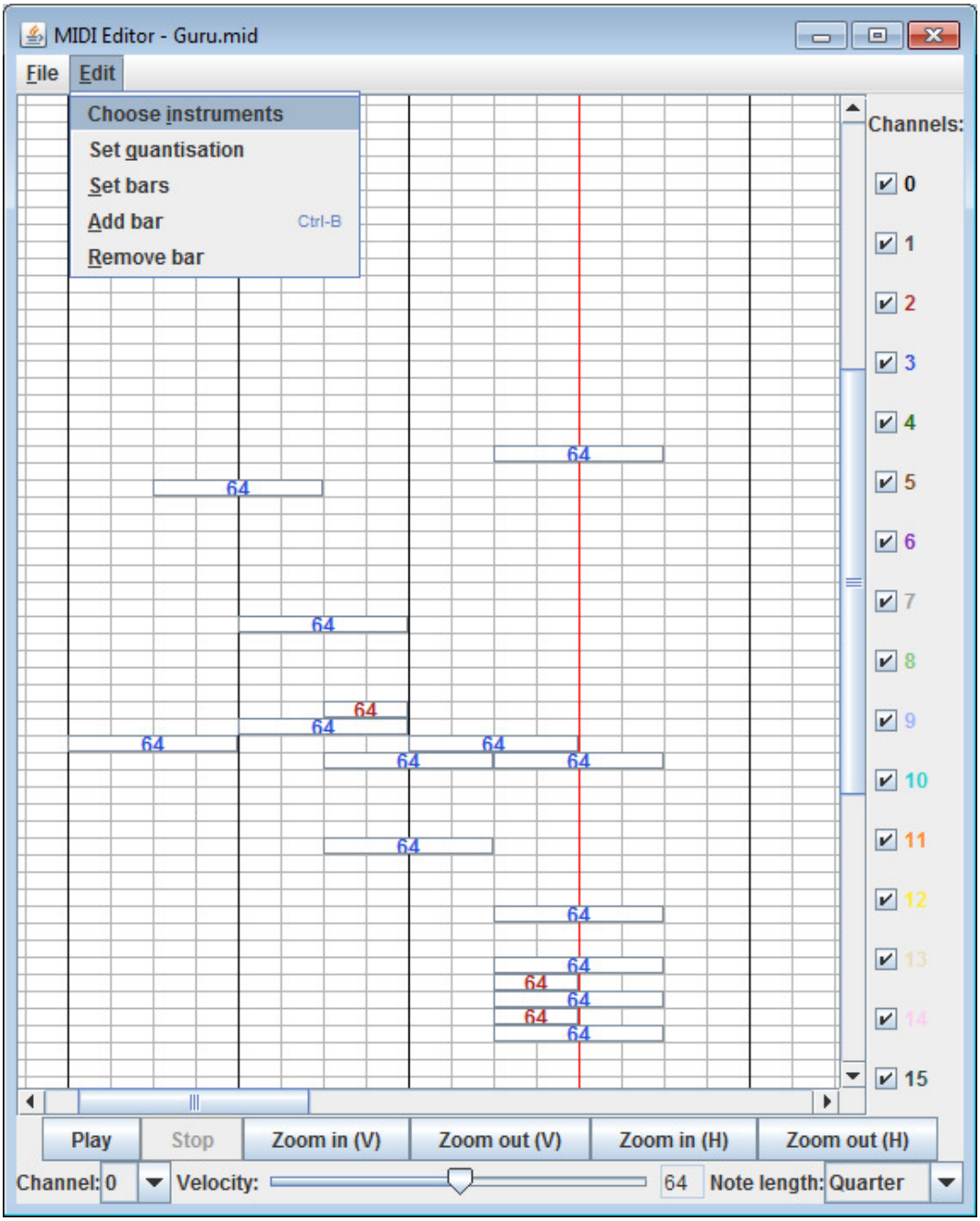
CS1006 Project 4 – MIDI Editor

Introduction

This practical is about implementing a MIDI sound file editor. MIDI files do not represents sounds, instead it represents a number of events from which sounds can be generated. The events explored in this project are NOTE_ON, NOTE_OFF and PROGRAM_CHANGE. The MIDI format is discussed in more detail in “The MIDI Format”-section.

Sample output



Discussion of Sample Output

The Midi Editor allows the user to start playing start and stop playing of a MIDI file, zoom in/out vertically (Zoom in (V) and Zoom out (V)) and horizontally (Zoom in (H) and Zoom out (H). Add new notes of channel, velocity and note-length selected using the graphical user interface components at the bottom of the window.

The MIDI format

There are two types of MIDI files: Type 0 and type 1. Type 0 only allows for one track onto which MidiEvents (such as NOTE_ON, NOTE_OFF and PROGRAM_CHANGE) can be added, while type 1 allows for several track. A track can contain up to 16 channels which can play different instruments simultaneously. Each note has a set key number (0-127, 127 being the highest), note velocity (0-127), channel (0-15) and note length (usually whole-, half-, quarter-, eighth- or sixteenth-notes). Instruments can be changed with PROGRAM_CHANGE events, and only instruments from a single sound bank can be played at the same time. Thus, the MIDI player is limited to a selection of 128 different instruments, but only up to 16 of them can be played simultaneously. The timing of a MIDI file is measured in ticks, and in this projects implementation there are 96 ticks per beat. The implementation in this project only allows one instrument per channel (i.e. the program change events are added at tick 0, and the instrument on a given channel cannot be changed while playing).

Key Features of Implementation

- Ability to add create new and load existing MIDI files of type 0.
- Ability to play type 0 MIDI files.
- Ability to add new notes with specified key, channel, velocity and note-length and time.
- Ability to remove existing notes.
- Ability to change the program (instrument) of each channel.
- Zoom in and out, vertically and horizontally.
- Filter which channels' note on/off events should be displayed.
- Quantisation features which shifts the start time of added notes to the nearest boundary, i.e. the nearest 1/4 note, 1/8 note or 1/16 note.
- Ability to set the length of a MIDI file in bars, allowing for more notes to be added.
- Title of the window changes as new files are created, files are saved or opened.

Design

Explanation of Classes

Class	Purpose
midied.Constants	Defines constants which are used elsewhere.
midied.FileChooser	Lets the user open or save a file of a given file format.
midied.InstrumentChooser	Lets the user choose which instruments each of the channels represent.
midied.MIDIEd	The window of the program, deals with the buttons (such as play and zoom), and the graphical interface components which deal with the properties of each note to be added, and the menus.
midied.NoteButton	Graphically represent single notes.
midied.NoteLength	Represents possible note lengths.
midied.PianoRollPanel	Deals with the editing and graphical representation of the notes in a MIDI file.

Adding notes

Notes are added when the user clicks the mouse anywhere on the PianoRollPanel (i.e. the grid in which notes are displayed). This is done through use of a MouseListener which adds NOTE_ON and NOTE_OFF MidiEvents to the track and NoteButtons to the PianoRollPanel's content pane. In order to understand what the first and second byte of ShortMessages mean the Midi.org was accessed (1).

Removing notes

Removing notes is done by clicking on existing notes. This feature is implemented through use of an ActionListener which listens for clicks on NoteButtons. When a NoteButton is pressed, the source of the click is obtained using the ActionEvent.getSource() method. This method returns an Object which, if found to be an instanceof NoteButton is cast to a NoteButton and then passed into the following method:

```
private void removeNote(NoteButton but) {
	// remove MidiEvents from track
	but.remove();
	// remove from grid
	remove(but);
	repaint();
	changeMade = true;
}
```

The MidiEvents associated with this button are removed from the track, the button is removed from the grid and the pianoRollPanel is repainted. The changeMade variable is used to determine is changes need to be saved before creating a new sound file, opening a different sound file or closing the program.

Note-channel

Midi files support up to 16 channels, which can all have separate notes playing simultaneously with different instruments. The GUI component used to choose which channel new notes are added onto is a ComboBox, this is suitable, because it takes up less space than RadioBoxes or a slider bar and represents a finite number of options.

Note-velocity

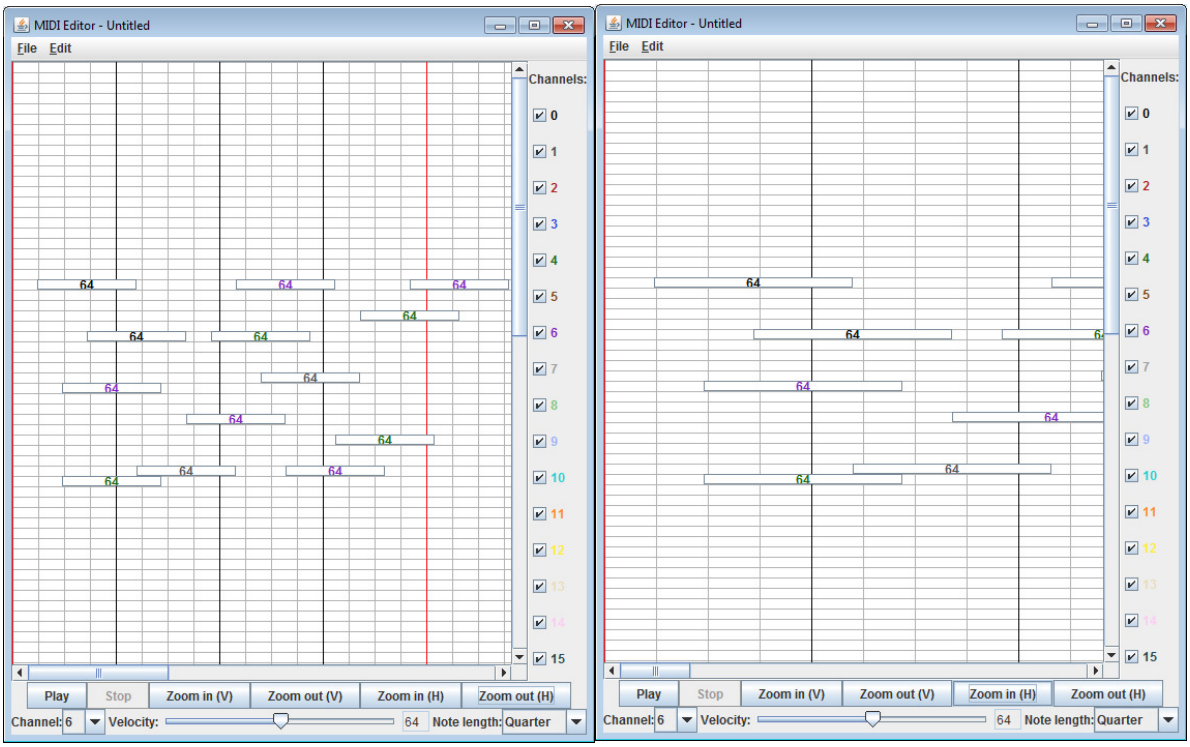
The note-velocity is the velocity at which a note is played. This effects the volume at which it is played, and can be compared with the intensity at which a key on a piano or keyboard is pressed down. The chosen GUI component is a JSlider which allows the user to choose any integer value between one and 127. A text-field next to the slider shows the currently chosen velocity. Alternatives to a JSlider could be to use a single textfield or a ComboBox, but a slider gives a better graphical representation of the velocity relative to how fast or slow it can be. Additionally, A ComboBox would not be appropriate, as the user could have to scroll through over 100 items to get to the desired velocity.

Note-length

The note-length can be set to five different values: Whole-, half-, quarter-, eighth- and sixteenth notes. The note length determines the time interval between the note-on and note-off events, in other words, how long a note is played for. The GUI component chosen to represent the note-length of new notes is a JComboBox, this allows the user to choose between a finite set of numbers. Alternatives to this solution could be radio-boxes, slider (as used for the velocity), or simply a text-field. However, radio boxes take up much more space, a slider would not be that appropriate as it is not desirable to have any value between one and a 16^{th}, and text-fields can be said to not be as user-friendly.

Horizontal zoom

The second picture is taken of the same piece of music, but is horizontally zoomed in.



This feature is implemented in the using the following code:

```
public void modifyHorizontalZoom(boolean in) {
	if (in && (horizontalZoom < 10)) {
		horizontalZoom++;
	} else if (!in && (horizontalZoom > 1)) {
		horizontalZoom--;
	}
	setHorizontalZoom(horizontalZoom);
}
```

```
private void setHorizontalZoom(int zoom) {
	beatWidth = horizontalZoom * NOTE_WIDTH_UNIT;
	beatScaleFactor = 100F / beatWidth;
	update();
}
```

The modifyHorizontalZoom method simply increases or decreases the horizontalZoom variable and calls setHorizontalZoom(int). setHorizontalZoom updates the beatWidth and beatScaleFactor and update() just updates the visible representation of the notes which are displayed.

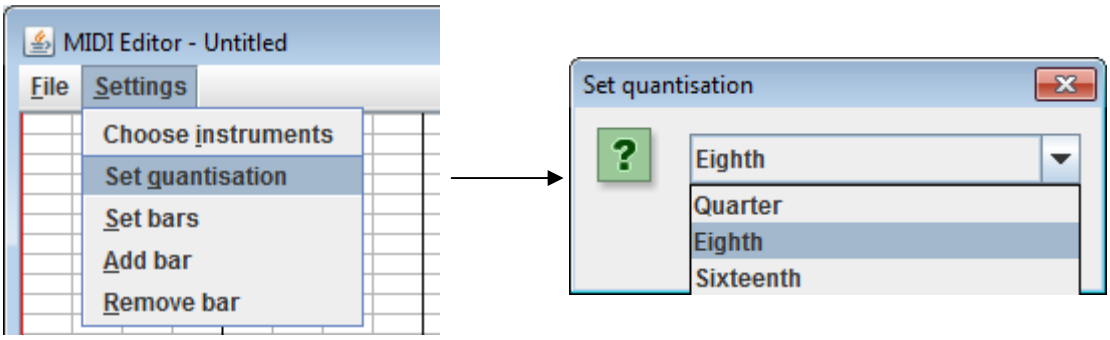
Play thread

The Play-button starts a thread which starts playing the open Sequence using a Sequencer. Using a Thread here makes it possible to add additional notes while the music is being played. The thread is required because the window freezes otherwise, and therefore it also allows the stop button to function.

Quantisation

Quantisation is “the process of converting a continuous range of values into a finite range of discreet values.” (2) In the case of the Midi Editor it refers to the process of correcting the timing of notes.

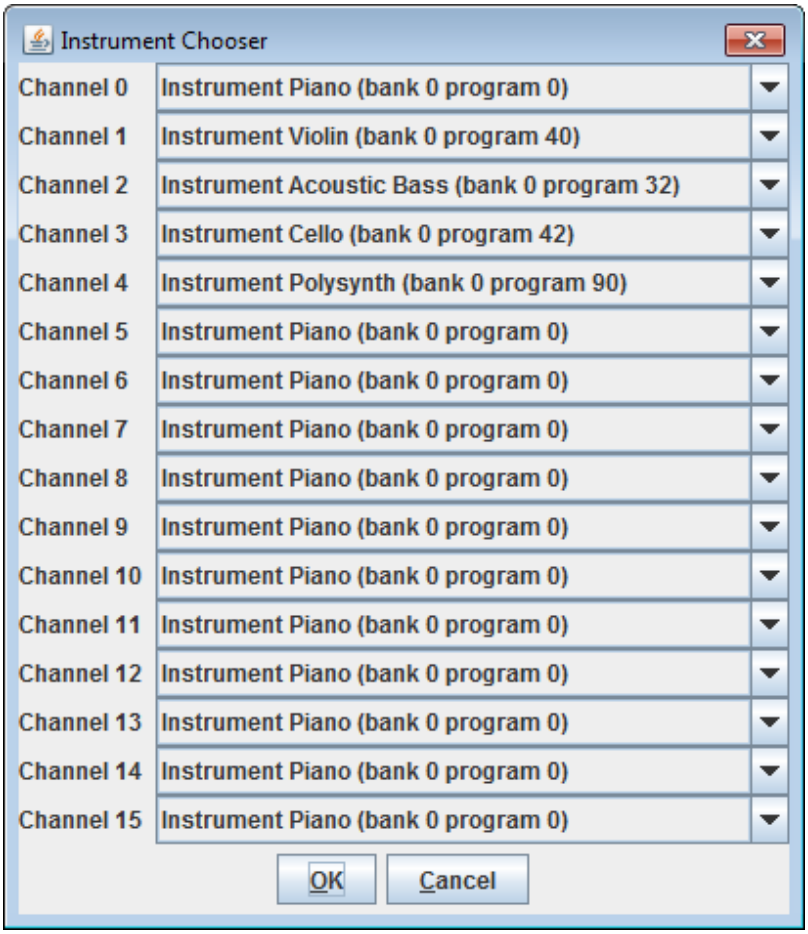
Quantisation is dealt with using the menus and a JComboBox displayed using JOptionPane.showMessageDialog:



This gives the user the possibility to make notes "stick" to intervals of a quarter, an eighth or a sixteenth of a note.

Program changes

Program change events are added using the "Choose instruments" option in the Settings menu. When it is clicked the following window is displayed:



This allows the user to choose between 128 different instruments for each of the 16 channels. When the instruments are chosen, program change events are added at tick 0 on each of the channels whose instruments are not program 0 (Piano).

Program change events are discovered in the findNotes method and stored in an array (programEvent) with up to one event per channel. When program changes are added, these events are also added to that array, so that they can be updated later (i.e. remove the old instrument event and create a new one).

Program change events are generated in the following way in the code:

```
public void changeProgram(int channel, int program) {
      if (getProgram(channel) == program) {
            // interrupt if the chosen program is the existing one.
            return;
```

```
        }
        if (programEvent[channel] != null)
                track.remove(programEvent[channel]);
        programEvent[channel] = createProgramChangeEvent(channel, program);
        track.add(programEvent[channel]);
        changeMade = true;
}

public MidiEvent createProgramChangeEvent(int channel, int program) {
        // The first zero has no function, the second zero is the tick.
        return createMidiEvent(ShortMessage.PROGRAM_CHANGE, channel, program,
                        0, 0);
}

public MidiEvent createMidiEvent(int command, int channel, int data1,
                int data2, long tick) {
        ShortMessage onMessage = new ShortMessage();
        try {
                onMessage.setMessage(command, channel, data1, data2);
        } catch (InvalidMidiDataException e) {
                theFrame.reportCriticalError(e);
        }
        return new MidiEvent(onMessage, tick);
}
```

In order to change the program of a channel a ShortMessage with the PROGRAM_CHANGE command, a specified channel, program, 0 as the second byte and tick 0 has to be added to the track.

With the current implementation it is not possible to change instrument in the middle of a track, only at the very beginning of each channel (i.e. at tick 0).

Testing & Limitations

A number of sample files have been included to show what the Midi Editor is capable of.

During testing a few of errors were discovered:

- Quantisation can make two notes appear on top of each other, which makes a different sound than if there were to only be a single note played.
- If notes are added at tick 0 and the instrument chooser is then used to choose different instruments, the notes added at tick 0 will still be played using program 0 (piano).
- If a MIDI sequence is played 17 times the MIDI Editor will no longer be able to play them.

Limitations include:

- Difficulty in knowing which key a note is representing. This could have been solved by having a keyboard on the left side of the PianoRollPanel.
- Inability to handle different type signatures than the 4/4 currently assumed.
- Inability to handle type 1 files. I.e. several tracks. This could have been dealt with using separate tabs for each of the tracks.
- Inability to add or display program change events after the first tick.
- If the default Java Synthesizer's bank does not contain 128 elements an error will occur. This assumes that the default Java Synthesizer is not the same for every computer.

Conclusion

In this project I created a program capable of editing type 0 MIDI files. It is capable of creating new, loading and saving files, adding (with specified key, channel, velocity, note length, and start time) and removing notes, program changes, vertical and horizontal zoom, filtering channel visibility, quantisation of notes and has the ability to set the length of a MIDI file. This means that it is possible to create rather complicated pieces of music using the MIDI editor. However, it is quite difficult to hit the right keys, because there is no way, other than counting lines, to see which key a note is.

If more time was given to the project, the following features could be added:

- Support for type 1 MIDI files.
- Support for instrument bank changes.
- Support for program changes anywhere on a channel (i.e. after tick 0).
- Ability to copy and paste notes.

Bibliography

1. MIDI Message Table 1. *MIDI Manufacturers Association.* [Online] [Cited: 1 May 2010.] http://www.midi.org/techspecs/midimessages.php.

2. Quantization. *Media College.com.* [Online] [Cited: 7 May 2010.] http://www.mediacollege.com/glossary/q/quantization.html.