo_dt_b_asin_title_o02_s00?ie=UTF8&psc=1.

[4] [Online]. Available: https://www.digikey.ca/en/products/detail/vishay-beyschlag-draloric-bc-components/CCF5510K0FKE36/9642551.

[5] [Online]. Available: https://www.digikey.ca/en/products/detail/c-k/D6L90-F2-LFS/1531278.

[6] [Online]. Available: https://www.digikey.ca/en/products/detail/cree-inc/C503B-BCS-CV0Z0461/1922944?utm_adgroup=LED%20Indication%20-%20Discrete&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Optoelectronics&utm_term=&productid=1922944&gclid=CjwKCAiA8Jf-BRB-EiwA.

[7] [Online]. Available: https://www.digikey.com/en/products/detail/chip-quik-inc/SBBTH1510-1/5978273.

[8] [Online]. Available: https://www.digikey.ca/en/products/detail/omron-automation-and-safety/A22RL-TG-M/2834845 .

APPENDICES

APPENDIX A: Java Magnifier Code

```
import javax.swing.*;
import org.jnativehook.GlobalScreen;
import org.jnativehook.NativeHookException;
import org.jnativehook.keyboard.NativeKeyEvent;
import org.jnativehook.keyboard.NativeKeyListener;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;

public class test extends JFrame implements NativeKeyListener {
   Robot robot;
   JFrame frame;
   int on = 0;
   int zoomFactor =2;
   PointerInfo pi;
   JPanel gui;
   JLabel output;
   Timer t;


   public test() throws AWTException {

       frame = new JFrame("Simple GUI");
       robot = new Robot();
       gui = new JPanel(new BorderLayout(2, 2));
       output = new JLabel("point to see zoom");
       gui.add(output, BorderLayout.PAGE_END);
       final int size = 200;
       final BufferedImage bi = new BufferedImage(size, size,
BufferedImage.TYPE_INT_RGB);
       final JLabel zoomLabel = new JLabel(new ImageIcon(bi));
       gui.add(zoomLabel, BorderLayout.CENTER);


       ActionListener zooomListener = (ActionEvent e) -> {
           pi = MouseInfo.getPointerInfo();
           Point p = pi.getLocation();
           Rectangle r = new Rectangle(
                   p.x - (size / (2 * zoomFactor)),
                   p.y - (size / (2 * zoomFactor)),
                   (size / zoomFactor),
                   (size / zoomFactor));
           BufferedImage temp = robot.createScreenCapture(r);
           Graphics g = bi.getGraphics();
           g.drawImage(temp, 0, 0, size, size, null);
```

```
            g.setColor(new Color(255, 0, 0, 128));
            int x = (size / 2) - 1;
            int y = (size / 2) - 1;
            g.dispose();
            zoomLabel.repaint();
            showInfo();
        };
        t = new Timer(40, zooomListener);
        t.start();
        GlobalScreen.addNativeKeyListener(this);
        while (true) {
            Point p = MouseInfo.getPointerInfo().getLocation();
            frame.setLocation((p.x + 55), (p.y + 20));
            updateRegion();
        }

    }

    public void updateRegion()
    {
        final int size = 200;
        BufferedImage bi = new BufferedImage(
                size, size, BufferedImage.TYPE_INT_RGB);
        JLabel zoomLabel = new JLabel(new ImageIcon(bi));
        pi = MouseInfo.getPointerInfo();
        Point p = pi.getLocation();
        Rectangle r = new Rectangle(
                p.x - (size / (2 * zoomFactor)),
                p.y - (size / (2 * zoomFactor)),
                 (size / zoomFactor),
                 (size / zoomFactor));
        BufferedImage temp = robot.createScreenCapture(r);
        Graphics g = bi.getGraphics();
        g.drawImage(temp, 0, 0, size, size, null);
        g.setColor(new Color(255,0,0,128));
        int x = (size/2)-1;
        int y = (size/2)-1;
        g.dispose();
        zoomLabel.repaint();
        showInfo();
    }

    public void stop() {
        t.stop();
    }

    public Component getGui() {
        return gui;
    }
```

```
    public void showInfo() {
        pi = MouseInfo.getPointerInfo();
        output.setText("Zoom: " + zoomFactor + "  Point: " + pi.getLocation());
    }


    public static void main(String[] args){
        try{
            GlobalScreen.registerNativeHook();
        } catch (NativeHookException ex){
            System.exit(1);
        }
        try {
            new test();
        } catch (AWTException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void nativeKeyPressed(NativeKeyEvent e) {

        if (e.getKeyCode() == NativeKeyEvent.VC_BACKQUOTE && on == 0) {
            zoomFactor = 2;
            frame.setDefaultCloseOperation(EXIT_ON_CLOSE);
            frame.setAlwaysOnTop(true);
            frame.setResizable(false);
            frame.add(getGui());
            frame.pack();
            frame.setAlwaysOnTop(true);
            frame.setVisible(true);

            WindowListener closeListener = new WindowAdapter() {
                @Override
                public void windowClosing(WindowEvent e) {
                    frame.dispose();
                }
            };
            on = 1;
            frame.addWindowListener(closeListener);

        } else if (e.getKeyCode() == NativeKeyEvent.VC_BACKQUOTE && zoomFactor
== 2 && on == 1) {
            zoomFactor = 4;
            on = 2;
        }else if (e.getKeyCode()==NativeKeyEvent.VC_BACKQUOTE && zoomFactor ==
4 && on == 2){
            zoomFactor = 6;
```

```
            on =3;
        }
        else if (e.getKeyCode() == NativeKeyEvent.VC_BACKQUOTE && on == 3) {
            frame.setVisible(false);
            on = 0;

        }
    }

    @Override
    public void nativeKeyReleased(NativeKeyEvent nativeKeyEvent) {

    }

    @Override
    public void nativeKeyTyped(NativeKeyEvent nativeKeyEvent) {

    }
}
```

APPENDIX B: Arduino Leonardo Hot-Key Code

```
/*
  Keyboard Message test

  For the Arduino Leonardo and Micro.

  Sends a text string when a button is pressed.

  The circuit:
  - pushbutton attached from pin 4 to +5V
  - 10 kilohm resistor attached from pin 4 to ground

  created 24 Oct 2011
  modified 27 Mar 2012
  by Tom Igoe
  modified 11 Nov 2013
  by Scott Fitzgerald

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/KeyboardMessage
*/

#include "Keyboard.h"
```

```
const int buttonPin = 4;          // input pin for pushbutton
int previousButtonState = HIGH;   // for checking the state of a pushButton
int counter = 0;                  // button push counter

void setup() {
  // make the pushButton pin an input:
  pinMode(buttonPin, INPUT);
  // initialize control over the keyboard:
  Keyboard.begin();
}

void loop() {
  // read the pushbutton:
  int buttonState = digitalRead(buttonPin);
  // if the button state has changed,
  if ((buttonState != previousButtonState)
      // and it's currently pressed:
      && (buttonState == HIGH)) {
    // increment the button counter
    counter++;
    // type out a message
    Keyboard.print("`");
    //Keyboard.print(counter);
    //Keyboard.println("");
  }
  // save the current button state for comparison next time:
  previousButtonState = buttonState;
}
```