CCA IXDSN-300-01/02
Interaction Design 3: Objects and Spaces
State Machine Walkthrough
Associated Code: T1_StateMachines
Last updated October 3, 2014

---

1. Defining the States

State machines are the basis of making more interesting interactions. They are one way of allowing systems to have memory, and to give them the ability to respond differently to the same type of input (for example, a button press under 5 seconds) depending on what's happened to them in the past.

In the case of state machines, the states are the basis for describing how the system will react to each type of input. It's the responsibility of the designer to choose what states the system will have. 2 systems could work identically and use different states, and that's perfectly fine.

Ideally, your selection of the systems states will result in the following:

> *for each possible input to your system, each state will have only one possible output*

For the LED example, let's say I simply chose two states:
State 0 = "Power On"
State 1 = "Power Off"

I have three possible inputs to my system:
Input 0 = Button pressed for < 5 seconds then released
Input 1 = Button pressed for > 5 seconds then released

Let's focus on how the system responds to each input in the "Power On" state:
State 0, Input 0 => could turn cyan, could turn magenta, could turn yellow
State 0, Input 1 => moves to Power Off state

For Input 1, there are 3 possible options! This means we need a better way of describing the states of this system. Let's try the following state definitions:
State 0 = "Power On - Cyan"
State 1 = "Power On - Magenta"
State 2 = "Power On - Yellow"
State 3 = "Power Off - Cyan"
State 4 = "Power Off - Magenta"
State 5 = "Power Off - Yellow"

Notice for completeness, I included the color even in the "Power Off" states. This is because when the system is turned off then on again, I want it to return to its most recent color, so I need to keep track of that even when it is OFF.

The above option only uses one state variable to describe the 6 different states, which can take values between 0 and 5. For my example, I'm going to use 2 state variables (powerState with 2 different states and colorState with 3 different states).

System Description	**Variable "State"** *Old State Description*	**Variable "powerState"** *New State Description*	**Variable "colorState"** *New State Description*
Power On, Cyan	0	1	0
Power On, Magenta	1	1	1
Power On, Yellow	2	1	2
Power Off, Cyan	3	0	0
Power Off, Magenta	4	0	1
Power Off, Yellow	5	0	2

---

2. Inputs

An **input** is anything you can do to the system to cause a change of state.

As mentioned earlier, for our example system the inputs are:

Input 0 = Button pressed for < 5 seconds then released
Input 1 = Button pressed for > 5 seconds then released

Note that I explicitly called out the release of the button. If you think about the problem description, I actually don't care about when the button is *pushed*. I really only want to know *when it is released*, and *how long it was on for* before it was released. If you don't make this distinction and accidentally only detect when the button is pressed, it's likely your system will cycle through colors very quickly whenever the button is held, instead of waiting until it is released.

You can define your Inputs before or after your states, but before moving forward it's always good to double check to see how your system's states respond to each input. If there's an input that doesn't change any states, get rid of it. If there's a behavior your system should have but doesn't, check to see if you need better state or input definitions.

---

3. Outputs

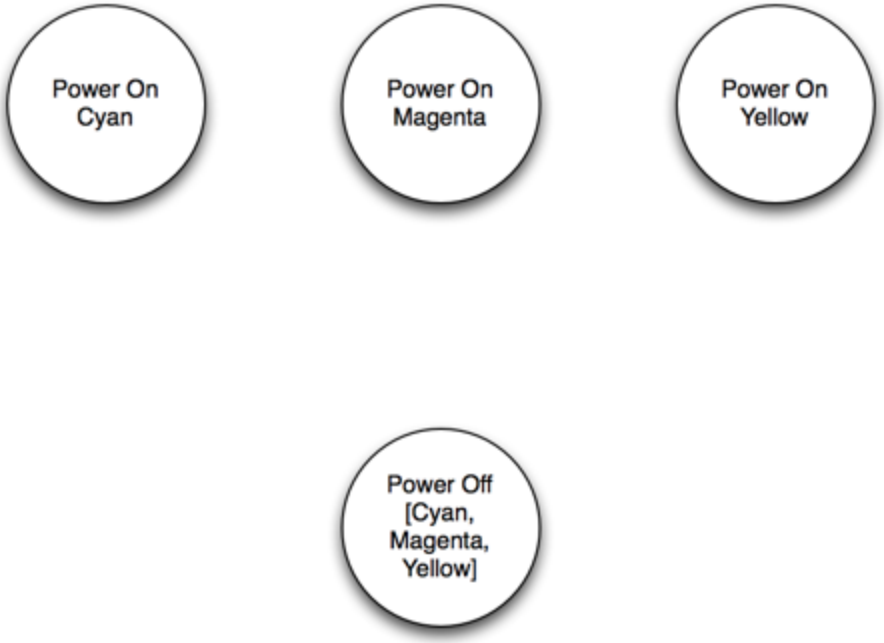
Power = 0, 1
Color = cyan, magenta, yellow

State Flow Diagram

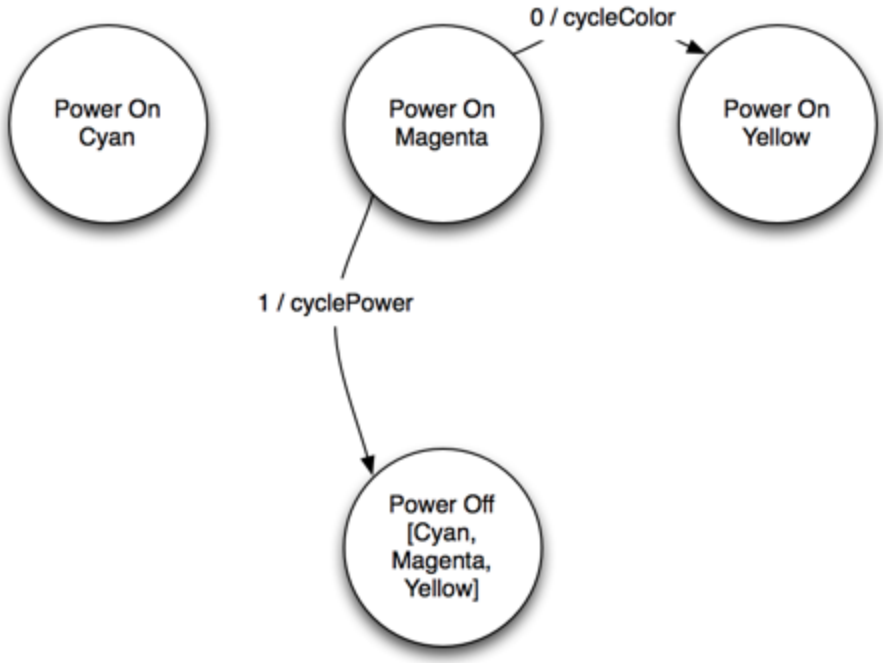
The state flow diagram consists of several elements:

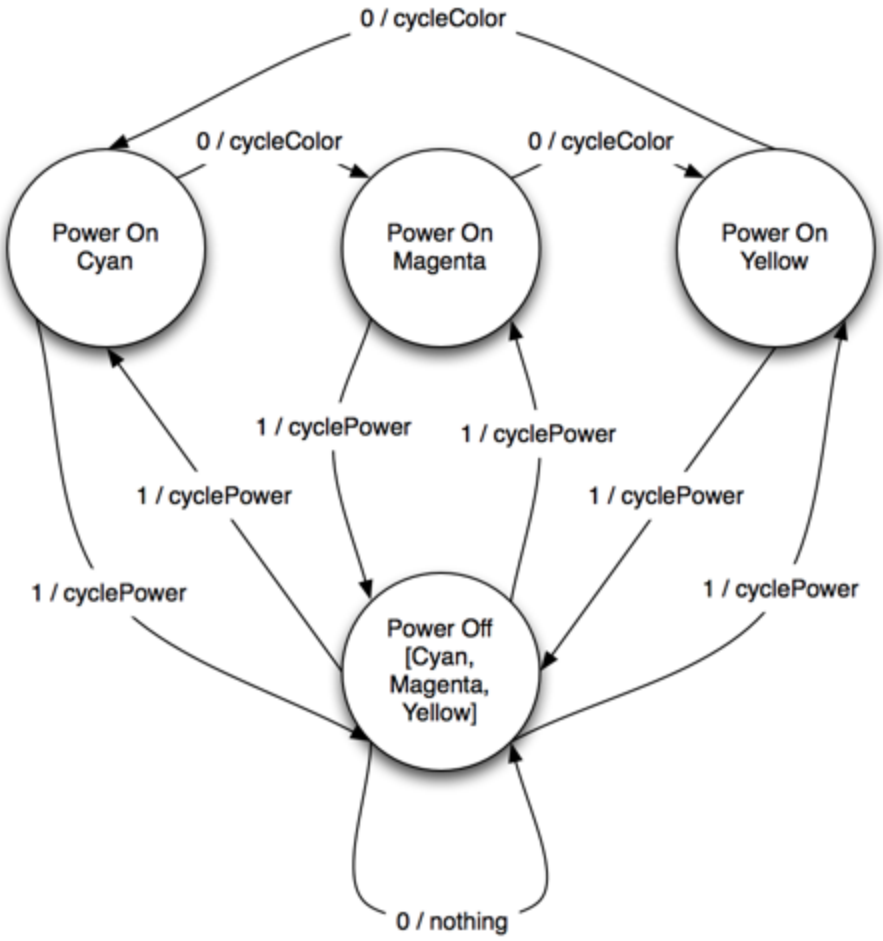
- *the states* of your system (including the outputs at each state)
- *transition arrows* - from a given state, these show which state your system transitions to next for each input option

1. Draw all possible states in circles. Make sure to include the outputs and their values.



2. Draw the transition lines. For each state, draw a transition line for each input. Do this until each state has a number of lines leaving it equal to the number of inputs of your system (2 in this case).


Power On
Cyan
Power On
Magenta
0 / cycleColor
Power On
Yellow
1 / cyclePower
Power Off
[Cyan,
Magenta,
Yellow]


0 / cycleColor
0 / cycleColor
0 / cycleColor
Power On
Cyan
Power On
Magenta
Power On
Yellow
1 / cyclePower
1 / cyclePower
1 / cyclePower
1 / cyclePower
1 / cyclePower
Power Off
[Cyan,
Magenta,
Yellow]
0 / nothing

Moving to Code

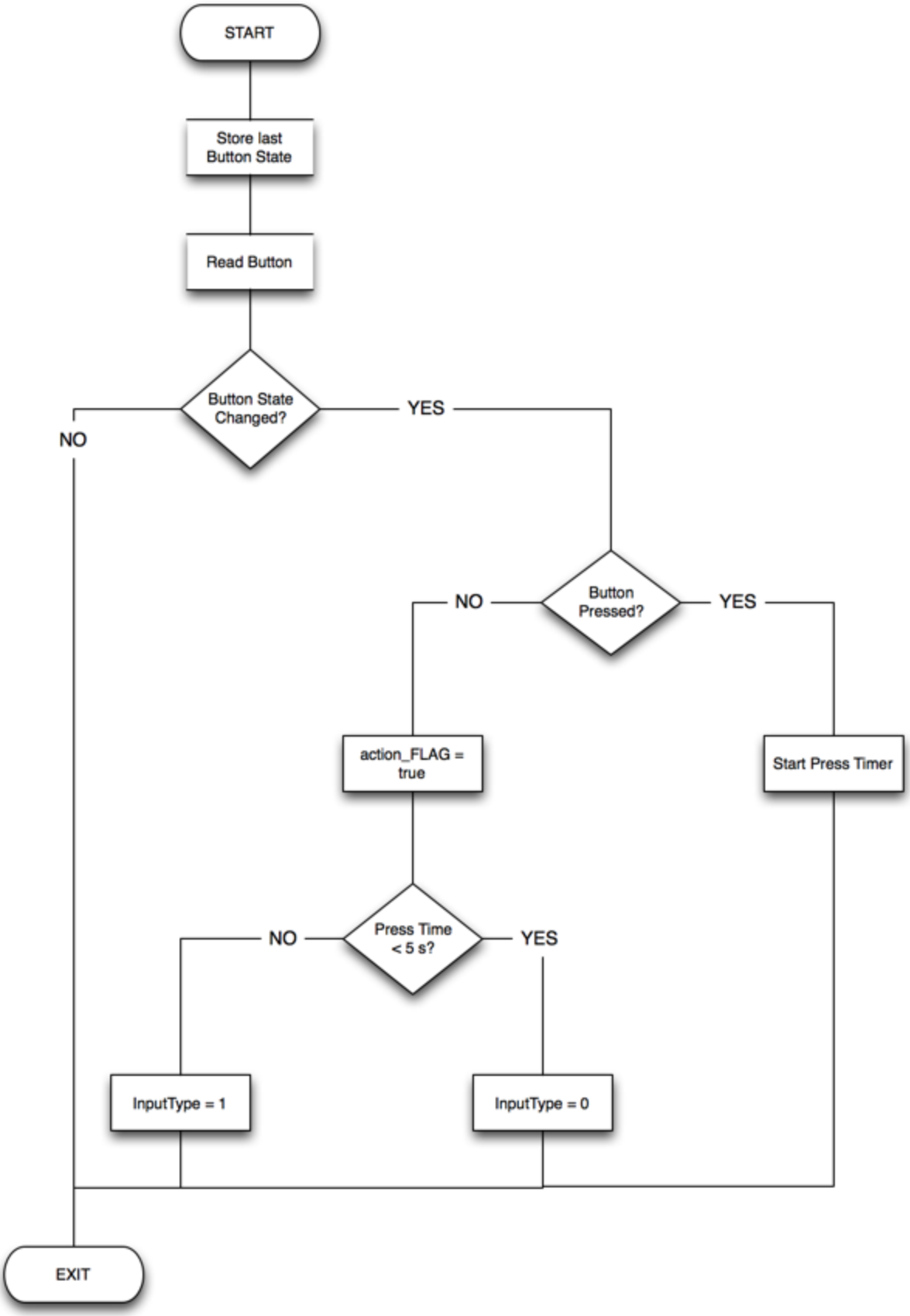
With state machines, there are 3 distinct sections to your code. Find the sample code here.

Structure

1. Read the sensors (buttonState) => Output the inputState (0 or 1 and the inputAction_FLAG)
2. Take in the currentState and inputState => Set the LED power and color
3. Pick next state based in inputState => Refer the state diagram to figure out what state is next

Section 1

Devise a logic flow that reads the button, and uses the last state of the button and a timer (to track when the button is pressed) to figure out the InputType and state of the inputAction_FLAG. My logic is below.


START
Store last Button State
Read Button
Button State Changed?
NO
YES
Button Pressed?
NO
YES
action_FLAG = true
Start Press Timer
Press Time < 5 s?
NO
YES
InputType = 1
InputType = 0
EXIT

Section 2

This section is easy. Using the current states of the system (in our case powerState and colorState), write logic to set the LED based on the current State of the system.

Section 3

Using the state diagram, write logic that uses the currentState and currentInput to select the next state.

For example, my logic needs to make sure that if the Power is Off and InputType = 0, the state of the system does not change (see how this transition arrow in the state diagram comes out and returns to the same state).

However, if Power is On and InputType = 0, the state transitions to the next state by cycling color.