JustLag Dance :
Afloja meneño

Medina Medina, David Alberto
Brito Ramos, Christian
Benlliure Jiménez, Mª Cristina
Martynas Zabulionis
López González, Néstor

27 de mayo de 2019

Índice general

Introducción

Este documento representa la memoria del trabajo final de la asignatura Creando Interfaces de Usuario para el curso 2018/19.

El objetivo del trabajo de curso consiste en la propuesta y desarrollo de un prototipo que haga uso de distintos elementos empleados a lo largo del curso, siendo estos:

1. Gráficos
2. Cámaras
3. Captura y producción de audio
4. Sensores

El trabajo que hemos desarrollado se basa en el mítico juego de consolas Just Dance ((JustLagDance, s.f.-a),(JustLagDance, s.f.-b)) y sus derivados. Una descripción simple de este juego popular consiste en que el jugador tiene que mostrar su destreza bailando, adquiriendo posturas corporales que la interfaz le indique con la finalidad de conseguir una gran puntuación. Para conseguir capturar las posturas del jugador, se hace uso de una Kinnect.

Nuestro prototipo es una versión muy simplificada del juego anteriormente mencionado, en el que el jugador puede seleccionar una canción de un repertorio que se encuentra en la pantalla principal, escribir su nombre de jugador y elegir entre una selección de máscaras una que llevará el esqueleto durante la partida, para seguidamente comenzar con el baile.

La metodología que sigue el baile es la siguiente:

1. Se muestra en una sección de la pantalla una silueta de persona haciendo una postura.
2. El usuario tendrá un límite de tiempo para conseguir hacer la postura lo mejor posible.
3. Una vez se ha cumplido ese tiempo, la puntuación obtenida al hacer dicha postura se suma a un marcador de puntos global.
4. Se muestra en pantalla otra postura y comienza el mismo procedimiento.
5. El proceso finaliza cuando se acaba la canción.

En la escena se muestra la estructura de un esqueleto humano resultado de un algoritmo de seguimiento del cuerpo que procesa la *Kinect v1.8* obteniendo como resultado la información de los puntos de articulación del mismo (DART, s.f.).

La *Kinect* solo ofrece la posición de cada punto de articulación en el plano XY por lo que es necesario analizar la información que de los sensores IR si queremos conocer la profundidad de cada uno de los puntos de articulación que conforman el esqueleto.

Al finalizar la canción se le mostrará al usuario un ranking top 10 de las mejores puntuaciones del juego.

Detección de posturas

El método empleado para la detección de posturas evalúa dos esqueletos, ambos capturados con la Kinnect. Llamaremos esqueleto a la detección de un ser humano por la Kinnect. Este esqueleto se almacena en memoria como un vector que contiene las cordenadas (X,Y,Z) de los puntos articulados del mismo.

Los datos son cargados de un CSV donde se persiste la información de la nube de puntos que define la postura modelo de un movimiento de baile. Cada postura de baile queda identificada con una clave única siguiendo el estándar `UUID versión 4`.

Una vez ya tenemos las dos posturas que vamos a evaluar, es necesario estandarizar la postura de tal forma que se pueda comparar. Lo que hemos hecho, es escoger el punto de la espina (SPINE_X,SPINE_Y,SPINE_Z) como punto de referencia, de tal manera que transladaremos el esqueleto completo al origen de coordenas resultando el punto anterior como el (0,0,0).

El siguiente paso a seguir es calcular la distancia euclídea de cada uno de los puntos del esqueleto que vamos a tomar de referencia con los puntos del esqueleto que vamos a evaluar. Este valor de la distancia, lo vamos a considerar el error en cada punto, por lo que si sumamos todas las distancias entre todos los puntos de ambos esqueletos, obtendremos el error global. Cuanto menor sea este valor, más similitud hay entre las posturas de los esqueletos. Si el valor total del error es 0, se considera que las dos posturas son la misma.

Método y materiales

3.1. Materiales

El desarrollo de este proyecto se ha llevado a cabo utilizando el IDE de desarrollo de aplicaciones *Java* de *JetBrains*, *IntelliJ*, y las siguientes herramientas:

- Librería *Processing* (Processing, s.f.).
- Librería *Kinect4WinSDK* (Chung, s.f.).
- Librería *Sound* (Foundation, s.f.).
- Librería *Minim*.
- Librería *Commons-CSV* de *Apache*
- *Kinect v1.8* (Microsoft, s.f.)
- Botones e iconos de juego desarrollados con *Photoshop* e *Illustrator*.

Las imágenes no propias del grupo que se usaron para el juego son las siguientes:

- La corona y las medallas de la pantalla de ranking
- Los iconos de la pantalla de juego
- Las máscaras de la pantalla previa al juego.

La mayoría de estas imágenes se obtuvieron de la siguiente página https://www.flaticon.com/ (Flaticon, s.f.). Donde desde la propia página se puede cambiar los colores del icono y el tamaño. Las máscaras del personaje se encontraron en google ya que se eligieron algunas en modo demo.

3.2. Método

Las siguientes clases que se definenen en este documente se organizan en los siguientes paquetes:

1. model
2. gui
3. control

3.2.1. Paquete `gui`

Clase `Main`

Para poder utilizar las primitivas de *Processing* esta clase debe heredar de `PApplet`. Para iniciar una aplicación de *Processing*, el método estático `main()` debe llamar a la primitiva `PApplet.main()` para indicar el nombre de la clase principal desde la cual se llama y sobrescriben los métodos primitivos: `settings()`, `setup()` y `draw()`.

```
1    public static void main(String[] args) {
2        PApplet.main("gui.Main");
3    }
```

El método primitivo `settings()` establece el tamaño de la pantalla llamando al método `size()` y pasándole la marca del renderizador de gráficos 3D (`P3D`).

```
1    @Override
2    public void settings() {
3        super.settings();
4        size(640, 480, P3D);
5    }
```

El método `setup()` inicializa las pantallas que serán utilizadas en *JustLag Dance* – pantalla principal, de preludio, de juego y de ranking.

```
1    @Override
2    public void setup() {
3        super.setup();
4        smooth();
5        stroke(255);
6
7        initializeUI();
8
9        screen = new MainScreen(this, UIResources, SOUND_DIR + "\\songs
     ");
10       gameScreen = new GameScreen(this, UIResources, SOUND_DIR + "\\
     fx\\count_beep.wav",
11               SOUND_DIR + "\\fx\\start_beep.wav", POSTURES_CSV,
     BASE_IMG_DIR + "\\postures\\");
12       gameScreen.setInitialCount();
13       preludeScreen = new PreludeScreen(this, UIResources,
     BASE_IMG_DIR + "\\masks\\");
14       rankingScreen = new RankingScreen(this, UIResources,
     RANKING_CSV);
15   }
```

El método privado `initializeUI()` inicializa todos los botones e iconos que serán utilizados en el juego.

```
1    private void initializeUI() {
2        UIResources = new HashMap<>();
3        UIResources.put(UISelector.NOW, loadImage(BASE_IMG_DIR + "\\ui
     \\ya.png", "png"));
4        UIResources.put(UISelector.START, loadImage(BASE_IMG_DIR + "\\
     ui\\start.png", "png"));
5        UIResources.put(UISelector.ONE, loadImage(BASE_IMG_DIR + "\\ui
     \\1.png", "png"));
6        UIResources.put(UISelector.TWO, loadImage(BASE_IMG_DIR + "\\ui
     \\2.png", "png"));
7        UIResources.put(UISelector.THREE, loadImage(BASE_IMG_DIR + "\\
     ui\\3.png", "png"));
```

```
        UIResources.put(UISelector.CHRONO, loadImage(BASE_IMG_DIR + "\\ui\\stopwatch.png", "png"));
        UIResources.put(UISelector.BACK, loadImage(BASE_IMG_DIR + "\\ui\\back.png", "png"));
        UIResources.put(UISelector.BACK_OVER, loadImage(BASE_IMG_DIR + "\\ui\\back_over.png", "png"));
        UIResources.put(UISelector.BACK_PRESSED, loadImage(BASE_IMG_DIR + "\\ui\\back_pressed.png", "png"));
        UIResources.put(UISelector.PAUSE, loadImage(BASE_IMG_DIR + "\\ui\\pause.png", "png"));
        UIResources.put(UISelector.PAUSE_OVER, loadImage(BASE_IMG_DIR + "\\ui\\pause_over.png", "png"));
        UIResources.put(UISelector.PAUSE_PRESSED, loadImage(BASE_IMG_DIR + "\\ui\\pause_pressed.png", "png"));
        UIResources.put(UISelector.TITLE, loadImage(BASE_IMG_DIR + "\\ui\\titulo.png", "png"));
        UIResources.put(UISelector.CROWN, loadImage(BASE_IMG_DIR + "\\ui\\crown.png", "png"));
        UIResources.put(UISelector.GOLD_MEDAL, loadImage(BASE_IMG_DIR + "\\ui\\gold-medal.png", "png"));
        UIResources.put(UISelector.SILVER_MEDAL, loadImage(BASE_IMG_DIR + "\\ui\\silver-medal.png", "png"));
        UIResources.put(UISelector.BRONZE_MEDAL, loadImage(BASE_IMG_DIR + "\\ui\\bronze-medal.png", "png"));

        UIResources.get(UISelector.NOW).resize(width / 10, 0);
        UIResources.get(UISelector.BACK).resize(0, UIResources.get(UISelector.BACK).height * height / 768);
        UIResources.get(UISelector.BACK_OVER).resize(0, UIResources.get(UISelector.BACK).height);
        UIResources.get(UISelector.BACK_PRESSED).resize(0, UIResources.get(UISelector.BACK).height);
        UIResources.get(UISelector.PAUSE).resize(0, UIResources.get(UISelector.PAUSE).height * height / 768);
        UIResources.get(UISelector.PAUSE_OVER).resize(0, UIResources.get(UISelector.PAUSE).height);
        UIResources.get(UISelector.PAUSE_PRESSED).resize(0, UIResources.get(UISelector.PAUSE).height);
        UIResources.get(UISelector.CHRONO).resize(0, height / 15);
        UIResources.get(UISelector.CROWN).resize(0,50);
        UIResources.get(UISelector.GOLD_MEDAL).resize(0,20);
        UIResources.get(UISelector.SILVER_MEDAL).resize(0,20);
        UIResources.get(UISelector.BRONZE_MEDAL).resize(0,20);
    }
```

El método primitivo `draw()` refresca el estado del juego y muestra la pantalla correspondiente en cada caso.

```
    @Override
    public void draw() {
        background(10, 255);
        rect(0,0,width,height);
        this.clear();
        if (isGameScreen) {
            gameScreen.show();
            if ("00:00".equals(gameScreen.getSong().timeLeft())) {
                isGameScreen = false;
                isRankingScreen = true;
                rankingScreen.loadScore(preludeScreen.getPlayerName(), gameScreen.getScore());
            }
        } else if (isPreludeScreen) {
            preludeScreen.show();
```

```
        } else if (isRankingScreen) {
            rankingScreen.show();
        } else {
            screen.show();
        }
    }
```

El método primitivo `mouseReleased()` lee los eventos de ratón en el momento en el que el usuario haga clic sobre cualquier botón de pantalla.

```
    @Override
    public void mouseReleased() {
        if (!isGameScreen && !isPreludeScreen && !isRankingScreen &&
    screen.mouseOverButtonJugar()) {
            isPreludeScreen = true;
        } else if (!isGameScreen && !isPreludeScreen && !
    isRankingScreen && screen.mouseOverButtonPrevious()) {
            screen.setPreviousSong();
        } else if (!isGameScreen && !isPreludeScreen && !
    isRankingScreen && screen.mouseOverButtonNext()) {
            screen.setNextSong();
        } else if (!isGameScreen && isPreludeScreen && !isRankingScreen
     && preludeScreen.mouseOverPlay()) {
            isGameScreen = true;
            isPreludeScreen = false;
            screen.getCurrentSong().stop();
            gameScreen.setSong(screen.getCurrentSong());
            gameScreen.setMask(preludeScreen.getSelectedMask());
        } else if (!isGameScreen && isPreludeScreen && !isRankingScreen
     && preludeScreen.mouseOverBack()) {
            isPreludeScreen = false;
            screen.getCurrentSong().stop();
            screen.getCurrentSong().play();
        } else if (isGameScreen && !isPreludeScreen && !isRankingScreen
     && gameScreen.mouseOverButtonPause()) {
            gameScreen.pause();
        } else if (isGameScreen && !isPreludeScreen && !isRankingScreen
     && gameScreen.mouseOverButtonBack()) {
            screen.getCurrentSong().stop();
            screen.getCurrentSong().play();
            isGameScreen = false;
            isPreludeScreen = false;
            gameScreen.setInitialCount();
        } else if (!isGameScreen && isPreludeScreen && !isRankingScreen
    ) {
            preludeScreen.mouseOverLeft();
            preludeScreen.mouseOverRight();
        } else if (!isGameScreen && !isPreludeScreen && isRankingScreen
     && rankingScreen.mouseOverContinue()) {
            isRankingScreen = false;
            rankingScreen.reset();
            gameScreen.setInitialCount();
            screen.getCurrentSong().stop();
            screen.getCurrentSong().play();
        }
    }
```

La pantalla de preludio lee los eventos de teclado a la espera de que el usuario introduzca su nombre de jugador.

```
    @Override
    public void keyPressed() {
```

```
        if (!isGameScreen && isPreludeScreen) {
            preludeScreen.keyboardTextArea();
        }
    }
```

A continuación, se gestionan los eventos primitivos de la cámara *Kinect* delegando el tratamiento de los mismo en la clase `GameScreen`.

```
    public void appearEvent(SkeletonData _s) {
        gameScreen.appearEvent(_s);
    }

    public void disappearEvent(SkeletonData _s) {
        gameScreen.disappearEvent(_s);
    }

    public void moveEvent(SkeletonData _b, SkeletonData _a) {
        gameScreen.moveEvent(_b, _a);
    }
```

Clase abstracta `Screen`

Se encarga de proporcionar el método abstracto `show()` a las clases `MainScreen`, `PreludeScreen`, `GameScreen` y `RankingScreen`.

Esta clase contiene, además, el mapa de recursos que será utilizado en el juego, así como los método `readDancerDataFromCSV()` que es necesarios para leer puntos de posturas modelos de nuestra biblioteca de posturas almacenadas en el fichero `data_postures.csv`.

Los métodos `writeDancerDataToCSV()` y el método priviado `generateHeaders()` son utilizados para escribir nuevos modelos en la biblioteca de posturas de baile.

```
    abstract void show();

    List<DancerData> readDancerDataFromCSV() {
        List<String> headers = generateHeaders(KinectAnathomy.
    NOT_TRACKED, KinectAnathomy.LABEL);

        HashMap<String, List<String>> allMoves = CSVTools.readCSV(
                Paths.get(DANCE_POSTURES_CSV_FILE),
                CSVFormat.EXCEL,
                headers.toArray(new String[headers.size()]));

        List<DancerData> ddl = new ArrayList<>();

        for (String key :
                allMoves.keySet()) {

            int i = 0;
            List<String> values = allMoves.get(key);
            PVector v = new PVector();
            HashMap<KinectAnathomy, PVector> data = new HashMap<>();

            for (String header :
                    headers) {
                if (header.endsWith("X")) {
                    v = new PVector();
                    v.x = Float.parseFloat(values.get(i));
                } else if (header.endsWith("Y")) {
                    v.y = Float.parseFloat(values.get(i));
```

```
                } else if (header.endsWith("Z")) {
                    v.z = Float.parseFloat(values.get(i));
                    data.put(KinectAnathomy.getEnumById(header.split("
    _Z")[0]), v);
                }
                i += 1;
            }
            ddl.add(new DancerData(parent, key, data));
        }

        return ddl;
    }

    protected boolean writeDancerDataToCSV(Kinect kinect, boolean
    printHeader) {
        DancerData dd = new DancerData(parent, kinect);

        List<String> headers = generateHeaders(KinectAnathomy.LABEL,
    KinectAnathomy.NOT_TRACKED);
        CSVFormat format;
        if (printHeader) {
            format = CSVFormat.EXCEL.withHeader(headers.toArray(new
    String[headers.size()]));
        } else {
            format = CSVFormat.EXCEL;
        }

        CSVTools.writeCSV(Paths.get(DANCE_POSTURES_CSV_FILE), format,
    dd.getDancerUUID(), dd.getAnathomyData());
        System.out.println("DANCE POSTURE SAVED (" + dd.getDancerUUID()
     + ")");

        return false;
    }

    private List<String> generateHeaders(KinectAnathomy label,
    KinectAnathomy notTracked) {
        List<String> headers = new ArrayList<>();
        headers.add("ID");
        for (KinectAnathomy ka :
                KinectAnathomy.values()) {
            if (!label.equals(ka) && !notTracked.equals(ka)) {
                headers.add(ka.getId() + "_X");
                headers.add(ka.getId() + "_Y");
                headers.add(ka.getId() + "_Z");
            }
        }
        return headers;
    }
```

Clase `MainScreen`

Su método `show()` se encarga de la creación de la pantalla principal, donde se puede elegir una canción para el juego. Se cambia la canción pulsando algún de los do botones (la previa o siguiente canción) y al cambiar ella se empieza a reproducirse. Después de elegirla se pulsa el botón `JUGAR`.

El botón `JUGAR` y el selector de canciones es mostrado por pantalla cuando se llama al método público `show()`.

```
    void show()
```

```
    {
        parent.background(10);
        parent.imageMode(parent.CENTER);
        parent.image(titulo, parent.width/2, parent.height / 3, parent.
    width / 1.5f, parent.height / 1.5f);

        // Anadimos el boton "JUGAR"
        parent.strokeWeight(5);
        if (mouseOverButtonJugar())
        {
            if (parent.mousePressed)
                parent.fill(100, 0, 0);
            else
                parent.fill(255, 0, 0);

            parent.stroke(100, 255, 0);
        }
        else
        {
            parent.stroke(0, 100, 255);
            parent.fill(255, 0, 0);
        }

        parent.rectMode(parent.CENTER);
        parent.rect(parent.width/2, parent.height/1.125f, parent.width
    / 7, parent.height / 12, 8);

        // Escribimos "JUGAR" en el boton
        parent.fill(0);
        parent.textSize(parent.height / 30.72f);
        parent.textAlign(parent.CENTER, parent.CENTER);
        parent.text("JUGAR", parent.width/2, parent.height/1.13f);

        // Hacemos reproducir la cancion

        // Anadimos campo de titulo de cancion
        parent.strokeWeight(1);
        parent.stroke(250);

        parent.fill(150, 150, 150);
        parent.rectMode(parent.CENTER);
        parent.rect(parent.width/2, parent.height/1.35f, parent.width /
     2, parent.height / 13, 2);

        // Anadimos el titulo de la cancion
        parent.fill(0);
        parent.textSize(parent.height / 30.72f);
        parent.textAlign(parent.CENTER, parent.CENTER);
        String song = SongsNames.get(currentSong);
        parent.text(song.substring(0, song.lastIndexOf('.')), parent.
    width/2, parent.height/1.36f);


        // Anadimos los botones "previa cancion" y "proxima cancion"
        parent.strokeWeight(3);
        if (currentSong == 0)
        {
            parent.stroke(50, 50, 50);
            parent.fill(100, 100, 100);
        }
        else
        {
```

```
60              if (mouseOverButtonPrevious())
61              {
62                  if (parent.mousePressed)
63                      parent.fill(0, 100, 0);
64                  else
65                      parent.fill(0, 255, 0);
66
67                  parent.stroke(100, 255, 0);
68              }
69              else
70              {
71                  parent.stroke(0, 100, 255);
72                  parent.fill(0, 255, 0);
73              }
74          }
75          parent.triangle(parent.width / 5.5f, parent.height/1.35f,
    parent.width / 4.5f, parent.height/1.35f - parent.height / 26,
    parent.width / 4.5f, parent.height/1.35f + parent.height / 26);
76
77          if (currentSong == numOfSongs - 1)
78          {
79              parent.stroke(50, 50, 50);
80              parent.fill(100, 100, 100);
81          }
82          else
83          {
84              if (mouseOverButtonNext())
85              {
86                  if (parent.mousePressed)
87                      parent.fill(0, 100, 0);
88                  else
89                      parent.fill(0, 255, 0);
90
91                  parent.stroke(100, 255, 0);
92              }
93              else
94              {
95                  parent.stroke(0, 100, 255);
96                  parent.fill(0, 255, 0);
97              }
98          }
99          parent.triangle(parent.width - parent.width / 5.5f, parent.
    height/1.35f, parent.width - parent.width / 4.5f, parent.height
    /1.35f - parent.height / 26, parent.width - parent.width / 4.5f,
    parent.height/1.35f + parent.height / 26);
100     }
```

La clase carga las canciones de cierta carpeta, dibuja los tres botones, campo del título de canción, pone el logo e implementa varias funciones. Las funciones `mouseOverButtonJugar()`, `mouseOverButtonPrevious()`, `mouseOverButtonNext()` para saber si los botones están pulsados. Las funciones `setNextSong()` y `setPreviousSong()` sirven para cambiar la canción. Y por último `getCurrentSong()` para manejar la canción actual.

```
1       Boolean mouseOverButtonJugar()
2       {
3           return parent.mouseX >= parent.width/2 - parent.width / 14 &&
    parent.mouseX <= parent.width/2 + parent.width / 14
4                   && parent.mouseY >= parent.height/1.125f - parent.
    height / 24 && parent.mouseY <= parent.height/1.125f + parent.
    height / 24;
5       }
```

```
6       Boolean mouseOverButtonPrevious()
7       {
8           if (currentSong == 0)
9               return false;
10
11          float coef = PApplet.map(parent.mouseX, parent.width / 5.5f,
        parent.width / 4.5f, 1, 0);
12          return parent.mouseX >= parent.width / 5.5f && parent.mouseX <=
         parent.width / 4.5f
13                  && parent.mouseY >= parent.height/1.35f - parent.height
         / 26 + (parent.height / 26 * coef)
14                  && parent.mouseY <= parent.height/1.35f + parent.height
         / 26 - (parent.height / 26 * coef);
15      }
16      Boolean mouseOverButtonNext()
17      {
18          if (currentSong == numOfSongs - 1)
19              return false;
20
21          float coef = PApplet.map(parent.mouseX, parent.width - parent.
        width / 5.5f, parent.width - parent.width / 4.5f, 1, 0);
22          return parent.mouseX >= parent.width - parent.width / 4.5f &&
        parent.mouseX <= parent.width - parent.width / 5.5f
23                  && parent.mouseY >= parent.height/1.35f - parent.height
         / 26 + (parent.height / 26 * coef)
24                  && parent.mouseY <= parent.height/1.35f + parent.height
         / 26 - (parent.height / 26 * coef);
25      }
26      public int getNumberOfSongs()
27      {
28          return numOfSongs;
29      }
30      void setNextSong()
31      {
32          Songs.get(currentSong).stop();
33          Songs.get(++currentSong).play();
34      }
35      void setPreviousSong()
36      {
37          Songs.get(currentSong).stop();
38          Songs.get(--currentSong).play();
39      }
40      public int getCurrentSongID()
41      {
42          return currentSong;
43      }
44      // No se si se necesita
45      Song getCurrentSong()
46      {
47          return Songs.get(currentSong);
48      }
```

Clase `PreludeScreen`

En la Clase PreludeScreen se agrupan todos los métodos que se encargan del diseño y de la lectura de la información para la pantalla previa al juego, donde el usuario indica su nombre y selecciona la máscara de su personaje.

Además, el jugador podrá seleccionar la máscara de baile de su avatar virtual, así como indicar su nombre de jugador.

```
    PreludeScreen(PApplet parent, HashMap<UISelector, PImage>
    uiResources, String maskPath) {
        super(parent, uiResources);
        this.parent = parent;

        MASKS_BASE_DIR = maskPath;

        playerName = "Jugador1";
        loadMask();
    }

    private void loadMask() {
        mask.add(parent.loadImage(MASKS_BASE_DIR + "mask8.png", "png"))
    ;
        mask.add(parent.loadImage(MASKS_BASE_DIR + "mask7.png", "png"))
    ;
        mask.add(parent.loadImage(MASKS_BASE_DIR + "mask1.png", "png"))
    ;
        mask.add(parent.loadImage(MASKS_BASE_DIR + "mask2.png", "png"))
    ;
        mask.add(parent.loadImage(MASKS_BASE_DIR + "mask3.png", "png"))
    ;
        mask.add(parent.loadImage(MASKS_BASE_DIR + "mask4.png", "png"))
    ;
        mask.add(parent.loadImage(MASKS_BASE_DIR + "mask5.png", "png"))
    ;
        mask.add(parent.loadImage(MASKS_BASE_DIR + "mask6.png", "png"))
    ;
    }

    public void show() {
        parent.background(0);
        parent.textMode(parent.SHAPE);
        parent.rectMode(parent.CORNER);
        parent.textMode(parent.CORNER);
        parent.textAlign(parent.CORNER, parent.CORNER);
        parent.imageMode(parent.CORNER);
        screenSelectName();
    }

    void screenSelectName(){
        parent.background(0);

        PImage back = UIResources.get(UISelector.BACK);
        parent.image(back, 10, 10);

        parent.textSize(26);
        parent.fill(255);
        parent.text("Nombre del jugador: ", 100, parent.height/5 );
        parent.stroke(100);
        parent.fill(100);

        parent.rect(100, parent.height/4, 310, 50); //rectangulo para
    el texto
        parent.fill(255);
        parent.text(playerName, 120, parent.height/3-5 );
        screenSelectMask();

        //Boton de Play
        parent.fill(85, 154, 232);
        parent.rect(parent.width/2-100/2, parent.height-70, 100, 50,7);
     //rectangulo para el texto
```

```
52          parent.fill(255);
53          parent.text("Jugar", parent.width/2 - 50/2 - 3, parent.height
    -38 );
54      }
55
56      String getPlayerName() {
57          return playerName;
58      }
59
60      void screenSelectMask() {
61          parent.fill(255);
62          parent.text("Mascara del jugador: ", 100, parent.height/2 );
63
64          PImage m = mask.get(maskSelected);
65          m.resize(0,100);
66          parent.image(m, 280, parent.height / 2 + 50);
67
68          if(maskSelected == 0) parent.fill(100); //si no se puede pulsar
     mas se pone en gris
69          parent.triangle(200,300,180,330,200,360);
70          parent.fill(255);
71          if (mask.size() -1 == maskSelected) parent.fill(100);
72          parent.triangle(440,300,460,330,440,360);
73      }
74
75      PImage getSelectedMask() {
76          return mask.get(maskSelected);
77      }
```

Por lo tanto nos encontramos con el control de los eventos de ratón y teclado para esa pantalla (`mouseOverBack()`, `mouseOverPlay()`, `mouseOverMainScreen()`, `mouseOverLeft()`, `mouseOverRight()`, keyboardTextArea()), los cuales se usan para el boton de `atras`, los botones de selección de máscara y el botón de `play`.

```
1       boolean mouseOverBack() {
2               //Boton de ir al menú principal
3           return (parent.mouseX >= 10 && parent.mouseX <= 60) && (parent.
    mouseY >= 10 && parent.mouseY <= 60);
4       }
5
6       boolean mouseOverPlay() {
7           //Boton de jugar
8           return (parent.mouseX >= parent.width / 2 - 100 / 2 && parent.
    mouseX <= parent.width / 2 - 100 / 2 + 100)
9                   && (parent.mouseY >= parent.height - 70 && parent.
    mouseY <= parent.height - 70 + 50);
10      }
11
12      boolean mouseOverMainScreen() {
13          return mouseOverPlay();
14      }
15
16      void mouseOverLeft() {
17          if ((parent.mouseX >= 180 && parent.mouseX <= 200)&&(parent.
    mouseY >= 300 && parent.mouseY <= 360) && maskSelected > 0)
18              maskSelected--;
19      }
20
21      void mouseOverRight() {
22          if ((parent.mouseX >= 440 && parent.mouseX<=460)&&(parent.
    mouseY >= 300 && parent.mouseY <= 360) && maskSelected<mask.size()
    -1)
```

```
            maskSelected++;
    }

    void keyboardTextArea() {
        if (parent.key == 8 && playerName.length()>0) //borrar
            playerName = playerName.substring(0,playerName.length()-1);
        if (playerName.length() >= 20) return; //límite de caracteres
        if (parent.key >= 'A' && parent.key <= 'z') playerName+= parent
    .key;
        if (parent.key == 32) playerName+= " ";
    }
```

Clase `GameScreen`

Se encarga de la creación de la cuenta atrás inicial al empezar a jugar y de la pantalla de juego. Muestra el tiempo restante de la canción, carga y visualiza las imágenes de reloj, de los botones atrás y pausar, también carga las imágenes de las posturas de cierta carpeta y las visualiza. Demuestra la puntuación actual y total, implementa el vuelo de la puntuación actual hacia la total e implementa agrandamiento del campo del puntuación total. Cambia la imágen de posturas aleatoriamente cada 4 segundos y realiza la cuenta atrás. Antes de demostrar la pantalla de juego, demuestra la cuenta atrás inicial.

Las funciones `mouseOverButtonBack()`, mouseOverButtonPause() sirven para saber si los botones están pulsados. `setInitialCount()` anula algunas variables y prepara para la cuenta atrás inicial. `pause()` hace pause o reanuda el juego.

Los métodos `appearEvent()`, `disappearEvent()` y `moveEvent()` son los responsables de recibir los eventos de la cámara *Kinect* y delegar la señal a la clase `Kinect`.

El método `setMask()` envía la imagen seleccionada por el jugador a la clase `Kinect` para que proceda a pintarla sobre la cabeza del avatar virtual.

```
    GameScreen(PApplet parent, HashMap<UISelector, PImage> UIResources,
     String countdownBeep, String startBeep,
              String csvPath, String baseImgDir) {
        super(parent, UIResources, csvPath);

        kinect = new Kinect(this.parent, null, null, null);
        kinect.setHandRadius(0);

        BASE_POS_DIR = baseImgDir;

        createFloor();

        preprocessDancerData();

        currentPosture = new Random().nextInt(ddl.size());

        yStep = (this.parent.height * 0.85f - this.parent.height / 20.f
     - 180) / STEPS;
        xStep = (this.parent.width - this.parent.width / 8.f - this.
    parent.width / 2.f - 110) / STEPS;

        countdown_beep = new SoundFile(this.parent, countdownBeep);
        start_beep = new SoundFile(this.parent, startBeep);
    }

```

```
23      private void preprocessDancerData() {
24          ddl = readDancerDataFromCSV();
25          translateToOrigin();
26      }
27
28      private void translateToOrigin() {
29          for (DancerData dd :
30                  ddl) {
31              Transformation.translateToOrigin(parent, dd, KinectAnathomy
    .SPINE);
32          }
33      }
34
35      public void show() {
36
37          if (initialCount)
38          {
39              if (startCount)
40              {
41                  startCount = false;
42                  time = parent.millis();
43              }
44              if (parent.millis() - time >= 1000)
45              {
46                  time = parent.millis();//also update the stored time
47
48                  if (counter <= 3) while(parent.millis() - time < 1000)
    {}
49
50                  if (counter >= 0)
51                  {
52                      if (counter == 0)
53                          start_beep.play();
54                      else
55                          countdown_beep.play();
56                      parent.imageMode(parent.CENTER);
57                      parent.image(UIResources.get(UISelector.
    getSelectorFromID(counter--)), parent.width / 2.f, parent.height /
    2.f);
58                  }
59                  else
60                  {
61                      initialCount = false;
62                      counter = 3;
63                      this.song.play();
64                  }
65              }
66              return;
67          }
68
69          kinect.doSkeleton(true);
70          kinect.refresh(KinectSelector.NONE, true);
71
72          DancerData liveDancer = new DancerData(parent, kinect);
73          Transformation.translateToOrigin(parent, liveDancer,
    KinectAnathomy.SPINE);
74
75          DancerData ddCSV = ddl.get(currentPosture);
76
77          if (counter == -1)
78          {
79              if (flyingCounter == STEPS)
```

```
80              {
81                  flyingCounter = 0;
82                  biggerCounter = 1;
83                  totalScore += flyingScore;
84              }
85              else if (biggerCounter == 0)
86                  ++flyingCounter;
87              else if (biggerCounter == STEPS_BIGGER)
88              {
89                  biggerCounter = 0;
90                  counter = 3;
91                  time = parent.millis();
92                  currentPosture = new Random().nextInt(ddl.size());
93              }
94              else
95                  ++biggerCounter;
96          }
97          else
98          {
99              parent.imageMode(parent.CENTER);
100             if (counter > 0) {
101                 PImage img = UIResources.get(UISelector.
    getSelectorFromID(counter));
102                 parent.image(img, parent.width / 2.f, parent.height /
    10.f, img.width / 4, img.height / 4);
103             } else if (counter == 0) {
104                 PImage img = UIResources.get(UISelector.NOW);
105                 parent.image(img, parent.width / 2.f, parent.height /
    10.f);
106             }
107             if (!pause && parent.millis() - time - pausedTime >= 1000)
108             {
109                 if (pausedTime > 0)
110                     pausedTime = 0;
111
112                 time = parent.millis();//also update the stored time
113                 --counter;
114             }
115         }
116         // Mostramos el moment score
117         parent.pushMatrix();
118         parent.translate(0,-80,200);
119         parent.ellipseMode(parent.CENTER);
120         parent.pushStyle();
121         parent.stroke(255, 50, 50);
122         parent.strokeWeight(10);
123         parent.fill(0);
124         parent.ellipse(parent.width / 2.f, parent.height * .86f, parent
    .width / 4.8f, parent.height / 10.f);
125         parent.popStyle();
126
127         parent.fill(255, 50, 50);
128         parent.textSize(parent.height / 15.72f);
129         parent.textAlign(parent.CENTER, parent.CENTER);
130
131         if (biggerCounter == 0)
132         {
133             parent.textMode(parent.SHAPE);
134             if (flyingCounter > 0)
135                 parent.text(flyingScore, parent.width / 2.f + xStep * (
    flyingCounter - 1), parent.height * 0.85f - yStep * (flyingCounter
    - 1));
```

```
136             else {
137                 flyingScore = getScore(liveDancer, ddCSV);
138                 parent.text(flyingScore, parent.width / 2f, parent.
    height * 0.85f);
139             }
140         }
141         parent.popMatrix();
142 
143         parent.imageMode(parent.CORNER);
144 
145         // Mostramos los botones "atras" y "pausa"
146         if (mouseOverButtonBack())
147         {
148             if (parent.mousePressed)
149                 parent.image(UIResources.get(UISelector.BACK_PRESSED),
    parent.width / 30, parent.height / 16);
150             else
151                 parent.image(UIResources.get(UISelector.BACK_OVER),
    parent.width / 30, parent.height / 16);
152         }
153         else
154             parent.image(UIResources.get(UISelector.BACK), parent.width
     / 30, parent.height / 16);
155 
156         if (mouseOverButtonPause())
157         {
158             if (parent.mousePressed)
159                 parent.image(UIResources.get(UISelector.PAUSE_PRESSED),
     parent.width / 10, parent.height / 16);
160             else
161                 parent.image(UIResources.get(UISelector.PAUSE_OVER),
    parent.width / 10, parent.height / 16);
162         }
163         else
164             parent.image(UIResources.get(UISelector.PAUSE), parent.
    width / 10, parent.height / 16);
165 
166 
167         // Mostramos el tiempo restante de la cancion
168         parent.pushMatrix();
169         parent.pushStyle();
170         parent.translate(130,-100,160);
171         parent.imageMode(parent.CENTER);
172         parent.image(UIResources.get(UISelector.CHRONO), parent.width /
     30, parent.height - parent.height / 13 + 5, 20,20);
173         parent.textMode(parent.SHAPE);
174         parent.fill(0);
175         parent.textSize(parent.height / 15.72f - 15);
176         parent.textAlign(parent.LEFT, parent.DOWN);
177         parent.text(song.timeLeft(), parent.width / 16.f, parent.height
     - parent.height / 20.f, 0); // Aqui cambiar scr a la variable de
    MainScreen
178         parent.popStyle();
179         parent.popMatrix();
180 
181         parent.pushStyle();
182         parent.fill(255);
183         parent.rectMode(parent.CENTER);
184         parent.stroke(0, 255, 90);
185         parent.rect(90,parent.height / 10 * 9 - 15,140,40, 1);
186         parent.popStyle();
187 
```

```
188         // Mostramos el score total
189         parent.fill(0);
190         parent.rectMode(parent.CENTER);
191         parent.stroke(0, 255, 90);
192         if (biggerCounter == 0)
193             parent.rect(parent.width - parent.width / 8.f, parent.
    height / 17.5f, parent.width / 5.f, parent.height / 15.f);
194         else
195             parent.rect(parent.width - parent.width / 8.f, parent.
    height / 17.5f, parent.width / 5 * 1.2f, parent.height / 15.f * 1.2
    f);
196 
197         parent.fill(0, 255, 90);
198         parent.textAlign(parent.CENTER, parent.CENTER);
199 
200         parent.textMode(parent.SHAPE);
201         if (biggerCounter == 0)
202             parent.textSize(parent.height / 15.72f);
203         else
204             parent.textSize(parent.height / 15.72f * 1.2f);
205         parent.text(totalScore, parent.width - parent.width / 8f,
    parent.height / 20f);
206 
207 
208         // Mostramos la postura y su rama
209         parent.imageMode(parent.CORNER);
210         PImage img = parent.loadImage(BASE_POS_DIR + ddl.get(
    currentPosture).getDancerUUID() + ".png");
211         img.resize(parent.width / 4, 0);
212         parent.image(img, parent.width - img.width, parent.height - img
    .height);
213         parent.stroke(255, 100, 255);
214         parent.line(parent.width - img.width, parent.height, parent.
    width - img.width, parent.height - img.height);
215         parent.line(parent.width - img.width, parent.height - img.
    height, parent.width, parent.height - img.height);
216 
217         makeFloor();
218     }
219 
220     private DancerData getDancerDataByUUID(String uuid) {
221         for (DancerData dd :
222                 ddl) {
223             if (dd.getDancerUUID().equals(uuid)) return dd;
224         }
225         return null;
226     }
227 
228     private int getScore(DancerData ddK, DancerData ddCSV) {
229         double err = Statistics.euclideanMSE(parent, ddK, ddCSV);
230         if (err > 2000 || ddCSV.getAnathomyData().size() == 0 || ddK.
    getAnathomyData().size() == 0) return 0;
231         return (int) Math.abs(err - THRESHOLD);
232     }
233 
234     public Boolean mouseOverButtonBack()
235     {
236         return !initialCount && parent.mouseX >= parent.width / 30 &&
    parent.mouseX <= parent.width / 30 + UIResources.get(UISelector.
    BACK).width
237                 && parent.mouseY >= parent.height / 16 && parent.mouseY
     <= parent.height / 16 + UIResources.get(UISelector.BACK).height;
```

```
238     }
239     public Boolean mouseOverButtonPause()
240     {
241         return !initialCount && parent.mouseX >= parent.width / 10 &&
        parent.mouseX <= parent.width / 10 + UIResources.get(UISelector.
        PAUSE).width
242                 && parent.mouseY >= parent.height / 16 && parent.mouseY
         <= parent.height / 16 + UIResources.get(UISelector.PAUSE).height;
243     }
244 
245     private void makeFloor() {
246         parent.pushMatrix();
247         parent.translate(-ROWS * SCALE / 2.f, 400, -COLS * SCALE /2.f);
248         parent.shape(floor);
249         parent.popMatrix();
250     }
251 
252     private void createFloor() {
253         parent.stroke(255);
254         parent.noFill();
255 
256         floor = parent.createShape();
257         for (int z = 0; z < COLS; z++) {
258             floor.beginShape(parent.QUAD_STRIP);
259             for (int x = 0; x < ROWS; x++) {
260                 floor.vertex(x * SCALE, 0, z * SCALE);
261                 floor.vertex(x * SCALE, 0, (z+1) * SCALE);
262             }
263             floor.endShape();
264         }
265     }
266 
267     void setSong(Song song) {
268         this.song = song;
269     }
270 
271     Song getSong() {
272         return song;
273     }
274 
275     int getScore() {
276         return totalScore;
277     }
278 
279     void setInitialCount()
280     {
281         biggerCounter = 0;
282         flyingCounter = 0;
283         initialCount = true;
284         startCount = true;
285         counter = 3;
286         totalScore = 0;
287         pause = false;
288         pausedTime = 0;
289     }
290 
291     void pause()
292     {
293         pause = !pause;
294         if (pause)
295         {
296             pausedTime = parent.millis();
```

```
297             song.pause(); // Aqui cambiar scr a la variable de
    MainScreen
298         }
299         else
300         {
301             pausedTime = parent.millis() - pausedTime;
302             song.play(); // Aqui cambiar scr a la variable de
    MainScreen
303         }
304     }
305
306     void appearEvent(SkeletonData _s) {
307         kinect.appearEvent(_s);
308     }
309
310     void disappearEvent(SkeletonData _s) {
311         kinect.disappearEvent(_s);
312     }
313
314     void moveEvent(SkeletonData _b, SkeletonData _a) {
315         kinect.moveEvent(_b, _a);
316     }
317
318     void setMask(PImage mask) {
319         kinect.setMask(mask);
320     }
```

Clase `RankingScreen`

Se encarga de la creación de la pantalla de ranking final, donde se lee un CVS que guarda la información de las puntuaciones. Para ello usamos la clase Table propia de processing donde permite ordenar el CSV llamando al método sort. Una vez ordenadas las puntuaciones las mostramos en pantalla hasta un máximo de 10 y añadimos medallas a las tres mejores. El método que se encarga de guardar la puntuación nueva en el CSV también se encuentra en esta clase, el cual es llamado una vez que se acaba la canción.

```
1     RankingScreen(PApplet parent, HashMap<UISelector, PImage>
      uiResources, String rankingCSV) {
2         super(parent, uiResources);
3         this.parent = parent;
4
5         RANKING_CSV = rankingCSV;
6
7         ranking = this.parent.loadTable(rankingCSV,"header");
8
9         title = UIResources.get(UISelector.TITLE);
10        title.resize(0, 120);
11    }
12
13    public void show() {
14        parent.textMode(parent.SHAPE);
15        parent.rectMode(parent.CORNER);
16        parent.textMode(parent.CORNER);
17        parent.textAlign(parent.CORNER, parent.CORNER);
18        screenRanking();
19    }
20
21    void loadScore(String name, int score){
22        TableRow res = ranking.findRow(name, "Name");
```

```
23          if (res == null) {
24              TableRow newRow = ranking.addRow();
25              newRow.setString("Name", name);
26              newRow.setInt("Score", score);
27          } else if (res != null && res.getInt("Score") < score) {
28              res.setString("Name", name);
29              res.setInt("Score", score);
30          }
31          parent.saveTable(ranking, RANKING_CSV);
32      }
33
34      void screenRanking(){
35          ranking.setColumnType(1, "int");
36
37          if (isFirstRun) {
38              ranking.sortReverse("Score");
39              isFirstRun = false;
40          }
41
42          parent.fill(255);
43          parent.textSize(26);
44          parent.text("Ranking: ", 100, parent.height/5-30 );
45          parent.textSize(18);
46          int Yvalue = parent.height/4;
47          parent.text("Nombre ", 100,Yvalue-10 );
48          parent.text("Puntuación ", 300, Yvalue-10 );
49          showRankingImg(Yvalue);
50          int max10 = 0; //Solo mostramos un max de 10
51          for (TableRow row : ranking.rows()) {
52              Yvalue +=25;
53              parent.text(row.getString("Name"), 100, Yvalue );
54              parent.text(row.getString("Score"), 300, Yvalue );
55              if(++max10 >= 10) break;
56          }
57
58          //Boton de Continuar
59          parent.fill(85, 154, 232);
60          parent.rect(parent.width/2-100/2, parent.height-70, 100, 50,7);
     //rectangulo para el texto
61          parent.fill(255);
62          parent.text("Continuar", parent.width/2 - 40, parent.height-38
    );
63      }
64
65      void showRankingImg(int Yvalue){
66          parent.image(UIResources.get(UISelector.CROWN), 240, parent.
    height/5-66);
67          parent.image(UIResources.get(UISelector.GOLD_MEDAL), 360,
    Yvalue+10);
68          parent.image(UIResources.get(UISelector.SILVER_MEDAL), 360,
    Yvalue+35);
69          parent.image(UIResources.get(UISelector.BRONZE_MEDAL), 360,
    Yvalue+60);
70          parent.image(title, 420, 30);
71      }
72
73      void reset() {
74          isFirstRun = true;
75      }
76
77      boolean mouseOverContinue() {
78          return (parent.mouseX >= parent.width / 2 - 100 / 2 && parent.
```

```
    mouseX <= parent.width / 2 - 100 / 2 + 100)
79              && (parent.mouseY >= parent.height - 70 && parent.
    mouseY <= parent.height - 70 + 50);
80  }
```

Clase `UISelector`

Esta clase enumerada es utilizada como selector de iconos y botones que serán utilizados por todas las pantallas del juego.

```
1  public enum UISelector {
2      NOW(-1),
3      START(0),
4      ONE(1),
5      TWO(2),
6      THREE(3),
7      CHRONO(4),
8      BACK(5),
9      BACK_OVER(6),
10     BACK_PRESSED(7),
11     PAUSE(8),
12     PAUSE_OVER(9),
13     PAUSE_PRESSED(10),
14     TITLE(11),
15     CROWN(12),
16     GOLD_MEDAL(13),
17     SILVER_MEDAL(14),
18     BRONZE_MEDAL(15);
19
20     private int id;
21
22     UISelector(int id) {
23         this.id = id;
24     }
25
26     public int getId() {
27         return id;
28     }
29
30     public static UISelector getSelectorFromID(int id) {
31         for (UISelector uiSelector :
32                 UISelector.values()) {
33             if (uiSelector.getId() == id) return uiSelector;
34         }
35         return null;
36     }
37 }
```

3.2.2. Paquete `control`

Clase `kinect.Kinect`

Esta clase es la interfaz que permite interactuar con la *Kinect v1.8*. El constructor acepta los siguientes parámetros:

parent Es una referencia a un objeto de la clase principal `KinectMe`.

pos Establece al posición inicial en el espacio tridimensional de la imágenes obtenidas desde la *Kinect v1.8*.

scale Establece la escala de la imagen.

skeletonRGB Se trata de un vector que ajusta el color del esqueleto que se muestra por pantalla.

```
1   public Kinect(PApplet parent, PVector pos, Float scale, Float[]
    skeletonRGB) {
2       this.parent = parent;
3       kinect = new kinect4WinSDK.Kinect(this.parent);
4       bodies = new ArrayList<SkeletonData>();
5       this.pos = pos;
6       this.scale = scale;
7       this.skeletonRGB = skeletonRGB;
8
9       if (this.pos == null) this.pos = new PVector(0,0,0);
10      if (this.scale == null) this.scale = 1.f;
11      if (this.skeletonRGB == null) this.skeletonRGB = new Float
    []{255.f, 255.f, .0f};
12
13      skelPositions = new HashMap<>();
14      doSkeleton = false;
15  }
```

El método `refresh()` permite refrescar las imágenes que se muestran por pantalla así como visualizar el esqueleto detectado por la *Kinect*. El primer parámetro de este método es un selector de la clase `KinectSelector` permite elegir entre las diferentes opciones de visualización. El segundo parámetro, es una variable booleana que permite visualizar por pantalla las imágenes obtenidas de la *Kinect* cuando se asigna a `true`.

```
1   public void refresh(KinectSelector selector, boolean onScreen) {
2       switch (selector) {
3           case RGB:
4               img = kinect.GetImage();
5
6               if (onScreen) {
7                   parent.image(img,
8                           pos.x, pos.y,
9                           640 * scale, 480 * scale);
10              }
11              break;
12          case DEPTH:
13              img = kinect.GetDepth();
14
15              if (onScreen) {
16                  parent.image(img,
17                          pos.x, pos.y,
18                          640 * scale, 480 * scale);
19              }
20              break;
21          case MASK:
22              img = kinect.GetMask();
23
24              if (onScreen) {
25                  parent.image(img,
26                          pos.x, pos.y,
27                          640 * scale, 480 * scale);
28              }
29              break;
30          default:
31              break;
```

```
32          }
33
34          // Calibrates skeleton to Cam
35          if (KinectSelector.RGB.equals(selector)) {
36              parent.pushMatrix();
37              parent.translate(xOffset, yOffset);
38          }
39
40          if(doSkeleton) bodyTracking();
41
42          if (KinectSelector.RGB.equals(selector)) parent.popMatrix();
43      }
```

El método `bodyTracking()` muestra por pantalla todos los esqueletos detectados por la *Kinect* con cada llamada al método `drawSkeleton()`. Este último, sondea todos los puntos de articulación que conforman el esqueleto detectado e imprime por pantalla cada una de las partes que conforman el esqueleto.

```
1       private void bodyTracking() {
2           for (int i = 0; i < bodies.size(); i++) {
3               drawSkeleton(bodies.get(i));
4           }
5       }
6
7       private void drawSkeleton(SkeletonData _s) {
8           collectPoints(_s);
9
10          // Body
11          drawBody();
12
13          // Left Arm
14          drawLeftArm();
15
16          // Right Arm
17          drawRightArm();
18
19          // Left Leg
20          drawLeftLeg();
21
22          // Right Leg
23          drawRightLeg();
24
25          drawPosition(_s);
26      }
```

Los puntos son sondeados con la llamada al método `collectPoints()`.

```
1       private void collectPoints(SkeletonData _s) {
2           PImage depthImg = kinect.GetDepth();
3           for (KinectAnathomy ka :
4                   KinectAnathomy.values()) {
5               skelPositions.put(ka, ka.getJointPos(_s, depthImg, parent.
        width, parent.height, xOffset, yOffset, skelPositions.get(ka)));
6           }
7       }
```

Cada una de las partes que componen el esqueleto son construidas llamando al método correspondiente. Cada método realiza una llamada a `DrawBone()` con dos selectores de puntos de articulación (`KinectAnathomy`) como argumentos.

```
1       private void drawRightLeg() {
2           DrawBone(KinectAnathomy.HIP_RIGHT,
```

```
                KinectAnathomy.KNEE_RIGHT);
        DrawBone(KinectAnathomy.KNEE_RIGHT,
                KinectAnathomy.ANKLE_RIGHT);
        DrawBone(KinectAnathomy.ANKLE_RIGHT,
                KinectAnathomy.FOOT_RIGHT);
    }

    private void drawLeftLeg() {
        DrawBone(KinectAnathomy.HIP_LEFT,
                KinectAnathomy.KNEE_LEFT);
        DrawBone(KinectAnathomy.KNEE_LEFT,
                KinectAnathomy.ANKLE_LEFT);
        DrawBone(KinectAnathomy.ANKLE_LEFT,
                KinectAnathomy.FOOT_LEFT);
    }

    private void drawRightArm() {
        DrawBone(KinectAnathomy.SHOULDER_RIGHT,
                KinectAnathomy.ELBOW_RIGHT);
        DrawBone(KinectAnathomy.ELBOW_RIGHT,
                KinectAnathomy.WRIST_RIGHT);
        DrawBone(KinectAnathomy.WRIST_RIGHT,
                KinectAnathomy.HAND_RIGHT);
    }

    private void drawLeftArm() {
        DrawBone(KinectAnathomy.SHOULDER_LEFT,
                KinectAnathomy.ELBOW_LEFT);
        DrawBone(KinectAnathomy.ELBOW_LEFT,
                KinectAnathomy.WRIST_LEFT);
        DrawBone(KinectAnathomy.WRIST_LEFT,
                KinectAnathomy.HAND_LEFT);
    }

    private void drawBody() {
        DrawBone(KinectAnathomy.HEAD,
                KinectAnathomy.SHOULDER_CENTER);
        DrawBone(KinectAnathomy.SHOULDER_CENTER,
                KinectAnathomy.SHOULDER_LEFT);
        DrawBone(KinectAnathomy.SHOULDER_CENTER,
                KinectAnathomy.SHOULDER_RIGHT);
        DrawBone(KinectAnathomy.SHOULDER_CENTER,
                KinectAnathomy.SPINE);
        DrawBone(KinectAnathomy.SHOULDER_LEFT,
                KinectAnathomy.SPINE);
        DrawBone(KinectAnathomy.SHOULDER_RIGHT,
                KinectAnathomy.SPINE);
        DrawBone(KinectAnathomy.SPINE,
                KinectAnathomy.HIP_CENTER);
        DrawBone(KinectAnathomy.HIP_CENTER,
                KinectAnathomy.HIP_LEFT);
        DrawBone(KinectAnathomy.HIP_CENTER,
                KinectAnathomy.HIP_RIGHT);
        DrawBone(KinectAnathomy.HIP_LEFT,
                KinectAnathomy.HIP_RIGHT);
    }
```

`DrawBone()` toma dos selectores de articulación y pinta una línea entre esos dos puntos originando, de este modo, una articulación.

En el caso de pintar el punto de articulación de las manos, se comprueba si el radio es mayor que 0. Si es así, se pinta una esfera del radio que se ha

especificado.

En este método se carga la máscara del avatar virtual localizando el punto de la cabeza.

```
    private void DrawBone(KinectAnathomy _j1, KinectAnathomy _j2) {
        parent.pushStyle();
        parent.noFill();
        parent.stroke(skeletonRGB[0], skeletonRGB[1], skeletonRGB[2]);

        int i = 0;
        if (KinectAnathomy.HIP_CENTER.equals(_j1) && KinectAnathomy.
    HIP_LEFT.equals(_j2)) {
            i = 1;
        }


        PVector joint1 = skelPositions.get(_j1);
        PVector joint2 = skelPositions.get(_j2);

        if (KinectAnathomy.HEAD.equals(_j1) && joint1 != null) {
            parent.pushMatrix();
            parent.translate(joint1.x, joint1.y, joint1.z);
            if (mask == null) {
                parent.pushStyle();
                parent.fill(255, 255, 0);
                parent.ellipse(0, 0, 35, 50);
                parent.popStyle();
            } else {
                parent.imageMode(parent.CENTER);
                parent.image(mask,0,0);
            }
            parent.popMatrix();
        }

        if (joint1 != null && joint2 != null) {
            parent.line(joint1.x, joint1.y, joint1.z,
                    joint2.x, joint2.y, joint2.z);

            if ((KinectAnathomy.HAND_LEFT.equals(_j2) || KinectAnathomy
    .HAND_RIGHT.equals(_j2))
                    && handRadius > 0) {
                parent.pushStyle();
                parent.fill(255,0,0,50);
                parent.pushMatrix();
                parent.translate(joint2.x, joint2.y, joint2.z);
                parent.sphereDetail(15);
                parent.sphere(handRadius);
                parent.popMatrix();
                parent.popStyle();
            }
        }

        parent.popStyle();
    }
```

La gestión de los eventos mencionados al final de la sección 3.2.1 y sección 3.2.1 son gestionados en los métodos `appearEvent(), disappearEvent()` y `moveEvent()` de esta clase.

```
    public void appearEvent(SkeletonData _s) {
        if (_s.trackingState == kinect4WinSDK.Kinect.
    NUI_SKELETON_NOT_TRACKED) {
```

```
            return;
        }
        synchronized(bodies) {
            bodies.add(_s);
        }
    }

    public void disappearEvent(SkeletonData skel) {
        synchronized(bodies) {
            for (int i=bodies.size()-1; i>=0; i--) {
                if (skel.dwTrackingID == bodies.get(i).dwTrackingID) {
                    bodies.remove(i);
                }
            }
        }
    }

    public void moveEvent(SkeletonData _b, SkeletonData _a) {
        if (_a.trackingState == kinect4WinSDK.Kinect.
    NUI_SKELETON_NOT_TRACKED) {
            return;
        }
        synchronized(bodies) {
            for (int i=bodies.size()-1; i>=0; i--) {
                if (_b.dwTrackingID == bodies.get(i).dwTrackingID) {
                    bodies.get(i).copy(_a);
                    break;
                }
            }
        }
    }
```

Enumerado `kinect.KinectAnathomy`

Este enumerado identifica los diferentes puntos de articulación detectados por la *Kinect v1.8*.

```
    HEAD("HEAD", Kinect.NUI_SKELETON_POSITION_HEAD),
    SHOULDER_CENTER("SHOULDER_CENTER", Kinect.
    NUI_SKELETON_POSITION_SHOULDER_CENTER),
    SHOULDER_LEFT("SHOULDER_LEFT", Kinect.
    NUI_SKELETON_POSITION_SHOULDER_LEFT),
    SHOULDER_RIGHT("SHOULDER_RIGHT", Kinect.
    NUI_SKELETON_POSITION_SHOULDER_RIGHT),
    SPINE("SPINE", Kinect.NUI_SKELETON_POSITION_SPINE),
    HIP_CENTER("HIP_CENTER", Kinect.NUI_SKELETON_POSITION_HIP_CENTER),
    HIP_LEFT("HIP_LEFT", Kinect.NUI_SKELETON_POSITION_HIP_LEFT),
    HIP_RIGHT("HIP_RIGHT", Kinect.NUI_SKELETON_POSITION_HIP_RIGHT),
    ELBOW_LEFT("ELBOW_LEFT", Kinect.NUI_SKELETON_POSITION_ELBOW_LEFT),
    WRIST_LEFT("WRIST_LEFT", Kinect.NUI_SKELETON_POSITION_WRIST_LEFT),
    HAND_LEFT("HAND_LEFT", Kinect.NUI_SKELETON_POSITION_HAND_LEFT),
    ELBOW_RIGHT("ELBOW_RIGHT", Kinect.NUI_SKELETON_POSITION_ELBOW_RIGHT
    ),
    WRIST_RIGHT("WRIST_RIGHT", Kinect.NUI_SKELETON_POSITION_WRIST_RIGHT
    ),
    HAND_RIGHT("HAND_RIGHT", Kinect.NUI_SKELETON_POSITION_HAND_RIGHT),
    KNEE_LEFT("KNEE_LEFT", Kinect.NUI_SKELETON_POSITION_KNEE_LEFT),
    ANKLE_LEFT("ANKLE_LEFT", Kinect.NUI_SKELETON_POSITION_ANKLE_LEFT),
    FOOT_LEFT("FOOT_LEFT", Kinect.NUI_SKELETON_POSITION_FOOT_LEFT),
    KNEE_RIGHT("KNEE_RIGHT", Kinect.NUI_SKELETON_POSITION_KNEE_RIGHT),
```

```
    ANKLE_RIGHT("ANKLE_RIGHT", Kinect.NUI_SKELETON_POSITION_ANKLE_RIGHT
    ),
    FOOT_RIGHT("FOOT_RIGHT", Kinect.NUI_SKELETON_POSITION_FOOT_RIGHT),
    LABEL("LABEL", Kinect.NUI_SKELETON_POSITION_SHOULDER_CENTER),
    NOT_TRACKED("NOT_TRACKED", Kinect.NUI_SKELETON_POSITION_NOT_TRACKED
    );

    private String id;
    private int skelID;

    KinectAnathomy(String id, int skelID) {
        this.id = id;
        this.skelID = skelID;
    }

    public String getId() {
        return this.id;
    }

    public int getSkelId() {
        return this.skelID;
    }
```

Las coordenadas de cada punto de articulación –incluida la profundidad– son calculadas en el método `getJointPos()`.

```
    public PVector getJointPos(SkeletonData _s, PImage depthImg, float
    width, float height, float xOffset, float yOffset, PVector original
    ) {
        if (_s.skeletonPositionTrackingState[this.skelID] !=
    KinectAnathomy.NOT_TRACKED.skelID) {
            PVector v = new PVector(_s.skeletonPositions[this.skelID].x
     * width + xOffset,
                    _s.skeletonPositions[this.skelID].y * height +
    yOffset,
                    0);
            return getDepth(v, depthImg, width, height, xOffset,
    yOffset, original);
        }
        return null;
    }
```

El cálculo de la profundidad se obtiene a partir de la imagen en blanco y negro que devuelve la librería `Kinect4WinSDK` tras llamar a la primitiva `GetDepth()`.

Para establecer la profundidad de cada punto de articulación se analiza cada uno de los píxeles de la imagen devuelta por `GetDepth()` y se toma el byte menos significativo. Posteriormente, se comprueba que el valor del byte se encuentre en el rango umbral del intervalo $[100, 250]$. El rango de profundidad se establece en el intervalo $[0, 360]$ y se asigna este valor al punto de articulación correspondiente.

```
    private PVector getDepth(PVector joint, PImage depthImg, float w,
    float h, float xOffset, float yOffset, PVector original) {
        PVector j = new PVector(joint.x, joint.y, joint.z);
        j = Transformation.translate(j, -xOffset, -yOffset, 0);

        int x = (int) j.x;
        int y = (int) j.y;

```

```
        int arrayPos = x + (y * depthImg.width) - 1;

        if (arrayPos < 0 || arrayPos >= depthImg.pixels.length) return
    original;

        int depthData = depthImg.pixels[arrayPos];

        float data = depthData & 0xFF;

        if (data >= 100 && data <= 250) {
            float depth = PApplet.map(data, 130, 230, 0, 360);
            joint.z = depth;
            return joint;
        }

        return original;
    }
```

Enumerado `kinect.KinectSelector`

Este enumerado permite seleccionar entre el tipo de imagen que se desea mostrar por pantalla:

- RGB
- DEPTH
- MASK
- NONE

```
public enum KinectSelector {
    RGB,
    DEPTH,
    MASK,
    NONE
}
```

Clase `algorithms.Statistics`

Contiene diversos métodos que servirán de ayuda para calcular los errores de las posturas cuando se estén evaluando. Dentro de esta clase podemos encontrar métodos que calculan la distancia euclídea, media, desviación estándar, obtención de puntos de una postura en un PVector, potencia, suma y normalización de puntos

```
    public static Double euclideanMSE(PApplet parent, DancerData pk,
    DancerData pCSV) {
        Double result = .0;
        Double totalPoints = .0;

        for (KinectAnathomy ka:
                KinectAnathomy.values()) {
            if (!KinectAnathomy.LABEL.equals(ka) && !KinectAnathomy.
    NOT_TRACKED.equals(ka)) {
                PVector kv = pk.getAnathomyVector(ka);
                PVector CSVv = pCSV.getAnathomyVector(ka);
```

```
10                  if (kv != null && CSVv != null) result += Math.pow(
    PVector.dist(kv, CSVv), 2);
11                  totalPoints += 1.;
12              }
13          }
14          return result / totalPoints;
15      }
16
17      public static PVector mean(DancerData dd) {
18          List<PVector> data = extractPoints(dd);
19          return getAvarage(data);
20      }
21
22      public static PVector standardDeviation(DancerData dd) {
23          PVector sd = new PVector();
24          List<PVector> data = extractPoints(dd);
25          PVector avarage = getAvarage(data);
26
27          for (PVector datum :
28                  data) {
29              sd.x = sd.x + (datum.x - avarage.x) * (datum.x - avarage.x)
    ;
30              sd.y = sd.y + (datum.y - avarage.y) * (datum.y - avarage.y)
    ;
31              sd.z = sd.z + (datum.z - avarage.z) * (datum.z - avarage.z)
    ;
32          }
33
34          sd.div(data.size() - 1);
35          sd.x = (float) Math.sqrt(sd.x);
36          sd.y = (float) Math.sqrt(sd.y);
37          sd.z = (float) Math.sqrt(sd.z);
38
39          return sd;
40      }
41
42      private static PVector getAvarage(List<PVector> data) {
43          PVector avarage = new PVector();
44
45          for (PVector v :
46                  data) {
47              avarage.x += v.x;
48              avarage.y += v.y;
49              avarage.z += v.z;
50          }
51
52          return avarage.div(data.size());
53      }
54
55      private static List<PVector> extractPoints(DancerData dd) {
56          List<PVector> data = new ArrayList<>();
57          for (KinectAnathomy ka :
58                  KinectAnathomy.values()) {
59              if (!KinectAnathomy.LABEL.equals(ka) && !KinectAnathomy.
    NOT_TRACKED.equals(ka)) {
60                  data.add(dd.getAnathomyVector(ka));
61              }
62          }
63          return data;
64      }
65
66      public static PVector correlate(DancerData dd1, DancerData dd2) {
```

```
67          List<PVector> x = normalize(dd1);
68          List<PVector> y = normalize(dd2);
69
70          if (x == null || y == null) return new PVector();
71
72          List<PVector> xy = new ArrayList<>();
73
74          for (int i = 0; i < x.size(); i++) {
75              PVector v = new PVector();
76              v.x = x.get(i).x * y.get(i).x;
77              v.y = x.get(i).y * y.get(i).y;
78              v.z = x.get(i).z * y.get(i).z;
79              xy.add(v);
80          }
81
82          List<PVector> xx = square(x);
83          List<PVector> yy = square(y);
84
85          PVector sumx = sum(x);
86          PVector sumy = sum(y);
87          PVector sumxy = sum(xy);
88          PVector sumxx = sum(xx);
89          PVector sumyy = sum(yy);
90
91          PVector sumx2 = new PVector();
92          sumx2.x = sumx.x * sumx.x;
93          sumx2.y = sumx.y * sumx.y;
94          sumx2.z = sumx.z * sumx.z;
95
96          PVector sumy2 = new PVector();
97          sumy2.x = sumy.x * sumy.x;
98          sumy2.y = sumy.y * sumy.y;
99          sumy2.z = sumy.z * sumy.z;
100
101         PVector sumxTimesSumy = new PVector();
102         sumxTimesSumy.x = sumx.x * sumy.x;
103         sumxTimesSumy.y = sumx.y * sumy.y;
104         sumxTimesSumy.z = sumx.z * sumy.z;
105
106         int n = dd1.getAnathomyData().size();
107         Double rx = (n * sumxy.x - sumx.x * sumy.x) / Math.sqrt((n *
    sumxx.x - sumx2.x) * (n * sumyy.x - sumy2.x));
108         Double ry = (n * sumxy.y - sumx.y * sumy.y) / Math.sqrt((n *
    sumxx.y - sumx2.y) * (n * sumyy.y - sumy2.y));
109         Double rz = (n * sumxy.z - sumx.z * sumy.z) / Math.sqrt((n *
    sumxx.z - sumx2.z) * (n * sumyy.z - sumy2.z));
110
111         return new PVector(rx.floatValue(), ry.floatValue(), rz.
    floatValue());
112     }
113
114     private static List<PVector> square(List<PVector> in) {
115         return in.stream().map(v -> {
116             PVector res = new PVector();
117             res.x = v.x*v.x;
118             res.y = v.y*v.y;
119             res.z = v.z*v.z;
120             return res;}).collect(Collectors.toList());
121     }
122
123     private static PVector sum(List<PVector> v) {
124         return v.stream().reduce(new PVector(), (v1, v2) -> v1.add(v2))
```

```
    ;
125 }
126
127 private static List<PVector> normalize(DancerData dd) {
128     List<PVector> norms = new ArrayList<>();
129     for (KinectAnathomy ka:
130             KinectAnathomy.values()) {
131         if (!KinectAnathomy.LABEL.equals(ka) && !KinectAnathomy.
    NOT_TRACKED.equals(ka)) {
132             PVector v1 = new PVector();
133             PVector v2 = dd.getAnathomyVector(ka);
134             if (v2 == null) return null;
135             v1.x = v2.x;
136             v1.y = v2.y;
137             v1.z = v2.z;
138             norms.add(v1.normalize());
139         }
140     }
141     return norms;
142 }
```

Clase `algorithms.Transformation`

Esta clase métodos estáticos necesarios para realizar translaciones y rotaciones ((Wikipedia, s.f.-b), (Wikipedia, s.f.-a)) de cuerpos tridimensionales. Es posible mover todos los puntos de cualquier postura almacenada en objetos `DancerData` con respecto a un punto de referencia colocado en el origen de coordenadas.

```
1  public static DancerData translateToOrigin(PApplet parent,
   DancerData dd, KinectAnathomy kaRef) {
2      if (dd == null) return null;
3
4      DancerData res = new DancerData(parent);
5
6      PVector refPoint = dd.getAnathomyVector(kaRef);
7
8      if (refPoint == null) return dd;
9
10     PVector v1 = new PVector(refPoint.x, refPoint.y, refPoint.z);
11
12     for (KinectAnathomy ka :
13             KinectAnathomy.values()) {
14         if (!KinectAnathomy.LABEL.equals(ka) && !KinectAnathomy.
   NOT_TRACKED.equals(ka)) {
15             PVector v = dd.getAnathomyVector(ka);
16             if (v != null) v.sub(v1);
17         }
18     }
19     return res;
20 }
21
22 public static PVector translate(PVector v, float x, float y, float
   z) {
23     v.x += x;
24     v.y += y;
25     v.z += z;
26     return v;
27 }
28
```

```
29      public static PVector rotateX(PVector v, float theta) {
30          v.x = v.x;
31          v.y = v.y * PApplet.cos(theta) - v.z * PApplet.sin(theta);
32          v.z = v.y * PApplet.sin(theta) + v.z * PApplet.cos(theta);
33          return v;
34      }
35
36      public static PVector rotateY(PVector v, float theta) {
37          v.x = v.x * PApplet.cos(theta) + v.z * PApplet.sin(theta);
38          v.y = v.y;
39          v.z = -v.x * PApplet.sin(theta) + v.z * PApplet.cos(theta);
40          return v;
41      }
42
43      public static PVector rotateZ(PVector v, float theta) {
44          v.x = v.x * PApplet.cos(theta) - v.y * PApplet.sin(theta);
45          v.y = v.x * PApplet.sin(theta) + v.y * PApplet.cos(theta);
46          v.z = v.z;
47          return v;
48      }
49
50      public static PVector superRotation(float x, float y, float z,
        float a, float b, float c, float u, float v, float w, float theta)
        {
51          PVector o = new PVector();
52
53          o.x = (a * (v*v + w*w) - u * (b*v + c*w - u*x - v*y - w*z)) *
        (1 - PApplet.cos(theta)) + x*PApplet.cos(theta) + (-c*v + b*w - w*y
         + v*z) * PApplet.sin(theta);
54          o.y = (b * (u*u + w*w) - v * (a*u + c*w - u*x - v*y - w*z)) *
        (1 - PApplet.cos(theta)) + y*PApplet.cos(theta) + (c*u - a*w + w*x
        - u*z) * PApplet.sin(theta);
55          o.z = (c * (u*u + v*v) - w * (a*u + b*v - u*x - v*y - w*z)) *
        (1 - PApplet.cos(theta)) + z*PApplet.cos(theta) + (-b*u + a*v - v*x
         + u*y) * PApplet.sin(theta);
56
57          return o;
58      }
```

Clase `csv.CSVTools`

Esta clase simplemente contiene dos estáticos métodos que tienen la función tanto de lectura de un fichero CSV como la de escritura.

```
1       public static void writeCSV(Path path, CSVFormat format, String
        uuid, HashMap<KinectAnathomy, PVector> dd) {
2           try {
3               CSVPrinter p = new CSVPrinter(new FileWriter(path.toString
        (), true), format);
4
5               List<Object> points = new ArrayList<>();
6               points.add(uuid);
7               for (KinectAnathomy ka :
8                       KinectAnathomy.values()) {
9                   if (!KinectAnathomy.NOT_TRACKED.equals(ka) && !
        KinectAnathomy.LABEL.equals(ka)) {
10                      PVector v = dd.get(ka);
11                      points.add(v.x);
12                      points.add(v.y);
13                      points.add(v.z);
14                  }
```

```
            }

            p.printRecord(points);
            p.close(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static HashMap<String, List<String>> readCSV(Path path,
    CSVFormat format, String... headers) {
        HashMap<String, List<String>> result = new HashMap<>();
        try {
            Iterable<CSVRecord> records = format.
    withFirstRecordAsHeader().parse(new FileReader(path.toString()));
            for (CSVRecord record :
                    records) {

                List<String> values = new ArrayList<>();

                for (String header :
                        headers) {
                    values.add(record.get(header));
                }
                result.put(record.get(headers[0]), values);
            }
            return result;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }
```

3.2.3. Paquete `model`

Clase `postures.DancerData`

La clase `DancerData` es utilizada como estructura de datos para acceder a los puntos que conforman el esqueleto de las medidas obtenidas con la *Kinect* y/o los modelos almacenados en la biblioteca de posturas de baile modelo.

```
    public DancerData(PApplet parent) {
        this.parent = parent;
        this.anathomyData = new HashMap<>();
        dancerUUID = UUID.randomUUID().toString();
    }

    public DancerData(PApplet parent, String dancerUUID, HashMap<
    KinectAnathomy, PVector> anathomyData) {
        this.parent = parent;
        this.dancerUUID = dancerUUID;
        this.anathomyData = anathomyData;
    }

    public DancerData(PApplet parent, Kinect kinect) {
        this.parent = parent;
        dancerUUID = UUID.randomUUID().toString();
        anathomyData = new HashMap<>();
        for (KinectAnathomy ka :
                KinectAnathomy.values()) {
```

```
19              if (!KinectAnathomy.LABEL.equals(ka) && !KinectAnathomy.
    NOT_TRACKED.equals(ka)) {
20                  PVector skelPos = kinect.getSkelPos(ka);
21                  if (skelPos != null) anathomyData.put(ka, new PVector(
    skelPos.x, skelPos.y, skelPos.z));
22              }
23          }
24      }
25
26      public String getDancerUUID() {
27          return dancerUUID;
28      }
29
30      public HashMap<KinectAnathomy, PVector> getAnathomyData() {
31          return anathomyData;
32      }
33
34      public PVector getAnathomyVector(KinectAnathomy ka) {
35          return anathomyData.get(ka);
36      }
37
38      public void drawDancerData() {
39          if (!anathomyData.isEmpty()) {
40              // Body
41              drawBody();
42
43              // Left Arm
44              drawLeftArm();
45
46              // Right Arm
47              drawRightArm();
48
49              // Left Leg
50              drawLeftLeg();
51
52              // Right Leg
53              drawRightLeg();
54          }
55      }
56
57      private void drawRightLeg() {
58          drawBone(anathomyData.get(KinectAnathomy.HIP_RIGHT),
    anathomyData.get(KinectAnathomy.KNEE_RIGHT));
59          drawBone(anathomyData.get(KinectAnathomy.KNEE_RIGHT),
    anathomyData.get(KinectAnathomy.ANKLE_RIGHT));
60          drawBone(anathomyData.get(KinectAnathomy.ANKLE_RIGHT),
    anathomyData.get(KinectAnathomy.FOOT_RIGHT));
61      }
62
63      private void drawLeftLeg() {
64          drawBone(anathomyData.get(KinectAnathomy.HIP_LEFT),
    anathomyData.get(KinectAnathomy.KNEE_LEFT));
65          drawBone(anathomyData.get(KinectAnathomy.KNEE_LEFT),
    anathomyData.get(KinectAnathomy.ANKLE_LEFT));
66          drawBone(anathomyData.get(KinectAnathomy.ANKLE_LEFT),
    anathomyData.get(KinectAnathomy.FOOT_LEFT));
67      }
68
69      private void drawRightArm() {
70          drawBone(anathomyData.get(KinectAnathomy.SHOULDER_RIGHT),
    anathomyData.get(KinectAnathomy.ELBOW_RIGHT));
71          drawBone(anathomyData.get(KinectAnathomy.ELBOW_RIGHT),
```

```
    anathomyData.get(KinectAnathomy.WRIST_RIGHT));
72      drawBone(anathomyData.get(KinectAnathomy.WRIST_RIGHT),
    anathomyData.get(KinectAnathomy.HAND_RIGHT));
73  }
74
75  private void drawLeftArm() {
76      drawBone(anathomyData.get(KinectAnathomy.SHOULDER_LEFT),
    anathomyData.get(KinectAnathomy.ELBOW_LEFT));
77      drawBone(anathomyData.get(KinectAnathomy.ELBOW_LEFT),
    anathomyData.get(KinectAnathomy.WRIST_LEFT));
78      drawBone(anathomyData.get(KinectAnathomy.WRIST_LEFT),
    anathomyData.get(KinectAnathomy.HAND_LEFT));
79  }
80
81  private void drawBody() {
82      drawBone(anathomyData.get(KinectAnathomy.HEAD), anathomyData.
    get(KinectAnathomy.SHOULDER_CENTER));
83      drawBone(anathomyData.get(KinectAnathomy.SHOULDER_CENTER),
    anathomyData.get(KinectAnathomy.SHOULDER_LEFT));
84      drawBone(anathomyData.get(KinectAnathomy.SHOULDER_CENTER),
    anathomyData.get(KinectAnathomy.SHOULDER_RIGHT));
85      drawBone(anathomyData.get(KinectAnathomy.SHOULDER_CENTER),
    anathomyData.get(KinectAnathomy.SPINE));
86      drawBone(anathomyData.get(KinectAnathomy.SHOULDER_LEFT),
    anathomyData.get(KinectAnathomy.SPINE));
87      drawBone(anathomyData.get(KinectAnathomy.SHOULDER_RIGHT),
    anathomyData.get(KinectAnathomy.SPINE));
88      drawBone(anathomyData.get(KinectAnathomy.SPINE), anathomyData.
    get(KinectAnathomy.HIP_CENTER));
89      drawBone(anathomyData.get(KinectAnathomy.HIP_CENTER),
    anathomyData.get(KinectAnathomy.HIP_LEFT));
90      drawBone(anathomyData.get(KinectAnathomy.HIP_CENTER),
    anathomyData.get(KinectAnathomy.HIP_RIGHT));
91      drawBone(anathomyData.get(KinectAnathomy.HIP_LEFT),
    anathomyData.get(KinectAnathomy.HIP_RIGHT));
92  }
93
94  private void drawBone(PVector v1, PVector v2) {
95      if (v1 != null && v2 != null) {
96          parent.pushStyle();
97          parent.stroke(255, 0, 0);
98          parent.strokeWeight(3.f);
99          parent.line(v1.x, v1.y, v1.z, v2.x, v2.y, v2.z);
100         parent.popStyle();
101     }
102 }
```

Clase `sound.Song`

En la clase `Song` se agrupa todos los métodos y atributos relativos a una canción. Una canción se genera a partir de un fichero de audio. Los métodos implementados permiten que una canción se pueda reproducir (`play()`), parar (`pause()`) y detener (`stop()`). También permite es posible mostrar el tiempo restante para que termine la canción, así como comprobar en cada momento si la canción está siendo reproducida.

Para la implementación de esta clase, se hace uso de la librería `Minim`, utilizando la clase `AudioPlayer` y sus métodos para poder hacer uso de las funciones de la clase `Song`.

```
public class Song {
    AudioPlayer cancion;
    Minim minim;

    public Song (PApplet parent, String ruta) {
        minim = new Minim(parent);
        this.cancion = minim.loadFile(ruta);
    }

    public void play() {
        cancion.play();
    }

    public void pause() {
        cancion.pause();
    }

    public void stop() {
        cancion.pause();
        cancion.rewind();
    }

    public long songDuration() {
        return cancion.length();
    }

    public long songPosition() {
        return cancion.position();
    }

    public String timeLeft() {
        return String.format("%02d:%02d",
                TimeUnit.MILLISECONDS.toMinutes(songDuration() -
    songPosition()),
                TimeUnit.MILLISECONDS.toSeconds(songDuration() -
    songPosition()) -
                        TimeUnit.MINUTES.toSeconds(TimeUnit.
    MILLISECONDS.toMinutes(songDuration() - songPosition()))
        );
    }

    public boolean isPlaying() {
        return cancion.isPlaying();
    }
```

Descripción del trabajo desarrollado por cada integrante

Todos los miembros del equipo participaron activamente en el proyecto. Los métodos de control de la kinect fue desarrollado por David

La obtención de puntos, el algoritmo que se encarga de obtener el error respecto al jugador, además de la corrección de diversos ajustes generales para el correcto funcionamiento y visualización de los elementos, fue desarrollado por David y Christian.

El sonido del juego y la selección de posturas para el funcionamiento del juego se encargó Nestor.

En lo referido a la pantalla inicial y la pantalla de juego principal se encargó de ello Martynas.

Finalmente, Cristina se encargó de las los dibujos del juego, como las figuras de las posturas o el diseño del título, además de encargarse de la pantalla de selección del jugador y la pantalla de ranking final.

Problemas encontrados junto con sus soluciones

Dentro de los problemas encontrados en el desarrollo del trabajo, no destaca ninguno en especial. Sobre todo problemas para encontrar librerías que se ajustaran a los requerimientos que necesitábamos y en general problemas con el modelo que evalúa el error de las posturas. Para éste último, ha sido necesaria la prueba de diversos mecanismos que hallaran un resultado óptimo, finalmente nos decantamos por el que estamos usando pero con cierto remordimiento de no haber podido usar algún método de inteligencia artificial por motivos de tiempo.

Referencias

Chung, B. (s.f.). *Kinect For Processing Library.* http://www.magicandlove.com/blog/research/control.kinect-for-processing-library/. Magic & Love Interactive. (Accessed: 2019-04-06)

DART. (s.f.). *Kinect Controller.* http://dartecne.wikidot.com/control.kinect-controller. DART: Dialogs between Art and Technology. (Accessed: 2019-04-06)

Flaticon. (s.f.). *Flaticon.* https://www.flaticon.com/. Processing. (Accessed: 2019-05-19)

Foundation, P. (s.f.). *Sound.* https://processing.org/reference/libraries/sound/index.html. Autor. (Accessed: 2019-04-06)

JustLagDance. (s.f.-a). *JustLag Dance.* https://github.com/david00medina/CIU-Repositorio/tree/master/TrabajoFinal. JustLag Team. (Accessed: 2019-05-27)

JustLagDance. (s.f.-b). *JustLag Dance.* https://drive.google.com/file/d/1wX15hfbd7weGG5fpa7o9KZRu132n0Rmb/view?usp=sharing. JustLag Team. (Accessed: 2019-05-27)

Microsoft. (s.f.). *Kinect for Windows SDK v1.8.* https://www.microsoft.com/en-us/download/details.aspx?id=40278. Autor. (Accessed: 2019-04-06)

Processing. (s.f.). *Processing: Javadoc.* https://processing.github.io/processing-javadocs/core/index.html. Autor. (Accessed: 2019-02-19)

Wikipedia. (s.f.-a). *Wikipedia: Rotation.* https://en.wikipedia.org/wiki/Rotation. Autor. (Accessed: 2019-02-19)

Wikipedia. (s.f.-b). *Wikipedia: Rotation Matrix.* https://en.wikipedia.org/wiki/Rotation_matrix. Autor. (Accessed: 2019-02-19)