# Block 1 Project

Understanding and Improvement of Traffic Lights System

## The Team:

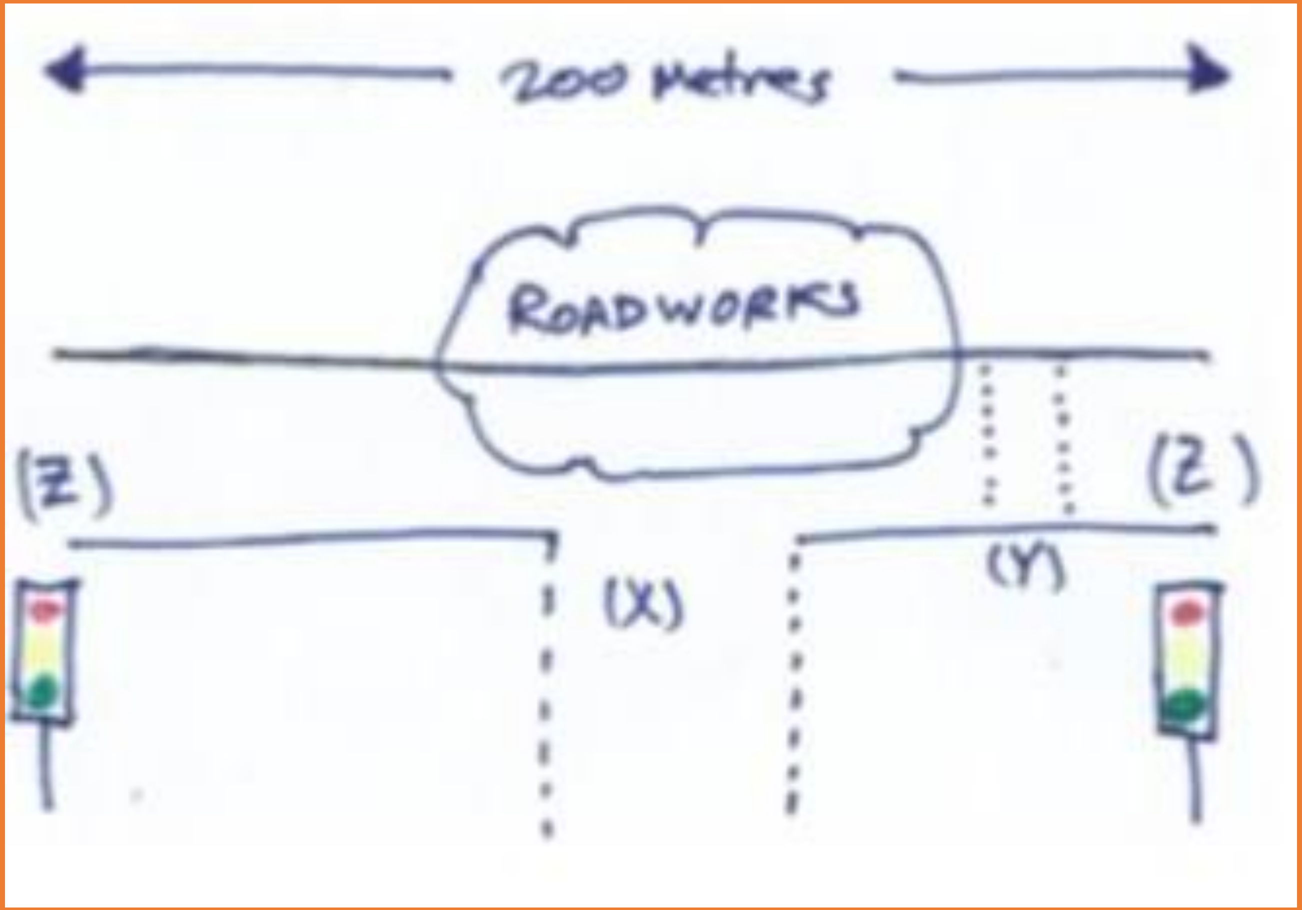
- We were asked to become a consultants for Transport For London and investigate the use of technology in traffic lights systems.
- Our team included:

Jan Ciepiela M00779169

Dhruvil Sachin Patel M00749715

Priyam Rameshchandra Patel M00754186

Haet Patel M00758340

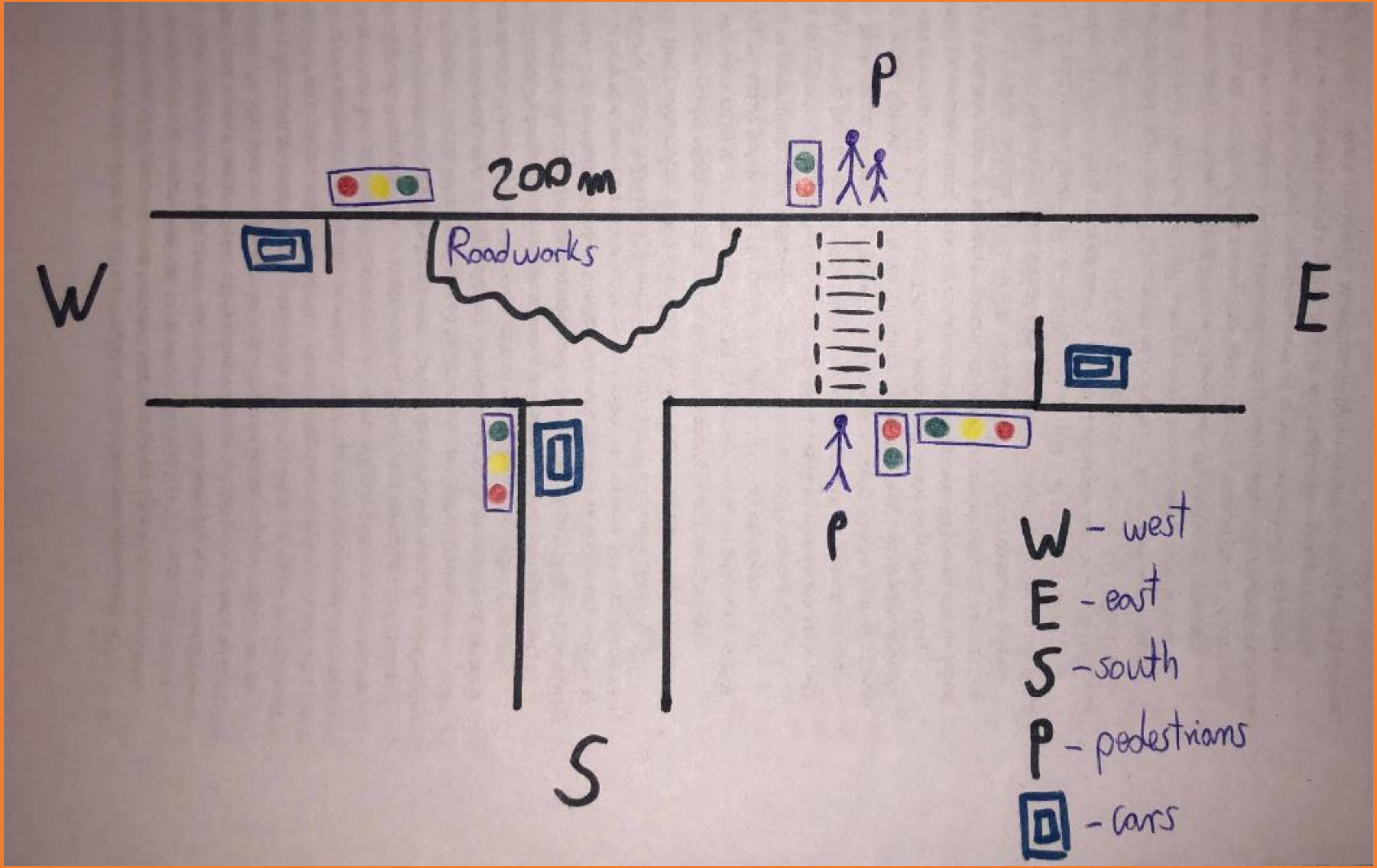


We were given this road layout along with some roadworks to work on and improve, so that's what our team shall do.

## Establishments:

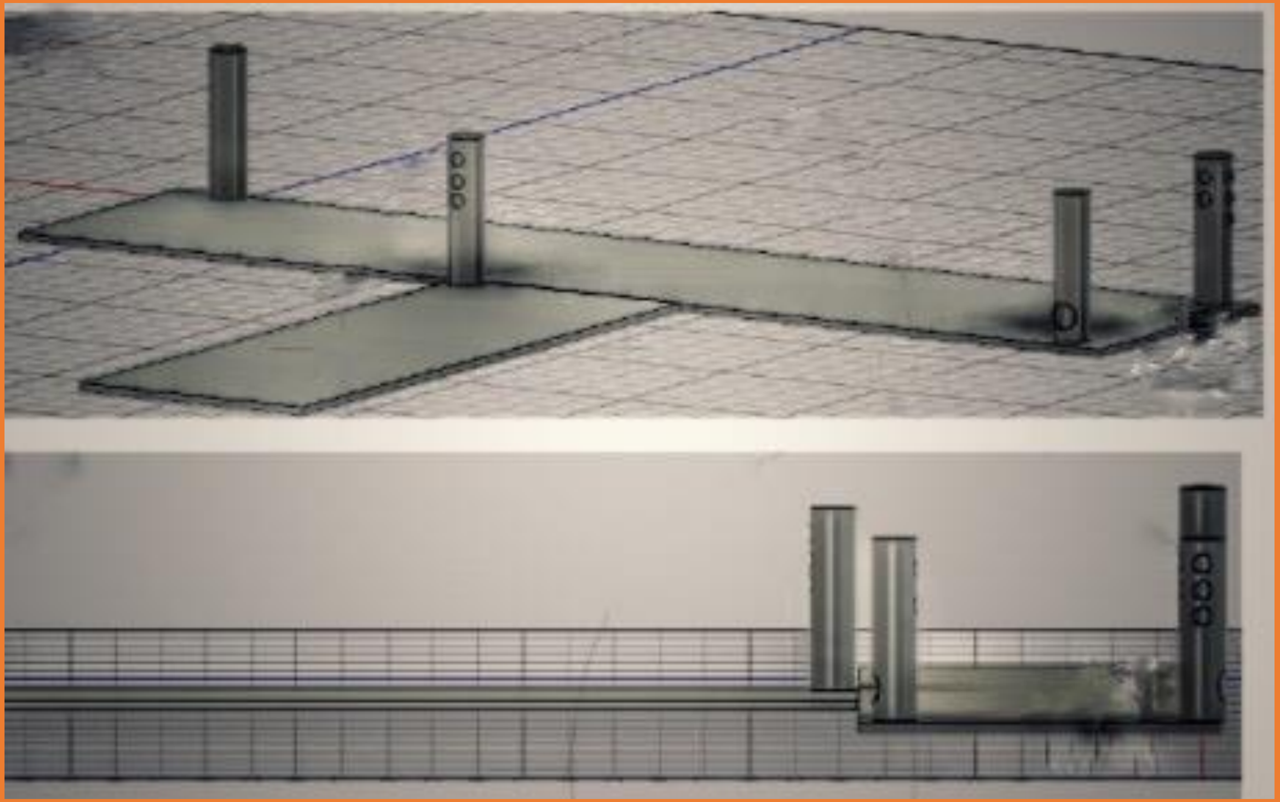
- We presume that all the drivers follow the road code.
- We assume that the roadworks take place in a built-up area.

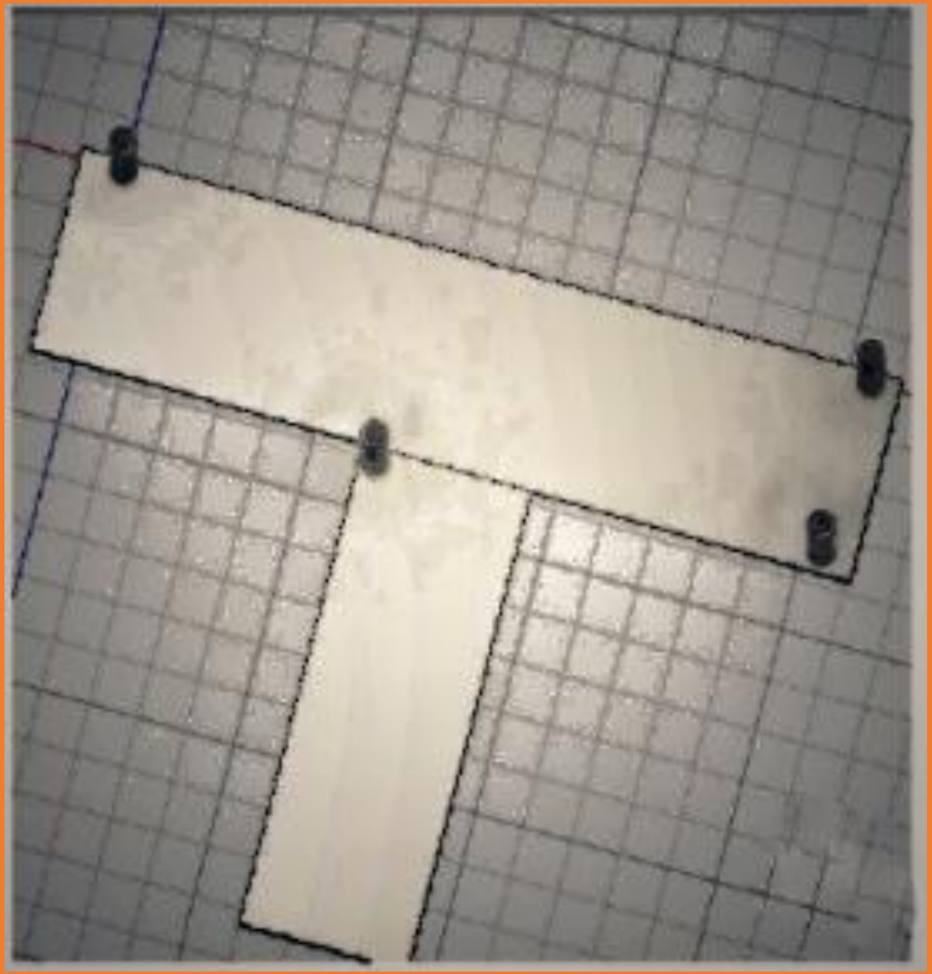
- Just to get started we made our own extended visual representation of where the lights should stand and how the traffic should work, including crossing for pedestrians and their own lights.



Right here you can see the legend of our visual representation <----------------

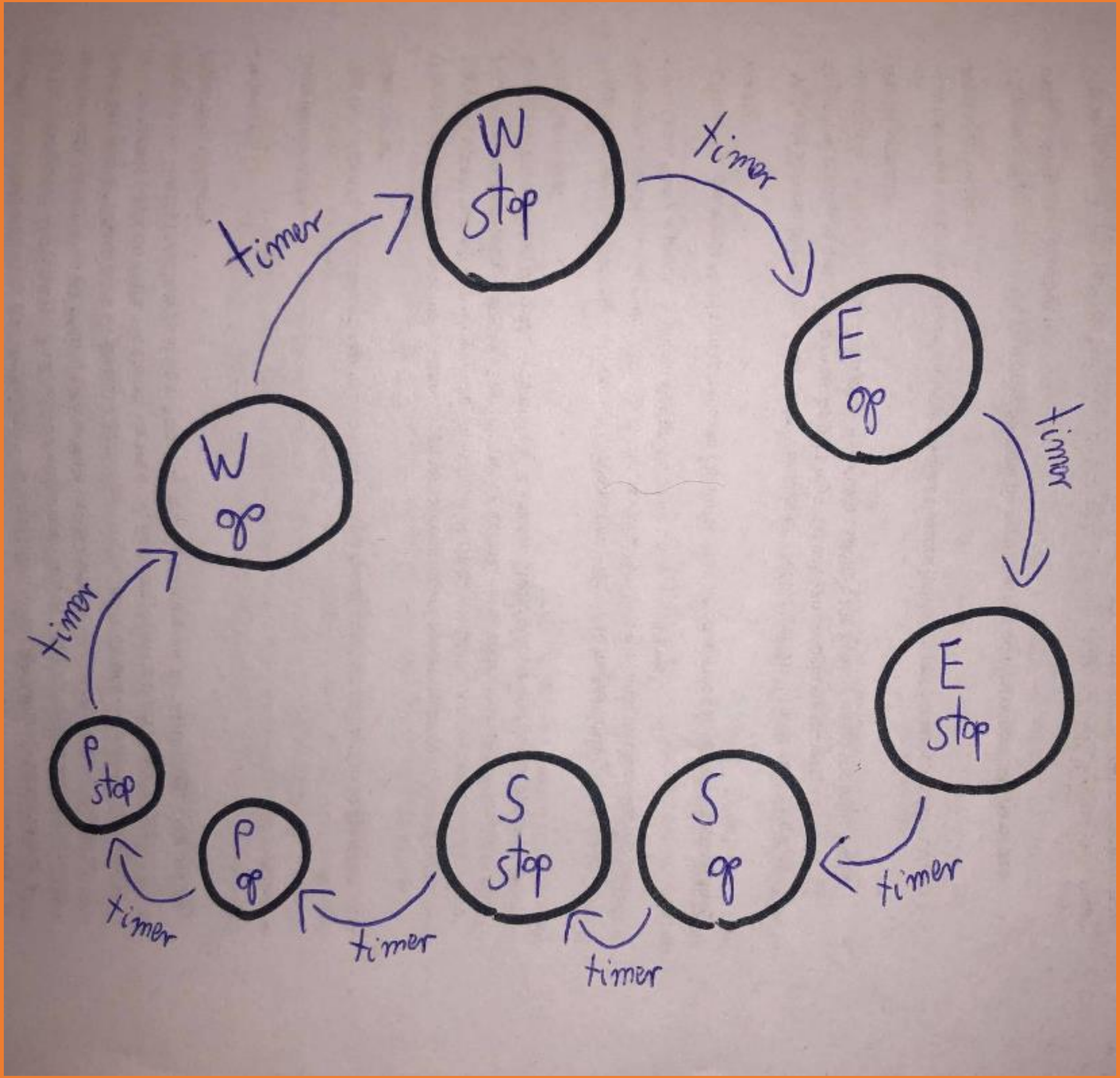
# An attempt to make a 3D version.



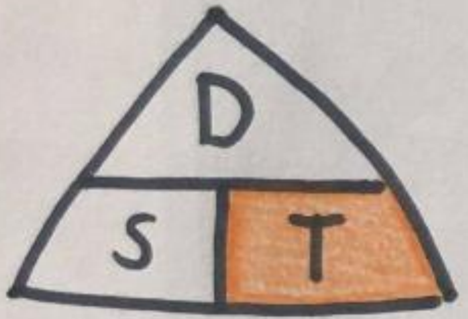


- To visualize what the states of our traffic lights will be we used FSM model.

Let us explain what's going on here. "W go" state indicates that the drivers coming from the west have the green light, hence they are free to go through the roadworks. "W stop" state means simply that drivers coming from the west must stop and aren't allowed to go. The same rule applies for all the other states. The transition "timer" is set to 20 seconds, allowing everyone to pass through roadworks easly as well as allowing the pedestrains to cross the street. Let us show you now why did we set the timer to 20 seconds.



- We assume that the roadworks take place in a built-up area. So an average speed would be around 30 mph. First we have to calculate how much time is it going to take for a driver to drive through 200 meters of roadworks. We will use the formula for Time.



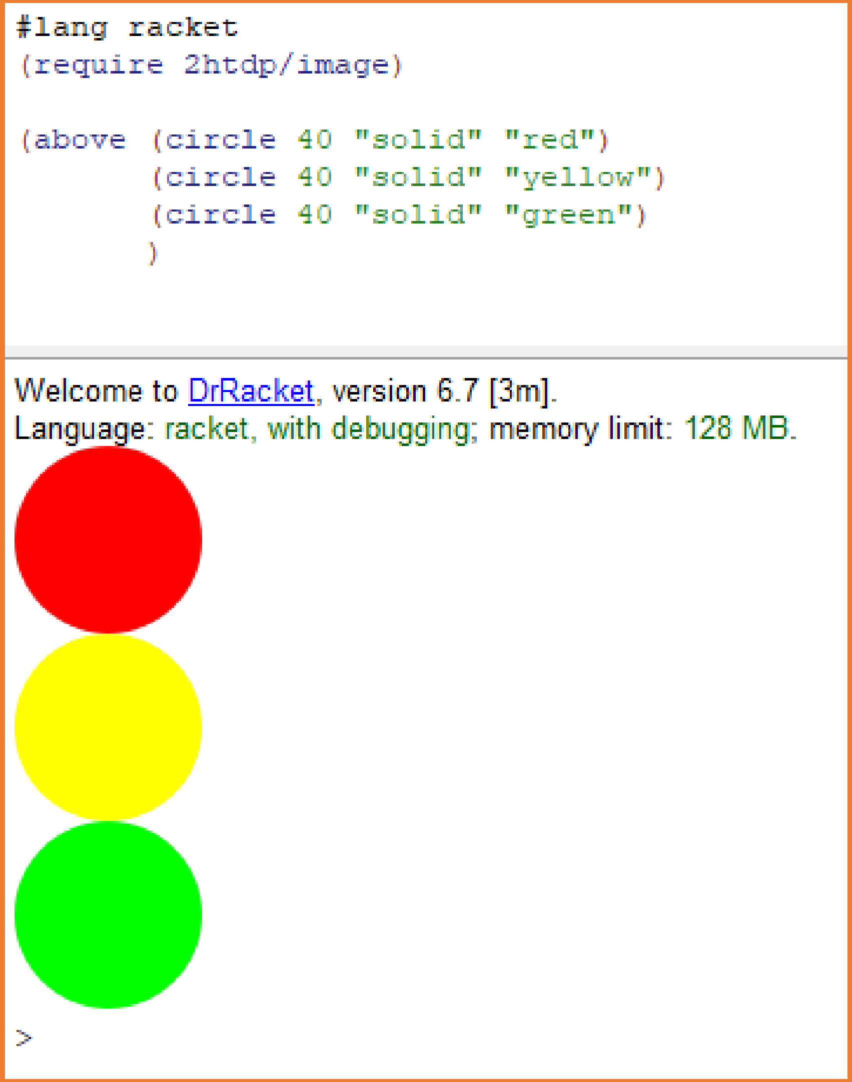
$$\text{Time} = \frac{\text{Distance}}{\text{Speed}}$$

**After calculation** we know that it will take aproximately 15 seconds. To make it more safe, we will round it up to **20 seconds**

## Now let's get into coding.

We shall use Dr.Racket version 7.9

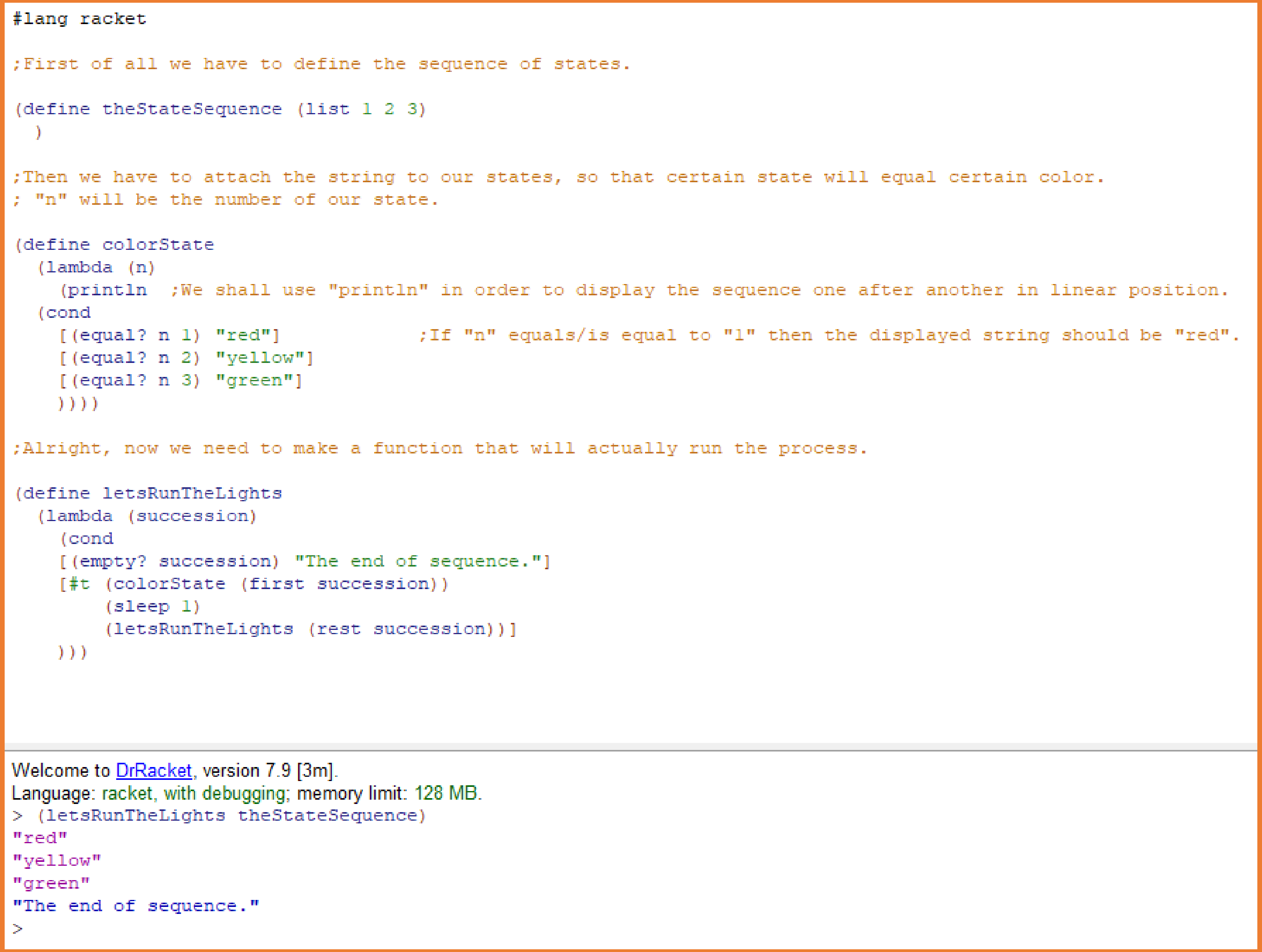
- So we started out simply by displaying how could the actual lights look like.



```
#lang racket
(require 2htdp/image)

(above (circle 40 "solid" "red")
       (circle 40 "solid" "yellow")
       (circle 40 "solid" "green")
       )
```

```
Welcome to DrRacket, version 6.7 [3m].
Language: racket, with debugging; memory limit: 128 MB.
>
```

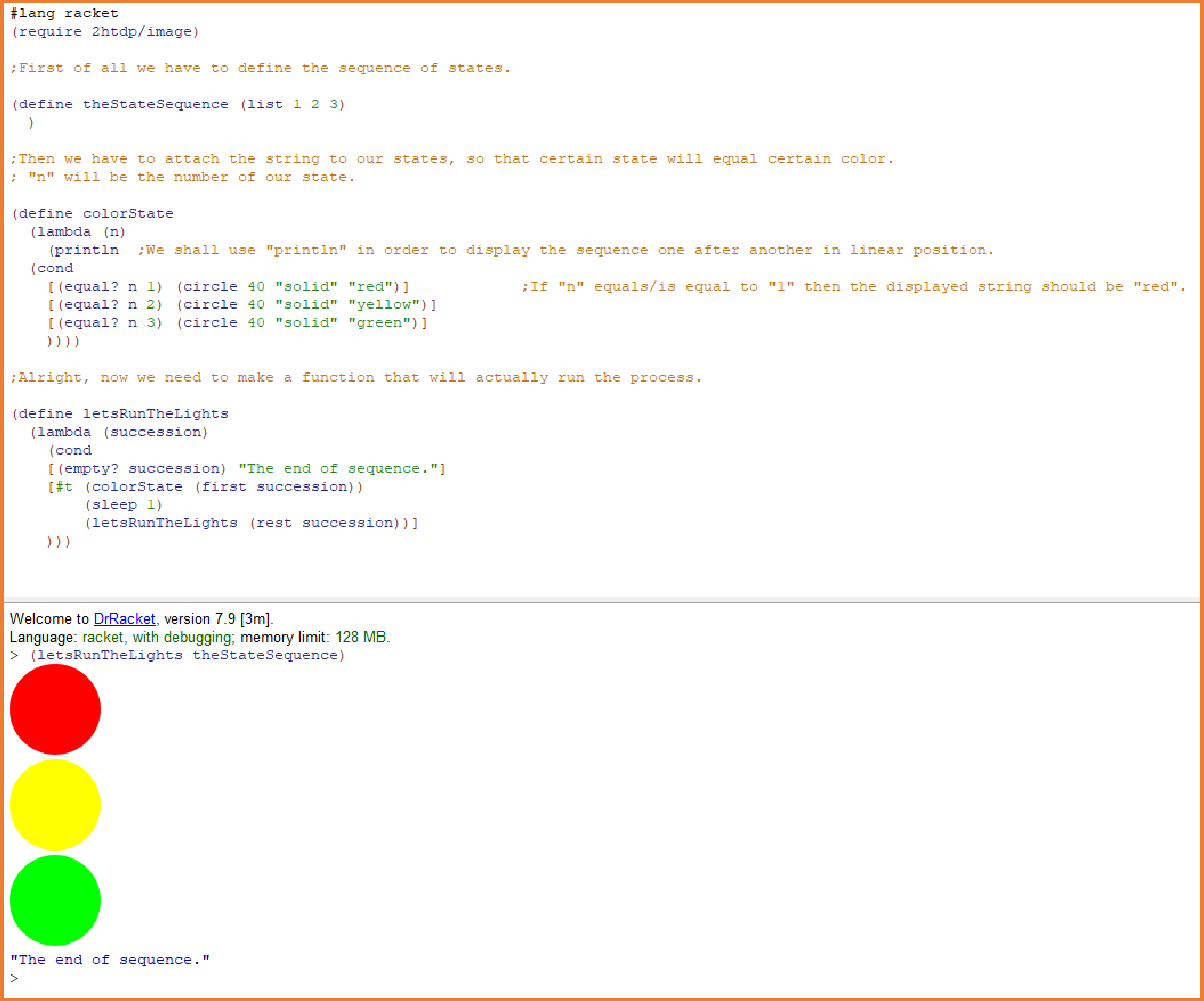
- Then, we made a basic code that would display the process in a form of string.



```
#lang racket

;First of all we have to define the sequence of states.

(define theStateSequence (list 1 2 3)
  )

;Then we have to attach the string to our states, so that certain state will equal certain color.
; "n" will be the number of our state.

(define colorState
  (lambda (n)
    (println  ;We shall use "println" in order to display the sequence one after another in linear position.
  (cond
    [(equal? n 1) "red"]            ;If "n" equals/is equal to "1" then the displayed string should be "red".
    [(equal? n 2) "yellow"]
    [(equal? n 3) "green"]
    ))))

;Alright, now we need to make a function that will actually run the process.

(define letsRunTheLights
  (lambda (succession)
    (cond
    [(empty? succession) "The end of sequence."]
    [#t (colorState (first succession))
        (sleep 1)
        (letsRunTheLights (rest succession))]
    )))
```

```
Welcome to DrRacket, version 7.9 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (letsRunTheLights theStateSequence)
"red"
"yellow"
"green"
"The end of sequence."
>
```

- Then, we implemented the visual representation of the lights in a form of "circles" to our code, so that it's more user-friendly.



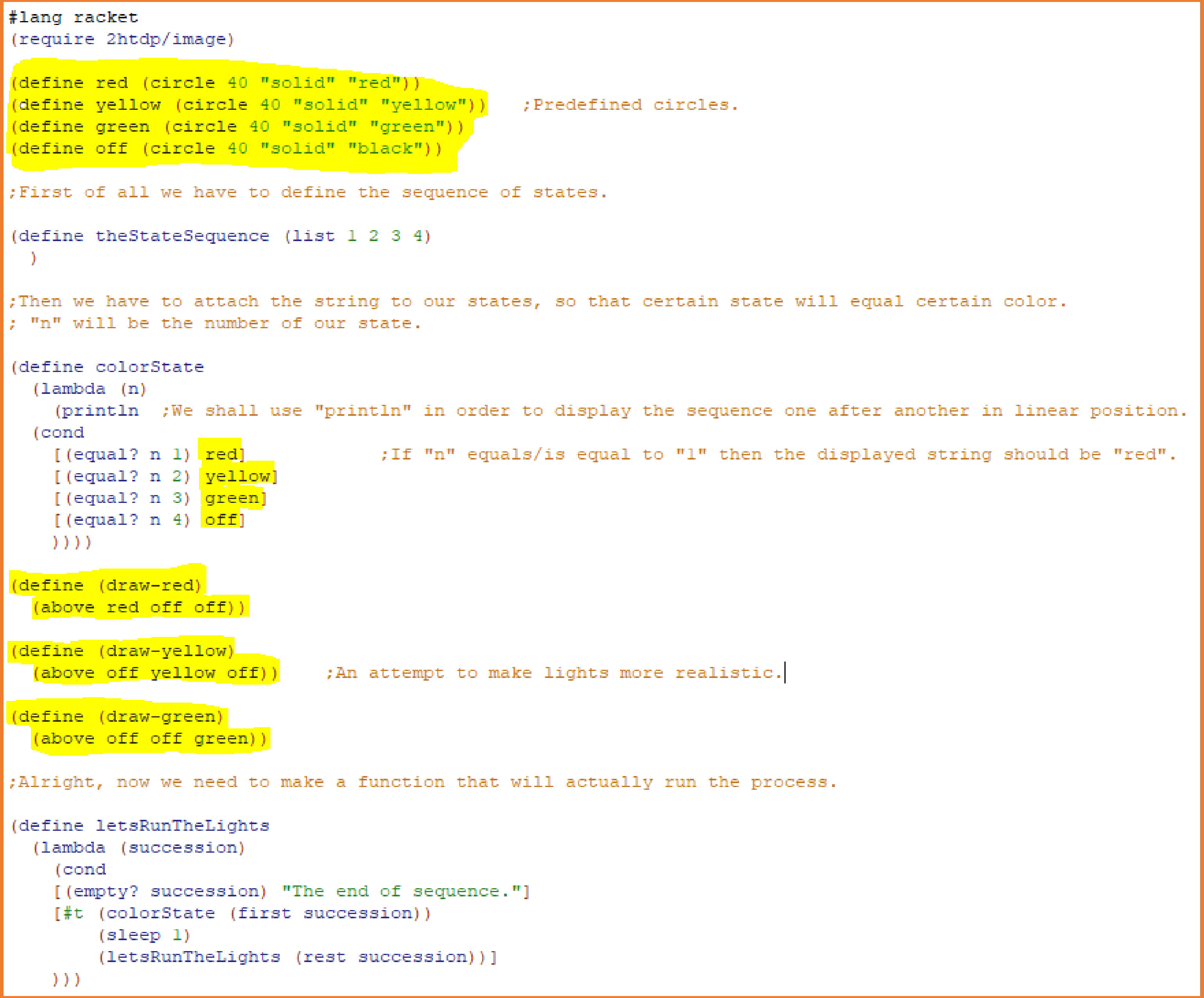
```
#lang racket
(require 2htdp/image)

;First of all we have to define the sequence of states.

(define theStateSequence (list 1 2 3)
  )

;Then we have to attach the string to our states, so that certain state will equal certain color.
; "n" will be the number of our state.

(define colorState
  (lambda (n)
    (println  ;We shall use "println" in order to display the sequence one after another in linear position.
  (cond
    [(equal? n 1) (circle 40 "solid" "red")]             ;If "n" equals/is equal to "1" then the displayed string should be "red".
    [(equal? n 2) (circle 40 "solid" "yellow")]
    [(equal? n 3) (circle 40 "solid" "green")]
    ))))

;Alright, now we need to make a function that will actually run the process.

(define letsRunTheLights
  (lambda (succession)
    (cond
    [(empty? succession) "The end of sequence."]
    [#t (colorState (first succession))
        (sleep 1)
        (letsRunTheLights (rest succession))]
    )))
```

```
Welcome to DrRacket, version 7.9 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (letsRunTheLights theStateSequence)
```

```
"The end of sequence."
>
```

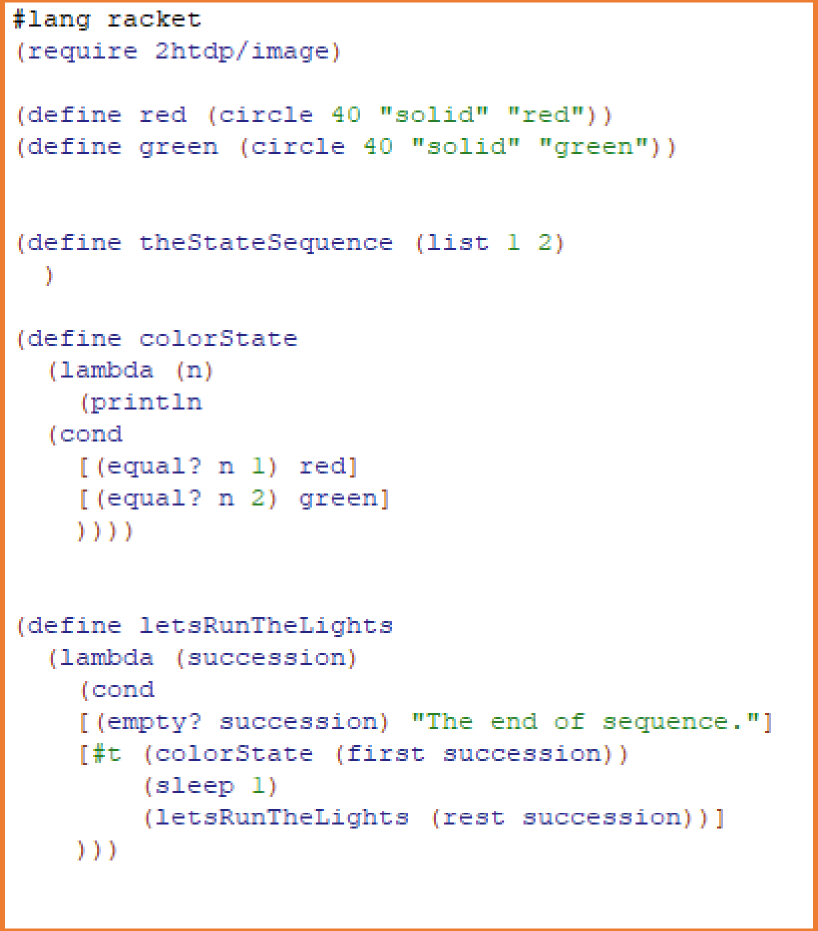
- Then, we added predefined circles for better clarity.

- And then, we tried to make lights more realistic.

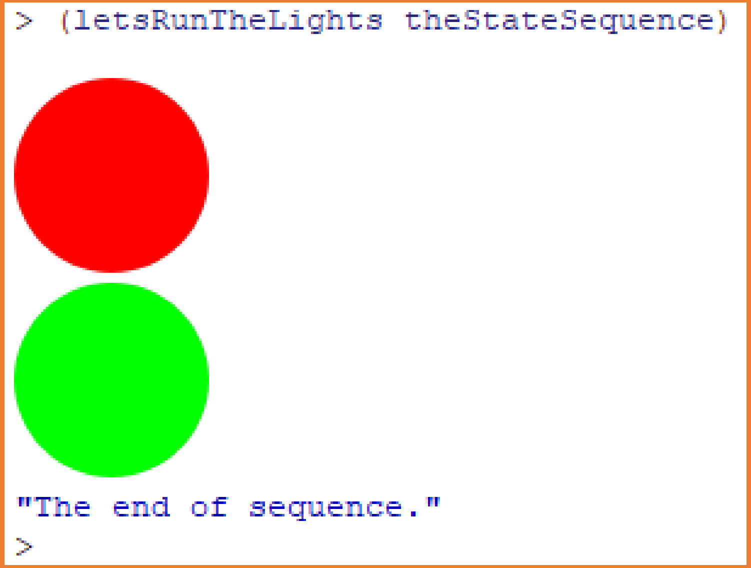


```
#lang racket
(require 2htdp/image)

(define red (circle 40 "solid" "red"))
(define yellow (circle 40 "solid" "yellow"))    ;Predefined circles.
(define green (circle 40 "solid" "green"))
(define off (circle 40 "solid" "black"))

;First of all we have to define the sequence of states.

(define theStateSequence (list 1 2 3 4)
  )

;Then we have to attach the string to our states, so that certain state will equal certain color.
; "n" will be the number of our state.

(define colorState
  (lambda (n)
    (println  ;We shall use "println" in order to display the sequence one after another in linear position.
  (cond
    [(equal? n 1) red]             ;If "n" equals/is equal to "1" then the displayed string should be "red".
    [(equal? n 2) yellow]
    [(equal? n 3) green]
    [(equal? n 4) off]
    ))))

(define (draw-red)
  (above red off off))

(define (draw-yellow)
  (above off yellow off))     ;An attempt to make lights more realistic.

(define (draw-green)
  (above off off green))

;Alright, now we need to make a function that will actually run the process.

(define letsRunTheLights
  (lambda (succession)
    (cond
    [(empty? succession) "The end of sequence."]
    [#t (colorState (first succession))
        (sleep 1)
        (letsRunTheLights (rest succession))]
    )))
```

- Here is another version of the code with the lights occuring at pedestrians crossing.



```
#lang racket
(require 2htdp/image)

(define red (circle 40 "solid" "red"))
(define green (circle 40 "solid" "green"))


(define theStateSequence (list 1 2)
  )

(define colorState
  (lambda (n)
    (println
  (cond
    [(equal? n 1) red]
    [(equal? n 2) green]
    ))))


(define letsRunTheLights
  (lambda (succession)
    (cond
    [(empty? succession) "The end of sequence."]
    [#t (colorState (first succession))
        (sleep 1)
        (letsRunTheLights (rest succession))]
    )))
```



```
> (letsRunTheLights theStateSequence)
```

```
"The end of sequence."
>
```

Now let's move to our written report.