The Algorithm

The output for the traffic simulation was made using the python module, turtle. Turtle draws specified vector graphics on a graphical user interface. An interactive pedestrian button, start and quit button, to exit the program were also added.

When the program is run, firstly the traffic lights and required, static text and buttons are added to the GUI. These are done when the *write_text*, *vehicle_light_box* and *pedestrian_light_box* functions are called. It was chosen to write these as functions and call them with specific parameters, such as x-y co-ordinates of where they were to be place on the GUI, as this saves re-writing the same piece of code multiple times – which makes the program more readable and easier to understand and also reduces the chance of having errors and makes it easier to debug. After the traffic lights are drawn on, the start and wait buttons are added – these were added after the traffic lights because when pressed they call functions which take the different traffic light objects as parameters that need to be previously defined. This in general is not very good practice as it can lead to 'crossed wires' in the program but was done due to the developer's lack of knowledge of OOP and will be discussed in the challenges faced section of the report.

The program is run by pressing the start button and can be quit at any time by pressing the quit button or red x in the corner of the tab. When the start button is pressed the *start_traffic_lights_cycles* function is called which in turn calls the *run_traffic_lights* function with parameters *mayfield_road*, *west_mains_road*, *pedestrian_crossing* which are the turtle objects associated with each of the traffic light drawings. The *run_traffic_lights* function runs an infinite loop which cycles through the states the traffic lights could be in and produces the required timings for which the traffic lights are in each state. Since an infinite loop causes all other functions to cease running on the current thread which the program is being executed on, the start button opens a new thread, on which this infinite loop executes, allowing the rest of the program to function as it was previously. This infinite loop consists of a for loop and state incrementor nested within a while loop. The state incrementor counts which of the 9 states the traffic lights should be in and the for loop, loops through these 9 states. Within the for loop there is a conditional statement that based on the state updates the traffic light. Each clause of the conditional statement increments the state, updates the colour on the traffic light and ceases program execution using the python module *time.sleep*, so that the traffic light is in each state for the required amount of time. After the for loop, the state is reset to 0, so that after the program goes through the 9 states it will repeat this process indefinitely, until it is quit. There is also a wait button running the *wait* function on the same thread as the quit button and rest of the GUI. The *wait* function is called when the wait button is pressed and sets the Boolean variable *waiting* to true, which causes changes to the timings for the traffic lights. The variable *waiting* starts off having value false, when pressed it is assigned a value true and then after the pedestrian lights turn have turned green and gone through the 'blackout' period is reset to being false again.

Traffic Light Timings

The different states of the traffic lights will be referred to, based on the corresponding state incrementors value at the beginning of each of the clauses in the conditional statement in the code for the program.

State 0: The green light at west main road is on for 10 seconds and then if the pedestrian wait button has not been pressed, an additional amount of time which is a random integer number of seconds between 0 and 30.

State 1: The yellow light at west main road is on for 3 seconds which is in agreement with the ltn2/95 document.

State 2: The yellow and red light at Mayfield Road is on for 2 seconds which is in agreement with the ltn2/95 document.

State 3: The green light at Mayfield Road is on for 10 seconds and then if the pedestrian wait button has not been pressed, an additional amount of time which is a random integer number of seconds between 0 and 30.

State 4: The yellow light at west main road is on for 3 seconds which is in agreement with the ltn2/95 document.

State 5: All 3 lights are then red for a random integer number of seconds between 1 and 3. This is in agreement with the ltn2/95 document, in reality the decision for the amount of time for the traffic lights to be red for would be based on sensors at the traffic light, a random number was used instead as the former would be beyond the scope of the project.

State 6 and 7: The green pedestrian light is on for a random amount of time between 4 and 7 seconds and then if the pedestrian wait button had been pressed, it is in the 'blackout' period for an amount of time which is a random integer number of seconds between 0 and 22, which is in agreement with the ltn2/95 document.

State 8: The yellow and red light at west mains Road is on for 2 seconds which is in agreement with the ltn2/95 document

This is deemed to be safe timings for the traffic lights as it gives plenty of time for cars and pedestrians to cross through the intersection and it is efficient as it is interactive as the timings are dynamic and change based on whether or not the wait button has been pressed. This means if there are no pedestrians looking to cross the road, it makes sense to let cars travel through the intersection for longer and if there are pedestrians the pedestrian green light should be on for longer.

Challenges Faced

Timings were not consistent between both documents given with traffic light timings, this meant extra thought needed to go into determining the specific timings, to ensure the traffic lights are safe and efficient. On the more technical side, as there was an infinite loop running, multi-threading was required to be used to allow the traffic light cycle and rest of the program to run asynchronously. Furthermore, due to the developers lack of knowledge of OOP and multithreading it was difficult to get the program to utilise these features and the program is perhaps not the most modular and so this is something to be improved upon in the future.

Appendix

```
#import neccessary modules
import turtle
import tkinter as tk
import time
import random
import threading

#function to start traffic light simulation when start button is pressed
def start_traffic_light_cycles():
    global waiting, mayfield_road, west_mains_road, pedestrian_crossing
    run_traffic_lights(mayfield_road, west_mains_road, pedestrian_crossing)

#function to set waiting variable to true which will alter the timings for
#the different states of the traffic lights
def wait():
    global waiting
    waiting = True

#function that uses turtle to write specfied text at a specified position
def write_text(start, text):
    pen = turtle.Turtle()
    pen.color("black")
    pen.width(3)
    pen.hideturtle()
    pen.penup()
    pen.goto(start[0],start[1])
    pen.pendown()
    pen.write(text)

#function that uses turtle to draw a vehicle traffic light
#at a specified position
def vehicle_light_box(start, width, height):
    pen = turtle.Turtle()
    pen.color("black")
    pen.width(3)
    pen.hideturtle()
    pen.penup()
    pen.goto(start[0],start[1])
    pen.pendown()
    pen.fd(width)
    pen.rt(90)
    pen.fd(height)
    pen.rt(90)
    pen.fd(width)
    pen.rt(90)
    pen.fd(height)
```

```
    red_light = turtle.Turtle()
    red_light.shape("circle")
    red_light.color("grey")
    red_light.penup()
    red_light.goto(start[0]+width/2, 40)
    red_light.pendown()

    yellow_light = turtle.Turtle()
    yellow_light.shape("circle")
    yellow_light.color("grey")
    yellow_light.penup()
    yellow_light.goto(start[0]+width/2, 0)
    yellow_light.pendown()

    green_light = turtle.Turtle()
    green_light.shape("circle")
    green_light.color("grey")
    green_light.penup()
    green_light.goto(start[0]+width/2, -40)
    green_light.pendown()

    #stores turtle objects of the different traffic light bulbs
    #so that the colour of these bulbs can be changed
    return [red_light, yellow_light, green_light]

#function that uses turtle to draw a pedestrian traffic light
#at a specified position
def pedestrian_light_box(start, width, height):
    pen = turtle.Turtle()
    pen.color("black")
    pen.width(3)
    pen.hideturtle()
    pen.penup()
    pen.goto(start[0],start[1])
    pen.pendown()
    pen.fd(width)
    pen.rt(90)
    pen.fd(height)
    pen.rt(90)
    pen.fd(width)
    pen.rt(90)
    pen.fd(height)

    red_light = turtle.Turtle()
    red_light.shape("circle")
    red_light.color("grey")
    red_light.penup()
```

```
    red_light.goto(start[0]+width/2, 40)
    red_light.pendown()

    green_light = turtle.Turtle()
    green_light.shape("circle")
    green_light.color("grey")
    green_light.penup()
    green_light.goto(start[0]+width/2, 0)
    green_light.pendown()

    #stores turtle objects of the different traffic light bulbs
    #so that the colour of these bulbs can be changed
    return [red_light, green_light]

#procedure that runs the main traffic light program, this takes the turtle
traffic light bulbs
#as input and updates them as required
#timings are also dynamic by the fact that pedestrian wait button can affect
these timings
def run_traffic_lights(mayfield_road, west_mains_road, pedestrian_crossing):
    global waiting
    #counts which state traffic light is in so that
    #it can be appropriatly updated
    state = 0
    #loops forever
    while True:
        #loops through the 9 states of the traffic light
        for n in range(0,9):
            if state == 0:
                #increments state
                state += 1
                #updates traffic light colour
                west_mains_road[1].color("grey")
                west_mains_road[0].color("grey")
                mayfield_road[0].color("red")
                west_mains_road[2].color("green")
                pedestrian_crossing[0].color("red")
                #keeps traffic lights in this state for
                #the appropriate amount of time
                time.sleep(10)
                #depending on whether or not pedestrian wait button has been
                #pressed allows green light to stay on for longer
                if waiting == False:
                    time.sleep(random.randint(0,30))
            elif state == 1:
                state += 1
                west_mains_road[2].color("grey")
                west_mains_road[1].color("yellow")
```

```
                time.sleep(3)
            elif state == 2:
                state += 1
                west_mains_road[1].color("grey")
                west_mains_road[0].color("red")
                mayfield_road[0].color("grey")
                mayfield_road[1].color("yellow")
                time.sleep(2)
            elif state == 3:
                state += 1
                mayfield_road[1].color("grey")
                mayfield_road[2].color("green")
                time.sleep(10)
                #depending on whether or not pedestrian wait button has been
                #pressed allows green light to stay on for longer
                if waiting == False:
                    time.sleep(random.randint(0,30))
            elif state == 4:
                state += 1
                mayfield_road[2].color("grey")
                mayfield_road[1].color("yellow")
                time.sleep(3)
            elif state == 5:
                state += 1
                mayfield_road[1].color("grey")
                mayfield_road[0].color("red")
                time.sleep(random.randint(1,3))
            elif state == 6:
                state += 1
                pedestrian_crossing[0].color("grey")
                pedestrian_crossing[1].color("green")
                time.sleep(random.randint(4,7))
            elif state == 7:
                state += 1
                pedestrian_crossing[0].color("red")
                time.sleep(3)
                #depending on whether or not pedestrian wait button has been
                #pressed allows blackout period to last for longer
                if waiting == True:
                    time.sleep(random.randint(0,22))
                    waiting = False
            elif state == 8:
                state += 1
                pedestrian_crossing[1].color("grey")
                west_mains_road[1].color("yellow")
                time.sleep(2)
        #resets state to 0, so that the for loop will execute the same way it
just did
```

```
        #indefinetly
        state = 0

#creates gui window for turtle drawing to ouput on
wn = turtle.Screen()
wn.title("Toucan Traffic Lights Simulation")
wn.bgcolor("white")
canvas = wn.getcanvas()
#starts off with the pedestrian light having not been pressed
waiting = False

#creates a quit button to close program when required
quit_button = tk.Button(wn._root, text="QUIT", command = wn._destroy)
quit_button.pack(side=tk.RIGHT)

#adds required static text and drawings
write_text([-205, 70], "Mayfield Road")
mayfield_road = vehicle_light_box([-200,60], 60, 120)
write_text([-10, 70], "West Mains Road")
west_mains_road = vehicle_light_box([0,60], 60, 120)
write_text([190, 70], "Crossing Signal")
pedestrian_crossing = pedestrian_light_box([200,60], 60, 80)

write_text([195, -60], "PEDESTRIANS")
write_text([195, -60], "PEDESTRIANS")
write_text([195, -70], "push button and wait")
write_text([195, -80], "for signal opposite")

#creates a start button which starts the simulation
#since the simulation is running on an infinite loop it starts it in a
#seperate thread so that the rest of the program can continue executing
start_button = tk.Button(canvas.master, text="START", command =
threading.Thread(target=start_traffic_light_cycles).start())
start_button.pack(side=tk.LEFT)

#creates intereactive pedestrian wait button which will allow the timings to
be changed
#depening on whether or not a pedestrian wants to cross the road
wait_button = tk.Button(canvas.master, text="WAIT", command = wait)
canvas.create_window(225, 100, window=wait_button)

#creates an infinite loop on a seperate thread that constantly check for
interactivity from the user
wn.mainloop()
```