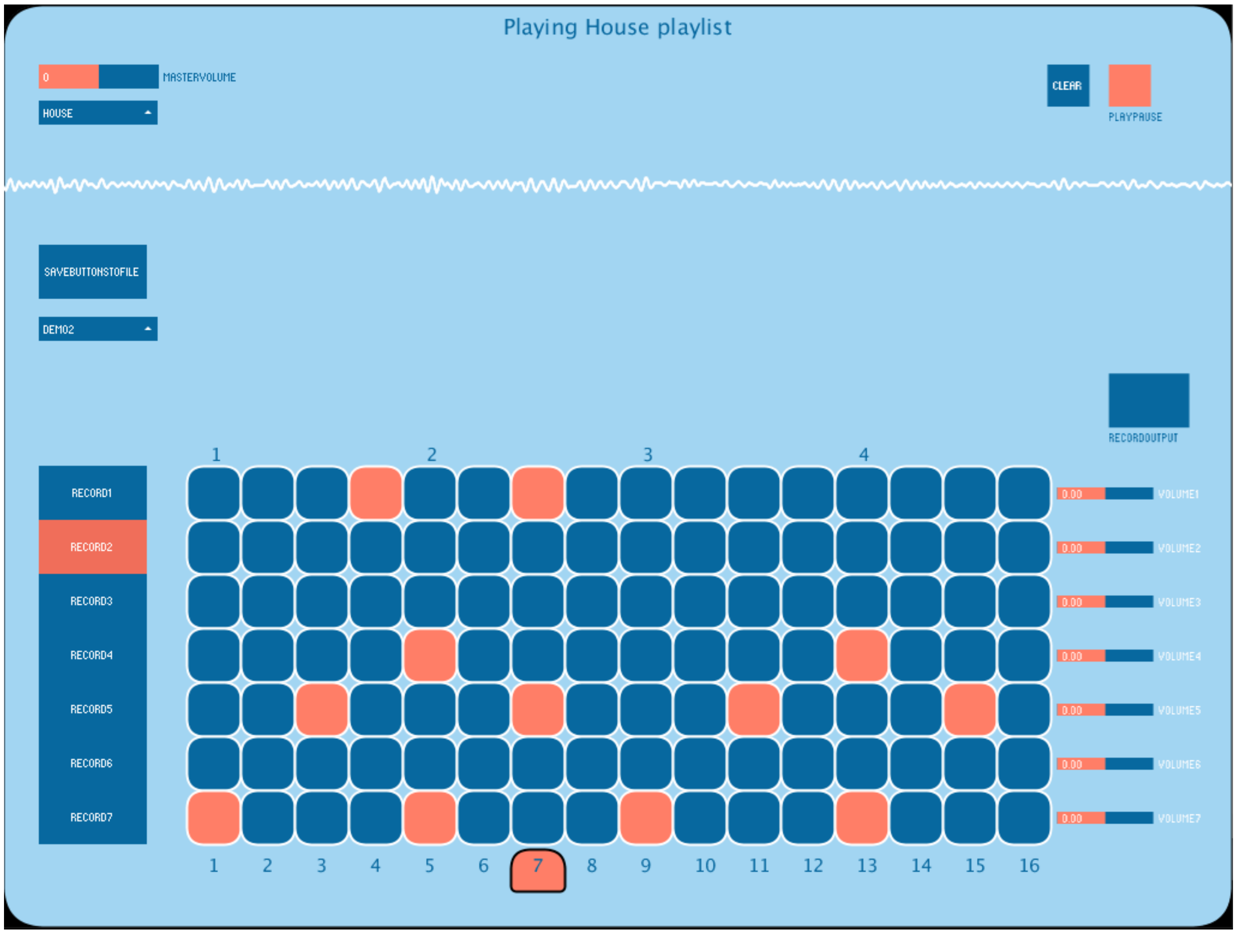
CS321: Music Programming - Project Description

Xavier Santamaria 19142820

Dec 2019

Drum Machine



1 Description of the project

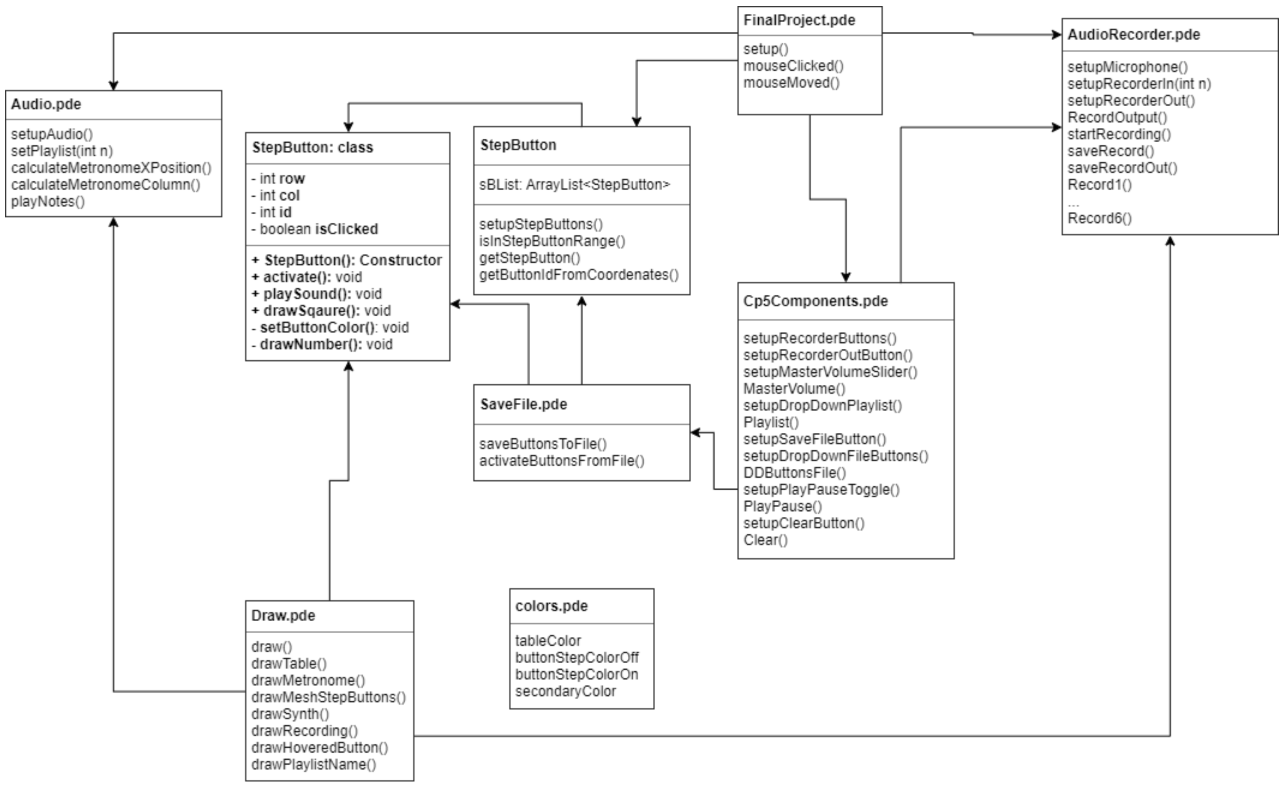
This project is for CS321 Music Programming and it consists of a Drum machine and its built using most of the concepts explained in class. A drum machine can produce songs given a group of sounds. Every row has the same sound and the buttons activated in the same row sound at the same time. The way the column sounds change is by 120bpm.

2 Features

1. Play sounds in 120 bpm

2. Modify the master volume
3. Change different sounds (Drop Down Playlist)
4. Setup buttons from a demo (Drop Down FileButtons)
5. Save buttons to use them later (Drop Down FileButtons, 3rd option)
6. Record your own sounds and play them (Drop Down Playlist, 3rd option) - One click to activate
7. Deactivate all the buttons
8. Play/Pause the music
9. Record song - 2 clicks: 1 for activate and 1 for stop the recording
10. When the new sound is recording the out is muted to not interfere in the recording of the sound

3 Structure of the code



The code is structured in files to have a better perception of where are all the methods. We have only one class, the **StepButton**. An object stepButton is one little rectangle this rectangle is an object so we can work more easily when we have to activate them and play a sound. Because it will be only object.activate() or object.playSound(). The other files have a common topic methods, like Cp5 is for the components of the library Cp5, AudioRecorder is for the input and output recorder methods, the colors.pde file is where the color variables are located so changing one variable from there affects the whole color interface.

4 Code

In this section I'm gonna explain the main functions in this code.

4.1 FinalProject.pde

The first file is the *FinalProject.pde*. This one contains the *setup()* function and the ones that interact with the mouse which are *mouseMoved()* and *mouseClicked()*.

4.1.1 Setup()

Setup() is the first function that runs in the program. In this function we set the size of the program, instantiate Cp5, and setup some Cp5 buttons, sliders, etc; and also setups recorders needed for recording input/output sounds.

```
void setup() {
  size(1024, 768);
  instantiateCP5();

  // Instantiates the minim and out Line and writes the names of the playlist
  // we have.
  setupAudio(); // ./Audio.pde

  // Instantiates the In Line so we can use the microphone later.
  setupMicrophone(); // ./AudioRecorder.pde

  // Setup the buttons that are in each row to record custom sounds.
  setupRecorderButtons(); // ./Cp5Components.pde

  // this recorderIn (input) is not used but I had to instantiate one recorder
  // otherwise the comprobations I do in the draw throw errors.
  setupRecorderIn(0); // ./AudioRecorder.pde

  // Setup the button to record output sound --> songs
  setupRecorderOutButton(); // ./Cp5Components.pde

  // Setup/ creates the recorder for the output sound to create songs
  setupRecorderOut(); // ./AudioRecorder.pde

  // Setup the slider for the master volume CP5
  setupMasterVolumeSlider(); // ./Cp5Components.pde

  // Setup the DropDownPlaylist to let user choose which playlist of sounds to use
  setupDropDownPlaylist(); // ./Cp5Components.pde

  // Save File = save in a file the buttons the user has clicked as a configuration
  setupSaveFileButton(); // ./Cp5Components.pde

  // Setup the dropDown to let the user choose the configuration of buttons they
      want to use
  setupDropDownFileButtons(); // ./Cp5Components.pde

  // Setup the CP5 button to play and pause the sounds
  setupPlayPauseToggle(); // ./Cp5Components.pde

  // Setup the Clear Button CP5 so the user can clear all the buttons in the drum
      machine
  setupClearButton(); // ./Cp5Components.pde

  // Step the buttons that once clicked produce sounds
  setupStepButtons(); // ./StepButton.pde
}
```

4.1.2 mouseClicked()

When the mouse is clicked, if position of the mouse is in the range of the StepButtons it gets the ID of the button it corresponds that location and activates that button.

```
void mouseClicked() {
  if (isInStepButtonRange(mouseX, mouseY)) {
    getStepButton(mouseX, mouseY).activate();
  }
}
```

4.1.3 mouseMoved()

When the mouse is moved, if it is in the StepButtonRange it gets the id of that button and stores it in the variable hoverButton that, otherwise, sets the hoverButton to -1 so no StepButton is hovered.

```
void mouseMoved() {
  if (isInStepButtonRange(mouseX, mouseY)) {
     hoverButton = getStepButton(mouseX, mouseY).id;
  }
  else {
    hoverButton = -1;
  }
}
```

4.2 Draw.pde

After running the *setup()* method, the next one is the *Draw()* which acts like a loop and it's constantly drawing in the screen. In this loop we first draw the background so in each iteration it refreshes the screen.

```
void draw() {
  background(0);
  strokeWeight(2);

  drawTable(); // Draw the background
  if (playToggle) drawMetronome(); // If play button is activated draw the
       metronome
  drawMeshStepButtons();
  drawSynth(); // Code to draw the wave of the output sounds
  drawRecording(); // Draw in screen 'Recording...' if we are recording
  drawHoveredButton(); // Emphasize the button the mouse in
  drawPlaylistName(); // Draw at the top of the screen which playlist are we using
}
```

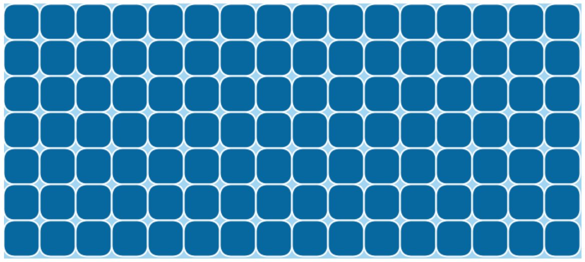
After drawing the background we draw the following components:

4.2.1 Metronome

Here we draw the rectangle that is moving when the play button is on. To draw that rectangle first we get the *xMetronome* this is the position we calculated. Then using this x position (because it moves horizontally only) we draw the rectangle using *rect()*. The function *calculateMetronomeXPosition()* returns the horizontal coordenates that are changing constantly in a 120bpm.

```
void drawMetronome() {
  xMetronome = calculateMetronomeXPosition();
  fill(buttonStepColorOn);
  rect(xMetronome, y +5, size, size -10, 80, 80, 5,5);
}
```

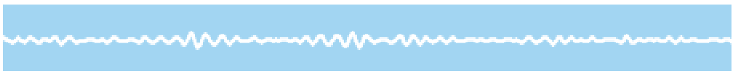
4.2.2 MeshStepButtons



This image is what this method draws. It iterates through the list of buttons and calls the method that actually draws that button/square. So at the end we have all the mesh of buttons drown.

```
void drawMeshStepButtons() {
  stroke(0);
  for(int i = 0; i < sBList.size(); i++){
    sBList.get(i).drawSquare();
  }
}
```

4.2.3 Synth

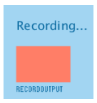


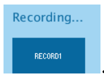
This function is to draw the wave.

```
void drawSynth() {
  translate(0,100);
  for( int i = 0; i < out.bufferSize() - 1; i++ )
  {
    // find the x position of each buffer value
    float x1 = map( i, 0, out.bufferSize(), 0, width );
    float x2 = map( i+1, 0, out.bufferSize(), 0, width );
    // draw a line from one buffer position to the next for both channels
    line( x1, 50 - out.left.get(i)*50, x2, 50 - out.left.get(i+1)*50);
    //line( x1, 150 - out.right.get(i)*50, x2, 150 - out.right.get(i+1)*50);
  }
  translate(0,-100);
}
```

4.2.4 Recording

This function draws 'Recording...' in the screen when

1. we are recording the output song 

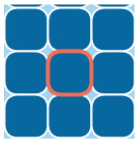
2. we are recording the input sound .

```
void drawRecording() {
  if(microhponeWorks) {

    // If we are recording a sound(trigger) it prints recording... during 2 seconds
        and then save the recorded sound
    if(recorder.isRecording()){
      recorderTimer++;
      text("Recording...", 30, height*.5 - 20);
      if(recorderTimer > 30){ // When time gets to its limited save the Sample sound
        saveRecord();
      }
    }
  }

  // if we are recording the Song, print in screen 'Recording...'
  if(recorderOut.isRecording()) {
    text("Recording...", width*.9, height*.4 - 20);
  }
}
```

4.2.5 HoveredButton



This method checks if the variable that stores the id of the button that is where the mouse is pointing, if it's different than -1 then it means that the mouse is pointing a button so we get that button and we draw it again so it is now the last button the be drawn and then we can emphasize by changing the color and with a bigger stroke.

```
void drawHoveredButton() { // If there is a button to hover, draw it
  if(hoverButton != -1) sBList.get(hoverButton).drawSquare();
}
```

4.2.6 PlaylistName

Playing Rock playlist

This draw function shows which playlist of sounds we are using.

```
void drawPlaylistName() {
  textAlign(CENTER);
  textSize(18);
  text("Playing␣" + playlistName[iPlaylist] + "␣playlist", width/2, 25);
  textAlign(LEFT);
  textSize(12);
}
```

5 Libraries used

In this section I'm going to explain the libraries I used to build this program and why.

5.1 Minim

5.1.1 AudioRecorder

It is used to record the input sounds and also to record the output song that is produced by the combination of sounds.

5.1.2 Sampler

The *Sampler[]* is used to store a list of sounds in the way we can trigger them anytime.

5.1.3 AudioOutput

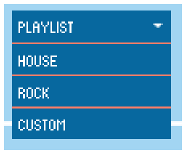
To patch all the sounds to a one output line so we can modify the gain of that group of sounds and also play the sounds so we can listen to them through that out line.

5.1.4 AudioInput

To be able to record our own sounds from the microphone.

5.2 ControlP5

5.2.1 DropDown

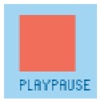


To let the user choose between different playlist of sounds.

5.2.2 Toggle

To clear all the buttons so the user doesn't have to deactivate all the buttons one by one.

5.2.3 Button



To play/pause the music.

5.2.4 Slider

To set the gain of the out sound.