

**Developers Guide**

**Three-Way Traffic Light System**

**What has been coded?**

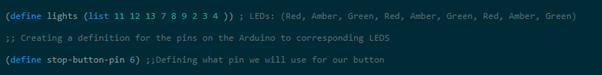
We have coded a system for a three way traffic light system, which includes a pedestrian crossing. The system has been developed in racket allowing it to interact with the users, upon button press it makes the sequence continue normally until all three traffic lights hit an all red state, this is where the delay on the timer increases allowing time for the pedestrians to cross safely. We have included comments within the code along with this document which shall assist any future developers using our code.

**Pre-Requirements**

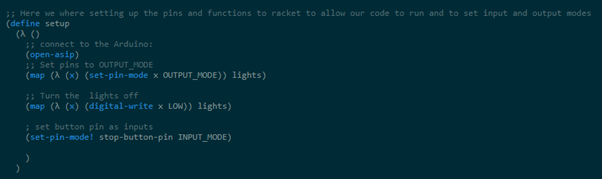
****

In order to get the arduino kits to function with the racket language you will have to link in two requirements files, that act as a library for our code to understand how to communicate directly with the pins on the arduino. Within racket you will also have to define you will be using the language racket. The pre-requirements should always be at the top of the racket code, and the requirements in this code have to be in the same directory as the main script.

**Definitions**



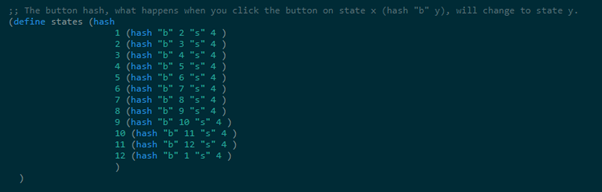
To set the lights to the pins which we have used on the Arduino, the first line of code in the image above is needed. This will set the LEDs to the given pin numbers where each set of traffic lights will be placed thus creating the red, orange and green for the three sets of lights in total. In our project we have included a passenger crossing. The second line in the image above is defining the “stop-button” to pin number six. This is where the button will be connected through the Arduino to take the input so when it is pressed the traffic lights go red and the passenger can cross.



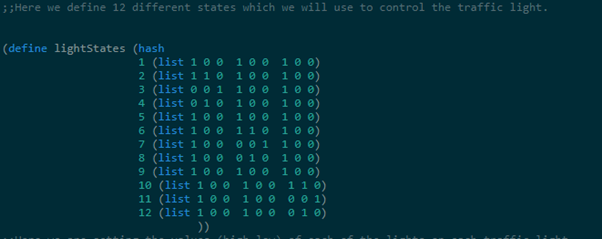
The part shown in the image above is where we define the setup. This is the section where “(open-asip)” is used in order for us to connect to the Auridon and let us use the files which we called to be required at the start. The next line we use the map function where we set the pin modes for our LEDs. Here the pin mode is set to output so that the visual representation of the traffic lights can be seen. The next part is where the same method is used but in this case, is used in order to make the LEDs turn off so when we run our sequence they follow the order we have set. The last line is the function where we set the button for the crossing on the input mode so that it can detect when it has been pressed to run the crossing sequence.



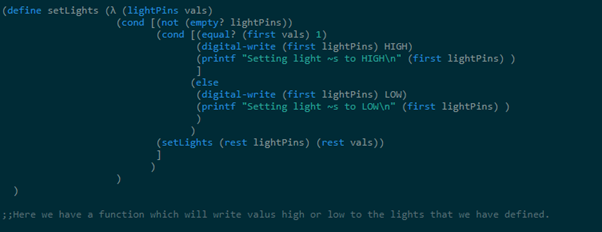
The code above shows where we defined the current state to be set to one. Which basically means that when we call for the current state it will always start from the first one.



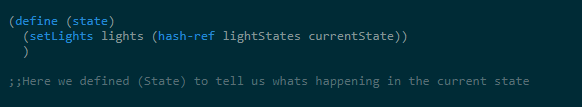
In this section we have the button hash. What happens in this stage is that when there is a button input given it will take the current state and will change it over to the next one given in the same line. For example, in 1 it will go from state 2 to state 4. This is done for all of the state so when there is an input for our crossing we can achieve a smooth transition inside of jumping a state.



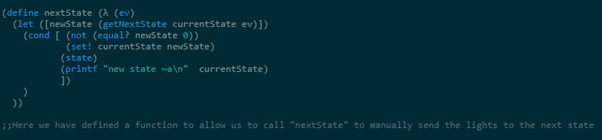
The section of code above shows the “lightstates” which is exactly as the defined name. this describes the position of each LED with each set. So, for the first set, which is the starting set, we can see that the list is 1 0 0 over three times. This will result in us having all of the traffic lights being red at the same time. And in the next one where it is 1 1 0 where the LED will be orange and the last part 0 0 1 where the LED will be green. Then the state is set again to be orange then back to the red. So, this is the rotation of our traffic lights for three sets.



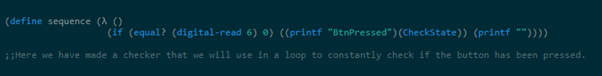
The block above shows the function which we implemented to set values for the LEDs that we have. The function itself runs through a conditional statement where is checks to see if the light pin value is 1 in which case it means that the next section will run, and we use digital-write to set the light pin to the high mode which means it will turn on. It also prints to show on the debugger what mode has been set. The same goes for the next section of the function where if the opposite is true then the values for the LED light pin will be set to low which mean it will be turned off. Once again the output to where as if it was set to high or low will be printed out for the debugger. In the end the ones that are not being used will be set the rest of the values which means they will of low or off.



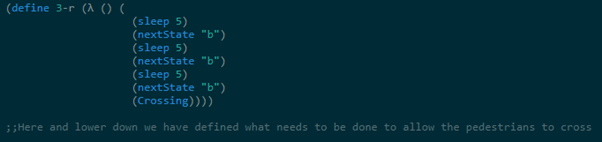
The image above shows the section where we defined “state” so that it will be able to tell us what is the current state. So, when it is running it will be able to see if the lights are on orange and give us the output of the state number which will be state four. This function is also important as this will be used to determine what state we are currently when someone gives an input for the pedestrian crossing.



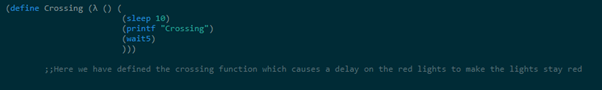
The function above displays the piece of code which lets us to display the next state of lights for our traffic lights. The function called “newState” works by checking the current state and then applying that to a new state where the condition works be checking to see if the new state and the value given is the same. If that is the case, then the current state is set to the new state where by then the state is displayed and the new state is also printed.



This block of code above is what we used in order to see if the button for the pedestrian crossing has been pressed. The code is constantly running in a loop to check and see if there has been an input for the crossing. As you can see the digital-read is used to constantly check on pin 6 which is the pin set to the button. And here we are seeing the value is equal to something. If that happens to be true, the sequence for the crossing will run however if not then the normal set for the traffic lights will be working constantly.



The image above shows the steps which are done in order for the crossing to function. With the code from the function check state combined we can get to this point where we are defining the “3-r” to run the sequence three times to with five seconds interval from changing to the next state before it runs the crossing function. So, with the function check state in mind if we check and see if the current state is 2 then “3-r” will run and go from green to orange to red and then run the crossing sequence. Likewise, if it was to be “2-r” then it will only run for the two set changes after finding out the state. So, for example “2-r” will run when the state is in orange. This means it will go from orange to red after a five second interval and then run the sequence for crossing.



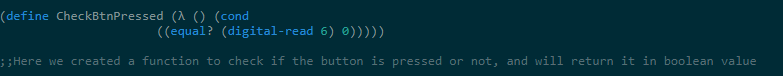
The image above shows the crossing sequence which I was talking about. What this does is run the crossing sequence which will go to the state where the lights are red. Once it is thereafter following the function I mentioned above it will sleep for ten seconds to allow the pedestrian to cross the street. It will also print out crossing in the debugger. After that is done and the lights are back it will run the function “wait5”. What that does is that it checks to see if the button has been pressed every ten milliseconds. And once that runs the process will repeat where the button input is checked and if there is an input the sequence will once again run.

**Buttons**

****

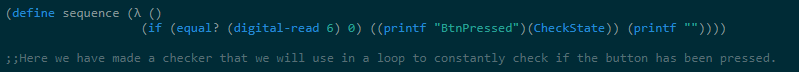
When adding a button we also have to ensure that we have code to detect what to do when given an input. We are using a predefined piece of code called “on-button-pressed” this is included within the Pre-Requirements. Our code as shown above will run a defined sequence (called sequence), when the stop-button-pin is pressed (We have previously defined stop-button-pin as 6 which is the pin used on input on an arduino )

**Condition**

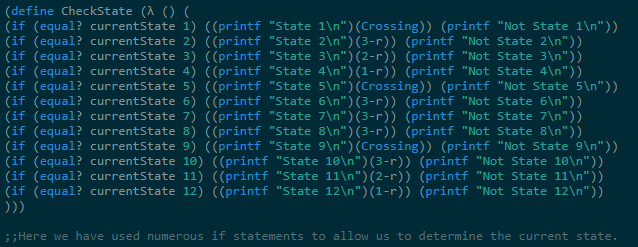
****

We have a function we have defined as CheckBtnPressed. We decided to go with a condition for this rather than an if statement as its more efficient in this instance. Our conditions checks if the value being given from pin 6 (reading the button) is equal to 0 (0 is the value when the button is being pressed), if the button is being pressed it will set CheckBtnPressed to true (a boolean value). We wanted this as a boolean value as we may need to in future development check the state of a button.

**If Statements**

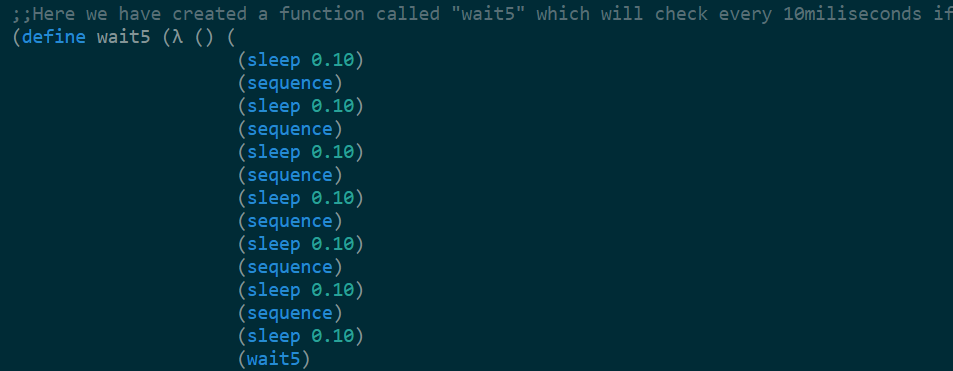
****

With one of our chunks of code that we will be calling to almost non stop within the sequence we have used an if statement. What our if statement does when (sequence) is called, is check if the value of pin 6 at that moment is equal to 0 (This means the button is pressed), and if its true it will print in our debug menu “BtnPressed” and run the sequence (Checkstate) (Which we will explain in a moment), if the button is not pressed it will print an empty string into the debug menu.



Within (CheckState) we have 12 if statements, what these will do check what the current state it, we do this using “(equal? currentState 12)” if the state is detected it will run a linked sequence which will take the light sequence back over to all red and allow the user to cross. We have included print messages to allow future debugging, once the button is pressed or (CheckState) is called, it will print the state it's in, and all the ones it's not in.

**Loop/Recursion**

****

During development we have had to adjust the code in order to get passed the issue of sleep between sequences stopping the button input from being detected as sleep pauses racket, so what we did to get past this was create a loop called wait 5 (this function waits 5 seconds then continues our sequence). What we are doing is setting a small delay and checking between delays if the button is being pressed then calling the sequence back to itself.