A diagram drawn on a blackboard

Description automatically generated

# Week 1 Worksheet: Finite State Machines

## Overview

The lab Objectives are:

1. Design a basic Finite State Machine (FSM)
2. Build an FSM using switch statements
3. Build an FSM with transition functions
4. Build a class-based FSM
5. Explore an example of an FSM in Unity

## Software

Primarily this tutorial work will require the use of a C++ IDE. Visual Studio is the recommended software to use, although any C++ development environment should be sufficient.

One of the tasks will require Unity. This may require you to create a Unity Account.

Visual Studio and Unity should be available on the lab machines.

There is also AppsAnywhere, that allows you access to some of the software we use. You can use this to install some of the software in the labs - <https://myapps.abertay.ac.uk/> .

## Getting Started

From the Week 1 content on MLS, download ‘**Finite State Machine - C++**’ from the “Week 1 Lab Resources” section. Extract the contents of the zip file. Open the Visual Studio Project.

Main.cpp will contain the below code. If you chose to use a different IDE or compiler setup, then copy the below code into a file and work from that.

#include <iostream>

using namespace std;

int main()

{

bool program\_running = true;

cout << "Starting Finite State Machine" << endl;

do {

/\*

Write your code here for the finite state machine example

\*/

} while (program\_running);

cout << "Ending Finite State Machine" << endl;

return 0;

}

# Task 1: Design a Basic Finite State Machine

Using the lecture slides as a guide, design a finite state machine that will demonstrate the behaviour described below. You can do this on a computer (such as a PowerPoint slide) or you can do it with pen and paper if you wish.

You will want to consider the states, transitions and overall functionality of the model.

1. A sports spectator is to have the following behaviours:
   1. By default they sit and watch the game
   2. If their team scores, they stand and cheer
   3. If the opposing team scores, they stand and boo
   4. After 15 seconds standing, for any reason, they sit down again
   5. If they are hungry at any time, they go to the food stall
   6. If they have food they sit and eat it and do not stand for any reason
   7. It takes five minutes to eat food, after which they no longer have food
   8. The spectator is never hungry while they have food

### Share and Compare

Once you have created your FSM design, compare it with 2-3 others in your area.

There are several ways you could model this behaviour, so there is no ‘right’ or ‘wrong’, so long as the desired behaviour is captured.

What aspects are the same? Are there any differences?

## Second Example

Using a directed graph, design a simple, single level Finite State Machine that could be used to create the following ‘emotional model’ for an NPC.

* Default state is CALM
* If NPC hears a noise at any time and there is no one else about then they become ALERT
* If the noise stops and nothing else happens then the NPC becomes CALM
* If NPC sees another character at any time they become AGITATED
* If other character appears to be friendly then they become CALM
* If the other character starts to act aggressively then they are SURPRISED
* If the other character is friendly and cracks a joke then they become HAPPY
* If the other character is aggressive and attacks they become ANGRY
* If the other character moves away then they become CALM

# Task 2: Building a FSM

## Switch Statements

Choose one of the examples from the previous task and implement it in a C++ program. It does not require visuals, it just needs to print messages onto the screen.

1. Create an *enum* that will contain the different states and populate it with each available state you have designed.
2. Create an instance of the *enum* at the beginning of your program. This will be used to determine which state you are in.
3. Add a switch statement to the do-while loop.
4. Add each *enum* value as a condition in the switch statement
5. For each member of the switch statement, add in the appropriate code so that it acts in accordance with your design.
6. Add in delays in your code when necessary
7. Use keyboard inputs to act as events if you need to
   1. For example, when you press the N key, it can trigger a ‘Noise’ event

Notes:

* Make sure you initialise the *enum* with a starting state
* The easiest way to implement and manage all the state code is to have a separate function for each
* The project contains both basic delay code (*sleep\_for*) and keyboard input code *(\_kbhit* and *\_getch*). Feel free to repurpose this code as you see fit.
  + The *\_kbhit* and *\_getch* commands will not work on all compilers. If this is the case, you can use whatever form of input handling you like.

## Transfer Functions

If you have not already done so, create a function that handles the transition between states. The following example code may be useful:

void swapState(STATES\* current\_state, STATES new\_state)

{

// Perform pre-switch function

\*current\_state = new\_state;

// Perform post-switch function

}

# Task 3: Class Based FSM

We are now going to rework the Finite State Machine so it uses classes rather than a switch statement.

1. Start by adding the following class into your code:

class BaseState

{

virtual void run() = 0;

};

1. Create a new class for each state of your program, making sure it inherits from BaseState.
2. In each child class, overwrite the run with code specific to that state
3. Create a single instance of each class in your program (you can do this by hardcoding them in or by using an std::map)
4. Create a pointer to a BaseState that will function as the current state the program runs.
5. Assign the pointer to a starting state.
6. Edit the transition function so it can swap the pointer from its current target to the new desired state
7. Replace the switch statement from previous tasks with something along the following lines (this is just pseudocode so it won’t work without some tweaking):



# Task 4: Unity Example

From the Week 1 content on MLS, download ‘**Finite State Machine - Unity**’ from the “Week 1 Lab Resources” section. Extract the contents of the zip file.

Open the project in Unity. If you receive any messages about versioning, select to use the latest installed version.

If you have not used Unity before, you may find the ‘Getting Started with Unity’ link on MLS useful.

## Exploring a FSM

The following graph indicates the structure of the AI in the game.

A close up of a logo

Description automatically generated

1. Run the game and view the AI running
   1. Write down what the events are the trigger a state change
2. Add a new state to the game where if the AI is chasing the player and the player is above them, try and get the AI to jump as well. Return to either chase or patrol state after the jump is complete.

# Open Ended Bonus Work

Invent something creative that extends the current Unity Finite State Machine. Extend the AI in any way that takes your fancy. You could add attack methods and states, add a stunned state if the player collides with the bot’s head etc.

You could also extend one of your C++ FSMs to incorporate one of the additional FSM models we covered in the lectures, such as Stack Based or Hierarchical FSMs.