MidiHero

Real Music in its Most Addictive Form

by Landon Judkins

Summary

MidiHero is a musical training game that applies all the principles of game learning to teaching people to play real instruments. The game is going to be modeled after successful song scrolling games such as DDR and GuitarHero but certain modifications will be made to make it effective as a tutorial program. The most prominent of these are that the player should be playing actual music with only information being provided by the game, and that the game will provide choice in gameplay. The player will interact with the game using a MIDI Controller such as a MIDI Keyboard. MidiHero will finally be a musical game that actually develops musical skill.

Analysis

MidiHero's core mission is to be an entertaining game that develops musical skill. This is a primary innovation of MidiHero as well. Unlike previous music games like DDR and Guitar Hero, becoming an expert MidiHero player will actually hold some value because the player will have developed useful musical skills. This mission requires that MidiHero interface with real instruments, cultivate musical ability, and still be an entertaining game. Much of the overall game design can be borrowed from existing music games, but a few key changes need to made. Whereas DDR and GuitarHero can present multiple difficulties for the same song easily, MidiHero has more restrictions on what it can have the player do since all of the input maps to MIDI output. In Guitar Hero for instance, an effort is made to make the notes seem to match the song but in reality there is no one-to-one correspondence between input and audio output, and the Guitar Hero controller is in fact incapable of representing enough notes to actually play the song. In MidiHero, the song can certainly be presented at different speeds and with different grading criteria, and the game may also isolate individual tracks, wait for player input, or any number of other training features, but the notes to be played need to map to real MIDI events. However, there isn't any reason that MidiHero can't provide choice for what notes to play. This will be the other main innovation in MidiHero. Instead of providing purely single-track linear game-play, MidiHero will allow for multiple paths through timing based pattern recognition. Amplitude, another simplified music scroller, allows the player to move between tracks, but this is done using track selecting input and does not allow divergence within the note playing interface. MidiHero relies solely on the MIDI Controller interface so the pattern based selection system will be necessary but also more appropriate because it will not break the flow of the music. This will not only allow the player to choose how the song will sound, but it will hopefully provide a MIDI game interface that could be easily used for other types of games like 2D platformers or even fighting games.

This interface presents a number of opportunities which MidiHero will explore. The first of these will be the use of multiple instruments by a single player. Using customized sound banks, range specific synthesizer relays, or a number of other approaches, MidiHero will allow the player to be using more than one instrument not only in the same song, but at the same time. For example, if the player were playing a song with a drum and guitar track, these could both be played at the same time using the right hand for the guitar and left for the drums using a standard percussion sound bank.

The game will also need a means of effectively presenting the desired user input (song) to the player. This will be accomplished by a right to left scroller much like the scroller found in DDR, but it will obviously need to be able to represent a much larger range of input. Initially this will be implemented by simply displaying the name of the note and having rows for all the different notes arranged sequentially, but a more refined version of this or a semi-standard staff notation may prove necessary.

A major part of MidiHero's effectiveness as a music tutorial and a game will rely on its ability to produce options in such a way that every path still sounds musical. Luckily, many of the basic properties of music are fairly objective. Transposition across scales is an example of musical capability

that is relatively simple to implement. MidiHero will include banks of musical phrases or riffs which it can transpose to fit in many different types of music. MidiHero may also dynamically generate even the building block musical phrases themselves, possibly through the use of existing software. The target experience that MidiHero aims to provide is that of assisted and evaluated improvisation. To put it more loosely, the game will “jam” with the player while also providing the player with music to play along with.

Requirements

Instant feedback: Feedback should be seemingly instantaneous. Input responses will be displayed in the render immediately following the input if possible, and multi-period algorithms will be displayed immediately following their completion. MidiHero will run at 60 FPS.

Music presentation: MidiHero will offer some variant of the standard staff notation. This variant should be readable to anyone who can read standard music notation with minimal explanation, and once a player masters reading the variant, they should be able to read sheet music with minimal explanation. There will also be an alternative musical notation that is immediately readable to people who cannot read the staff. This may resemble the prototype's display which basically writes out the note as a string and places it on the appropriate row.

Effective Learning curve: MidiHero should not be a frustrating game to learn. The game should adequately provide challenging but reasonable game play for all skill levels from novice to intermediate musicians. This will be verified by user testing (see Testing Plan).

Visual Cues: The target audience for this game usually respond well to visually cued timing so MidiHero will provide this in a similar fashion to Guitar Hero and DDR. These cues will include, but not be limited too, beat bars, frame divisions, and note position.

Beat Representation: MidiHero will recognize the beat of the sequence and provide this information to the player both visually and acoustically. The visual cues will likely take the form of beat bars and/or frame divisions, while the audio cues may take the form of many MIDI sounds varying from handclaps (DDR) to a simple metronome. The audio cues will of course be optional and possibly only available in training or lower difficulties.

Sequence Playback: MidiHero will be able to play back the sequence partially or entirely in time with the player's display and grading. Specifically, MidiHero will be able to handle the case of playing one or more tracks off of a sequence with a number of other tracks remaining by playing back the remaining tracks in addition to playing the player's tracks at an independently adjustable volume factor, which could be zero.

Dynamic Music Generation: Through the use of transposition and categorized musical phrases (i.e. Jazz licks, classical ostinato, etc.), MidiHero will create music dynamically based on the users choices. This dynamic process will support key transitions (i.e. C Major to D Minor) during gameplay. This will include both a change in the options presented to the player but also in the accompaniment, which will still need to match the player. MidiHero may also incorporate the generation of musical phrases possibly with the use of external libraries.

Appeal to music gamers: MidiHero should appeal to gamers who enjoy GuitarHero, DDR, and other similar games.

MIDI Controllers: MidiHero will be for MIDI Controllers exclusively. The game will handle audio output for player input so a MIDI Synthesizer will not be necessary.

Complete MIDI Controller Interface: MidiHero will only require the use of a MIDI controller as input. There will also be keyboard shortcuts, but all interactive sessions will present a MIDI interface. There will of course be QWERTY keyboard commands and shortcuts, but none will be necessary for

standard game play.

Input Playback: MidiHero will optionally play back the MIDI input from the player for a variety of sound banks. This will allow MIDI controllers to be used in addition to MIDI synthesizers. Most importantly, MidiHero will support multiple instrument playback simultaneously.

Persistent Player Data: Player profiles will contain persistent information about the player's previous records. This will include but is not limited to high-scores, best grades, longest streak, and unlocked songs.

Platform: MidiHero will run on Mac OS X and Windows without installation or different versions. The folder containing the executable should be able to be moved so if it were on a Flash drive, it could be played on any Mac or Windows computer which supports the most recent Java runtime environment. However, the game will only be able to access MIDI devices available through the system's MIDI System so the proper installation of keyboards will be up to the user.

Modest CPU usage: MidiHero is not a graphically or computationally intensive game so it should be able to run on most modern systems. Specifically, MidiHero should run smoothly on 1Ghz computer with 512 Mb of RAM and a light load or better.

Possible Additions

Musically Appropriate Campaign: MidiHero could have a Single Player mode that motivates users to complete an introductory music course. This course would complement the grading in creating musically aligned incentives. This course will be based in part on existing courses but it will also take into account the differences between the MidiHero experience and traditional keyboard training.

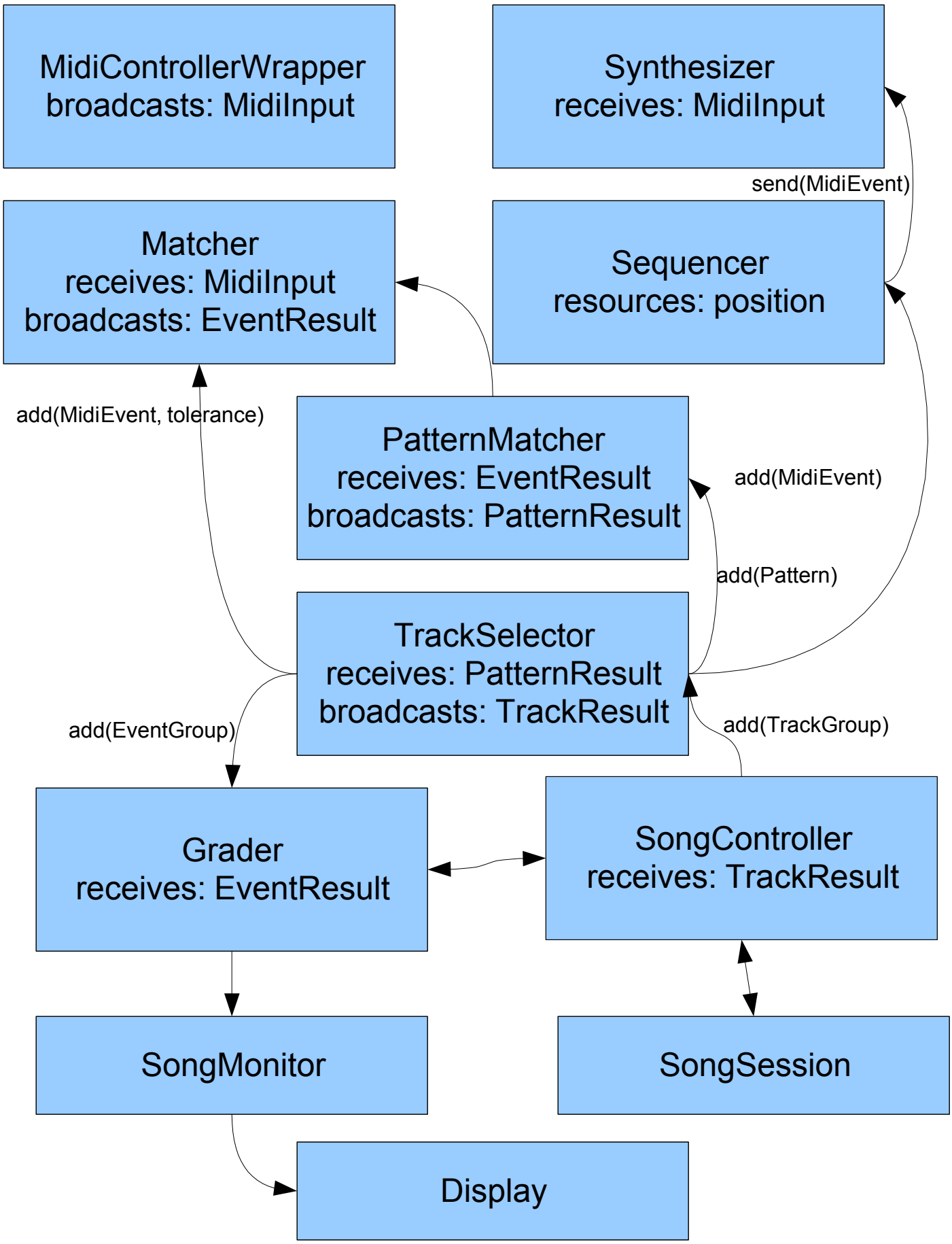
Networked Multiplayer: The ability to synchronize the playback of a song on two different computers running MidiHero would open up many possibilities. This would allow players to play together in a way similar to Rock Band, but they would actually be closer to a real band since they would be generating the music. This would make use of MidiHero's strict tempo policy which does not “forgive” incorrect timing by adjusting the playback position, meaning that all players would necessarily be playing in time with each other if they were scoring well. It would also allow for directly competitive play.

Musical MIDI Editor: MidiHero could also provide a system for editing MIDI sequences through a playback based interface. The interface could allow users to isolate tracks simply by playing the desired notes. A trademark of many successful game editors is their ability to exploit user experience with the game to improve the effectiveness and entertainment value of a limited interface (i.e. Forge on Halo 3 or the editor in Big Small World).

Alternate MIDI Input Mini-game: Since the MIDI interface is intended to be extensible, MidiHero could include a mini-game that uses the MIDI Controller as input to something unconventional like a 2D platformer or fighting game.

Audio Input: MidiHero could provide the user with the option of using a non-digital instrument, specifically the guitar. Using direct input from an electric guitar or possibly microphone input from an acoustic guitar, the game could approximately convert it to MIDI, possibly using probabilistic grading biased towards the desired input.

High Level Design



Design

MIDI based Musical Model: MidiHero will use a simplified note sequence system similar to that used in MIDI in addition to MIDI itself. MIDI represents notes as paired ON and OFF messages, but MidiHero's simplified note sequence will use a single message with a duration to represent notes. This representation will be used for the display code.

Matcher: The matching system operates on a tolerance based system similar to those found in games like Guitar Hero and DDR. The Matcher compares MIDI input to pending MIDI track events by dividing up EventResults into three general categories: unmatched input, expired track events, and matched events. Unmatched input occurs when an input event is received that does not match any track events within a tolerance before and after its position. Expired track events occur when track events go unmatched a tolerance past their occurrence. Matched events occur when an input event falls within one tolerance of a track event of the same note. The tolerance does not necessarily reflect what is an acceptable error on input. In the current system, a match event with an error of just under the tolerance receives a failing grade.

TrackSelector: The TrackSelector allows the Song Controller to simply provide a TrackGroup containing the selection pattern and track body of a group of related tracks (like two exclusive choices at a fork), and let the TrackSelector handle most of the interaction with the Grader, PatternMatcher, Sequencer, Synthesizer, and Matcher.

Scoring: The scoring system will likely appear to the user as a near clone of DDR's scoring system with added information available from the output of grading. Scoring will likely include a letter grade, a point total, and a best streak. It may also include awards similar to those commonly given in multiplayer FPS games (i.e. “Best Rhythm”,”Most Consistent”, etc.).

Scroller Display: The note display will be given by a right-to-left scroller system based on the current position of the MIDI sequence being played. The graphical coordinate system will be based on MIDI position relative to the current position for the x value and the y value will be based on the note.

Dynamic Music Generation: Track objects, which contain both patterns for the user to play as well as MIDI accompaniment, will be dynamically generated using transposition and categorized musical phrase banks. This will allow for game play that was not possible with fixed path songs like infinite but continuously changing songs.

Implementation

SGE Event System: SGE uses a type-based event system which usually uses universal, ordered broadcast of events to all components listening for that type of message. If necessary, SGE allows for more fine-grained control over message passing but this probably won't be necessary (NOTE: this is why some of the components in the design seem disconnected; see the broadcast and receive fields of each component).

SGE Session Based Interface: MidiHero will rely on the SGE Session and GUI library for the user interface. MidiHero primarily utilizes MIDI input for both navigation and commands, but keyboard events will also be accessible through SGE. Mouse support may be added if it proves beneficial, but there are no planned uses of it.

SGE Graphics: MidiHero will make use of the SGE graphical model which is currently in

reconstruction due to a flaw in Mac OS X Java 2D implementation. The new system will be implemented using JOGL, the standard Java OpenGL binding.

Scroller Display: The configuration for this scroller system will be accessible through the SongSession Env. This configuration will provide all the information necessary for Components to render themselves in the appropriate location for the current position in the sequence.

EventPainter: In addition to a ScrollerConfiguration, the SongSession Env contains an EventPainter that will handle the display of Images and MIDI events. This will provide the final layer of abstraction on display which ScrollerComponents will use to display information such as beats, frame divisions, notes, wrong input, graded notes, and grades without concern for their graphical representation.

Player: The Player interface will provide persistent information about the player's record to the rest of the game. This will include but is not limited to high-scores, best grades, longest streak, and unlocked songs.

Dynamic Music Generation: Track objects will be implemented using MIDI sequences for both accompaniment and the player's patterns. On success, a pattern will queue a new track that has been shifted to begin directly after the end of the pattern (musically speaking, not literally) and transposed to fit the current scale or mode of the music. This will allow all Tracks to start at 0 and be in a common mode (usually) like C Major when defined and still be easily applied in many situations.

Testing Plan

General Game Errors: SGE will provide a high enough layer of abstraction that most game errors having to do with user interfaces, startup, shutdown, threading, and those general issues will be isolated to within SGE.

Unit Testing: MidiHero will include unit testing for all of its major algorithms using JUnit. This should be very effective and easy to implement because MIDI input is easily scripted. Of course, this will not be the only kind of testing applied.

Automated MIDI Testing: Testing the matching system for basic correctness will be fairly straightforward. Because of the way the SGE event model works, a scripted player could simply receive pattern selections and broadcast MIDI input to match a specific pattern exactly. It could also receive pattern selection results to verify the match. Scripting incorrect input is also very simple. If the MIDI input from the previous method were one more than the tolerance early or late, the note should not be matched, although it may match a different note. Grading would also be easily verifiable in a similar manner.

Interface Testing: A primary innovation of MidiHero is the use of a musical instrument as the input device. This is an unconventional system and testing it requires user testing to make sure that it is effective more than just technically correct and free of bugs.

Personal Testing: I personally intend to master MidiHero to get a good feeling for the game. I match the profile of my target audience so this testing should provide much of the feedback necessary to design an effective display and grading scheme.

Long Term User Testing: I also have a number of long term testers who will play the game throughout

the development process. Some of this testing has already begun and the results have been excellent. My most committed tester has shown not only that musical aptitude translates well into MidiHero skill, but also that the game is entertaining and addictive enough to have a stronger draw than most conventional video games to at least some members of the target demographic. I hope to recruit at least ten such testers with a fairly even mix of those with prior musical training and those without.

Musical Testing: I intend to get feedback from musicians and music theorists. They should be able to offer a good deal of incite into how to make improvisation sound musical. The Brown music department and the musicians I know should be able to provide this feedback.

Project Timeline/Milestones

April 4th
MidiHero4.0 test-user release
Features:
auto-detection of MIDI Controllers,
library of at least four levels,
dynamic paths and accompaniment,
multiple instruments simultaneously,
tolerance-based matching,
multi-path compatible scoring,
basic display (see Storyboard)

April 11th
Present current status of Long Term testing results

April 18th
Present plans for possible extensions including guitar and networked multiplayer

April 25th
Present Long Term testing results

May 6th
Final deliverable: MidiHero5.0 (See Requirements for features)

May 12th
Postmortem report

Evaluation Criteria

Requirements (%)

Presentation: 30

- 5 Instant feedback
- 10 Music presentation
- 10 Visual Cues
- 5 Beat Representation

Experience: 30

- 10 Effective Learning curve
- 10 Appeal to music gamers
- 5 MIDI Keyboard
- 5 Complete MIDI Keyboard Interface

Music: 30

- 10 Transposition across modes and scales
- 10 Viable bank of musical phrases or alternative
- 5 Multi-soundbank Sequence Playback
- 5 Beat Representation

System: 10

- 4 Persistent Player Data
- 3 Portable
- 3 Modest CPU usage

Storyboard[1]

Starting Up

The game will start and automatically connect to any attached keyboard. The player will then have an opening menu with approximately the same options as those shown below. The player will interact with the menu by entering two note sequences. The menu selection will be moved by the difference between the first and second notes or if same note is struck twice, the currently selected option is chosen. Double clicking any note will take the player to the song menu.

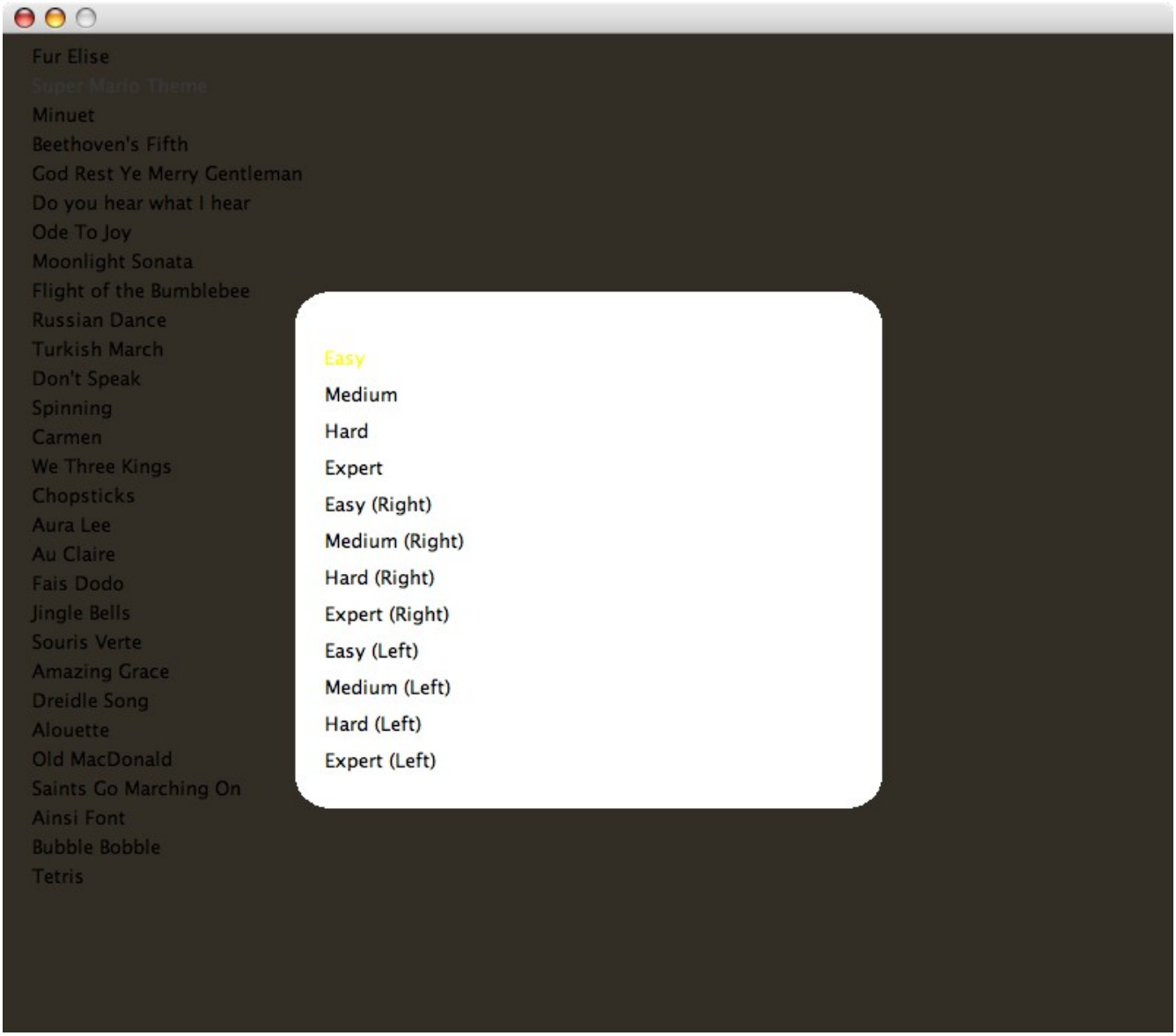
1 These images are from Frets on Fire, Guitar Hero, MidiHero2.5, and DDR, respectively. I intend to model MidiHero after the other music games in the same sort of fashion as they have all copied each other so the borrowing of these images seemed appropriate and more accurate than I was likely to produce.

Song Menu

The song menu will give the player a list of available songs and display the players previous records for the songs. This will be done in a similar way to that shown below. This menu will respond to the same two-note sequence interface as the main menu. Once a player chooses a song, they will be given a mode menu for that song.

Mode Menu[2]

The mode menu allows the player to choose which version of the song they would like to play. These versions will differ by speed, grading criteria and even notes. Certain versions will allow you to play one hand at a time. This menu uses the standard MidiHero navigation interface used so far. Once a mode is selected a Song Session begins.



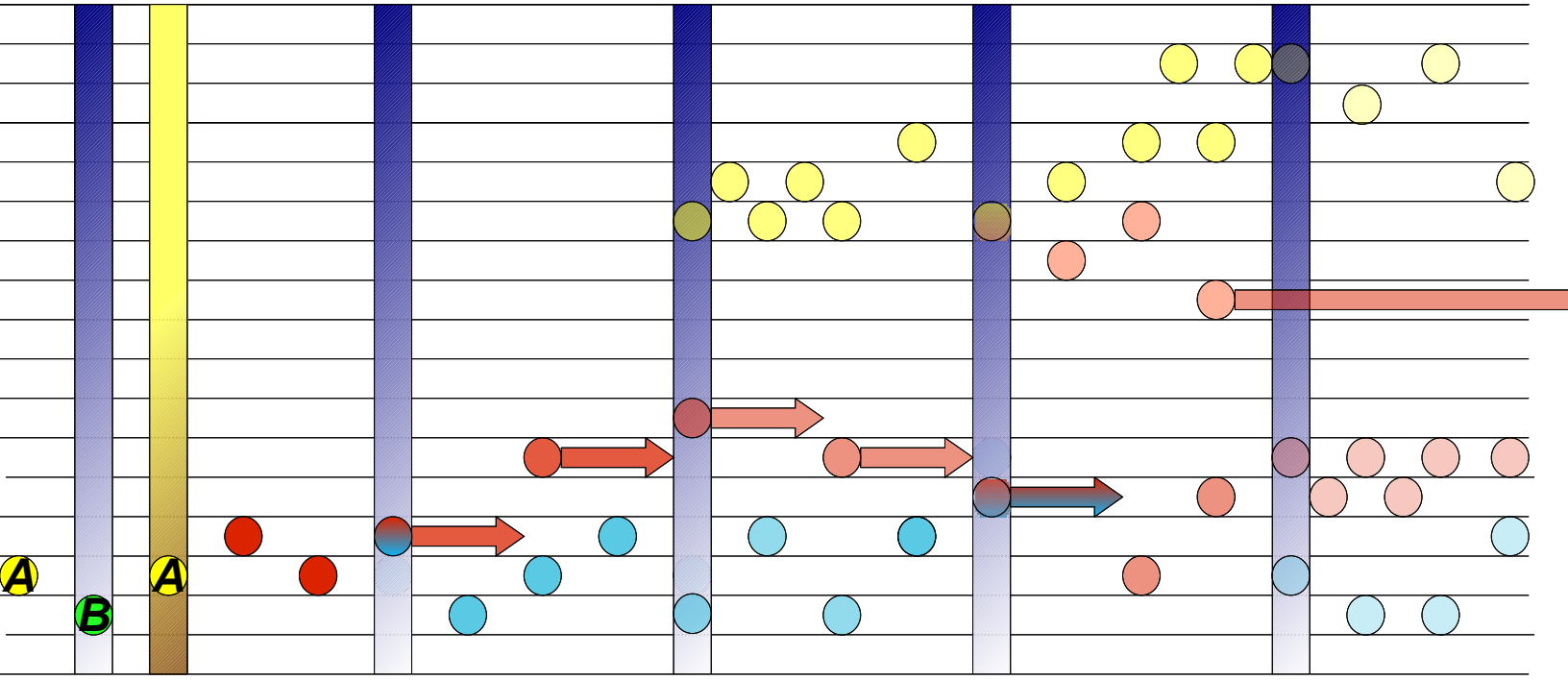
2 The real mode menu will of course have better graphics.

Song Session

The Song Session is really the heart of the game. This is where the user plays the song in time with scrolling display and the song playback. This session will incorporate a number of visual aids in addition to those shown below, such as beat bars and frame divisions (see earlier descriptions for more detail). The basic idea is to hit the correct note as it occurs in the song and passes over the guide bar. The game will give instant grades to user input. The players other score information will be displayed at the top of the screen. This information includes high-score, multiplier, streak, and best streak as shown below. Once the song is over a Result Session begins.

The interesting part about the MidiHero Song Session as opposed to other music scrollers is that it will provide the player with choices. For a simple two path choice, the display will show the beginning of both paths and the player simply plays along with the one they want to take. This will also be used for optional parts like a drum beat for the left hand that can be played or ignored with the current track continuing either way.

The example section below shows the player currently on a left-handed part that is about to branch in the next frame. One frame beyond that, an optional right-handed part begins which branches as the two left handed parts converge. Since the number of paths could increase with a branching of even more than 2 per frame, the later paths become increasingly transparent based on the total number of options.



Result Session

This is the final session before returning to the Song Menu. This session displays the player's results including score, final grade, best streak, and any other information that the grader has collected. The result session will also notify the user if they have broken a highscore.

THE END