```java
package hevs.fragil.patapon.music;

import java.util.Arrays;
import java.util.Vector;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.g2d.Animation.PlayMode;

import ch.hevs.gdx2d.lib.GdxGraphics;
import ch.hevs.gdx2d.lib.interfaces.DrawableObject;
import hevs.fragil.patapon.drawables.SpriteSheet;
import hevs.fragil.patapon.mechanics.Param;
import hevs.fragil.patapon.units.State;

/**
 * This class manages drums sequences,
 * and principally returns a new State depending of the recognized sequence.
 */
public class Sequence implements DrawableObject {
    //actual sequence
    private Vector<Drum> melody = new Vector<Drum>();
    private Vector<Drum> toDraw = new Vector<Drum>();
    private boolean sequenceInProgress = false;
    private boolean pause = true;
    private float sigmapisTimeCounter;
    private float drawCountDown;
    private float sinceLastDrum;
    private float sinceLastRythm;
    private int feverScore = 0;
    private static SpriteSheet drums;


    public State add(Drum d, float lastRythm){
        toDraw.add(d);
        drawCountDown = Param.NOTE_REMANENCE;

        if(!sequenceInProgress){
            sinceLastDrum = 0;
            sequenceInProgress = true;
        }
        if(pause)
            pause = false;
        else if (sigmapisTimeCounter > 0){
            pause = true;
            clearFever();
            endSequence();
            return State.IDLE;
        }

        feverScore += juge();

        sinceLastRythm = lastRythm;
        sinceLastDrum = 0;

        melody.add(d);

        State a = getAction();
        return a;
    }
    /**
     * @return value between 0 and 100
     */
    public int getFever() {
        return Math.min(feverScore, 100);
    }

    public void clearFever() {
        feverScore = 0;
    }

    /**
     * @return the corresponding action
     * */
    private State getAction(){
        if(melody.size() >= 4){
            //compare the last 5 ones
            int startIndex = Math.max(melody.size()-5, 0);
            int lastIndex = Math.min(5, melody.size());
            Drum[] last5Notes = new Drum[lastIndex];
            Drum[] last4Notes = new Drum[4];

            //get the last 4 or 5 notes in an array
            for(int i = 0 ; i < lastIndex ; i++){
                last5Notes[i] = melody.elementAt(i + startIndex);
            }

            //check if we need another array of 4 elements (equals function)
            //when checking for the 5 and the 4 last notes
            if(last5Notes.length >= 5){
                last4Notes = Arrays.copyOfRange(last5Notes,last5Notes.length-4, last5Notes.length);
            }

            //go through all possible actions and compare the current sequence to them
            //when a match is found, return the corresponding action
            for(int i = 0; i < Param.COMBOS.length; i++){
                if(Arrays.equals(last5Notes,Param.COMBOS[i]) || Arrays.equals(last4Notes,Param.COMBOS[i])){
                    System.out.println("Sequence " + State.values()[i] + " recognized !");
                    endSequence();
                    sigmapisTimeCounter = 2f;
                    return State.values()[i];
                }
            }

            //indicates bad sequence
            System.out.println("No possible sequence found... Fever goes down !");
            clearFever();
            endSequence();
            return State.IDLE;
        }

        return null;
    }
    public void step(){
        float dt = Gdx.graphics.getRawDeltaTime();

        sigmapisTimeCounter -= dt;
        sinceLastDrum += dt;
        verify();

        Vector<Drum> toRemove = new Vector<Drum>();
        for (Drum d : toDraw) {
            if(!sequenceInProgress){
                drawCountDown -= dt;
                if(drawCountDown <= 0){
                    toRemove.add(d);
                }
            }
        }
        //Remove elements
        for (Drum n : toRemove) toDraw.remove(n);
        toRemove.removeAllElements();
    }
    private void endSequence(){
        melody.removeAllElements();
        sequenceInProgress = false;
    }
    public void verify(){
        if(!pause){

            if((sequenceInProgress && sinceLastDrum > Param.MUSIC_BAR + Param.PASS)
                    || (!sequenceInProgress && sinceLastDrum > 5*Param.MUSIC_BAR + Param.PASS)){

                System.out.println("too long ! : " + sinceLastDrum);
                pause = true;
                clearFever();
                endSequence();
            }
        }
    }
    /**
     * returns a value depending of the user rythm phase
     * @return fever value between 15 and 0, -1 is a fail
     */
    private int juge(){
        float delay = Math.min(sinceLastRythm, Param.MUSIC_BAR-sinceLastRythm);
        if (delay < Param.PERFECT) {
            return 15;
        } else if (delay < Param.EXCELLENT) {
            return 10;
        } else if (delay < Param.GOOD) {
            return 5;
        } else if (delay < Param.PASS) {
            return 1;
        } else {
            System.out.println("Bad rythm ! : " + sinceLastRythm);
            pause = true;
            clearFever();
            endSequence();
            return 0;
        }
    }
    @Override
    public void draw(GdxGraphics g) {
        //draw elements
        int index = 0;
        for (Drum d : toDraw) {
            int x = (Param.CAM_WIDTH / 2 - 200) + (index % 4 * 100);
            float alpha = drawCountDown / Param.NOTE_REMANENCE;
            drums.drawFrameAlpha(d.ordinal(), x, 600, alpha);
            index++;
        }
    }
    /**
     * This is only to load files in the PortableApplication onInit method
     */
    public static void loadSprites(String url) {
        drums = new SpriteSheet(url, 1, 4, 0.2f, false, PlayMode.NORMAL);
    }
}
```