Fields, Methods, and Random Numbers Practice

Music Player Lab

# Main Objectives

* Class design including fields, constructors, and methods.
* Further explore the method API for Greenfoot.
* Utilize Math.random and other math operations.
* Optional Goal: Practice with Arrays or Lists.

# Prior Knowledge

* This project uses a several classes from the Greenfoot package including methods from World, Actor, GreenfootImage, GreenfootSound, and Greenfoot itself.
* Students need to know the basics of writing classes with fields, constructors, and methods.
* String concatenation.
* For perfect motion of the Notes to follow precise diagonal lines, some form of either basic right triangle trig would need to be used or the algebra 1 level of finding an equation for a line.

# Greenfoot Background

* This lab does require knowledge and use of many of the Greenfoot classes.
* It will use the GreenfootSound API as its primary usage of Greenfoot.

# Getting Started

Install the Greenfoot IDE: <https://www.greenfoot.org/download>

* Greenfoot went through many changes beginning with the 3.0.0 release. Version 2.4.2, although older, tends to be one of the most stable versions but is missing a few of the newer additions.

In your GuitarfootHero folder run the project file with the Greenfoot icon to begin.

# Demonstration

View a sample demonstration of what the GuitarFoot Hero lab could look like here:

<https://www.greenfoot.org/scenarios/23671>

# Instructions

The lab can be broken into two separate concepts, one without arrays or lists which could be completely fairly early on in a student’s learning or one with arrays, without much changes to the original lab’s goals otherwise.

Differences: Without an array or list will generate random notes falling from the screen at random times. The program with arrays or lists will have notes that must be pre-defined. Two parallel arrays or lists could be used to keep track of the note that will play next as well as the milliseconds since the program began that the note must be put on the screen. It is rather tedious to build the arrays to match a full song, which is a trade off.

## Personalization

Depending on the student’s level of comfort, this lab could be presented simply with the instructions: “Build the best version of Guitar(foot) Hero you can”. This will give students a sense of ownership over what they build, allowing them to find or draw their own images, decide what small touches they will add into the program, etcetera.

### Minimum requirements examples:

* Notes must, over time, come from the top of the screen and move towards the bottom in some form of ‘lane’.
* Frets must have a logical keyboard key that, when pressed, shows the Fret has been pressed in a visual way.
* Frets must have a second keyboard key which all Frets share to represent strumming the strings.
* When a Fret is strummed with both its own keyboard key and the strumming key and a Note is nearby the Note should be scored and removed from the screen.
* If a Note hits the bottom of the playing area, the Note should be removed, and the score should be reduced.
* Some visual representation of the Score should appear on the screen and update when Notes are either hit correctly or when they are missed entirely.

## More In-Depth Instructions, but not a Step-by-Step set:

### Working with the Note Class

* May work best as a subclass of SmoothMover to allow decimal precision movement. To import SmoothMover in Greenfoot go to Edit -> Import Class -> SmootherMover.
* Each Note, when constructor, should be told which type of Note it will be. The choice of how this is done is up to the student. Depending on the type of Note, the image of the Note should be set correctly. If the Note will be moving in diagonal directions the value of these diagonal movement amounts should be chosen as well.
* When a Note acts, it should move slightly down the screen and either the X value should stay the same or it should adjust based on the angle the Note is moving in.
* If the Note hits the edge of the screen it should be removed from the screen.
* If the Note hits the edge of the screen the score should be lowered.
* For students who wish to scale the size of the note based on how high it is from the top of the screen it is best to first set the image back to its original image, then scale this new image to the correct size. The current Y coordinate would be useful in determining the current size the note should be.

### Working with the Fret Class

* Each Fret, when built, should be told which type of Fret it will be. This type will determine which images the Fret should use as well as which keyboard key it will use. The likely two best choices for keyboard keys are either the number values, 1, 2, 3, 4, 5, or the F values, F1, F2, F3, F4, and F5. The F values make more sense for hand placement, however websites will not be able to use them correctly and some keyboards have difficulty holding down many F keys at once.
* When the Fret’s keyboard key is pressed, the Fret should update to a second image representing the Fret is being pushed down.
* When the Fret’s keyboard key is pressed and the strum key is pressed (backspace or enter are good options, backspace does not work well for websites), the Fret should determine if a Note is near. If so, it should remove the Note and update the score correctly.
* A better Fret class will determine how close the Note was to the center of the Fret. The closer the Note is to the center of the Fret, the higher the score increases. Helpful images popping up saying ‘too soon’ or ‘too late’ can also be utilized for a more authentic feeling game.

### Working with the World Class

* The World should immediately place Frets on the screen in the correct locations.
* The World should play the music file for the game.
* Each act, the world should give a random percent chance that one (or more) notes will appear on the screen. If so, the World should build the Notes and place the Notes in the correct lane.
* Students may decide if the music file will be played just once or repeatedly. If repeatedly, then a GreenfootSound object will need to exit which can be placed on a playLoop.
* Adding in a feature which allows 2 or more notes to appear simultaneously will make for a more authentic feeling game.
* If students wish to not randomly choose notes, they will need to build an array or List which determines which notes will be played and when.

### Other Potential Classes:

Instructions – Displays the instructions on the screen.

Notification – Displays images like “Too Early” or “Too Late” or “Perfect” when Notes are hit.

Score – Keeps track of and displays the current score. Could also include a ‘streak’ counter keeping track of how many Notes have been successfully hit in a row.

# Extensions and Modifications

* A beginning screen may be added which allows different settings to be chosen - difficulty level or song choice being examples.
* A game over screen may be added which shows statistics of the game. How many of each type of correct key were scored. How many were missed.
* Mapping a song to notes instead of using random notes can be done. These notes and the timing of the notes could be written directly into an array or list, or could be stored separately in a file and read using some form of file reading class.
* Keeping track of long-term high schools could be done for students who have learned how to write to a file and read from a file.
* For a simpler note falling, students could elect for a more rectangular board for notes to fall straight down instead of at angles.

