Assignment 2.1

Steinarr Hrafn Höskuldsson

September 13, 2022

Part 1

A new project was created in PlatformIO and a finite state diagram using plantUML was written.

```
@startuml
[*] --> Red
Red --> Green : Go
Green --> Yellow : Stop
Yellow --> Red : Timeout


Red : entry/ Turn on Red Light
Red : exit/ Turn off Red Light
Yellow : entry/ Turn on Yellow Light
Yellow : entry/ Turn on timer
Yellow : exit/ Turn of Yellow Light
Green : entry/ Turn on Green Light
Green : exit/ Turn off Green  Light
@enduml
```

Listing 1: state.wsd

Which generated the following diagram:

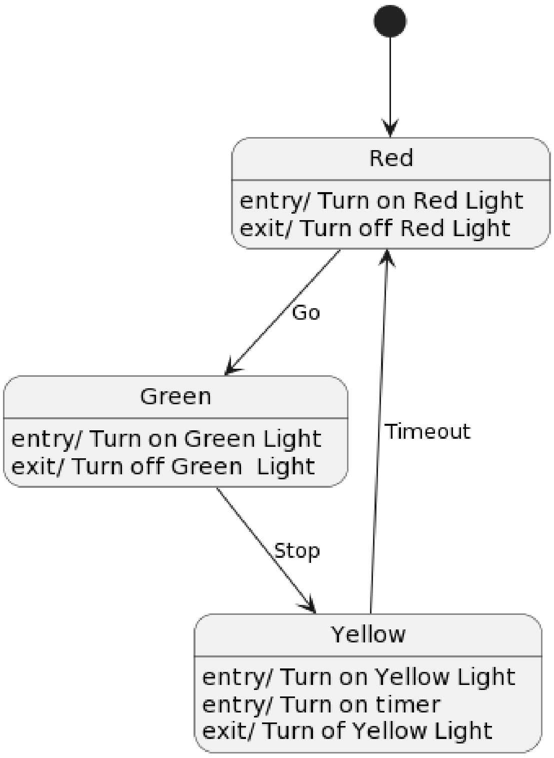
Red
entry/ Turn on Red Light
exit/ Turn off Red Light
Go
Green
entry/ Turn on Green Light
exit/ Turn off Green Light
Timeout
Stop
Yellow
entry/ Turn on Yellow Light
entry/ Turn on timer
exit/ Turn of Yellow Light

Figure 1: Finite State Machine Diagram of a simplified traffic light.

Part 2

The state machine was implemented using a switch case setup. The code can be seen in appendix A. Running the code resulted in the following behaviour, note that lines encased in {} are comments added afterwards.

```
Part 2, State Machine, based on Refactoring Gurus example
RedLight->ON

{sending: g}

I received: 103
I received a go command
RedLight->OFF
GreenLight->ON

{sending s}

I received: 115
I received a stop command
GreenLight->OFF
YellowLight->ON

{there was a delay here}

YellowLight->OFF
RedLight->ON
```

Part 3

The State behaviour was implemented based on the example from Reference Guru. The Context and State Definitions were moved to Context.h and each class definition was moved into its own header file. Here is the program. All relevant header files can be seen in Appendix A

```
#include <Arduino.h>
#include "Context.h"
#include "Red.h"
#include "Green.h"
#include "Yellow.h"
// State Behaviour based on the C++ example at Refactoring Guru

// nothing happens in loop, we never get there.
void loop(){}


Context *context;
void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
  delay(100);
  Serial.println("Part 3, State Machine, based on Refactoring Guru's example");
  char command;
  context = new Context(new Red);

  while (true)
  {
    // wait for some time
    context->do_work();
    if (Serial.available() > 0)
    {
      // read the incoming byte:
      command = Serial.read();

      // say what you got:
      Serial.print("I received: ");
      Serial.println(command, DEC);
      // you can compare the value received to a character constant, like ’g’.

      switch (command)
      {
      case ’g’:
        Serial.println("I received a go command");
        context->transition_to(new Green);
        break;
      case ’s’:
        Serial.println("I received a stop command");
        context->transition_to(new Yellow);

        break;
      }
    }

  }

  delete context;
}
```

Listing 2: main.cpp in Part 3

The behaviour when run was exactly the same as an be seen in Part 2.

A Code

```
#include <Arduino.h>

int command = 0; // for incoming serial data

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
  delay(100);
  Serial.println("Part 2, State Machine, based on Refactoring Guru’s example");
}

enum State
{
  RED,
  YELLOW,
  GREEN
};

State state = State::RED; // Define state variable and set the initial state

bool go_command, stop_command;
unsigned long yellow_timeout_start;

void loop()
{
  // put your main code here, to run repeatedly:

  // send data only when you receive data:
  if (Serial.available() > 0)
  {
    // read the incoming byte:
    command = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(command, DEC);

    // you can compare the value received to a character constant, like ’g’.

    switch (command)
    {
    case ’g’:
      Serial.println("I received a go command");
      go_command = true;
      break;
    case ’s’:
      Serial.println("I received a stop command");
      stop_command = true;
      break;
    }
  }
  switch (state)
  {
  case State::RED:
    if (go_command)
    {
      go_command = false;
      // RED exit action
      Serial.println("Red light -> OFF");
      // event1 transition action
      // set new target state
      state = State::GREEN;
      // target state entry action
      Serial.println("Green Light -> ON");
    }
    break;

  case State::GREEN:
    if (stop_command)
    {
      stop_command = false;
      // GREEN exit action
      Serial.println("Green Light -> OFF");

      // event2 transition action
      // set new target state
      state = State::YELLOW;

      // target state entry action

      Serial.println("Yellow Light -> ON");
      yellow_timeout_start = millis();
    }
    break;

  case State::YELLOW:
    if (millis() - yellow_timeout_start > 1000)
    {
      // YELLOW exit action
      Serial.println("Yellow Light -> OFF");

      // event2 transition action
      // set new target state
      state = State::RED;

      // target state entry action

      Serial.println("Red Light -> ON");
    }
    break;
  }
}
```

Listing 3: main.cpp in Part 2

```
#ifndef CONTEXT_DEFINED
#define CONTEXT_DEFINED
/**
 * The base State class declares methods that all concrete states should
 * implement and also provides a backreference to the Context object, associated
 * with the State. This backreference can be used by States to transition the
 * Context to another State.
 */


class Context; // Forward declaration to allow the definition of a pointer to
    Context without compile error
```

```
class State
{
  /**
   * @var Context
   */

protected:
  Context *context_;

public:
  virtual ~State()
  {
  }

  void set_context(Context *context)
  {
    this->context_ = context;
  }

  virtual void on_do() = 0;

  virtual void on_entry() = 0;

  virtual void on_exit() = 0;


  // ...
};

/**
 * The Context defines the interface of interest to clients. It also maintains a
 * reference to an instance of a State subclass, which represents the current
 * state of the Context.
 */

class Context
{
    /**
     * @var State A reference to the current state of the Context.
     */

private:
    State *state_;

public:
    Context(State *state) : state_(nullptr)
    {
        this->transition_to(state);
    }

    ~Context()
    {
        delete state_;
    }

    /**
     * The Context allows changing the State object at runtime.
     */

    void transition_to(State *state)
    {
        if (this->state_ != nullptr)
        {
            this->state_->on_exit();
            delete this->state_;
        }

        this->state_ = state;

        this->state_->set_context(this);

        this->state_->on_entry();
    }

    /**
     * The Context delegates part of its behavior to the current State object.
     */

    void do_work()
    {
        this->state_->on_do();
    }
};

#endif
```

Listing 4: Context.h

```
#ifndef RED_H
    #define RED_H
    #include "Context.h"
    #include <Arduino.h>

    class Red : public State
    {
    public:
        void on_do() override
        {
        }

        void on_entry() override
        {
            Serial.println("RedLight->ON");
        }

        void on_exit() override
        {
            Serial.println("RedLight->OFF");
        }
    };
#endif
```

Listing 5: Red.h

```
#include <Arduino.h>
#include "Context.h"
// State Behaviour based on the C++ example at Refactoring Guru


class Green : public State
{
public:
  void on_do() override
  {
  }

  void on_entry() override
  {
    Serial.println("GreenLight->ON");
  }

  void on_exit() override
  {
    Serial.println("GreenLight->OFF");
  }

};
```

Listing 6: Green.h

```
#include <Arduino.h>
#include "Context.h"
#include "Red.h"
// State Behaviour based on the C++ example at Refactoring Guru


class Yellow : public State
{
public:
  void on_do() override
  {
    if (millis() - timer_start > 1000)
    {
      this->context_->transition_to(new Red);
    }
  }

  void on_entry() override
  {
    Serial.println("YellowLight->ON");
    timer_start = millis();
  }

  void on_exit() override
  {
    Serial.println("YellowLight->OFF");
  }

private:
  unsigned long timer_start;
};
```

Listing 7: Yellow.h