Assignment 2

Team number: 86
Team members

Name	Student Nr.	Email
Boyee Zhang	2757826	b.zhang3@student.vu.nl
Lyudmil Anastasov	2781875	l.anastasov@student.vu.nl
Emre Yalcin	2751909	e.yalcin3@student.vu.nl
Jorrit Bogaard	2786510	j.d.bogaard@student.vu.nl

Summary of changes from Assignment 1

Author(s): Boyee

Question 1: delete the notes

Overview:

During this project we will design and engineer a software based on the Drums Machine system inspired by the online version of Roland TR-808, which was given as an example in the Project Guide, drumbit and Virtual Drums MUSICCA.

The main types of users of our software will be musicians, people whose hobbies revolve around music and pretty much anyone willing to give it a try.

The software will work primarily from the console with commands like:

(example of command), (example of command2)... (JAVAFX)!!!!

Although we plan on implementing a GUI at a later stage of development, with clickable buttons like in online version of Roland and drumbit, initially. Regardless of whether we make a GUI or not, there will be a visual representation of the drums playing somewhere in the middle of the screen.

Our primary objective is to create a digital tool that allows anyone to simulate a wide range of percussion instruments and create complex rhythms and beats.

-sources: drumbit | Online drum machine;

You should have delete the notes you have taken for yourself. It summarizes the application and goal, nice.

Feedback: we already deleted the notes, and I think the overview is clear for users to understand our guidelines for this application.

Question 2: explain which parts are essential and which are bonus

Feature 1

All features are well defined. Good work. Maybe we could just focus on why some features are essential and some are bonus.

Name: Drum Components and Audio Feedback in Real Time

Description:

This feature forms the core of the interactive experience in the drum machine software. It ...us drum components, such as kicks, snares, hi-hats, and cymbals, in real-time through a graphica... traditional drum pad layout. Each time a user clicks or taps on a specific drum component on the screen, the software instantly generates the corresponding drum sound, providing immediate audio feedback.

.

Champion: Emre Yalcin

inspiration:https://www.musicca.com/drums

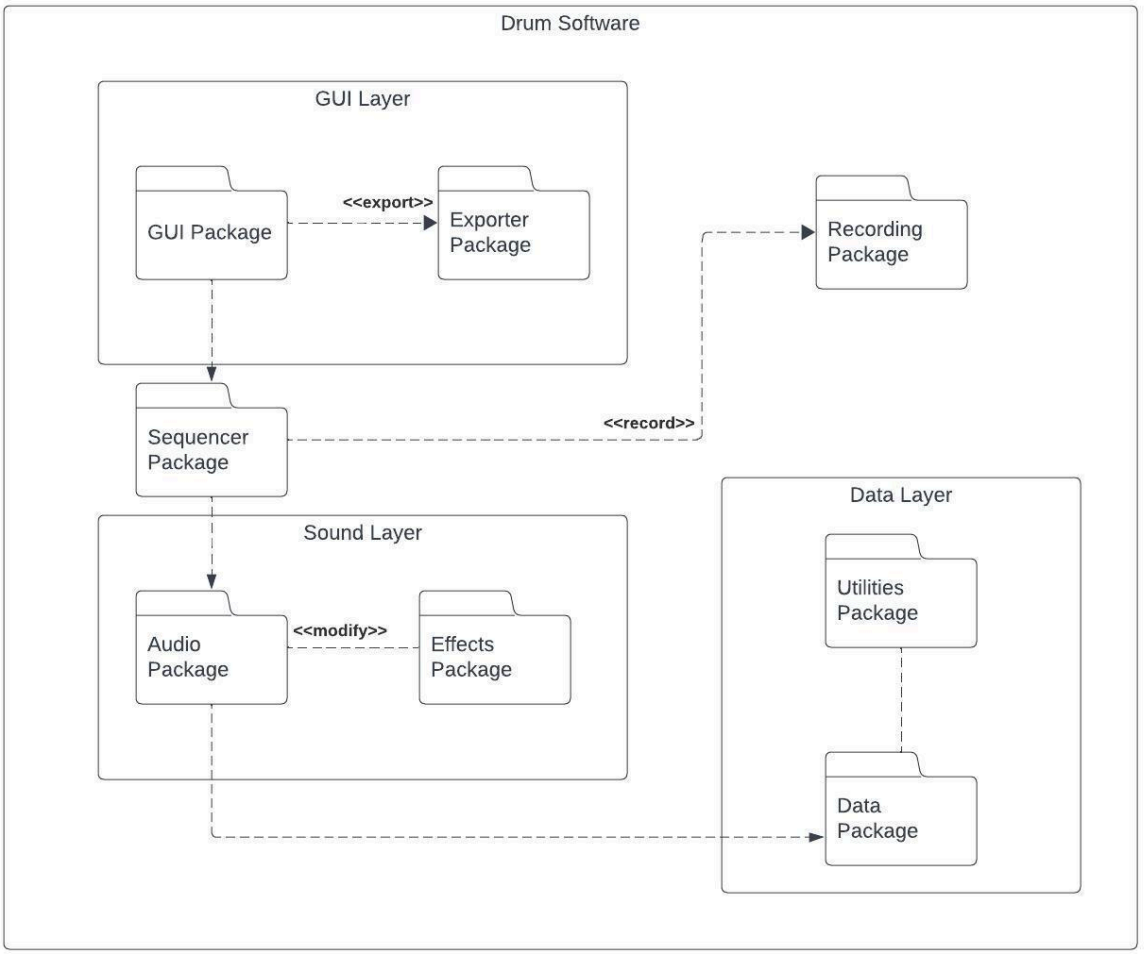
Feedback: We will add the reason for the distinction between basic and bonus features, here is a brief explanation of the reason, we are designing a simple musical instrument simulator, so we mainly focus on the realisation of the various functions of the instrument simulator, such as different instruments can produce the corresponding sound, you can program different combinations of musical instruments "musical sequence data ".

As for other features, such as exporting our programmed music, this is a more upgraded feature, not just a basic feature of an instrument simulator, so we set it as a bonus feature.

Package Diagram

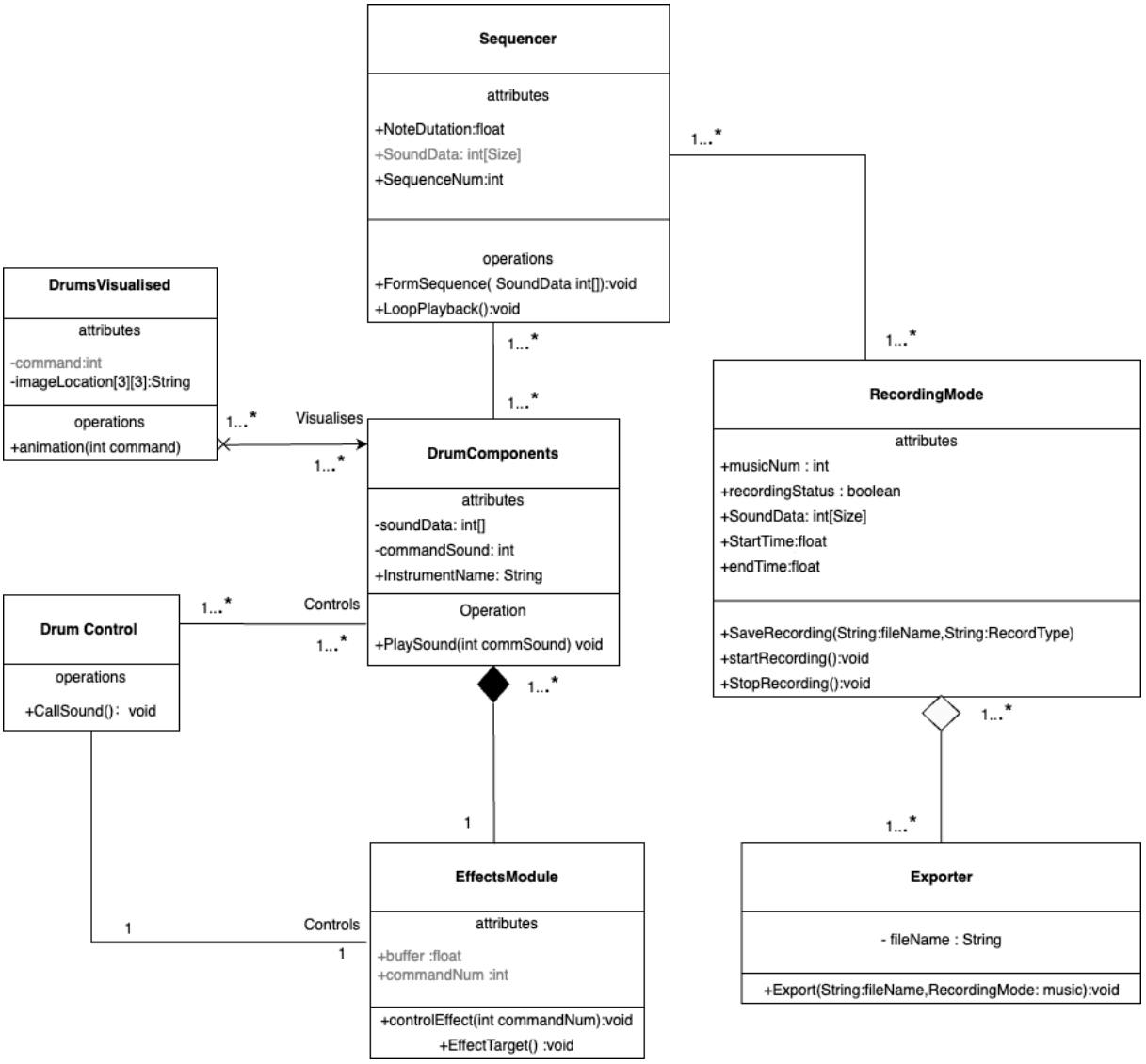
Author(s): Emre Yalcin

The package diagram illustrates the system's architecture across various layers of functionality. At the base, the Data Layer contains the Utilities Package, which provides shared services, and the Data Package for persistent storage operations. The Sound Layer, situated above, encompasses the Audio Package responsible for generating drum sounds, and the Effects Package which applies audio modifications. The GUI Layer, at the top, includes the GUI Package for user interface operations and the Exporter Package for exporting sequences. The Sequencer Package, bridging the GUI and Sound Layers, manages the order and timing of sounds. Additionally, the Recording Package, linked to the GUI Layer, handles audio capture. These packages interact through dependencies indicated by dashed arrows, with the directionality of the arrows representing the flow of data and control.



Class diagram

Author(s): Boyee, Lyudmil



DrumComponents

Representing various musical instruments or small parts of musical instruments, such as a drum, or a gong

Most relevant attributes

1. soundData: Different sound file data is present depending on the instrument, e.g. drums and trumpets have different sound files.
2. commandSound: Depending on different command numbers, it represents different instruments.

Most relevant operation

1. playSound: Play different sound files based on different inputs.

Most relevant association

Since almost all of the other classes in our system design will go into play as a result of interacting with the drum component, I'll describe the association of these classes to the drum component again when I say other classes

DrumVisualised

You can perform basic animation of some instruments on the picture according to the input of different instrument commands, for example, when the drum command is input, the drum will vibrate.

Most relevant attributes

1. Command: Representing different numbered instruments
2. imageLocation: Represents the location of the file where the different images are located

Most relevant operation

1. Animation: Play different instrument playback effects according to the entered instrument number.

Most relevant association

Binary association：DrumsVisualised can access different drum components to get their different command numbers(visible attributes), and play different corresponding animations according to the different command numbers.

Sequencer

You can perform basic animation of some instruments on the picture according to the input of different instrument commands, for example, when the drum command is input, the drum will vibrate. (More about sound management than animations)

Most relevant attributes

1. SoundData: Successive combinations of different instrument numbers represent a section of notes or drum rhythms
2. NoteDuration: Duration of notes

Most relevant operation

1. formSequence：Instrument combinations that play different sequences and note durations depending on the Sounddata

Most relevant association

Binary association：The sequencer may be linked to different drum components to produce a musical fragment.

RecordingMode

You can perform basic animation of some instruments on the picture according to the input of different instrument commands, for example, when the drum command is input, the drum will vibrate.

Most relevant attributes

1. recordingStatus: Indicates the current recording status, possibly a boolean value (true/false), to determine whether the software is currently in recording mode.
2. startTime: Records the start time of the recording session, used for calculating recording duration or note timestamps.
3. endTime: Records the end time of the recording session, used for calculating recording duration or note timestamps.

Most relevant operation

1. startRecording(): Initiates the recording process. Sets the recordingStatus to true and records the start time of the recording session.

2. stopRecording(): Stops the recording process. Sets the recordingStatus to false and records the end time of the recording session.
3. saveRecording(filename): Saves the recorded music to a file. This operation saves the note information from recordedNotes to the specified file for later playback or editing purposes.

Most relevant association

Binary association：for each music sequence, we need to use one recording mode to recording it, different recording modes have different numbers to indicate it difference.

Exporter

The exporter class is responsible as the name suggests for exporting the sounds or sequences of sounds that have previously been recorded with the class record. The sounds being previously recorded are a mandatory prerequisite. The targeted format is MP3.

Most relevant attributes

1. fileName:indicate the exporter music file name

Most relevant operation

1. Export: export the specific recording music.

Most relevant association

aggregation: exporter file could be independent as a file(beacuse not all the exporter file is recording file), but recording file could produce exporter file.

EffectsModule

Allows musicians to enhance and modify the sound of each drum component using a variety of audio effects. Musicians can apply a range of effects such as chorus, delay, distortion, and phaser to individual drum elements like the kick, snare, hi-hats, or to the entire drum kit. For instance, adding a chorus effect can give a fuller and more resonant sound to a snare drum, while a delay effect can create echoing beats that add rhythm complexity.

Most relevant attributes

commandNum: indicates different drum component numbers

Most relevant operation

controlEffect：Influencing the instrument playback of different drumcomponents

Most relevant association

composition：The effects are there to affect the different drum components, and without the drum components there would be no effect, or at least the class effect would not have any functionality and would not be able to exist independently.

Drum Control

A master console that allows you to modulate the sound of different drum components, allowing you to choose precisely which ones to use.

Most relevant operation

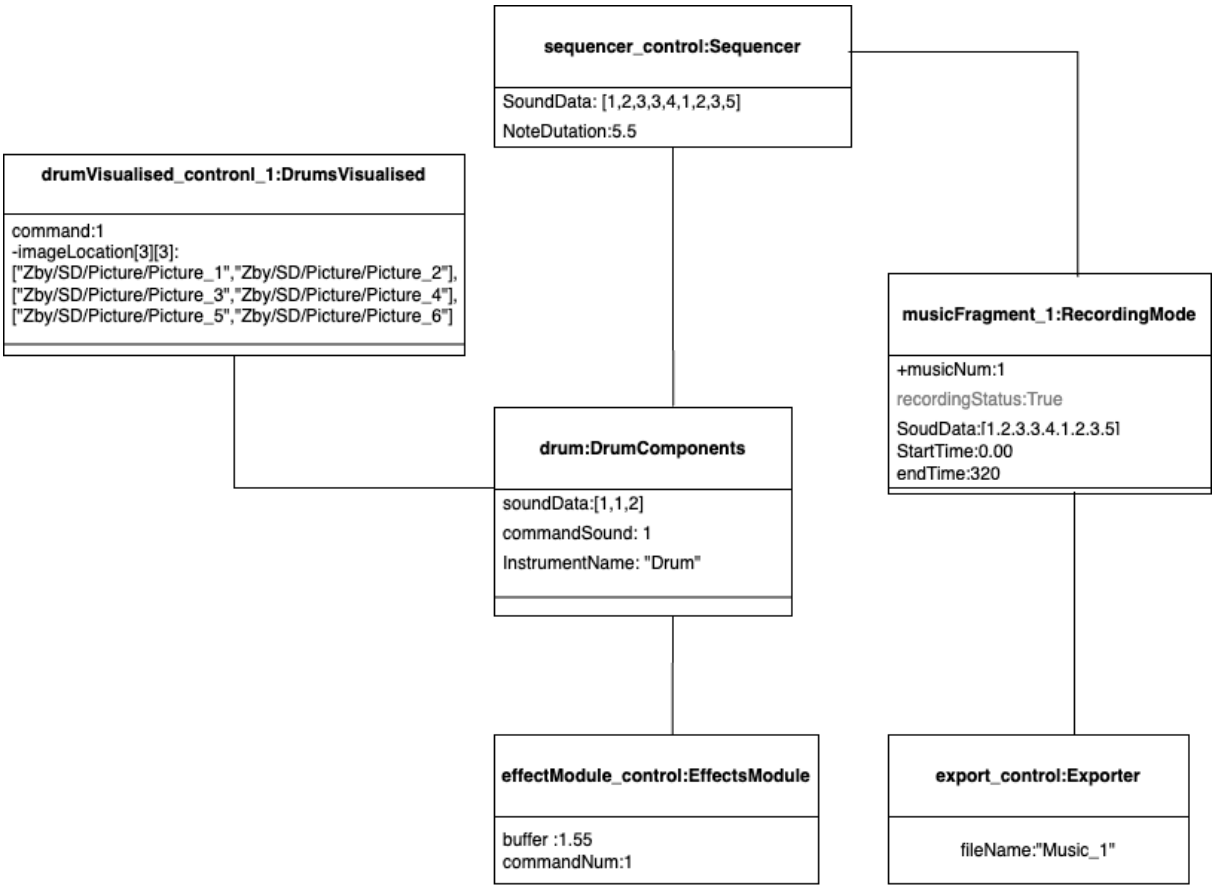
1. callSound: Play the sound of different components

Most relevant association

Binary association：control effects and different drum components to make sounds.

Object diagram

Author(s): Boyee

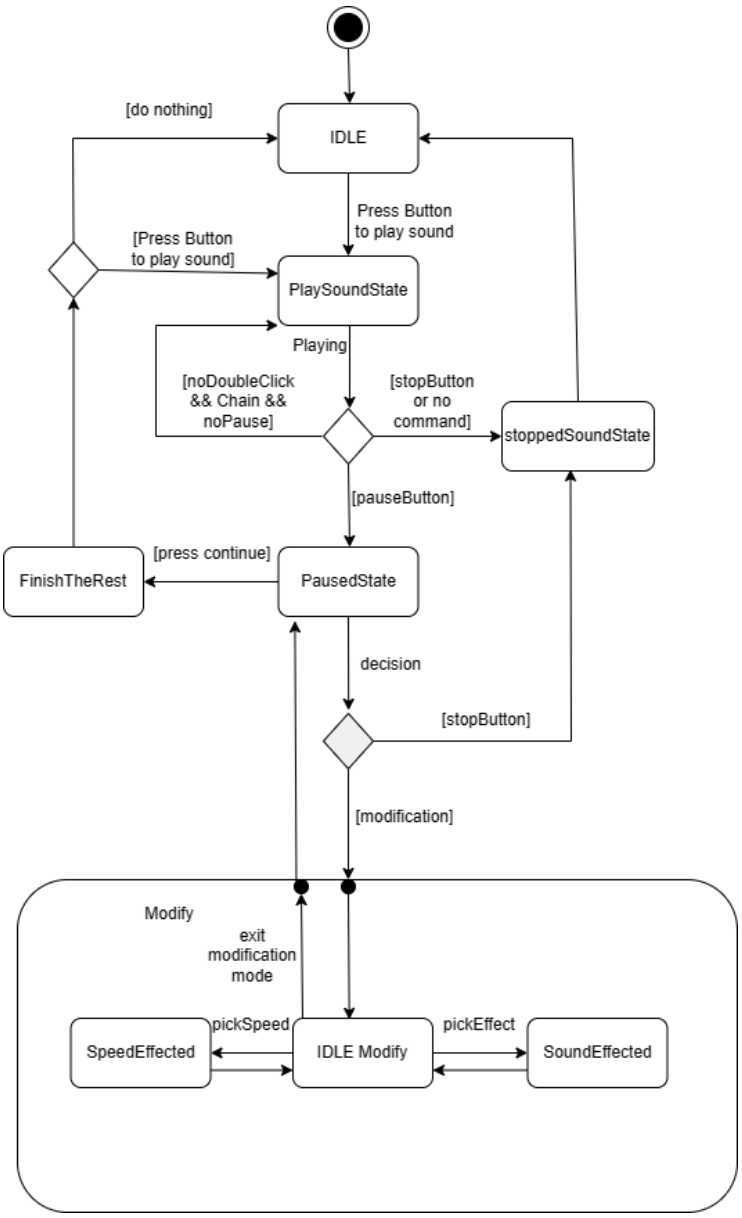


This is a “snapshot” of the scenario of a user creating several drum components, such as a bass drum, a gong, and a barrel drum, and changing the timbre and duration of some of the drum components via EffectsModule, and finally exporting the music fragment.

This current snapshot shows that we have created a drum instance, and through the effect module we can modify the timbre, and tempo of this drum, and through the drum visual instance we can control the animation performance of this drum, and at the same time the sequencer is controlling the other instruments, for example, 1 stand for drums, 2 stands for gongs, and 3 stands for heavy drums. The duration of the music clip is 5.5 seconds, and finally the bit rate and format of the exported music is set by recording, and finally music_1 is successfully exported.

State machine diagrams

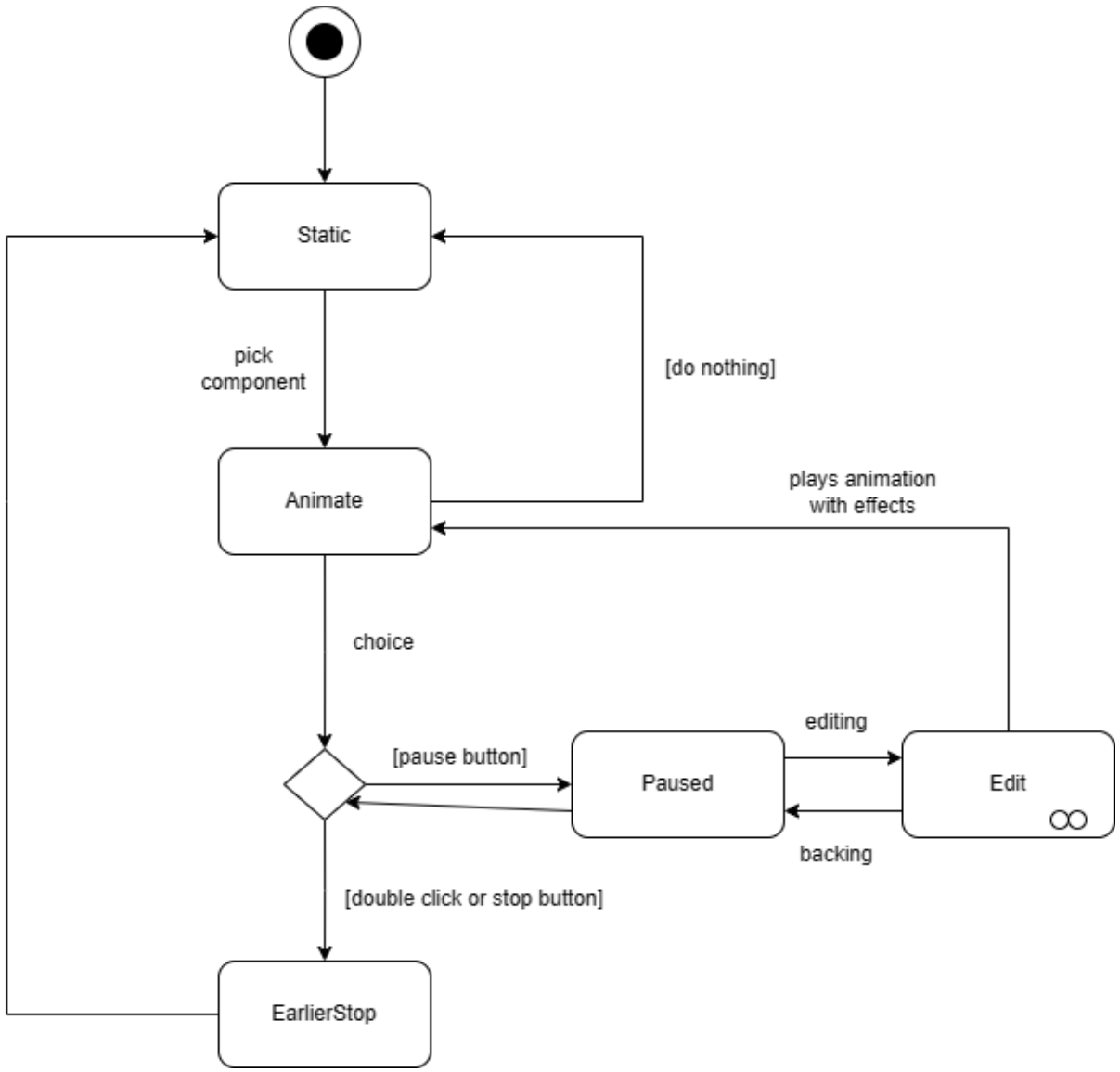
Author(s): Lyudmil Anastasov
Class: Drum Control



- States:
    1. IDLE - as the name suggests, during this state no sound is played and nothing else is done too
    2. PlaySoundState - this state is responsible for playing any of the different sounds we have prepared and in order to do that, the user has to click on one of the buttons on the GUI, or use the command line input. Once a sound is selected, it begins playing. Unlike the IDLE state, which only goes to the PlaySoundState, here we have quite a few options, namely: we can use the stop button to cut shortly the sound currently playing, or we can do nothing and we arrive at the same state, which is stop sound; another option is using the pause button, which leads to the PausedState; the final option is to return directly to PlaySoundState and play another sound, before the other one has finished playing.

3. PausedState - as the name indicates this is the state where the sound is paused with precision exactly when the user clicks the indicated button on the GUI. From here we have 3 options once more: finish the rest of the sound, go to the modification section or halt the sound completely.
4. IDLE_MODIFY - here we are once again doing nothing but we are presented with options to change the effect of the sound that will continue to be played.
5. SPEED_EFFECTED - the speed of the sound played has been increased
6. SOUND_EFFECTED - another type of effect on the sound (chorus..) Both affected states go back to IDLE_MODIFY. From where we can again choose to modify or exit the modification part and go back to the paused state.
7. FinishTheRest - rest of the sound has been played. Goes back to IDLE.
8. MODIFY is a composite state.

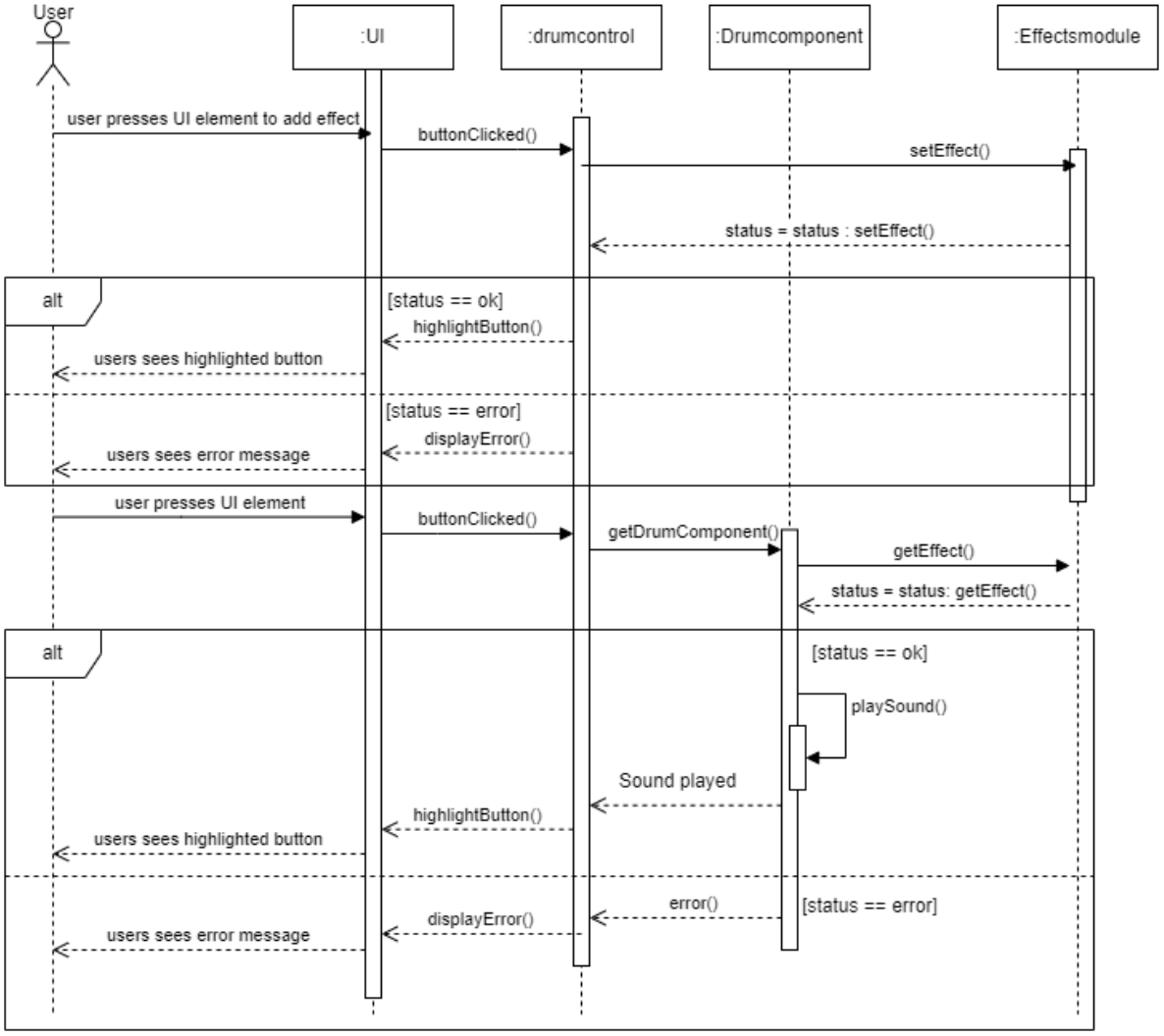
Class : DrumsVisualised
Author: Lyudmil Anastasov
This state machine diagram is similar to the previous one in the sense that it also has to use the effects module, so having a state machine exactly for it is convenient and we would not have to recreate it in this diagram. It has the same functionality as before, the difference here is that when we exit it, instead of hearing the effects, we see them in the Animate state. That is if we want that to happen, because otherwise we can cut the animation short and finish and go back to IDLE state. And finally we are once again presented with a choice after getting to the PAUSED state.



Sequence diagrams

Author(s): Jorrit Bogaard

Sequence diagram showing the user setting an effect and then pressing a drum button


User
:UI
:drumcontrol
:Drumcomponent
:Effectsmodule
user presses UI element to add effect
buttonClicked()
setEffect()
status = status : setEffect()
alt
[status == ok]
highlightButton()
users sees highlighted button
[status == error]
displayError()
users sees error message
user presses UI element
buttonClicked()
getDrumComponent()
getEffect()
status = status: getEffect()
alt
[status == ok]
playSound()
Sound played
highlightButton()
users sees highlighted button
error()
[status == error]
displayError()
users sees error message

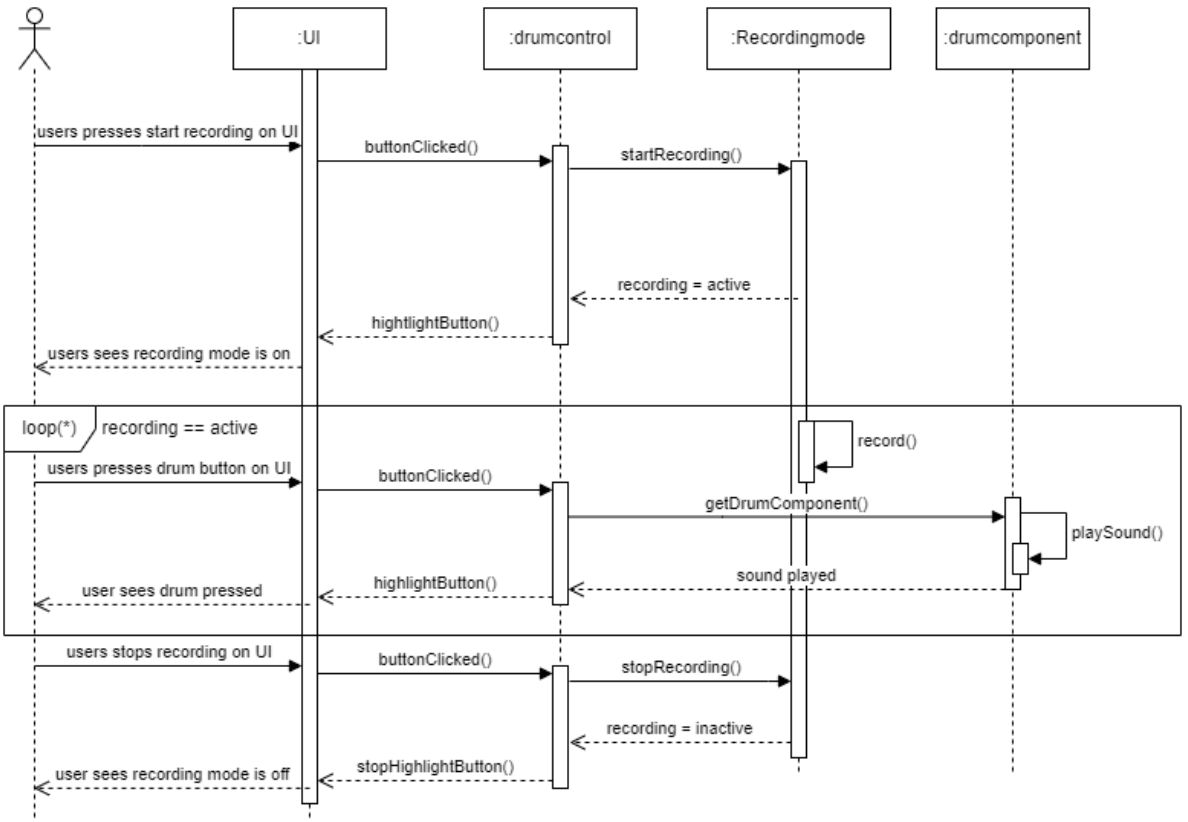
This sequence diagram shows the user setting an effect for the drums and then pressing a drum to hear the drum sound with the applied effect.

The interaction partners in this sequence diagram are the user and the classes UI, drumcontrol, DrumComponent and EffectsModule.

The user first presses a button on the UI to set an effect. The UI will then call the buttonClicked() function to let drum control know which button was clicked. Drumcontrol now knows it was an effect button so it calls setEffect with the appropriate effect and the Effectsmodule will now set this effect to active. It will respond to drumcontrol with a status code. If the status code is ok, DrumControl will call highlightButton(). This will cause the UI to highlight the pressed button so the user can see that the effect is active. If the status returns with error, DrumControl will call displayError(). This will cause the UI to show an error message to the user indicating that something went wrong when setting the effect.

Now that an effect is set to active the user wants to hear it so the user presses a drum button on the UI. The UI will again call the buttonClicked() function to let drumcontrol know which drum was pressed. Drumcontrol will then call getDrumComponent to let the corresponding drum component know it has to play a sound. Before playing the sound, the DrumComponent first checks if there are active effects. It does this by calling getEffect(). Effectsmodule will respond with the same messages as before. If the status is ok the sound will be played with the active effect. If not, an error message will be displayed.

Sequence diagram showing how a user would use the recording mode



This sequence diagram shows the user starting the recording mode, then entering some drums and then stopping the recording.

The interaction partners in this sequence diagram are the user and the classes UI, drumcontrol, Recrodingmode and Drumcomponent.

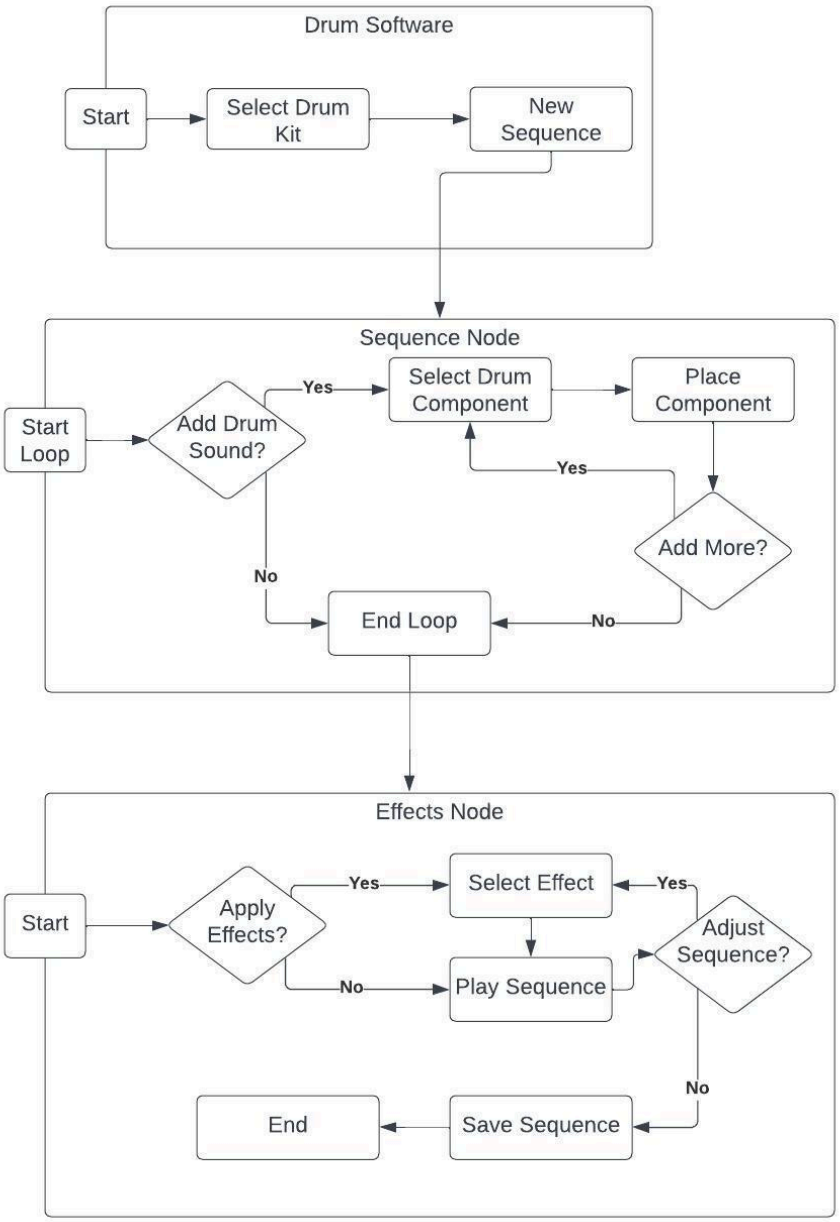
The user first presses the start recording button on the UI. The UI will then call the buttonClicked() function. Drumcontrol now knows that the user wants to start the recording and calls startRecording. Recordingmode responds by setting the recording variable to active. Drumcontrol will then call highlightButton(). The user can now see that the recording mode is on.

Now that recording mode is active, the loop in the sequence diagram will execute since recording == active. This loop will keep executing until the user presses the stop recording button on the UI. This will set the recording variable to inactive similarly to how it was set to active earlier.

While the loop is executing the user can press as many drum buttons as he likes. And the record() function keeps being called which will save the data of the pressed drum such as duration and type of drum.

Activity Diagram(Optional)

This activity diagram represents the workflow for creating a drum sequence in drum machine software. Starting with selecting a drum kit and initiating a new sequence, users can add and place drum components in a loop, apply optional effects, and finally play and save their customised sequence.



Time Logs

Team number	89		
Member	**Activity**	**Week number**	**Hours**
Lyudmil Anastasov	Define functional features	1	3
Jorrit Bogaard	Define functional features	1	2
Boyi Zhang	Define functional features	1	2
Emre Yalcin	Define functional features	1	3
Lyudmil Anastasov	Time log contribution	1	1
Jorrit Bogaard	Time log contribution	1	1
Boyi Zhang	Time log contribution	1	1
Emre Yalcin	Time log contribution	1	1
Lyudmil Anastasov	Contract	2	1
Jorrit Bogaard	Contract	2	1
Boyi Zhang	Contract	2	1
Emre Yalcin	Contract	2	1
Lyudmil Anastasov	Class Diagram	3	3
Jorrit Bogaard	Sequence Diagrams	3	5
Boyi Zhang	Class Diagram	3	3
Emre Yalcin	Package Diagram	3	5
Lyudmil Anastasov	State Machine Diagrams	3	5
Boyi Zhang	Object Diagram	3	4
Emre Yalcin	Activity Diagram	3	3
Boyi Zhang	Changes	3	4
		TOTAL:	**50**