Building a MIDI processor using Markov models

In this section you are going to carry out some experiments with the variable Markov modelling library I have created. In a later section you will adapt the project you create here into a full JUCE MIDI processing plugin.

Preparing the CMake project

I have created a starter project for you. The project includes a Markov model library that I have created in C++. The latest version of the library is available on Github[1] but the starter project includes a 'current-at-the-time-of-writing' version of the files for simplicity. I would like to make you aware of some features of the starter project:

1. The project is a JUCE plugin project
2. The plugin target has 'IS_MIDI_EFFECT', 'NEEDS_MIDI_INPUT' and 'NEEDS_MIDI_OUTPUT' set to TRUE as it will be a MIDI effect-type plugin
3. The project includes the files for the Markov library I have created
4. The project is configured to build the Markov library as a separate module which is a good idea as the library should not need to change. If you do want to customise it, you can do that of course and the build system will detect that and rebuild the library
5. The project has an additional executable target called markov-expts which is a simple, non-JUCE dependent executable you can use for simple experiments

Once you have the starter project downloaded and unpacked, generate the project in your usual way, open it up in your usual IDE and build it.

Build a Markov Model

Let's dive straight in and build some Markov models in the markov-expts project. Open the file 'MarkovExperiments.cpp'. You should see an empty main function in there. Add some lines to the main function as follows:

```
#include "MarkovManager.h"

int main()
{
    MarkovManager mm{};
    mm.putEvent("A");
    mm.putEvent("B");
    mm.putEvent("A");
```

[1] https://github.com/yeeking/MarkovModelCPP

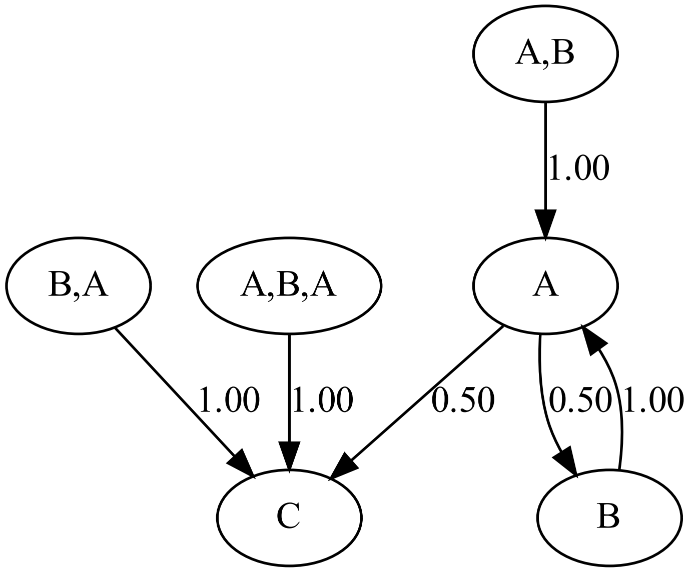

Figure 1: Example of the Markov model generated by the code shown in figure [code:markov-first-example]

```
    mm.putEvent("C");
    std::cout << mm.getModelAsString() << std::endl;
}
```

[code:markov-first-example]

You should see some output like this:

```
1,A,:2,B,C,
1,B,:1,A,
2,A,B,:1,A,
2,B,A,:1,C,
3,A,B,A,:1,C,
```

Considering what you know about Markov models, what do you think that output represents? Figure 1 illustrates the model as it stands after running that code.

You can pass any strings you like as the states for the model. Here is the signature for the putEvent function from MarkovManager.h:

```
    void putEvent(state_single symbol);
```

'state_single' is the type for the argument. If you ctrl-click on that type in your IDE, you should see this, from MarkovMode.h:

```
typedef std::string state_single;
```

This means that state_single is just a renamed std string type.

Generate from a Markov Model

Now you have a model you can use it to generate a new sequence. Here is a for loop that queries the model:

```
for (auto i=0;i<5;++i){
    state_single next = mm.getEvent();
    std::cout << "Next state " << next << std::endl;
}
```

I see output like this:

```
Next state B
Next state A
Next state C
Next state C
Next state C
```

It will be different each time. What is going on here? This is a variable Markov model, so that means it tries to find the highest order output that has an available observation following it. You can query the model for the order of the previous event using the getOrderOfLastEvent function:

```
for (auto i=0;i<5;++i){
    state_single next = mm.getEvent();
    int order = mm.getOrderOfLastEvent();
    std::cout << "Next state " << next << " order " << order << std::endl;
}
```

I see output like this (which might be different each time):

```
Next state C order 0
Next state A order 0
Next state B order 1
Next state A order 2
Next state C order 3
```

Here is an explanation of that:

1. Next state C order 0: the manager has no memory as this is the first event. So it selects a random state from the distriution of all observed states (A,B or C) - this is zero order.
2. Next state A order 0: the manager now has a memory of 'C', so it starts by trying to look up states following on from 'C'. There are no states observed after 'C' so it again samples from the distribution of all states (zero order)
3. Next state B order 1: the memory is now 'C,A' so it first looks for states following 'C,A' (2nd order) and there are no observations following that. Then it tries states following 'A' (1st order) and it finds two options: B and C. It chooses B - first order.
4. Next state A order 2: the memory is now 'C,A,B', so it tries states following 'C,A,B', of which there are none, then 'A,B'. 'A,B' has one observation following it: 'A'. So you get 'A' which is second order since the lookup as 'A.B'
5. Next state C order 3: now we use a third order lookup of 'A,B.A' and choose the only option which is 'C'.

Things to explore with Markov modelling

Try creating a larger model and generating from it. For fun, try some experiments with words. You could read words from a text file, build a Markov model of the word sequence then generate new sequences in that style. Another interesting thing to do is to create a chatbot that builds a model of the person it is talking to as the conversation proceeds. One of my first experiences with artificial intelligence was interacting with a similar chatbot on my Commodore Amiga computer in the early 1990s. A cover-disk from an Amiga magazine contained several examples of chatbots and one of them worked like this. I am not sure how sophisticated the underlying model was or exactly what kind of model it used.