



FALMOUTH
UNIVERSITY

COMP712: Classical Artificial Intelligence

Workshop: Finite State Machine (FSM)

Dr Daniel Zhang @ Falmouth University
2023-2024 Study Block 1

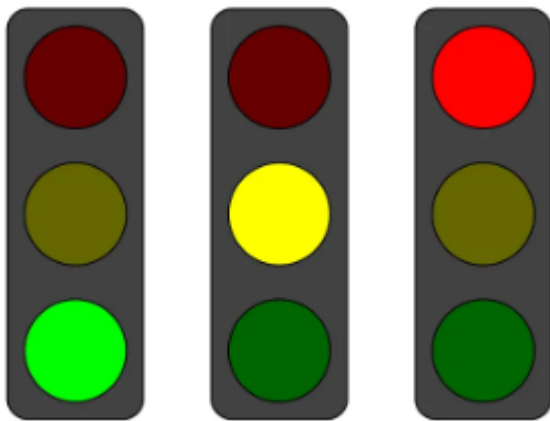


Table of Contents

- [COMP712: Classical Artificial Intelligence](#)
- [Workshop: Finite State Machine \(FSM\)](#)
- [Introduction](#)
- [Understanding Finite State Machines](#)
 - [Components of a Finite State Machine](#)
 - [A Practical Example: Traffic Lights FSM](#)
 - [The Logic](#)
 - [Adjacency List](#)
- [Task 1: Traffic Light FSM](#)
 - [The Repository](#)

- [The Code Structure](#)
- **The Task**
- [Task 2: Vending Machine FSM](#)
 - **The Task**
- [Submit Your Code](#)
- [Further Reading](#)

Introduction

[Top](#)

Finite State Machines (FSMs) are powerful tools in computer science and programming. They provide a structured way to model complex systems, making them an essential concept for any aspiring programmer or computer scientist. In this session, we will delve into the world of Finite State Machines, exploring their fundamental principles and applications. We'll also demonstrate how to implement FSMs using Python, a versatile and popular programming language.

Understanding Finite State Machines

[Top](#)

At its core, a Finite State Machine is a mathematical model that describes a system's behaviour by dividing it into a finite number of states, transitions between these states, and the conditions triggering these transitions. FSMs are excellent for representing systems with distinct modes or conditions. They are widely used in various domains, such as robotics, gaming, natural language processing, and more.

Components of a Finite State Machine

[Top](#)

Before we delve into Python programming, let's understand the essential components of an FSM:

- **States** : These represent the distinct conditions or modes of the system. For instance, in a traffic light system, the states could be 'Green,' 'Yellow,' and 'Red.'
- **Transitions** : These define the conditions under which the system switches from one state to another. Transitions are often triggered by specific *Events (It is also considered as **Conditions** or **Signals**)* or criteria.

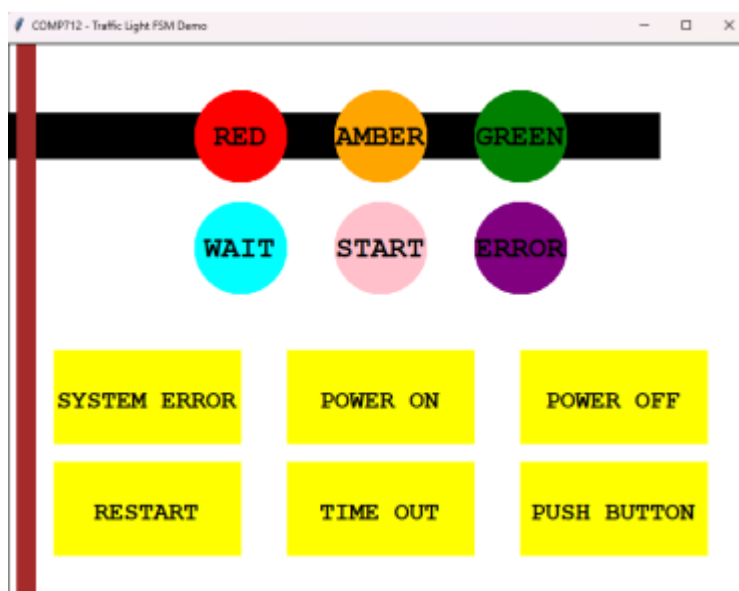
- **Actions** : Actions are tasks or operations performed when a transition occurs. They can range from simple outputs to complex calculations.
- **Initial State** : This is the starting point of the FSM, where the system begins.
- **Final State** : The final state indicates the end of a process or scenario.

Note: In certain FSMs, transitions and actions may be very similar, as no additional actions are required beyond state transitions.

A Practical Example: Traffic Lights FSM

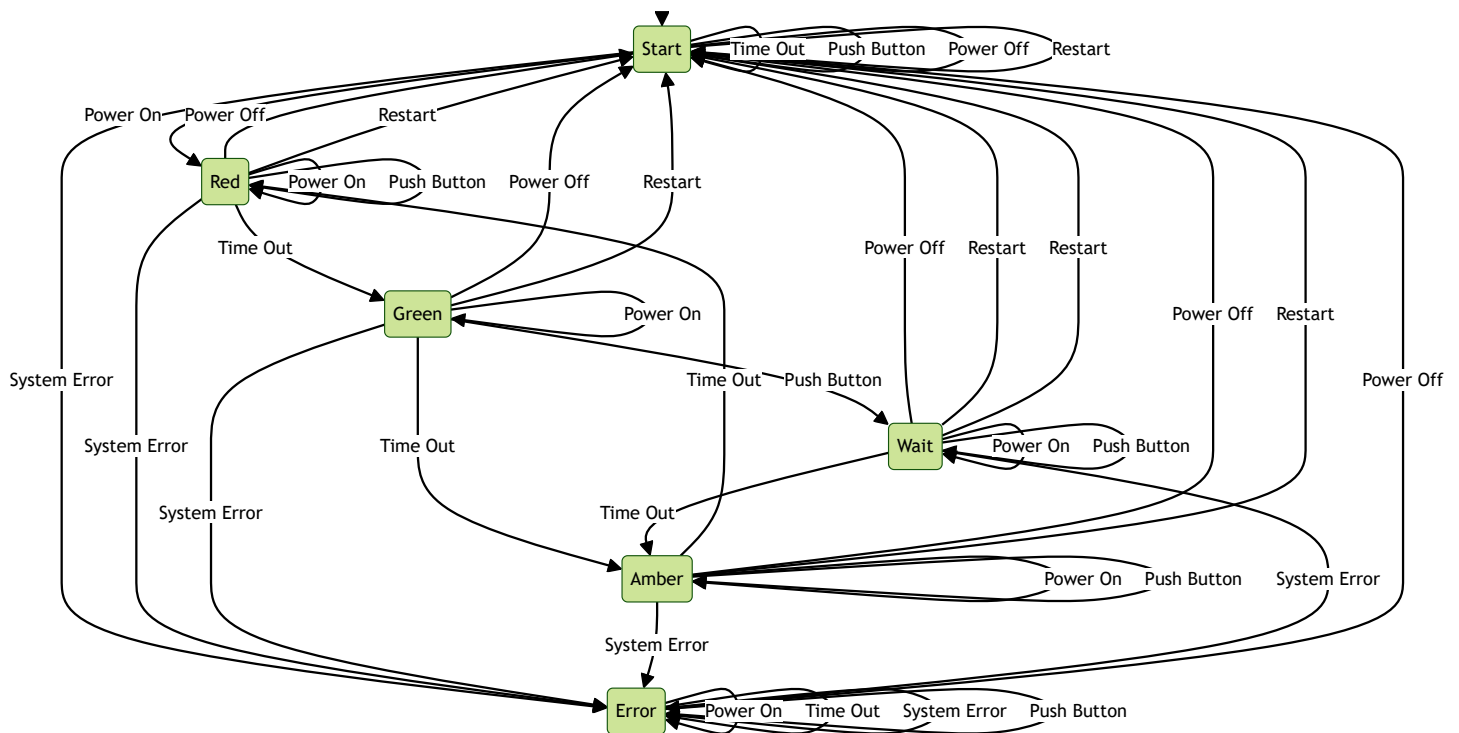
[Top](#)

Let's consider a simple example of a traffic lights model. The FSM for the traffic lights could have states like ' Red ', ' Green ', ' Amber ', etc. Signals occur as the time elapses or pedestrians pushed the button for crossing. The transition operations need to change the lights' states.



The state diagram of the Traffic Light FSM is illustrated below.

Traffic Lights FSM



The Logic

[Top](#)

The logic follows a specific sequence of states in a finite state machine (FSM) based on time intervals and certain conditions:

- From RED , GREEN , and AMBER states, after a designated timeout duration, it transitions sequentially through each state.
- Pressing the button during GREEN shifts the state to WAIT and restarts the timeout.
- In AMBER and RED , pressing the button won't trigger a transition to WAIT since it has already waited.
- After a specific timeout on WAIT , it moves to AMBER , and subsequently to RED .
- If an error occurs at any time, it switches to the ERROR state.
- Upon receiving the power-on signal, if any light is already ON, it remains unchanged except for transitioning from START to RED .
- Restart or power-off signals force the FSM to return to the START status.

When comparing the state transition diagram above to the definition in the previous section, we can describe the Traffic Light FSM as follows:

- **State** : The Traffic Light FSM encompasses the states of Start , Red , Green , Amber , Wait , and Error .
- **Signals** : The FSM responds to signals including Power On , Power Off , Time Out , Push Button , Restart , and System Error .
- **Actions** : State transitions between different states.
- **Initial State** : The initial state for this FSM is Start .
- **Final State** : The determination of the final state is context-dependent. For instance, Error might be considered the final state, although the Restart signal can trigger a transition to change the light to the Red state. Multiple states can be included in the final state set to terminate the FSM at various points.

Adjacency List

Top

In practical applications, an adjacency list becomes particularly useful when the state diagram grows too complex to explore easily. It serves as a collection of unordered lists, primarily used to represent a finite graph. Each unordered list within an adjacency list describes the set of transitions originating from a specific vertex in the graph. This representation is one of several commonly employed methods for graph representation in computer programming.

For relatively simple state machines, explicitly writing out adjacency lists might seem unnecessary. However, for more intricate systems, you'll likely find yourself creating a form of it, making it a valuable practice.

In the context of this specific discussion, we won't adhere to the precise definition of an adjacency list, as we're not describing a **"set of neighbours"** but rather a **"set of transitions"**, as shown in the table below (it doesn't list all of the transitions illustrated in the above state diagram).

State	Transition to
Start	Red OR Error OR Start ...
Red	Start OR Green OR Error ...
Green	Amber OR Wait OR Error ...
Amber	Red OR Wait OR Error ...
Wait	Red OR Error ...

State	Transition to
Error	Error OR Start ...

Task 1: Traffic Light FSM

[Top](#)

This task is to implement the FSM for this Traffic Lights model according to the state diagram above.

The Repository

[Top](#)

The repository below contains the template code of an FSM implementation in Python.

Fork the repository (NOT clone!) and work on your fork. This will enable you to submit a pull request in the end.

<https://github.falmouth.ac.uk/Daniel-Zhang/COMP712-Finite-State-Machine.git>

The Code Structure

[Top](#)

The repository contains 4 python file.

- `fsm.py` : it is the implementation of a simple FSM with all the components introduced above except `Actions`, which is the task for you to implement in a separate script during this workshop. *This design will make sure the FSM model response to any model if the transitions and actions are defined properly somewhere.*
- `fsm_demo.py` : it is a template python script that you are going to working on.
- `demo.pyc` : it is an example of completed implementation of Traffic Light FSM, which can be simulated by running `python demo.pyc`
 - It was compiled using `python 3.10.11`, which should work with all `python 3.x` version. Please let the tutor know if it doesn't work and a newer version will be updated.
- `demo_gui.pyc` : This interactive version features a simple GUI that fully implements the FSM diagram you've seen above. It aids in completing tasks by visualizing transitions between different

states. Run it using `python demo_gui.pyc` .

The Task

[Top](#)

- complete the implementation of the Traffic Lights FSM
 - make sure it works as expected
 - increase the number of actions to simulate the process longer
 - add one or more states to the model
- You don't need to create a GUI; a command-line demonstration suffices.

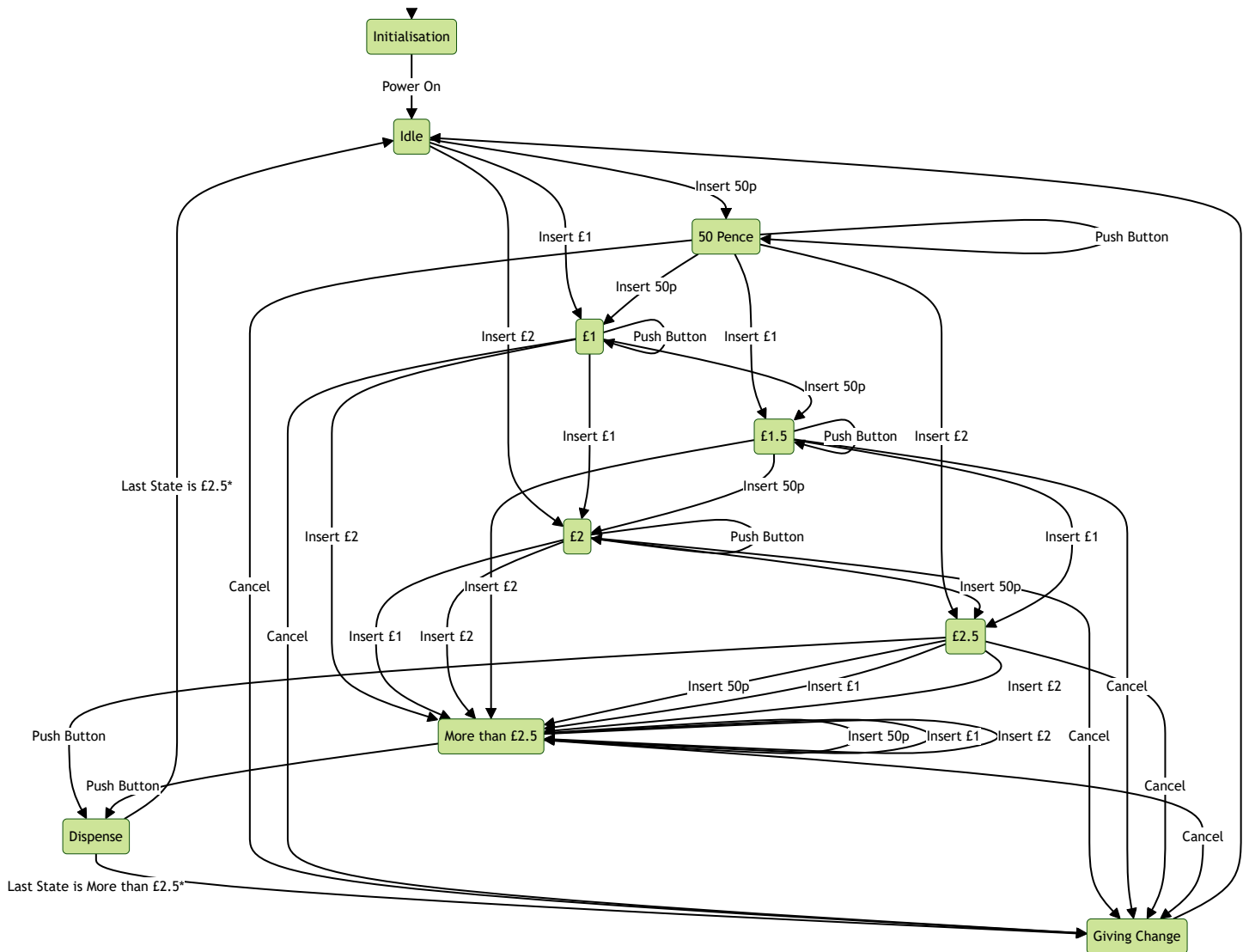
Task 2: Vending Machine FSM

[Top](#)

This task is to implement a Vending Machine FSM that sells Coca to the user. This special machine only accepts 50p, £1, and £2 coins and each can of Coca costs £2.5 (a little bit expensive 😊). Additionally, we will assume the vending machine never runs out of stock for economic reasons. Similarly, it never runs out of change. This example is a perfect computational problem to model with a finite state machine!

The state diagram is illustrated below.

Vending Machine FSM



The Task

[Top](#)

- Work out the adjacency list to help you understand the diagram if necessary.
- Create a new Python file (.py), either inherit from the FSM model defined in fsm.py or start from scratch.
- Implement the Vending Machine FSM to meet the requirements outlined in the state diagram.
- You don't need to create a GUI; a command-line demonstration suffices.

Hint: you might want to make use of the last state data member defined in FSM class.

Submit Your Code

[Top](#)

Feel free to submit a pull request to the original repository to showcase your work if you wish to share your solution with others.

Further Reading

[Top](#)

- [Wiki - Finite State Machine](#)
- [Python State Machine](#)
- [An implementation of FSM](#)