Lisp in Summer Projects Submission

Submission Date	2013-10-23 17:55:29
Full Name	Janne Nykopp
Country	Finland
Project Name	Notewhacker
Type of software	gui app
General category	game
LISP dialect	Commmon Lisp
GitHub URL	https://github.com/jnykopp/notewhacker
Did you start this project?	Yes, all the code is written by me
Project Description	I want to describe my project in this form.
Purpose	Notewhacker is a game for learning to read sheet music: Random notes or chords scroll on the game screen. The player has to hit the corresponding keys (or buttons etc.) of the instrument. Duration and rhythm are ignored. Emphasis is on muscle memory, learning note names is not needed.
Function	The game displays a staff and randomized chords of notes, generated at certain intervals. Chords scroll from right to left with leftmost chord being a target chord. The game also reads MIDI data from an instrument. When notes for the target chord are played, player gets points, scrolling speed increases, and the target chord is removed. Wrong notes are displayed with red note markers. When a chord scrolls off the staff, a life is lost and scrolling

	speed halves. When all lives are gone, score is shown, and player is offered a restart.
Motivation	I play accordion. Most accordions have a bass system — Stradella bass — which is quite different from other instruments. Existing games similar to Notewhacker do not support Stradella bass. Also, combining MIDI accordion and Common Lisp together was fun!
Audience	This game is created for anyone who wants to learn to read sheet music and has a MIDI instrument. It is especially created for accordion players, as in future there will be an option for special handling of Stradella bass.
Methodology	* Graphics The game uses OpenGL for graphics. Each string, number, notehead, clef, and accidental is a textured quad. Vecto is used to render TrueType fonts into textures. Visual elements are drawn as a collection of GL-quads and lines. I first tried to use Lispbuilder-sdl's surfaces but couldn't get alpha blending to work properly, and changed to OpenGL. Using effects, e.g. rotating, scaling, color change, and fading seems to be easier with OpenGL. Graphics are designed to scale with the window size, though implementation's not ready yet. All coordinates (except the last-minute code) are defined as multiples of note head width and height, and textures can be scaled thanks to TrueType fonts. I'm planning on using Reactive Programming by http://common-lisp.net/project/cells/ library for propagating values. * Midi Midi part is Unix-specific at the moment. OSS Midi device is opened for reading. There is a separate thread which reads the device perpetually and fills a ring buffer with read octets. Reading is asynchronous so the reading thread can be stopped at any time. Common Lisp standard doesn't offer asynchronous I/O. Notewhacker

now only works with SBCL and CCL, which have implementation specific asynchronous I/O (i.e. read-byte with short timeout). Other Common Lisp implementations may offer similar interfaces. They aren't yet supported.

This approach seems more straightforward than using asynchronous I/O libraries which require system level C-libraries.

* Sheet music typesetting

Chords can be drawn in different key signatures and for G and F clefs. For key signatures, each staff has a table for mapping Midi key number modulo octave to a choice of positions, as a pitch can sometimes be drawn in two ways: e.g. key 68 can be drawn as G♯4 or A♭4. The clef and key signature can be changed on the fly so that only chords created after the change will be drawn relative to the new clef or key signature. This allows exercises on changing key signature, for example.

Typesetting works a chord at a time. The algorithm takes the set of Midi key numbers (generated targets or midi data) and staff, and outputs a list of drawing commands to display the chord, including accidentals, ledger lines etc. Noteheads and accidentals very close to each other are drawn so that they won't overlay each other, i.e. upside down or translated on x-axis.

* Engine

Lispbuilder-sdl is used for event handling, game window management etc.

Conclusion

The game is playable and it works well. It can already be used to practice notes in G-clef with C-major key.

Unfortunately I couldn't devote much time to this project so many things are unfinished. With little effort, changing key signature can be implemented. Most of the code for this is there already.

Second staff line with F-clef is also something that I will implement specifically to get the Stradella bass part covered. Good amount of this work is also done already.

Target notes are now completely random. I plan on a system with chords being created according to e.g. scales. Notes where player makes often mistakes should be emphasized.

Window resizing is not implemented at all. This is something I want to try and write with the Cells library. Also I will create a menu system which can be operated with the Midi instrument. These require more work.

Also usability should be improved. User shouldn't be required to modify the code to define the midi device. The game could detect the right device or offer the user to enter it upon startup.

Build Instructions

Easiest is to use quicklisp. Download the code of this project (https://github.com/jnykopp/notewhacker/archive/master.zip) and move the extracted directory "notewhacker-master" to quicklisp's local projects directory with name "notewhacker". Then load the project using quicklisp: (ql:quickload "notewhacker").

Test Instructions

The test cases can be run after building by following commands:

```
(5am:run! 'notewhacker::raw-buffer-tests)
(5am:run! 'notewhacker::matcher-tests)
(5am:run! 'notewhacker::midi-tests)
```

Execution Instructions

Before starting the program, make a change to file midi.lisp.

Change the row

```
(defparameter *midi-device-pathname* #p"/dev/midi1"
```

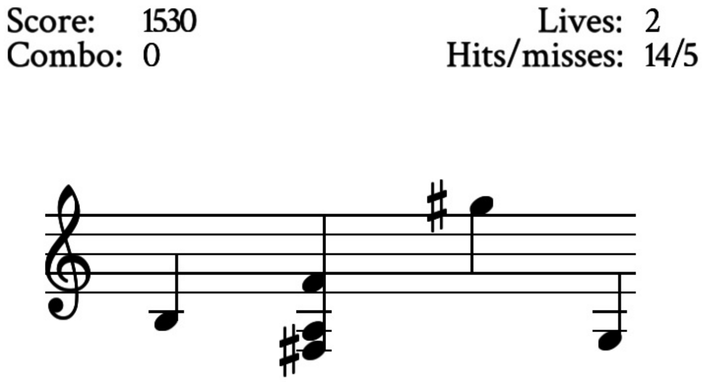
to point to your midi device file. Then recompile the defparameter or re-quickload by (ql:quickload "notewhacker").

Start the game from REPL with command (notewhacker:main).

Describe any bugs or caveats

If the midi device file is missing, the midi reader thread will throw an exception which is uncaught.

On some graphics cards all of the textures have a thin transparent line going across them. This happens e.g. on Nvidia Quadro NVS 4200M. The software was developed using Intel HD4000 graphics card, which doesn’t expose this bug.

Screen shots	 Score: 1530 Combo: 0 Lives: 2 Hits/misses: 14/5 notewhacker2.png
Official	I have read rules and have abided by them. I am 18 years of age or older. I am not living in Brazil, Quebec, Saudi Arabia, Cuba, Iran, Myanmar (Burma), North Korea, Sudan, or Syria.

Text