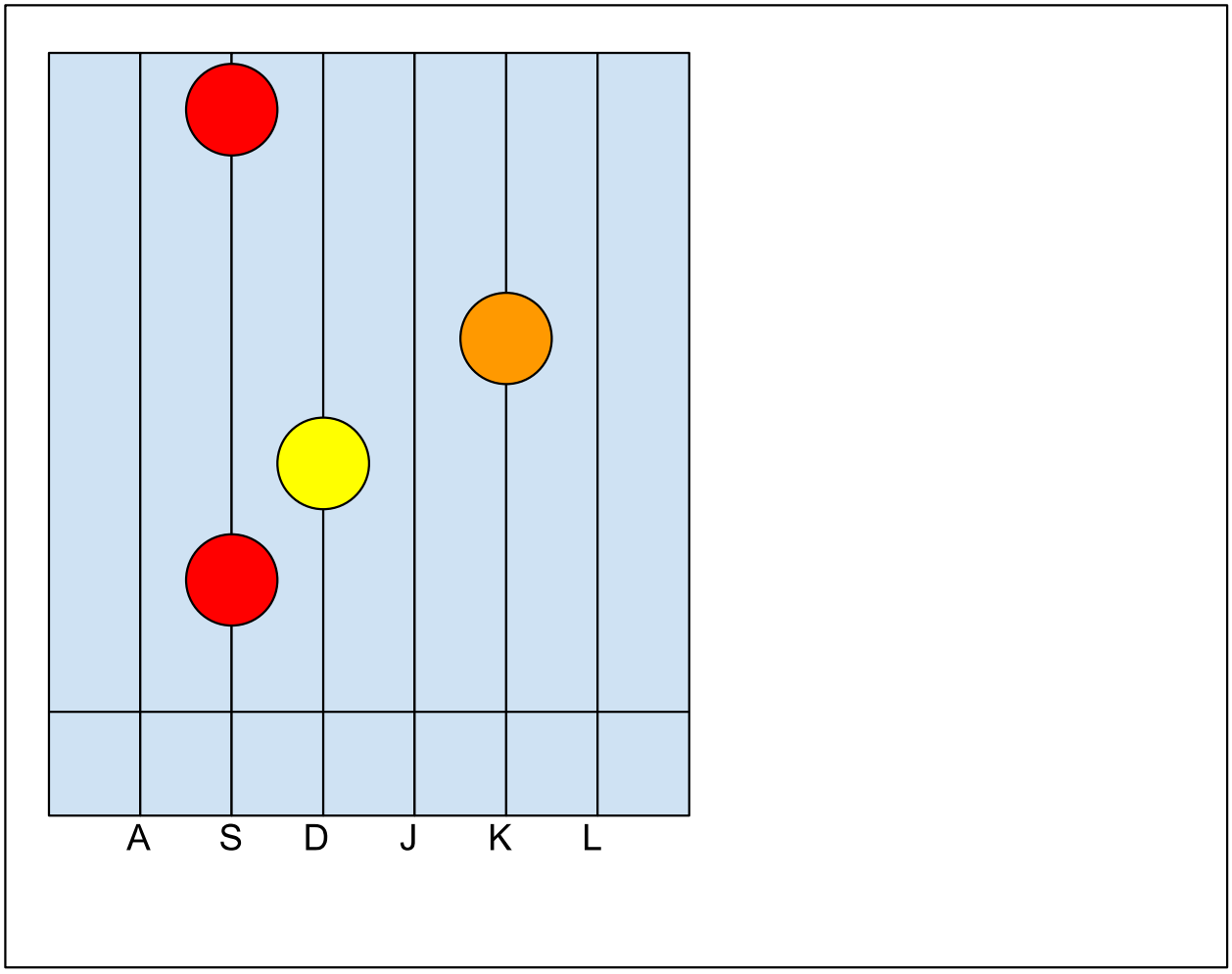
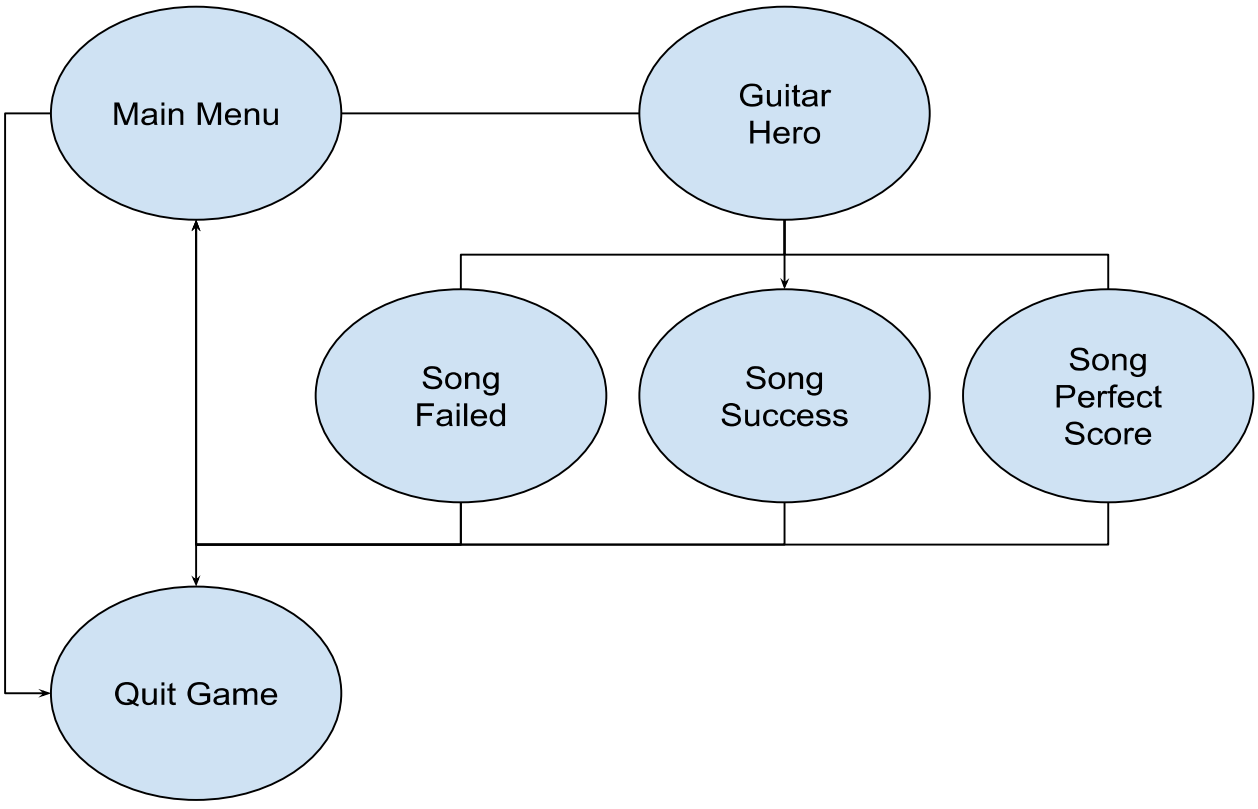
Idea 1: Guitar Hero clone

- Player has to play a simple guitar hero style beat map (Plants vs Zombies Grasswalk)
- 6 note lanes mapped out to A S D J K L
- Forseeable issues:
    - Timing music with notes will be very hard in Processing
    - Mapping each key to a specific time measure in processing will take a long time
    - Notes move at the same speed regardless of how many there are on screen
- Forseeable player enjoyment:
    - If timing is done right, player will have fun with a rhythm game
    - If timing is off, it will suck for the player

Functions & Data:

- Grasswalk BPM: 115
    - Will have to speed song up to 120bpm to match frames
- Notes will take 2 seconds reach from the top of the screen to where the player has to hit them
    - Note velocity done with PVector
- Player doesn't have to be frame perfect, there will be a certain distance from the bar the player can hit the note on measured with PVector
- Metronome Function needed for timing
    - Using modulo to measure 1/8 notes in the song (Grasswalk never uses 1/16 notes)
    - Second modulo increases by 1 every 8 1/8 notes to indicate which bar of the song is playing
    - Saves information into a PVector
- Note Function
    - Will create notes based on the current measure of the Metronome Function
    - Metronome Function has to be 2 seconds ahead of proper song Metronome so notes are created and move down the track in proper timing
    - Unlike Guitar Hero, there will be no perspective so notes will stay the same size as they descend the neck
    - Note timings will be saved in an Array (there will be a lot of notes… roughly 534)
- Loss Case:
    - Players will have 5 lives, if they miss a note they will lose 1. 0 lives = game over
    - Every 20 consecutive notes hit, player regains 1 life
- Win Case:
    - If the player makes it to the end of the song, they win! Score is based on notes struck and will show their score /534 at the end
        - Special screen for a Full Combo


A
S
D
J
K
L


Main Menu
Guitar Hero
Song Failed
Song Success
Song Perfect Score
Quit Game

Pseudo Code:

- Metronome Function
    - Function that will be called when the game starts proper
    - Uses 2 Int values, Bar and Note
    - Adds +1 to int Note every %15 frames
    - If Notes +1 would equal 9
        - Set Note to 1
        - Increase Int Bar by +1
    - Adds values to a PVector called TimeMeasure
        - X = Bar
        - Y = Note
        - Not used for Vector math, just for data storage
- Note Timings Array:
    - LONG array containing the time measures of each note in the song
    - Gets its values from a separate text file (easier to write)
        - X = Bar
        - Y = Note
        - Z = Type of note
            - 1 - 6 int value corresponding with which lane the note will fall under
    - Referenced for creating the next notes
        - Int NodeIndex will increase the index of the array by 1 every time a note is created to avoid having to read the entire array
        - Code checks if Metronome Vector's X and Y are equal to the X and Y of the note at the current index
            - If true, creates the note
- Note Class
    - Creates each note
    - Constructor requires 1 parameter
        - Int NoteType to determine which lane the note spawns in
    - Created into an object array to be able
        - Used to call the .move function for each of them
        - Used to call nullify the array index when the player hits them and "destroy" the object

Implementation Log

Timing the Notes

- In order for a rhythm game to function properly the notes need to appear ahead of time to give the player the necessary time to react to them, and they need to reach the area of where the player will hit them when the correct note is played in the song
- Creating notes early and note speed:
    - Notes should be created around 2 seconds before they are played so the player has enough time to react to them
    - Note highway is around 350 pixels long, with the bar indicating when notes should be pressed being around 300px from the top
    - Notes need to move at a speed of 2.5px/frame to reach the bar in 2 seconds
    - If the notes are moving too slowly I can increase the speed to 3.3px/frame, giving the player around 1.5s to react
- Note Timing:
    - Rhythm games need to be nearly perfectly timed in order to function properly, meaning notes need to appear exactly when they should in accordance to the beats
    - Im going to need to create a Metronome function in order to measure the time
        - 2 Modulos to determine the Bar, and Beat
            - Note variable which increases by 1 every 15 frames. Resets back to 1 if it would hit 9
            - Bar variable which increases by 1 every 8 Notes. Does not reset
            - 3rd Value is a number between 1 and 6 to determine which lane the note will go down
            - Each of these will be saved into the X Y Z variables of a 3D PVector.
    - This allows me to map the song out easily if I can get access to the note chart or even sheet music of the song
    - Beatmap will likely be written into a text file and read by the program

- Implementation Results:
    - Metronome Function worked PERFECTLY, keeps exact time of the song
    - Realized that the variables weren't necessary, I could just do metronome.x or .y and add the values directly to the metronome PVector
    - Metronome doesn't need to have the type of the note in it at all.
    - Used a Table object to read a .cvs file and assign the values to entries in an ArrayList, worked like a charm

Using PVector for all necessary components

- I don't wanna commit the same mistake as last assignment and miss the things I have to use PVector for, those were dumb wasted marks so I'm gonna plan each use out

- **Position & Distance**
    - Position will be used for 2 different things:
        - Making sure the notes spawn in their correct column
        - Tracking the note's current position to calculate the different from the hit bar in order to create leniency
    - This will allow me to have an array of PVectors numbered 1 - 6 which will correspond with the type of note, spawning them in the correct lane
    - Also allows me to track how far the note is from the bar, giving the player x-many pixels of leniency so they don't have to hit frame perfect notes
- **Velocity**
    - Easiest PVector to get out of the way
    - Used to measure the speed the notes go down the lane
- **Acceleration**
    - Tricky PVector to implement because notes will move at a static speed.
    - Will likely use in some kind of menu
    - Ideas:
        - Background animation of main menu
        - Confetti if the player wins
        - Rain if the player loses
    - Don't want to use in actual gameplay to give the notes their full needed processing power

- Implementation Results:
    - Everything worked perfectly as intended.
    - Position is used to track the location of the notes for the purpose of hit and miss tracking, for the rain to remove the object if it's off screen, and to track the position of each lane's hit zone
    - Velocity is used in the notes to make sure they are going down the lanes and by the rain to make sure its falling diagonally
    - Acceleration is used in the rain in the background of the lose screen, making it accelerate diagonally across the screen.

Emergent Issues

- Turbo Final Note
    - Final note in the ArrayList moves WAY faster than all the others, unsure as to why
        - Found out why: since the noteIndex wasn't increasing after the last note, it was simply creating multiple references to the same note, thus applying it movement multiple times per frame
        - Fixed by only running the check to create a new note if the noteIndex is lesser than or equal to notes.size()-1

- Notes not resetting
    - Notes weren't resetting when the player would replay the game
        - Reason: By assigning the notes.get(i) directly it was simply tying that note to the activeNotes arraylist
            - Solution: changed it so instead of assigning the object from activeNotes to an index in activeNotes, instead it creates a new object using the values from notes.get(i), essentially creating a "copy" of that note while keeping the information intact within the notes arraylist

- Hard Coded values
    - Adding more notes via the Beatmap.csv file adds more notes however other variables that control how long the game goes on for are still hard coded
        - Made the timeLimit PVector's bar value equal the bar value of the last note in the csv file +5 (adds a tail to the song)
    - Changed several other hard coded values to be variable based.