

```
00001: package hevs.fragil.patapon.music;
00002:
00003: import java.util.Arrays;
00004: import java.util.Vector;
00005:
00006: import com.badlogic.gdx.Gdx;
00007: import com.badlogic.gdx.graphics.g2d.Animation.PlayMode;
00008:
00009: import ch.hevs.gdx2d.lib.GdxGraphics;
00010: import ch.hevs.gdx2d.lib.interfaces.DrawableObject;
00011: import hevs.fragil.patapon.drawables.SpriteSheet;
00012: import hevs.fragil.patapon.mechanics.Param;
00013: import hevs.fragil.patapon.units.State;
00014:
00015: /**
00016:  * This class manages drums sequences,
00017:  * and principally returns a new State depending of the recognized sequence.
00018:  */
00019: public class Sequence implements DrawableObject {
00020:     //actual sequence
00021:     private Vector<Drum> melody = new Vector<Drum>();
00022:     private Vector<Drum> toDraw = new Vector<Drum>();
00023:     private boolean sequenceInProgress = false;
00024:     private boolean pause = true;
00025:     private float sigmapisTimeCounter;
00026:     private float drawCountDown;
00027:     private float sinceLastDrum;
00028:     private float sinceLastRythm;
00029:     private int feverScore = 0;
00030:     private static SpriteSheet drums;
00031:
00032:
00033:     public State add(Drum d, float lastRythm){
```

```
00034:         toDraw.add(d);
00035:         drawCountDown = Param.NOTE_REMANENCE;
00036:
00037:         if(!sequenceInProgress){
00038:             sinceLastDrum = 0;
00039:             sequenceInProgress = true;
00040:         }
00041:         if(pause)
00042:             pause = false;
00043:         else if (sigmapisTimeCounter > 0){
00044:             pause = true;
00045:             clearFever();
00046:             endSequence();
00047:             return State.IDLE;
00048:         }
00049:
00050:         feverScore += juge();
00051:
00052:         sinceLastRythm = lastRythm;
00053:         sinceLastDrum = 0;
00054:
00055:         melody.add(d);
00056:
00057:         State a = getAction();
00058:         return a;
00059:     }
00060:     /**
00061:      * @return value between 0 and 100
00062:      */
00063:     public int getFever() {
00064:         return Math.min(feverScore, 100);
00065:     }
00066:
```

```
00067:     public void clearFever() {
00068:         feverScore = 0;
00069:     }
00070:
00071:     /**
00072:      * @return the corresponding action
00073:      * */
00074:     private State getAction(){
00075:         if(melody.size() >= 4){
00076:             //compare the last 5 ones
00077:             int startIndex = Math.max(melody.size()-5, 0);
00078:             int lastIndex = Math.min(5, melody.size());
00079:             Drum[] last5Notes = new Drum[lastIndex];
00080:             Drum[] last4Notes = new Drum[4];
00081:
00082:             //get the last 4 or 5 notes in an array
00083:             for(int i = 0 ; i < lastIndex ; i++){
00084:                 last5Notes[i] = melody.elementAt(i + startIndex);
00085:             }
00086:
00087:             //check if we need another array of 4 elements (equals function)
00088:             //when checking for the 5 and the 4 last notes
00089:             if(last5Notes.length >= 5){
00090:                 last4Notes = Arrays.copyOfRange(last5Notes,last5Notes.length-4, last5Notes.length);
00091:             }
00092:
00093:             //go through all possible actions and compare the current sequence to them
00094:             //when a match is found, return the corresponding action
00095:             for(int i = 0; i < Param.COMBOS.length; i++){
00096:                 if(Arrays.equals(last5Notes,Param.COMBOS[i]) || Arrays.equals(last4Notes,Param.COMBOS[i])){
00097:                     System.out.println("Sequence " + State.values()[i] + " recognized !");
00098:                     endSequence();
00099:                     sigmapisTimeCounter = 2f;
```

```
00100:         return State.values()[i];
00101:     }
00102: }
00103:
00104:     //indicates bad sequence
00105:     System.out.println("No possible sequence found... Fever goes down !");
00106:     clearFever();
00107:     endSequence();
00108:     return State.IDLE;
00109: }
00110:
00111:     return null;
00112: }
00113: public void step(){
00114:     float dt = Gdx.graphics.getRawDeltaTime();
00115:
00116:     sigmapisTimeCounter -= dt;
00117:     sinceLastDrum += dt;
00118:     verify();
00119:
00120:     Vector<Drum> toRemove = new Vector<Drum>();
00121:     for (Drum d : toDraw) {
00122:         if(!sequenceInProgress){
00123:             drawCountDown -= dt;
00124:             if(drawCountDown <= 0){
00125:                 toRemove.add(d);
00126:             }
00127:         }
00128:     }
00129:     //Remove elements
00130:     for (Drum n : toRemove) toDraw.remove(n);
00131:     toRemove.removeAllElements();
00132: }
```

```
00133:     private void endSequence(){
00134:         melody.removeAllElements();
00135:         sequenceInProgress = false;
00136:     }
00137:     public void verify(){
00138:         if(!pause){
00139:
00140:             if((sequenceInProgress && sinceLastDrum > Param.MUSIC_BAR + Param.PASS)
00141:                 || (!sequenceInProgress && sinceLastDrum > 5*Param.MUSIC_BAR + Param.PASS)){
00142:
00143:                 System.out.println("too long ! : " + sinceLastDrum);
00144:                 pause = true;
00145:                 clearFever();
00146:                 endSequence();
00147:             }
00148:         }
00149:     }
00150:     /**
00151:      * returns a value depending of the user rythm phase
00152:      * @return fever value between 15 and 0, -1 is a fail
00153:      */
00154:     private int juge(){
00155:         float delay = Math.min(sinceLastRythm, Param.MUSIC_BAR-sinceLastRythm);
00156:         if (delay < Param.PERFECT) {
00157:             return 15;
00158:         } else if (delay < Param.EXCELLENT) {
00159:             return 10;
00160:         } else if (delay < Param.GOOD) {
00161:             return 5;
00162:         } else if (delay < Param.PASS) {
00163:             return 1;
00164:         } else {
00165:             System.out.println("Bad rythm ! : " + sinceLastRythm);
```

```
00166:         pause = true;
00167:         clearFever();
00168:         endSequence();
00169:         return 0;
00170:     }
00171: }
00172: @Override
00173: public void draw(GdxGraphics g) {
00174:     //draw elements
00175:     int index = 0;
00176:     for (Drum d : toDraw) {
00177:         int x = (Param.CAM_WIDTH / 2 - 200) + (index % 4 * 100);
00178:         float alpha = drawCountDown / Param.NOTE_REMANENCE;
00179:         drums.drawFrameAlpha(d.ordinal(), x, 600, alpha);
00180:         index++;
00181:     }
00182: }
00183: /**
00184:  * This is only to load files in the PortableApplication onInit method
00185:  */
00186: public static void loadSprites(String url) {
00187:     drums = new SpriteSheet(url, 1, 4, 0.2f, false, PlayMode.NORMAL);
00188: }
00189: }
```