jrullan Update README.md History

1 contributor

▸ Details 162 lines (114 sloc) | 5.71 KB ...

Micropython Statemachine

This is a port from a library I developed for Arduino a while back. Now that I am working with Raspberry Pi Pico I decided to port it to Micropython. This library includes the Neotimer library (https://github.com/jrullan/micropython_neotimer) for convenience but it does not require it.

Explanation

This library implements a basic State Machine. State machines are used extensively in controllers and systems to guarantee that the program behaves deterministically. This means that at any point in time you know exactly the state of the application and only certain behaviors are allowed to happen while in a particular state. The application can only be in one state at any point in time. To change from one state to another there must be a transition condition logic that determines if a transition should occur.

Please note that if multiple transitions are attached to a state, it will evaluate them in the order they were attached and the first one it finds that evaluates to True is the one that takes effect.

The state logic and its transition's conditions are implemented as functions in your program for flexibility.

States

States contain the machine logic of the program. The machine only evaluates the current state until a transition occurs that points to another state.

To evaluate a piece of code only once while the machine is in a particular state, you can use the `state_machine.execute_once` attribute. It is **True** each time the machine enters a new state until the first transition is evaluated.

Transitions

Each state can have multiple transitions attached. Transitions require two parameters,

1. The transition test function that returns a boolean value indicating whether or not the transition met the criteria defined in the function and
2. The target state to transition to

If none of the attached transitions evaluate to true, then the machine stays in the current state.

An alternative way to specify the transitions without creating transition functions is to use the state machine's `force_transition_to()` method. This method will force the transition to another state, bypassing any transitions attached to a particular state. This approach also has the benefit of not requiring the creation of transition objects, and the code could be leaner. One way of using this feature is to use it inside the state logic as in the example below:

```
def state1_logic():
    global counter
```

```
    if myMachine.execute_once:
        myTimer.start()
        counter += 1
        print("State 1 Logic: Blinking LED")

    if myTimer.finished():
        myMachine.force_transition_to(state2)   #<---- If timer has finished f
```

Example Template

```
from neotimer import *
from statemachine import *
from machine import Pin

state_machine = StateMachine()
myTimer = Neotimer(1000)
led = Pin(25,Pin.OUT)

#============================================================
# States Logic Functions
#============================================================
def state0_logic():
    # Referenced global variables
    # ----> Here <----

    # Code that executes just once during state
    if state_machine.execute_once:
        print("Machine in State 0")
        myTimer.start()

    # Code that executes continously during state
    led.off()


def state1_logic():
    # Referenced global variables
    # ----> Here <----

    # Code that executes just once during state
    if state_machine.execute_once:
        print("Machine in State 1")
        myTimer.start()

    # Code that executes continously during state
    led.on()

#============================================================
# Add states to machine (Also create state objects)
```

```
#=============================================================
# Create States
state0 = state_machine.add_state(state0_logic)
state1 = state_machine.add_state(state1_logic)

#=============================================================
# State Transitions Functions (optional)
#=============================================================
# Create Transition Functions
def delay_transition():
    if myTimer.finished():
        return True
    else:
        return False

#=============================================================
# Attach transitions to states (optional)
#=============================================================
state0.attach_transition(delay_transition, state1)
state1.attach_transition(delay_transition, state0)


# Main Loop: Run the state machine here
while True:
    state_machine.run()
```

Raspberry Pi Pico - Multicore

You could leverage your Pico dual core architecture by running the state machine in a separate core, leaving one core entirely for other non deterministic code (continuous functions).

```
import _thread

...

# This is the main loop running on the second core (Core 1)
def state_machine_logic():
    while True:
        state_machine.run()

# Start state_machine_logic() on second core
_thread.start_new_thread(state_machine_logic, ())

# Main Loop: Use for functionality not related to the states logic
while True:
    #---> do whatever here, the machine will run on the other core <---
    ...
```