FSM - Finite State Machine

A finite-state machine (FSM) is a computational model of an abstract machine that can be in exactly one of a finite number of states at any given moment. Depending on the programming logic of the state machine, particularly in game engines or microcontroller units (MCU), there is a process that loops, for example, the game loop or the MCU processing loop. The loop is continuously polling for the programmed conditions of the state inputs. The FSM can change from one state to another in response to these inputs where the change from one state to another is called a transition. The FSM is defined by an arbitrary list of its states, its initial state, and the inputs that trigger each transition.

Keeping in mind the best programming practices, all decisions made are maintained within the states themselves and any processes for the states are maintained in a central manager. The state machine is separate and independent from the list of states and the manager.

CFP-FSM

The C style function pointer finite state machine (CFP-FSM) is one of the simplest state machines to write in the C programming language. The pointer function besides simply calling a function in the normal way, it can also be used to call functions. Even though the function is not a variable, there is a memory location that can be assigned to a pointer. The function's address is the entry point of the function. On line 34, this is a function pointer is declared as *currentState* within the pointer function *void (*pointer)()*.

```
23
24 #include <Arduino.h>
25
26 #include "Timer.h"
27 #include "Bitwise.h"
28
29 // Global Objects
30 uno::Timer timer;
31 nmr::Bitwise<int> bitwise;
32
33 // Global Function Pointer
34 void (*currentState)();
35
```

How does function pointer work? The function pointer is used to assign a *named*-state function from any number of the listed arbitrary states that points-to or calls. For example on line 166, *currentState* is assigned *greenRedState*.

```
158       // Process Cross Avenue Conditionals
159       if (bitwise.IsBitNumberSet((uint16_t)crossRR))
160       {
161         bitwise.ClearBitNumber((uint16_t)crossRR);
162         Serial.print("GREEN - RED");
163         bitwise.SetBitNumber((uint16_t)mainGR);
164       }
165       // Transition from overlapState to greenRedState
166       currentState = greenRedState;
167     }
```

What is *greenRedState*? On line 76, *greenRedState* in one of three state functions used in the example of a traffic lights FSM at the intersection of Main Street and Cross Avenue.

```
74
75  // state functions
76  void greenRedState()
77  {
78    // State Conditional - 5s
79    if (timer.isTimer(5000))
80    {
81      timer.resetTimer();
82      processOnce = true;
83    }
84  }
85
86  void amberState()
87  {
88    // State Conditional - 2.5s
89    if (timer.isTimer(2500))
90    {
91      timer.resetTimer();
92      processOnce = true;
93    }
94  }
95
96  void overlapState()
97  {
98    // State Conditional - 1s
99    if (timer.isTimer(1000))
100   {
101     timer.resetTimer();
102     processOnce = true;
103   }
104 }
105
106 // manager function
107 void lightManager()
108 {
109   if (processOnce)
110   {
111     if (currentState == greenRedState)
112     {
113       // Process Main Street Conditionals
114       if (bitwise.IsBitNumberSet((uint16_t)mainGR))
115       {
```

When and where is the function pointer called? Usually there is single call yet it depends where the programmer wants to place the call. In the function *void LightManager() that starts on line 106*, the last statement of the function of *LightManager* on line 177 is the call function *currentState()*. Lines 116 through 163 were removed for clarity.

```
164       }
165       // Transition from overlapState to greenRedState
166       currentState = greenRedState;
167     }
168
169     Serial.print(" - bit number: ");
170     Serial.print(bitwise.GetBitNumber());
171     Serial.print(" ");
172     Serial.println(bitwise.PrintBinaryBits());
173     processOnce = false;
174   }
175
176   //
177   currentState();
178 }
```

The FSM controls a simple traffic light at an intersection of Main Street and Cross Avenue but it is written in the C programming language. There is actually no state machine to speak about except the state manager here substitutes as the state machine. This is just a simple demonstration in how the mechanics of a finite state machine works. Once mastered, we move on to the OOP-FSM, our old friend C++, an object oriented programming driven finite state machine.

Carpenter Software
Article 1004-CFP-FSM-001 date 20221220.
by Jesse Carpenter