Assignment 7

Overview

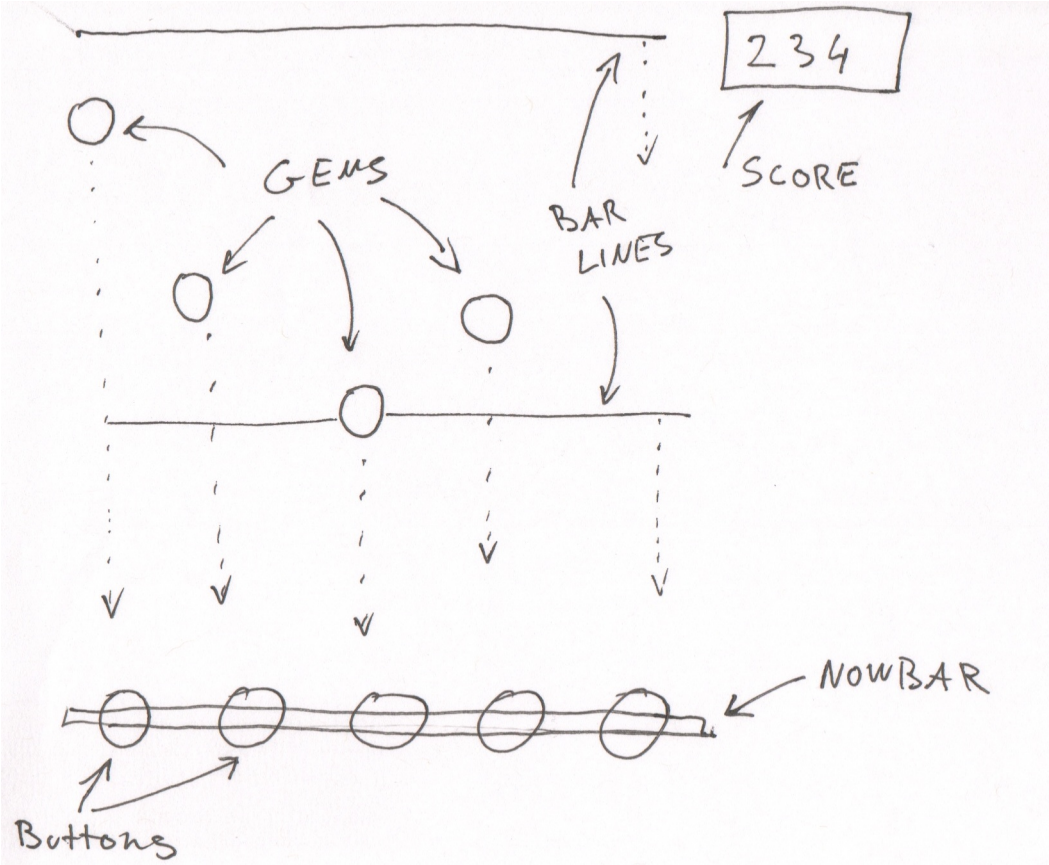
In this assignment, you will make a simplified Guitar Hero game, focusing on the core game mechanic (without the background graphics / characters, etc...). The main simplification is that we will use the computer keyboard as input device instead of a plastic guitar. Playing a note is therefore simply a matter of pressing the right keyboard key at the right time, as opposed to holding down the correct fret button(s), and strumming at the right time.

Since this assignment is fairly involved, you are advised to start early.

Due Dates

Late breaking news! This assignment is now split into two due dates. Parts 0, 1, 2 are due on 11:59pm, April 5. Part 3 is due on 11:59pm, April 12. However, you are free to complete the entire assignment on by April 5 if you wish.

Specification



Hit a key ('p') to start/toggle the song playing – this will start both the background audio and the guitar-solo audio.

Visual Display

As the song plays, *Gems* (visual display of notes which you will author to correspond to some notes of the guitar solo) will start "falling down" from the top of the screen and approach the *NowBar*, located towards the bottom of the screen. Each Gem belongs to one of 5 *Lanes*. There are also 5 *Buttons (visual indicators),* attached to the NowBar. When a Gem reaches the NowBar, it will intersect with the corresponding Button of the Lane.

In addition to Gems, *BarLines* also "fall down" from the top of the screen. Each BarLine represents the beginning of a measure / bar in the music. When Gems and Barlines travel down the screen, their velocity could be either in pixels per second or pixels per tick. Choosing pixels per second implies that all objects travel at a constant speed. As the song tempo changes, the *density* of the objects changes. Choosing ticks per second implies that BarLines are always spaced the same distance apart. As the song tempo changes, the *speed* of the objects changes. At Harmonix, we chose pixels per second for Guitar Hero and Rock Band. For this assignment, you can choose either style.

A *Score Display* in the top corner shows the player's score.

Player Actions

5 keyboard keys are mapped to activate the five Buttons.

The player attempts to "hit" the Gems by pressing the keyboard key associated with the Lane of a Gem when the Gem intersects the NowBar. There are three possible outcomes:

- *Hit*: The Gem was successfully hit. More specifically, a key was pressed when a Gem was within a time window centered on the NowBar and the key's Lane matches the Gem's Lane. The window is called the *Slop Window* and is +/- 100ms (ie, a total of 200ms in width)
- *Miss*: There are two miss conditions
    - *Temporal Miss*: A key was pressed, but no Gem was within the Slop Window.
    - *Lane Miss*: A key was pressed with a Gem inside the Slop Window, but in the wrong Lane. Any Gem that was Lane Missed is marked as unhittable.
- *Pass*: A gem passed a point in time (after passing the NowBar) where it can no longer be hit (ie, it is past the Slop Window).

Game Reactions

A well-designed game has direct visual and auditory feedback to inform the player about what's going on and how they are doing. The reactions for GH are:

Button Press: When a key is pressed and released, the associated Button's visual display changes accordingly.

Hit: The Gem shows that it was hit (for example, an explosion, flair, or some other exciting display). The Gem should then disappear or otherwise stop flowing down the screen to indicate it is no longer in play. The solo audio track unmutes. Points are earned.

Pass: The Gem changes visual state to show that it can no longer be hit. The solo audio track mutes.

Miss: A miss sound plays (ie, a WaveSnippet) to indicate a missed note. A Lane Miss has the same visual effect as a Pass (it can no longer be hit).

Part 0: Choose a Song

I've put up a few songs that we used in the original Guitar Hero game. These are multitrack songs, where the main guitar part is split from the rest of the background part so that each song consists of two stereo audio files. Pick a song.

Part 1: Annotate the Song

Create a tempo map for the song (as you did in Assignment 5).

Create a gem data file that describes the locations (Lane and Tick) of the Gems. Authoring the Gem data is a bit of an art form. The Gem pattern should be musical and follow the contours of the solo guitar line so it "feels right" to play it. You can choose how difficult you want the Gem pattern to be. You do not need to have a Gem match every note of the guitar solo. In the easy levels of Guitar Hero/Rock Band, Harmonix authored far fewer gems than guitar notes. Only the most difficult level had a (mostly) 1-1 correspondence between Gems and guitar notes.

You can create your own file format as you see fit. I created the following file format, as shown by example:

```
1:........
2:2214.3201232.202
2:2214.3201232.202
2:2214.3201232.202
2:2214.32012324323
2:12324323
1:4.3
4:343.
2:2...32
```

The set of numbers on a line represent a sequence of Gems, all of which have the same duration. Each number is the Gem's lane. A period (.) is a rest (no Gem in that time slot). The <num>: indicates the Gem duration for that line. 1: means a sequence of quarter notes. 2: means a sequence of eight notes, etc...

This format is very compact and easy to type. The limitation is that it can only represent one gem per time location (ie, it does not allow for "chords"). But that's OK for this assignment. You can use this file format, or create your own as you see fit.

Parse the Gem file and store the data in the class *GemData.* You can do a rough pass initially to validate your file format. Then in Part 3, you can refine your Gem authoring so it really feels right.

Part 2: Basic Graphics and Audio

Create the graphical and audio elements of the game:

- **GemDisplay**: draws a single gem
- **ButtonDisplay**: draws a single button
- **BeatMatchDisplay**: draws Gems, BarLines, NowBar, and Buttons. Animates the Gems flowing down the screen (hint: create a Translate() object and scroll Gems and Barlines by animating trans.y).
- **AudioControl**: plays both background and solo audio files.

Instantiate the BeatMatchDisplay and the AudioControl. The Gems and BarLines should flow down the screen towards the NowBar as the song plays.

At the end of this part, nothing is interactive. Gems and BarLines simply flow down as the song plays.

Part 3: Interactive Elements and Game Logic

In this part, you will finish the game by adding all the interactive elements and audio/graphical Game Reactions.

- **GemDisplay**: implement functions to change the Gem's graphical look as needed: *on_hit*, *on_pass*, and *on_update* (to help with hit animation).
- **ButtonDisplay**: implement *on_down*, and *on_up*, to change graphical look when the user presses a key.
- **Player**: Create the game logic that implements the interactive behavior described in the Spec. Player receives button up/down events from MainWidget. It figures out what to do and calls functions on BeatMatchDisplay and AudioControl to make the appropriate reactions happen.
- **BeatMatchDisplay**: implement the functions called by Player that affect visual display of Gems and Buttons.
- **AudioControl**: implement mute / unmute for the solo tracks, and play the miss-sound-effect with WaveSnippet or WaveFileGenerator.

Lastly, refine your Gem authoring so that it feels right and fun to play.

Finally...

No need to make a movie. Please have good comments in your code. When submitting your solution, submit a zip file that has all the necessary files. Upload your zip file to Stellar / Homework.