HCI-Adherent Graphical User Interface for a Traffic System

An honors contract executed by

Ryan Lane

with oversight from Dr. Benjamin Ruddell.

EGR 102

Introduction to Engineering Design II

Spring 2011

Department of Engineering
Arizona State University at the Polytechnic campus

Executive Summary
This report communicates the produced solution to an honors contract for EGR 102. The project is to design and develop a Python-based GUI that follows Human-Computer Interface best practices which interfaces with a hardware traffic system. Human-Computer Interface design is briefly summarized and its implementation in the solution discussed. The final GUI design, developed through the use of wire-frames (visual design), use narratives (interaction design), and object-oriented layout (implementation design), controlled at least one and up to five different intersections simultaneously (the particular intersections activated and chosen by the user). Through the implementation of various HCI design principles, the final GUI proved to be a highly usable solution with low-user friction. Designing future GUIs according to HCI principles is highly recommended.

Table of Contents

List of Figures

Introduction

Intersection traffic systems, ubiquitous necessities for directing vehicular traffic in cities, also serve as prime engineering exercises. From their incredible usefulness, to the varied human factors that affect their performance, to issues concerning their optimization, intersection systems pose important design questions that promise meaningful and useful solutions.

Because of the crucial role traffic systems play, a human user must ultimately be able to monitor and directly control them. This necessitates an interface through which the user interacts with the system. Since the advent of computers, the way human users interact with them has been an important, but often undervalued, consideration in system design. If the goal of technology is to the enhance and ease the life of people, then the friction introduced by computerized technology must be absolutely minimized. Fortunately, the now-thriving field of Human-Computer Interaction asserts guiding design principles developed through a substantial body of research. These HCI principles allow interface developers to coherently design interfaces that drastically reduce user friction and connect the user with desired functionality in a logical and effective manner.

The object of this report concerns the development of an HCI-adherent user interface for a five intersection traffic system—an extension of the central semester project for the EGR 102 class.

Project Definition

The project is to design and develop a Python-based GUI that follows Human-Computer Interface best practices which interfaces with a hardware traffic system. The GUI should show the states of the intersections present in the traffic system and allow the user to manually change the state of each intersection and also switch the system to operate according to an automatic algorithm.

<u>Criteria</u>

• The GUI should be highly user-friendly and closely follow defined HCI design guidelines
• The GUI should match the states of the hardware with as little update delay as possible

<u>Constraints</u>

• The GUI should be deployable for any number of intersections (up to 5) within the same Python program.
• The GUI should allow the user to manually change the state of each intersection
• Each intersection in the GUI should be implemented using Finite State logic
• Each intersection should be able to handle either 4 or 3 lights, depending on their location and the isec module's particular capacity for the intersection.
• The GUI and report is to be submitted by the end of the semester.

Methods
The HCI guidelines used in the development of this user interface consist of the following principles and concrete suggestions:

8 Golden Principles of Interface Design (Shneiderman 74-75)
1) Strive for consistency.
2) Cater to universal usability.
3) Offer informative feedback
4) Design dialogs to yield closure.
5) Prevent errors.
6) Permit easy reversal of actions.

HCI Concrete Suggestions
• Standardize task sequences (62).
• Use checkboxes for binary choices (62).
• Standardize colors, terminology, and icons during the design process (63).
• Use only two levels of contrast intensity (64).
• Use up to three different fonts (64).
• Use up to four standard colors (64).
• Keep data-entry transactions consistent (65).
• Minimize input actions by the user (65).

Concept Generation
Concepts were generated by wire-framing initial ideas on Adobe's iOS app *Ideas* (**Figures 1-3**) and crafting a user narrative to draw out specific use cases. Both approaches were relied on heavily before any coding began, helping to ensure that ideas about implementation would not hamper the design of a solution that fully met the user's needs.

User Narrative
Michelle sits down at the computer and launches the GUI, being greeted by a configuration screen. She chooses how many intersections she wants in the traffic system and where they go. The center intersection is already in place and activated as the 1st intersection. The other numbers of the intersection are interchangeable. The center intersection is a four-way intersection. The other intersections are three-way.

Michelle then hits the "Activate Traffic System" button. The configuration screen melts away into an active traffic system, live with sensor data and fully functional. Any intersections that she did not activate are visibly inactive.

Michelle changes the lights for the intersections when he sees that a sensor tells her a car is there. She gets up for coffee and switches the program into Automatic mode. Immediately, the "change intersection" icon for each intersection fades and the intersections operate according to an algorithm that biases toward keeping the outer thoroughfares clear. Michelle comes with her coffee, feeling energized and ready to once again take the reigns—she switches the system back into Manual mode and continues running the lights. When all the cars are gone and Michelle is finished, she hits the "Quit" button.

Wire-frames

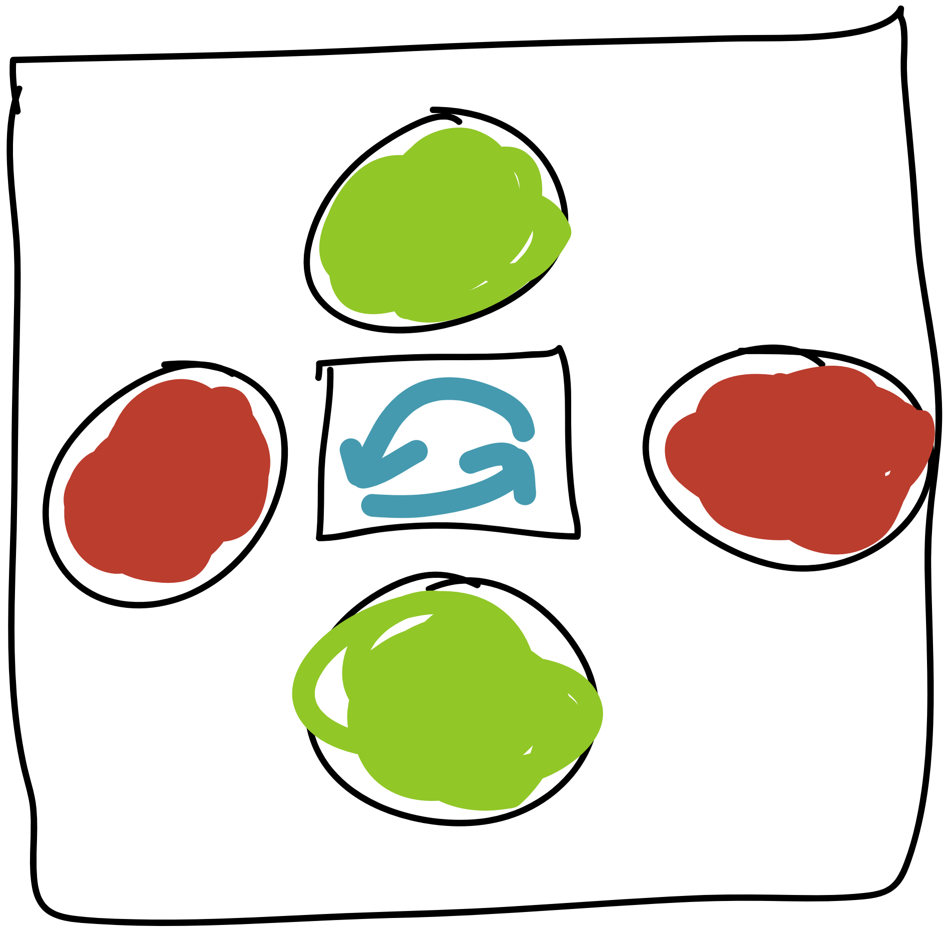

Figure 1
Wire-frame of an intersection.

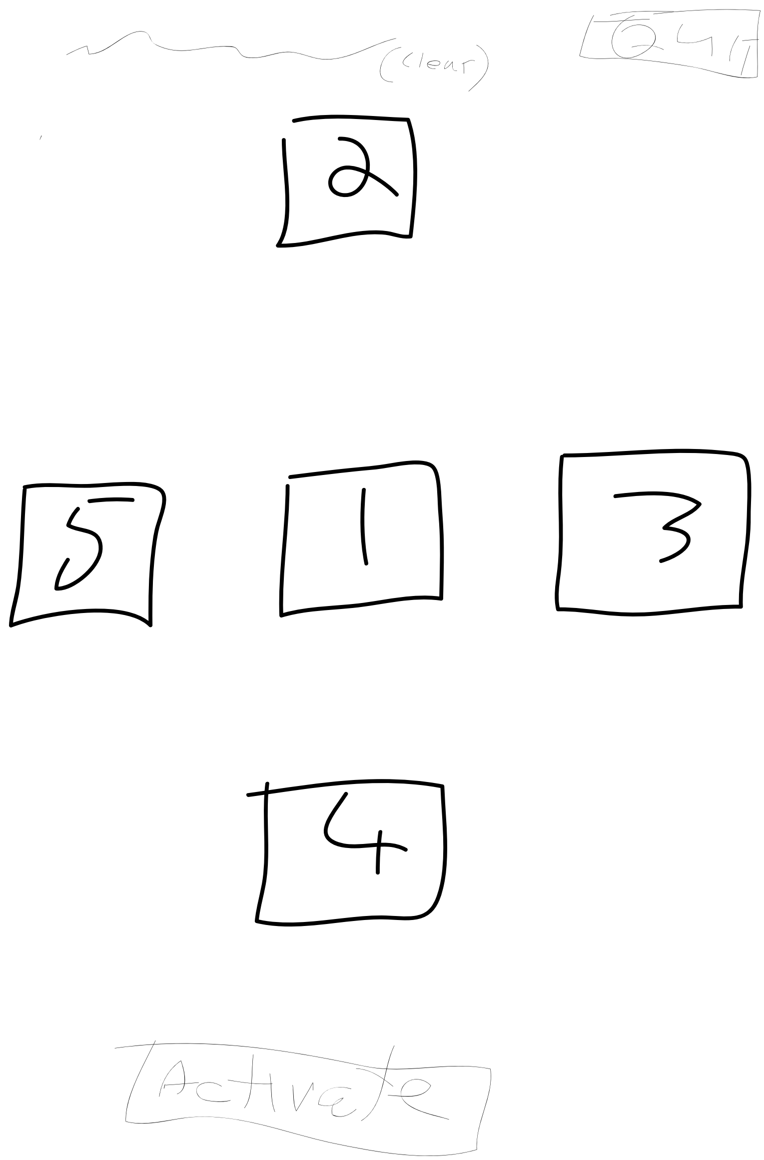

Figure 2
Wire-frame of the launch screen.

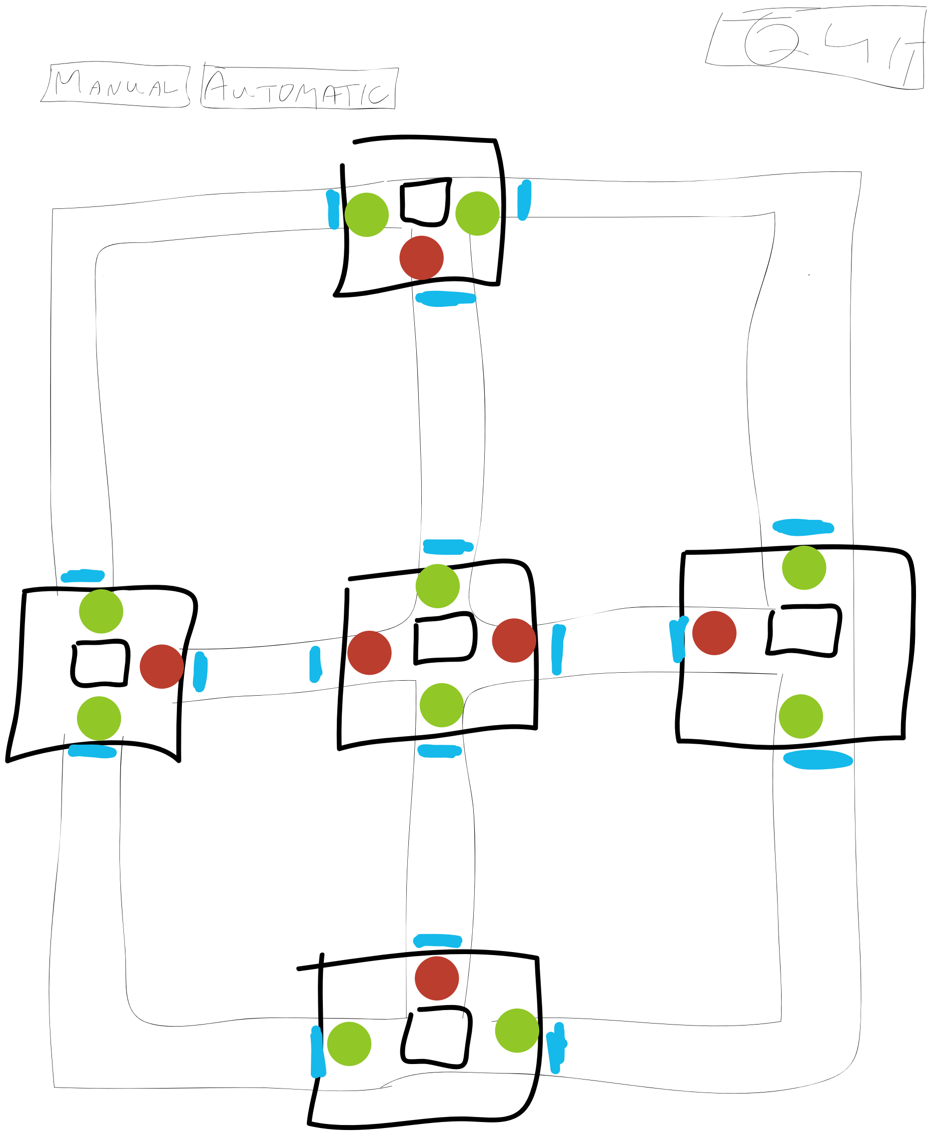

Figure 3
Wire-frame of the main screen.

Selection of Concepts

One of the ideas generated during the initial concept brainstorming was to extend the GUI up to 9 different intersections. However, after drafting a wire-frame of the main screen, it became immediately apparent that the given hardware system couldn't support a 9 intersection arrangement. Given the fact that a light-based intersection is only useful if there are two conflicting traffic directions (resulting in a minimum of three traffic lights per intersection), the hardware would only be able to support an additional four intersection (as shown in **Figure 4**) if the center intersection could support 8 distinct lanes of traffic. Since the hardware was not configured in this way, a 5 intersection configuration was selected as the GUI concept to be implemented.

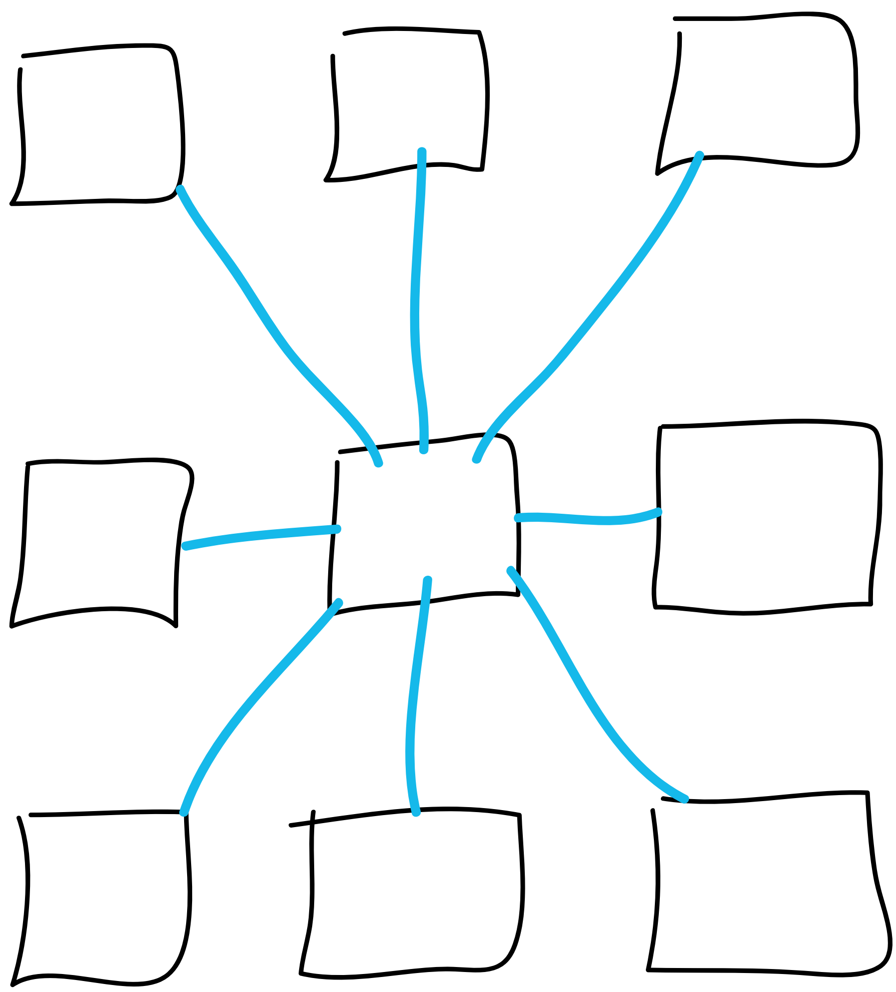

Figure 4
Rejected 9 Intersection GUI Concept

Results

<u>Programmatic Structure</u>

The final structure of the code was composed of three central classes:

• Light
• Intersection
• TrafficGUI

Each of those classes are instantiated by the class that follows. The source code itself may be found in <u>Appendix 2</u>**.**

<u>Finite State Machine Implementation</u>

The FSM used in the intersection design and the automatic algorithm applied to each one is the same developed by Team 7 in the 10AM EGR 102 section (this author's team). More details are covered in that team's report, but the basic FSM transition table (**Figure 7**) and state diagram (**Figure 6**) are found in <u>Appendix 1</u>. In addition to the light timings presented in the FSM documentation, it is important to note that the final design contained a 0.5 second delay after

<u>HCI Adherence</u>

The following are the HCI principles and suggestions stated at the beginning of the report and how they were implemented in the final design (**Figure 5 and Figure 6**).

<u>8 Golden Principles of Interface Design</u>

1) *Strive for consistency*.
    - All dynamic GUI components (i.e. lights and intersections) were designed as software objects, maintaining both logic and UI consistency.
2) *Cater to universal usability*.
    - Skeuomorphic road representations clarified the GUI's purpose and function without relaying on excessive textual instruction.
3) *Offer informative feedback.*
    - Real-time light sensors provided necessary system information in an incredibly accessible and comprehensible form.
4) *Design dialogs to yield closure*.
    - While no error dialogs were present in the final implementation, all of the program's communication with the user appears in plain English.
5) *Prevent errors*.
    - Many aspects of the GUI could potentially be customized (such as color of intersections and roads), but available choices were kept to an absolute minimum in order to reduce program complexity and minimize errors. Designing the architecture according to object-oriented methodology solidified this aim as well.
6) *Permit easy reversal of actions*.
    - The initial launch screen is the only configuration screen the user must interact with before using the GUI's functionality, and the intersection configurations there are easily undone and redone.

HCI Concrete Suggestions

- *Standardize task sequences.*
    - Changing the light states of individual intersections while in “Manual” mode is consistent across the GUI.
- *Use checkboxes for binary choices.*
    - Checkboxes (technically radio buttons in this case) were used to make the binary switch between “Manual” and “Automatic” modes.
- *Standardize colors, terminology, and icons during the design process.*
    - GUI light colors represent the colors of the hardware lights which themselves represent the colors of actual streetlights. This recognizable consistency simplified the GUI and places less burden on the user in regard so comprehending the design.
- *Use only two levels of contrast intensity.*
    - In the launch screen, there are two distinct levels of contrast for the intersection squares: light gray (same as the background) with a dashed outline for unselected squares, and a sharp black text-on-white scheme for selected ones.
- *Use up to three different fonts.*
    - Helvetica is the only font used in the GUI (and this report, coincidentally).
- *Use up to four standard colors.*
    - This HCI suggestion was graciously put aside, as a rich representation of a traffic system warranted more than four standard colors, especially considering that the lights inherently require three.
- *Keep data-entry transactions consistent.*
    - There is only one way to do any of the actions defined by the GUI: click on squares to initiate their intersections, click on the centers of intersections to change their state while in “Manual” mode, and select the appropriate radio button to switch between “Manual” and “Automatic.”
- *Minimize input actions by the user.*
    - The only user input recognized by the GUI entails the initial intersection selection, manual intersection state change, and “Manual”/”Automatic” mode toggling.

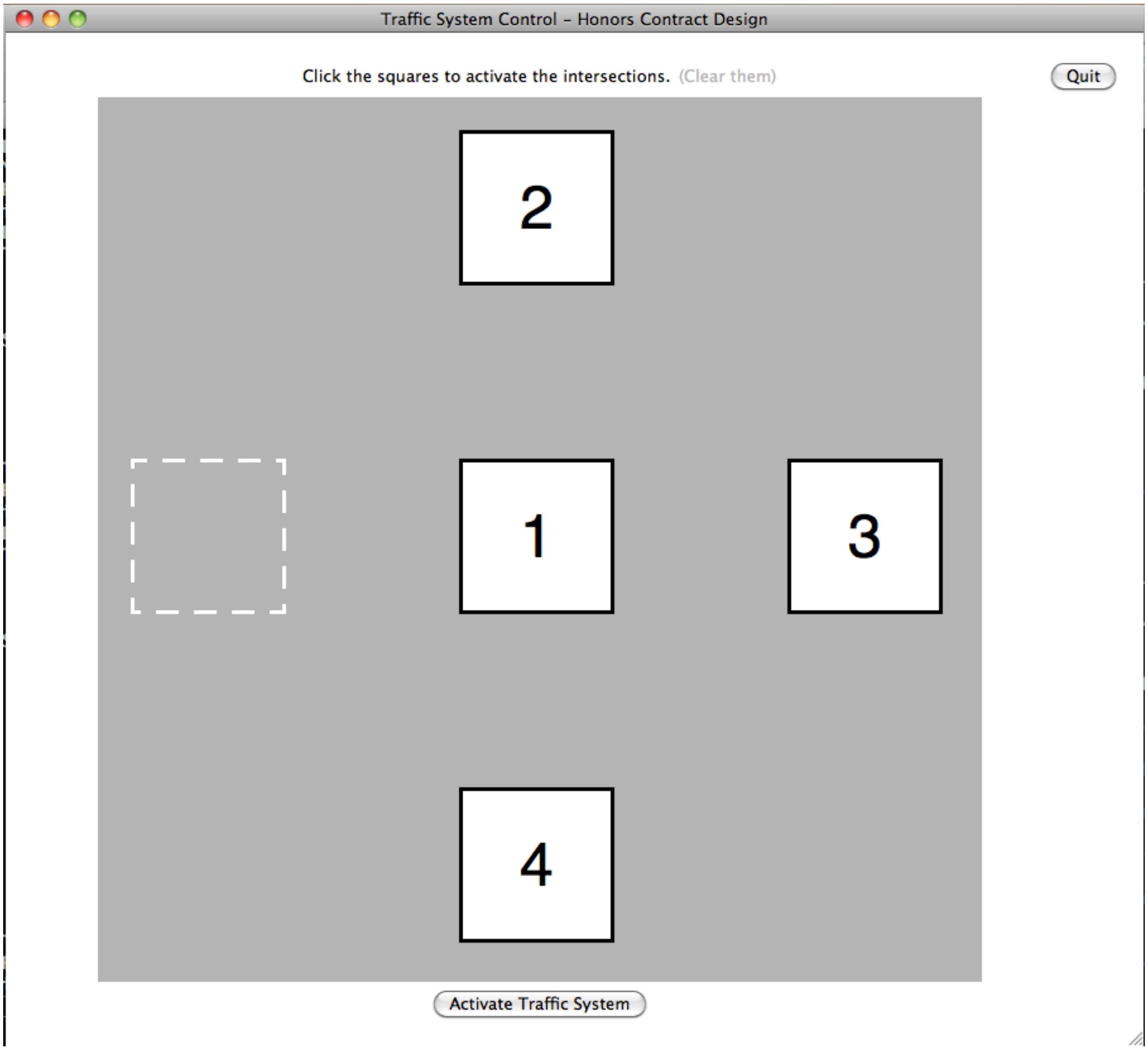

Figure 5
The launch screen of the final design.

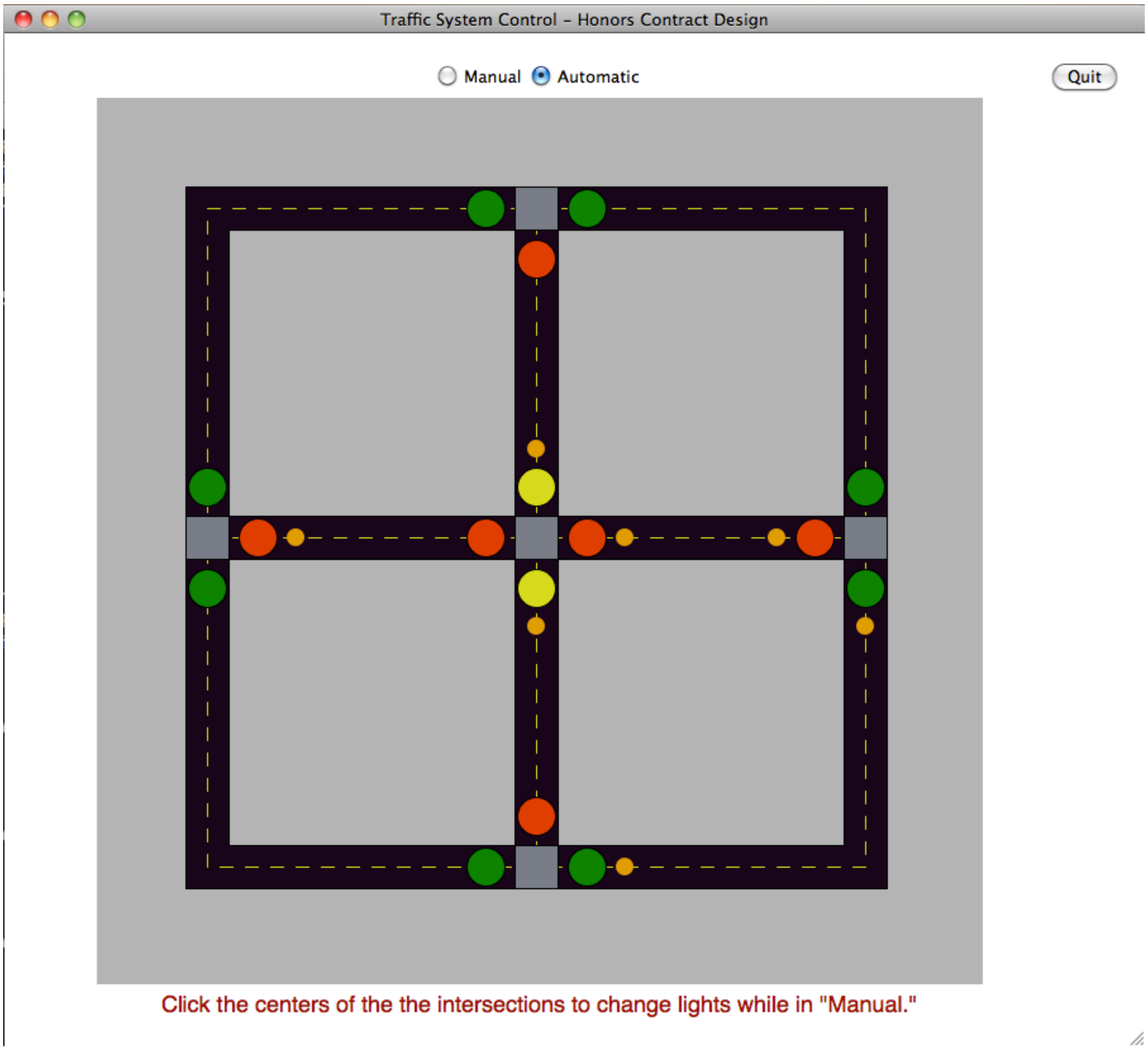

Figure 6
The main screen of the final design.

Criteria and Constraints Success

The final GUI design successfully fulfilled all defined criteria and constraints. Both criteria of the GUI being highly user-friendly and have minimized hardware updating were achieved via high HCI-compliance and an average update time of 0.0625 sec. All constraints, as defined in the Problem Definition at the beginning of the report, were successfully met.

Conclusion

The final design solution met the primary project goals of building an HCI-directed interface for the hardware traffic system in EGR 102. As the content of this report attests to, designing interfaces with a primary focus on the user and a secondary focus on implementation details yields highly usable results—most importantly with low friction experienced by the user. Designing interfaces in this manner is highly recommended for future EGR 102 GUI projects.

Acknowledgements
Many thanks to the following individuals: Dr. Benjamin Ruddell, Ramón Anguamea Lara, Richard Whitehouse, Dr. Darryl Morrell, Dr. Chen-Yuan Kuo, Guoxin Li, Sumit Nair, Chrissy Wicklund, and Johnathan Barone. This project could not have been pursued without your gracious guidance and hard work.

References

Shneiderman, Ben, and Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-computer Interaction*. Boston, Mass.: Pearson, 2005. Print.

Appendices

Appendix 1: Finite State Diagram and Transition Table for Single-Intersection Algorithm

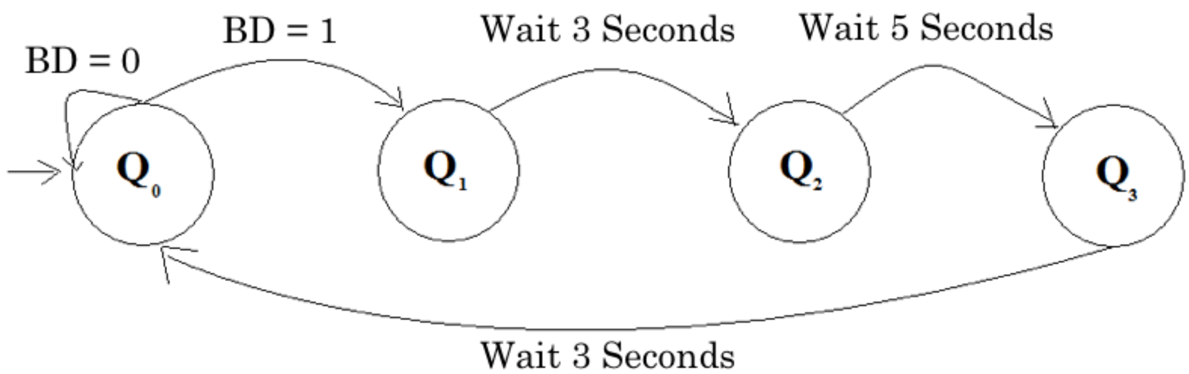

Figure 7
The state diagram for the single-intersection algorithm.

STATE TRANSITION TABLE

*** = Input does not initiate a state change**

STATE INFORMATION		INPUTS			
Cur_state	Light Value	BD Sensor 1	BD Sensor 0	Time.sleep(3)	Time.sleep (5)
Q_0	AC-G,BD-R	Q_1	Q_0	*	*
Q_1	AC-Y,BD-R	*	*	Q_2	*
Q_2	AC-R,BD-G	*	*	*	Q_3
Q_3	AC-R,BD-Y	*	*	Q_0	*

Figure 8
The state transition table for the single-intersection algorithm.

Appendix 2: Source Code

```
# This GUI has been designed and developed by Ryan Lane. The algorithm for
the Intersection.Automatic() method was developed by Ryan Lane, Sumit Nair,
Johnathan Barone, and Guoxin Li.


from Tkinter import *
import time
import isec

color_bank = {"green":'#067302',"yellow":'#D2D914',"red":'#D92B04', "sensor":
'#D98E04',
'switch':'#686E75','switch_hover':'#9BAAC1','streets':'#150517',"background":
'dark grey'}

class Light():
    def __init__
(self,Intersection,name,sensor_orientation,intersection_center_x,intersection
_center_y):
        self.root = Intersection.root
        self.intersection_number = Intersection.intersection_number
        self.Canvas = Intersection.Canvas
        self.light_name = name
        self.sensor_orientation = sensor_orientation
        if sensor_orientation == "up":
            center_x = intersection_center_x
            sensor_center_x = intersection_center_x
            center_y = intersection_center_y - 40
            sensor_center_y = intersection_center_y - 70
        elif sensor_orientation == "down":
            center_x = intersection_center_x
            sensor_center_x = intersection_center_x
            center_y = intersection_center_y + 40
            sensor_center_y = intersection_center_y + 70
        elif sensor_orientation == "left":
            center_x = intersection_center_x - 40
            sensor_center_x = intersection_center_x - 70
            center_y = intersection_center_y
            sensor_center_y = intersection_center_y
        elif sensor_orientation == "right":
            center_x = intersection_center_x + 40
            sensor_center_x = intersection_center_x + 70
            center_y = intersection_center_y
            sensor_center_y = intersection_center_y
        self.center = {"x":center_x,"y":center_y}
        self.sensor_center = {"x":sensor_center_x,"y":sensor_center_y}
        self.drawLight()
        self.drawSensor()

    def drawLight(self):
        x = self.center["x"]
        y = self.center["y"]
```

```
        self.gui = self.Canvas.create_oval(x-15,y-15,x+15,y+15, fill="",
outline="black")

    def drawSensor(self):
        x = self.sensor_center["x"]
        y = self.sensor_center["y"]
        self.sensor = self.Canvas.create_oval(x-7,y-7,x+7,y+7, fill="",
outline="")

    def turnLight(self,switch,color):
        color_letter = None
        if color == "green":
            color_letter = 'G'
        elif color == "yellow":
            color_letter = 'Y'
        elif color == "red":
            color_letter = 'R'
        if switch == "on":
            self.Canvas.itemconfig(self.gui,fill=color_bank[color])
            if self.light_name != "C" and self.light_name != "D":
                isec.light_on(self.intersection_number,self.light_name
+color_letter)
        elif switch == "off":
            if self.light_name != "C" and self.light_name != "D":
                isec.light_off(self.intersection_number,self.light_name
+color_letter)
        self.root.update()

    def turnSensor(self,switch):
        if switch == 'on':
            self.Canvas.itemconfig(self.sensor,fill=color_bank["sensor"])
        elif switch == 'off':
            self.Canvas.itemconfig(self.sensor,fill="")

    def checkSensor(self):
        result = None
        if isec.sense(self.intersection_number,self.light_name+'1') == True:
            self.turnSensor('on')
            result = True
        elif isec.sense(self.intersection_number,self.light_name+'1') ==
False:
            self.turnSensor('off')
            result = False
        return result

class Intersection():
    def __init__(self,root,parent_canvas,square,intersection_number):
        self.root = root
        self.Canvas = parent_canvas
        self.square = square
        self.intersection_number = intersection_number
        self.Lights = {}

    def buildIntersection(self):
```

```
        light_orientations = {"A":self.square
["orientation"],"B":None,"C":None,"D":None}
        x = self.square["x"]
        y = self.square["y"]
        if light_orientations["A"] == 'up':
            light_orientations["B"] = 'right'
            light_orientations["C"] = 'down'
            light_orientations["D"] = 'left'
        elif light_orientations["A"] == 'down':
            light_orientations["B"] = 'left'
            light_orientations["C"] = 'up'
            light_orientations["D"] = 'right'
        elif light_orientations["A"] == 'left':
            light_orientations["B"] = 'up'
            light_orientations["C"] = 'right'
            light_orientations["D"] = 'down'
        elif light_orientations["A"] == 'right':
            light_orientations["B"] = 'down'
            light_orientations["C"] = 'left'
            light_orientations["D"] = 'up'
        for i in light_orientations:
            if i != "D" or self.square["number_of_lights"] == 4:
                self.Lights[i] = Light(self,i,light_orientations[i],x,y)
        self.switch_gui = self.Canvas.create_rectangle(x-17,y-17,x+17,y
+17,fill=color_bank["switch"], outline = "black")
        self.Canvas.tag_bind(self.switch_gui, "<Button-1>", lambda dummy:
self.bumpState())
        self.Canvas.tag_bind(self.switch_gui, "<Enter>", lambda dummy:
self.Canvas.itemconfig(self.switch_gui,fill=color_bank["switch_hover"]))
        self.Canvas.tag_bind(self.switch_gui, "<Leave>", lambda dummy:
self.Canvas.itemconfig(self.switch_gui,fill=color_bank["switch"]))

        self.currentState = 0
        self.moveGUIToState(self.currentState)

    def bumpState(self):
        if self.currentState != 3:
            self.currentState += 1
        else:
            self.currentState = 0
        self.moveGUIToState(self.currentState)

    def moveGUIToState(self,newState):
        if newState == 0:
            self.Lights["B"].turnLight('off','yellow')
            self.Lights["B"].turnLight('on','red')
            if self.square["number_of_lights"] == 4:
                self.Lights["D"].turnLight('off','yellow')
                self.Lights["D"].turnLight('on','red')
            time.sleep(0.75)
            self.Lights["A"].turnLight('off','red')
            self.Lights["A"].turnLight('on','green')
            self.Lights["C"].turnLight('off','red')
            self.Lights["C"].turnLight('on','green')
```

```
        elif newState == 1:
            self.Lights["A"].turnLight('off','green')
            self.Lights["A"].turnLight('on','yellow')
            self.Lights["C"].turnLight('off','green')
            self.Lights["C"].turnLight('on','yellow')
        if newState == 2:
            self.Lights["A"].turnLight('off','yellow')
            self.Lights["A"].turnLight('on','red')
            self.Lights["C"].turnLight('off','yellow')
            self.Lights["C"].turnLight('on','red')
            time.sleep(0.75)
            self.Lights["B"].turnLight('off','red')
            self.Lights["B"].turnLight('on','green')
            if self.square["number_of_lights"] == 4:
                self.Lights["D"].turnLight('off','red')
                self.Lights["D"].turnLight('on','green')
        elif newState == 3:
            self.Lights["B"].turnLight('off','green')
            self.Lights["B"].turnLight('on','yellow')
            if self.square["number_of_lights"] == 4:
                self.Lights["D"].turnLight('off','green')
                self.Lights["D"].turnLight('on','yellow')

    def checkSensors(self):
        call_for_light_change = False
        wait_a_bit = False
        for i in self.Lights:
            result = self.Lights[i].checkSensor()
            if (i == "A" or i == "C") and result == True:
                wait_a_bit = True
            elif (i == "B" or i == "D") and result == True:
                call_for_light_change = True
        return [call_for_light_change,wait_a_bit]

    def smart_sleep(self,wait_time):
    # This method checks the auto/manual state so as to quicken the program's
response time to user input.
        for i in range(wait_time*2):
            if self.currentMode == "auto":
                time.sleep(0.5)
            else:
                break

    def Manual(self):
        self.currentMode = "manual"

    def Automatic(self):
        self.currentMode = "automatic"
        if self.currentState == 1 or self.currentState == 3:
                self.ChangeLights(1,1.5)
        if self.currentMode == "automatic":
            sensor_status = self.checkSensors()
            if self.currentState == 0 and sensor_status[0] == True:
                  if sensor_status[1] == True:
```

```
                    self.smart_sleep(3)
                self.ChangeLights(2,1.5)
            if self.currentState == 2:
                self.root.update() # This keeps the Automatic/Manual button
from remaining in the depressed state during time.sleep().
                self.smart_sleep(3)

                self.ChangeLights(2,1.5)
                time.sleep(1)

    def ChangeLights(self,repeats,wait_time):
        self.checkSensors()
        # if AC is Green, then go through the light changes to end up with
BD-Green
        if self.currentState == 0:
           self.bumpState()
        elif self.currentState == 1:
           self.bumpState()
           time.sleep(0.5)
        # if AC is Green, then go through the light changes to end up with
BD-Green
        elif self.currentState == 2:
           self.bumpState()
        elif self.currentState == 3:
           self.bumpState()
           time.sleep(0.5)
        isec.print_lights()
        self.root.update()
        time.sleep(wait_time)
        if repeats != 1:
            self.ChangeLights(repeats-1,wait_time)

class TrafficGUI(Frame):

    def __init__(self,master=None):
        Frame.__init__(self,master)
        self.master.title("Traffic System Control - Honors Contract Design")
        self.grid(padx=20,pady=20)
        self.resetIntersections()
        self.createLaunchScreen()

        isec.simulate_hardware = True
        isec.hw_init()
        #isec.hw_init(simulate_hardware = False)

    def createLaunchScreen(self):
        self.Quit = Button(self, text="Quit", command=self.destroy)
        self.Quit.grid(row=0,column=2,sticky=E)
        self.TopText = Label(self, text="Click the squares to activate the
intersections.")
        self.TopText.grid(row=0,column=0,sticky=E)
        self.Clear = Label(self, text="(Clear them)",fg="dark gray")
        self.Clear.bind("<Enter>",lambda dummy:self.Clear.config(fg="dark
red"))
```

```
        self.Clear.bind("<Leave>",lambda dummy:self.Clear.config(fg="dark
gray"))
        self.Clear.bind("<Button-1>",lambda dummy:self.ClearSquares())
        self.Clear.grid(row=0,column=1,sticky=W)

        self.Activate = Button(self, text="Activate Traffic System",
command=self.ActivateTrafficSystem)
        self.Activate.grid(row=2,columnspan=2)

        self.Canvas = Canvas(self, width=700,height=700,background=color_bank
["background"])
        self.Canvas.grid(row=1,columnspan=2,padx=50)

        self.generateSquareStructures()
        self.loadSquareGUIs(reset=False)

    def generateSquareStructures(self):
        # This function populates Square Data
        centers_x = [350,90,350,610,350]
        centers_y = [90,350,350,350,610]
        orientations = ['right','up','up','down','left']
        number_of_lights = [3,3,4,3,3]
        self.squares = []
        for i in range(len(centers_x)):
            square_info = {"id":i,"x":centers_x[i],"y":centers_y
[i],"orientation":orientations[i],"number_of_lights":number_of_lights
[i],"gui":None,"label":None,"label_window":None}
            self.squares.append(square_info)

    def loadSquareGUIs(self,reset):
        for i, square in enumerate(self.squares):
            if reset == False:
                 x = square["x"]
                 y = square["y"]
                 square["gui"] = self.Canvas.create_rectangle(x-60,y-60,x+60,y
+60,fill=color_bank["background"],dash=15, dashoffset=5, outline ="white",
width=3)
            else:
                 self.Canvas.itemconfig(square["gui"], fill="dark gray",
dash=15, dashoffset=5, outline="white")
                 self.Canvas.delete(square["label_window"])
            if i == 0:
                self.Canvas.tag_bind(square["gui"], "<Button-1>", lambda
dummy: self.registerIntersection(self.squares[0]))
            elif i == 1:
                self.Canvas.tag_bind(square["gui"], "<Button-1>", lambda
dummy: self.registerIntersection(self.squares[1]))
            elif i == 2:
                self.registerIntersection(self.squares[2])
            elif i == 3:
                self.Canvas.tag_bind(square["gui"], "<Button-1>", lambda
dummy: self.registerIntersection(self.squares[3]))
            elif i == 4:
```

```
                self.Canvas.tag_bind(square["gui"], "<Button-1>", lambda
dummy: self.registerIntersection(self.squares[4]))

    def ClearSquares(self):
        self.resetIntersections()
        self.loadSquareGUIs(reset=True)

    def registerIntersection(self,square):
        if square["id"] != 2:
            intersection_number = self.intersection_queue.next()
            self.Canvas.tag_unbind(square["gui"], "<Button-1>")
        else:
            intersection_number = 1
        self.Intersections[intersection_number] = Intersection
(self,self.Canvas,square,intersection_number)

        self.Canvas.itemconfig(square["gui"], fill="white", outline="black",
dash=1)

        self.squares[square["id"]]["label"] = Label(self, text=str
(intersection_number),font=('Helvetica',48),bg="white")
        self.squares[square["id"]]["label_window"] =
self.Canvas.create_window(square["x"],square["y"],window=square["label"])

    def resetIntersections(self):
        intersection_range = range(2,6)
        self.intersection_queue = iter(intersection_range)
        self.Intersections = {}

    def drawRoads(self):
        self.Canvas.create_rectangle(73,73,627,627,fill=color_bank
["streets"],outline="black")

        self.Canvas.create_rectangle(107,107,333,333,fill=color_bank
["background"],outline="black")
        self.Canvas.create_rectangle(367,107,593,333,fill=color_bank
["background"],outline="black")
        self.Canvas.create_rectangle(107,367,333,593,fill=color_bank
["background"],outline="black")
        self.Canvas.create_rectangle(367,367,593,593,fill=color_bank
["background"],outline="black")


        self.Canvas.create_rectangle(90,90,610,610,fill="",outline=color_bank
["yellow"],dash=10)
        self.Canvas.create_rectangle
(350,90,610,350,fill="",outline=color_bank["yellow"],dash=10)
        self.Canvas.create_rectangle
(90,350,350,610,fill="",outline=color_bank["yellow"],dash=10)

        self.update()

    def ActivateTrafficSystem(self):
        for square in self.squares:
```

```
            self.Canvas.delete(square['gui'])
            self.Canvas.delete(square['label_window'])
        self.Activate.grid_remove()
        self.TopText.grid_remove()
        self.Clear.grid_remove()
        self.BottomText = Label(self, text="Click the centers of the the
intersections to change lights while in \"Manual.\"",fg="dark red",font=
('Helvetica',18))
        self.BottomText.grid(row=2,columnspan=2)

        self.mode_var = IntVar()
        self.ManualButton = Radiobutton(self, text="Manual",
variable=self.mode_var, value="manual", command=self.toggleMode)
        self.ManualButton.select()
        self.ManualButton.grid(row=0,column=0,sticky=E)
        self.AutomaticButton = Radiobutton(self, text="Automatic",
variable=self.mode_var, value="automatic", command=self.toggleMode)
        self.AutomaticButton.grid(row=0,column=1,sticky=W)
        self.drawRoads()
        for i in self.Intersections:
            self.Intersections[i].buildIntersection()
        self.mode = "manual"
        self.update()
        self.Manual()

    def toggleMode(self):
        if self.mode == "manual":
            self.mode = "automatic"
            self.Automatic()
        elif self.mode == "automatic":
            self.mode = "manual"
            self.Manual()

    def Manual(self):
        for i in self.Intersections:
            self.Intersections[i].Manual()
        while self.mode == "manual":
            for i in self.Intersections:
                self.Intersections[i].checkSensors()
                self.update()
            for i in range(8):
                time.sleep(0.0625)
                self.update()

    def Automatic(self):
        while self.mode == "automatic":
            self.update()
            for i in self.Intersections:
                self.Intersections[i].checkSensors()
                self.Intersections[i].Automatic()
            for i in range(8):
                time.sleep(0.0625)
                self.update()
```

```
if __name__ == "__main__":
    TrafficApp = TrafficGUI()
    TrafficApp.mainloop()
```