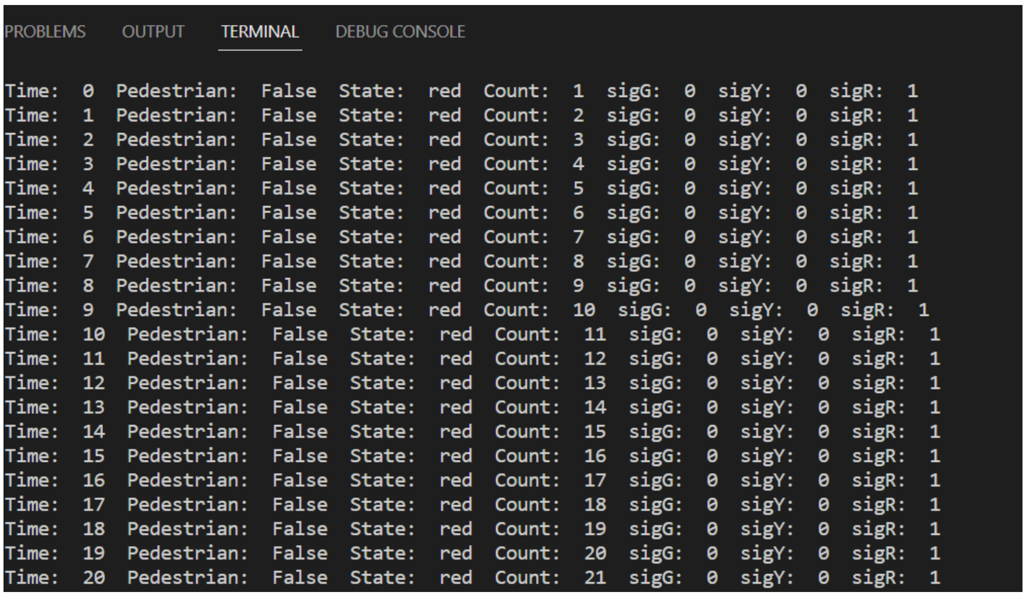
Assignment 2

Explanation:

The implementation code has been built based on following assumptions and steps:

- Create a class named TrafficLight to represent the state machine. This class should have the following attributes: state, count, sigG, sigY, sigR, and pedestrian.
- Initialize the attributes in the constructor with the starting state "red", count=0, sigG=False, sigY=False, sigR=True, and pedestrian=False.
- Implement a method tick() that advances the simulation by one second. This method should increment the count by 1 and update the state based on the conditions described in the state machine diagram.
- Implement methods turn_green(), turn_yellow(), and turn_red() that update the signals sigG, sigY, and sigR respectively to turn on the corresponding light.
- Create a function run_simulation() that creates an instance of the TrafficLight class, and runs the simulation for 200 seconds. This function should log the time, inputs, system state, and outputs to the console every second.
- At 70th second, call the turn_green method to set pedestrian to True.
- At the end of the simulation, print the final state of the system.



```
Time:  0  Pedestrian:  False  State:  red  Count:  1  sigG:  0  sigY:  0  sigR:  1
Time:  1  Pedestrian:  False  State:  red  Count:  2  sigG:  0  sigY:  0  sigR:  1
Time:  2  Pedestrian:  False  State:  red  Count:  3  sigG:  0  sigY:  0  sigR:  1
Time:  3  Pedestrian:  False  State:  red  Count:  4  sigG:  0  sigY:  0  sigR:  1
Time:  4  Pedestrian:  False  State:  red  Count:  5  sigG:  0  sigY:  0  sigR:  1
Time:  5  Pedestrian:  False  State:  red  Count:  6  sigG:  0  sigY:  0  sigR:  1
Time:  6  Pedestrian:  False  State:  red  Count:  7  sigG:  0  sigY:  0  sigR:  1
Time:  7  Pedestrian:  False  State:  red  Count:  8  sigG:  0  sigY:  0  sigR:  1
Time:  8  Pedestrian:  False  State:  red  Count:  9  sigG:  0  sigY:  0  sigR:  1
Time:  9  Pedestrian:  False  State:  red  Count:  10  sigG:  0  sigY:  0  sigR:  1
Time:  10  Pedestrian:  False  State:  red  Count:  11  sigG:  0  sigY:  0  sigR:  1
Time:  11  Pedestrian:  False  State:  red  Count:  12  sigG:  0  sigY:  0  sigR:  1
Time:  12  Pedestrian:  False  State:  red  Count:  13  sigG:  0  sigY:  0  sigR:  1
Time:  13  Pedestrian:  False  State:  red  Count:  14  sigG:  0  sigY:  0  sigR:  1
Time:  14  Pedestrian:  False  State:  red  Count:  15  sigG:  0  sigY:  0  sigR:  1
Time:  15  Pedestrian:  False  State:  red  Count:  16  sigG:  0  sigY:  0  sigR:  1
Time:  16  Pedestrian:  False  State:  red  Count:  17  sigG:  0  sigY:  0  sigR:  1
Time:  17  Pedestrian:  False  State:  red  Count:  18  sigG:  0  sigY:  0  sigR:  1
Time:  18  Pedestrian:  False  State:  red  Count:  19  sigG:  0  sigY:  0  sigR:  1
Time:  19  Pedestrian:  False  State:  red  Count:  20  sigG:  0  sigY:  0  sigR:  1
Time:  20  Pedestrian:  False  State:  red  Count:  21  sigG:  0  sigY:  0  sigR:  1
```

- Define the states:

Define the different states of the traffic light - Red, Green, Yellow, and Pending.
Also, define the initial state of the traffic light, which is Red.

- Implement the input:
  In this scenario, the input is the presence of a pedestrian. It is a binary input, meaning it can either be present or absent.
  For the simulation, you can use a variable to represent the pedestrian input, for example, pedestrian = False.
- Define the variables:
  Define a variable to keep track of the count, for example, count = 0.
  Define variables to store the outputs, for example, sigG = False, sigY = False, sigR = False.
- Implement the logic:
  Based on the state of the traffic light and the input, implement the logic described in the problem statement.
  For example, if the current state is Red and the count is less than 60, increase the count by 1. If the count reaches 60, change the state to Green.
  If the state is Green and the pedestrian is present, change the state to Yellow. If the state is Green and the count is less than 60, increase the count by 1.
  If the state is Yellow, decrease the count by 1. If the count reaches 0, change the state to Red.
- Based on the state, set the outputs (sigG, sigY, sigR) accordingly.
- Implement the simulation loop:
  Start a loop that will run for 200 seconds.
  In each iteration, update the inputs, state, and outputs based on the logic implemented in the previous step.
  Log the inputs, state, and outputs to the console.
- End the simulation:
  End the simulation after 200 seconds.