Universidad de Córdoba

Escuela Politécnica Superior

Grado en Ingeniería Informática

Especialidad Computación

Trabajo de Fin de Grado

Colección de videojuegos de destreza multiplataforma

Manual de Código

Autor:
Juan Martos Cáceres

Director:
Manuel Jesús Marín Jiménez

Córdoba - 28 de Junio de 2018

Dr. Manuel Jesús Marín Jiménez profesor del Departamento de Informática y Análisis Numérico de la Universidad de Córdoba.

CERTIFICA

Que el proyecto fin de carrera titulado: *"Colección de videojuegos de destreza multiplataforma"*, ha sido realizado bajo mi dirección por Juan Martos Cáceres, cumpliendo, a mi juicio, con los requisitos exigidos en este tipo de proyectos.

Fdo. **Dr. Manuel Jesús Marín Jiménez**

Córdoba, 28 de Junio de 2018

El alumno Juan Martos Cáceres con D.N.I. 30.999.954-W ha realizado el proyecto presentado en esta memoria junto con la dirección de Manuel Jesús Marín Jiménez.

Fdo. **Juan Martos Cáceres**

Córdoba, 28 de Junio de 2018

Índice general

Índice de figuras

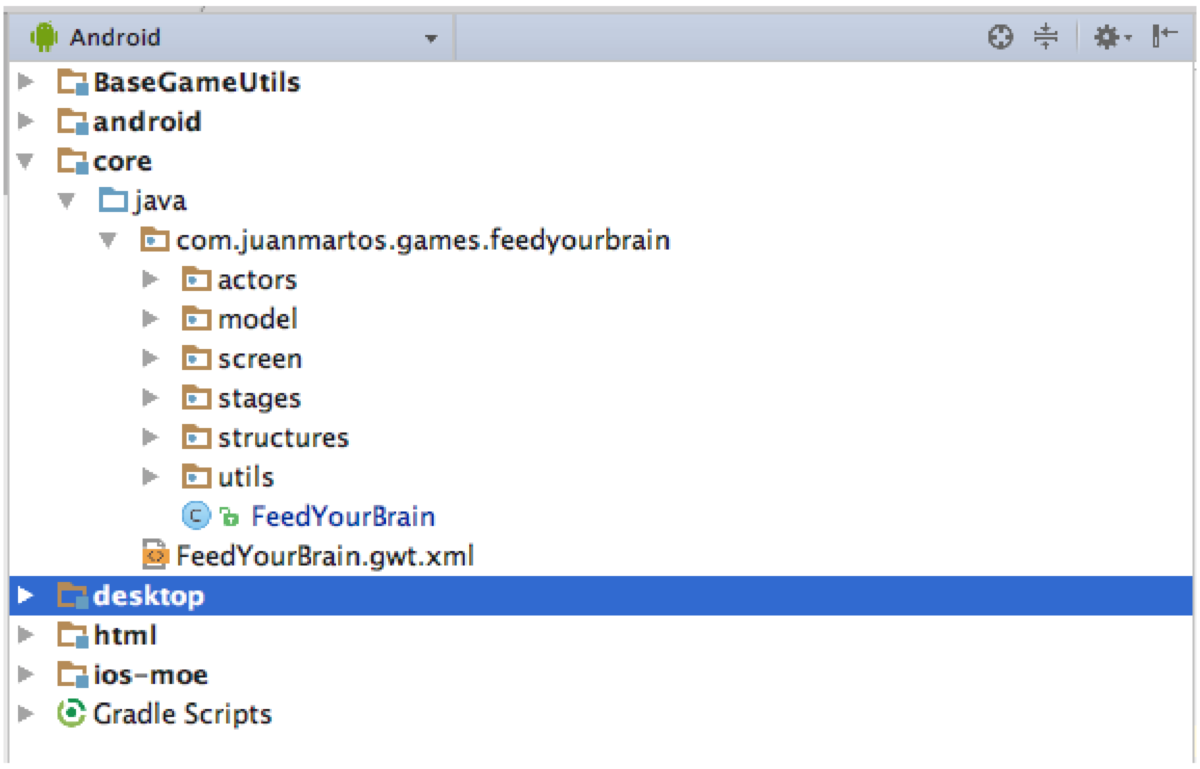
Capítulo 1

Introducción

El presente manual de código se corresponde con el proyecto realizado para la construcción de la aplicación *Colección de videojuegos de destreza multiplataforma*. A lo largo del mismo se describirá la forma en la que se encuentra estructurado y la indicación del código correspondiente a los ficheros. En este manual se van a describir los ficheros de código.

1.1. Organización de los ficheros

Las aplicaciones libGDX organizan sus ficheros en varios proyectos, tenemos un proyecto por cada tipo de aplicación y un proyecto central denominado 'core'. En el entorno de desarrollo de libGDX multiplataforma, instalado en Android Studio, puede observarse la división en la Figura 1.1



```
▶ BaseGameUtils
▶ android
▼ core
    ▼ java
        ▼ com.juanmartos.games.feedyourbrain
            ▶ actors
            ▶ model
            ▶ screen
            ▶ stages
            ▶ structures
            ▶ utils
              FeedYourBrain
          FeedYourBrain.gwt.xml
▶ desktop
▶ html
▶ ios-moe
▶ Gradle Scripts
```

Figura 1.1: Estructura de una aplicación para libGDX

Algunas de las carpetas de de la Figura 1.1 carecen de utilidad de cara al programador, de forma que no han sido empleadas durante el desarrollo de la aplicación y, por lo tanto, no serán explicadas.

1.2. Core

De esta forma, la división empleada en los ficheros por el proyecto Core para libgdx consta de las siguientes carpetas:

- src (Source). En esta carpeta se encuentran todos los ficheros en lenguaje Java que codifican la aplicación.

Capítulo 2

Ficheros Java

En esta sección, se presentan los ficheros con el código Java.

2.1. FeedYourBrain

```java
package com.juanmartos.games.feedyourbrain;

import com.badlogic.gdx.Game;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Net;
import com.badlogic.gdx.assets.AssetManager;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.Texture.TextureFilter;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.juanmartos.games.feedyourbrain.screen.AbstractScreen;
import com.juanmartos.games.feedyourbrain.screen.AssociationScreen;
import com.juanmartos.games.feedyourbrain.screen.EndScreen;
import com.juanmartos.games.feedyourbrain.screen.GameScreen;
import com.juanmartos.games.feedyourbrain.screen.LoadingScreen;
import com.juanmartos.games.feedyourbrain.screen.MainScreen;
import com.juanmartos.games.feedyourbrain.screen.MathScreen;
import com.juanmartos.games.feedyourbrain.screen.MemoryScreen;
import com.juanmartos.games.feedyourbrain.screen.ModeScreen;
import com.juanmartos.games.feedyourbrain.screen.SequenceScreen;
import com.juanmartos.games.feedyourbrain.screen.VisualScreen;
import com.juanmartos.games.feedyourbrain.screen.WeightScreen;
import com.juanmartos.games.feedyourbrain.utils.Api;
import com.juanmartos.games.feedyourbrain.utils.ApiInterface;
import com.juanmartos.games.feedyourbrain.utils.DatabaseFeedYourBrain;
import com.juanmartos.games.feedyourbrain.utils.Log;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;

/**
 * FeedYourBrain extends the class Game and provides methods for managing the views and resources
 */
public class FeedYourBrain extends Game {

    /**
     * Main Screen
     */
    public AbstractScreen mainScreen;

    /**
     * Games Screen
     */
    public AbstractScreen modeScreen;

    /**
     * Help Screen
     */
    public AbstractScreen helpScreen;

    /**
     * Play Screen
     */
    public AbstractScreen playScreen;

    /**
     * End Screen
     */
    public AbstractScreen endScreen;

    /**
     * Level Screen
     */
    public AbstractScreen levelScreen;

    /**
     * Score Select Screen
     */
    public AbstractScreen scoreSelectScreen;

    /**
     * LeaderBoard Screen
     */
    public AbstractScreen leaderBoardScreen;

    /**
     * LeaderBoardGame Screen
     */
    public AbstractScreen leaderBoardGameScreen;

    /**
     * Achievement Screen
     */
    public AbstractScreen achievementScreen;

    /**
     * Credits Screen
     */
    public AbstractScreen creditsScreen;

    /**
     * Association Game Screen
     */
    public GameScreen associationScreen;

    /**
     * Visual Game Screen
     */
    public GameScreen visualScreen;

    /**
     * Math Game Screen
     */
    public GameScreen mathScreen;

    /**
     * Logic Game Screen
     */
    public GameScreen logicScreen;

    /**
     * Memory Game Screen
     */
    public GameScreen memoryScreen;

    /**
     * Sequence Game Screen
     */
    public GameScreen sequenceScreen;

    /**
     * Weight Game Screen
     */
    public GameScreen weightScreen;

    /**
     * Loading Screen
     */
    public LoadingScreen screenLoading;

    /**
     * Resource manager
     */
    private AssetManager manager;

    /**
     * Texture manager draw
     */
    private SpriteBatch batch;

    /**
     * Loading start date
     */
    private long dateStart;

    /**
     * Loading final date
     */
    private long dateEnd;

    /**
     * Global Score
     */
    private int globalScore;

    /**
     * Database Handler
     */
    private DatabaseFeedYourBrain databaseFeedYourBrain;

    /**
     * Sound success
     */
    public static final int SUCCESSSOUND = 0;

    /**
     * Sound error
     */
    public static final int ERRORSOUND = 1;

    public Api api;

    public FeedYourBrain(){

    }

    /**
     * Called when the application is first created.
     */
    @Override
    public void create() {
        this.setDatabaseFeedYourBrain(new DatabaseFeedYourBrain());
```

```
        this.api = new Api();

        this.manager = new AssetManager();
        this.batch = new SpriteBatch();

        this.screenLoading = new LoadingScreen(this);

        this.login();
        //this.actionResolver.login();

        this.loadAssets();

        this.setScreen(this.screenLoading);
    }

    public void login() {
        String udid = MyPreferences.getId();

        if (udid.equals("")) {
            this.createUser();
        } else {
            this.api.login(udid, new ApiInterface() {
                @Override
                public void success(Net.HttpResponse httpResponse) {
                    Log.log("Login: success");
                }

                @Override
                public void failed() {
                    createUser();
                }

                @Override
                public void cancelled() {
                    createUser();
                }
            });
        }
    }

    public void createUser() {
        this.api.createUser(new ApiInterface() {
            @Override
            public void success(Net.HttpResponse httpResponse) {
                Log.log("Create user: success");
            }

            @Override
            public void failed() {
                Log.log("Create user: failed");
            }

            @Override
            public void cancelled() {
                Log.log("Create user: cancelled");
            }
        });
    }

    /**
     * Load all application assets
     */
    public void loadAssets() {

        this.dateStart = new java.util.Date().getTime();

        /**
         * Load textures for loading screen
         */
        this.manager.load("data/loading/background.png", Texture.class);
        this.manager.load("data/loading/bar_down.png", Texture.class);
        this.manager.load("data/loading/bar_up.png", Texture.class);

//      /**
//       * Load textures for all characters
//       */
//      for (int i = 0; i <= 9; ++i)
//          this.manager.load("data/text/" + i + ".png", Texture.class);
//
//      for (int i = 65; i <= 90; ++i) {
//          char c = (char) i;
//          this.manager.load("data/text/Lower_" + c + ".png", Texture.class);
//          this.manager.load("data/text/Upper_" + c + ".png", Texture.class);
//      }
        for (int i = 1; i < 7; ++i) {
            this.manager.load("data/background/bug" + i + ".png", Texture.class);
        }

        /**
         * Load texture for background
         */
        this.manager.load("data/background/background.png", Texture.class);

        /**
         * Load texture for box
         */
        this.manager.load("data/background/box.png", Texture.class);

        /**
         * Load textures for main buttons
         */
        this.manager.load("data/main/button.png", Texture.class);
        this.manager.load("data/main/button_play.png", Texture.class);
        this.manager.load("data/main/button_games.png", Texture.class);
        this.manager.load("data/main/button_score.png", Texture.class);
        this.manager.load("data/main/button_help.png", Texture.class);
        this.manager.load("data/main/button_play_push.png", Texture.class);
        this.manager.load("data/main/button_games_push.png", Texture.class);
        this.manager.load("data/main/button_score_push.png", Texture.class);
        this.manager.load("data/main/button_help_push.png", Texture.class);

        /**
         * Load textures for main buttons english lang
         */
        this.manager.load("data/main/button_play_en.png", Texture.class);
        this.manager.load("data/main/button_games_en.png", Texture.class);
        this.manager.load("data/main/button_score_en.png", Texture.class);
        this.manager.load("data/main/button_help_en.png", Texture.class);
        this.manager.load("data/main/button_play_push_en.png", Texture.class);
        this.manager.load("data/main/button_games_push_en.png", Texture.class);
        this.manager.load("data/main/button_score_push_en.png", Texture.class);
        this.manager.load("data/main/button_help_push_en.png", Texture.class);

        //22

        /**
         * Load textures for exit buttons
         */
        this.manager.load("data/main/button_exit_mini.png", Texture.class);
        this.manager.load("data/main/button_exit_mini_push.png", Texture.class);

        /**
         * Load textures for games buttons
         */
        this.manager.load("data/main/button_math.png", Texture.class);
        this.manager.load("data/main/button_memory.png", Texture.class);
        this.manager.load("data/main/button_visual.png", Texture.class);
        this.manager.load("data/main/button_logic.png", Texture.class);
        this.manager.load("data/main/button_sequence.png", Texture.class);
this.manager.load("data/main/button_weight.png", Texture.class);
this.manager.load("data/main/button_math_push.png", Texture.class);
this.manager.load("data/main/button_memory_push.png", Texture.class);
this.manager.load("data/main/button_visual_push.png", Texture.class);
this.manager.load("data/main/button_logic_push.png", Texture.class);
this.manager.load("data/main/button_sequence_push.png", Texture.class);
this.manager.load("data/main/button_weight_push.png", Texture.class);

/**
 * Load textures for game buttons english lang
 */
//TODO: Add sequence and weight en, now in es
this.manager.load("data/main/button_math_en.png", Texture.class);
this.manager.load("data/main/button_memory_en.png", Texture.class);
this.manager.load("data/main/button_visual_en.png", Texture.class);
this.manager.load("data/main/button_logic_en.png", Texture.class);
this.manager.load("data/main/button_sequence_en.png", Texture.class);
this.manager.load("data/main/button_weight_en.png", Texture.class);
this.manager.load("data/main/button_math_push_en.png", Texture.class);
this.manager.load("data/main/button_memory_push_en.png", Texture.class);
this.manager.load("data/main/button_visual_push_en.png", Texture.class);
this.manager.load("data/main/button_logic_push_en.png", Texture.class);
this.manager.load("data/main/button_sequence_push_en.png", Texture.class);
this.manager.load("data/main/button_weight_push_en.png", Texture.class);

/**
 * Load textures for back buttons
 */
this.manager.load("data/main/button_back_mini.png", Texture.class);
this.manager.load("data/main/button_back_mini_push.png", Texture.class);

//50

/**
 * Load textures for game buttons
 */
this.manager.load("data/main/button_start.png", Texture.class);
this.manager.load("data/main/button_exit.png", Texture.class);
this.manager.load("data/main/button_back.png", Texture.class);
this.manager.load("data/main/button_continue.png", Texture.class);
this.manager.load("data/main/button_restart.png", Texture.class);
this.manager.load("data/main/button_start_push.png", Texture.class);
this.manager.load("data/main/button_exit_push.png", Texture.class);
this.manager.load("data/main/button_back_push.png", Texture.class);
this.manager.load("data/main/button_continue_push.png", Texture.class);
this.manager.load("data/main/button_restart_push.png", Texture.class);

/**
 * Load textures for game buttons english lang
 */
this.manager.load("data/main/button_start_en.png", Texture.class);
this.manager.load("data/main/button_exit_en.png", Texture.class);
this.manager.load("data/main/button_back_en.png", Texture.class);
this.manager.load("data/main/button_continue_en.png", Texture.class);
this.manager.load("data/main/button_restart_en.png", Texture.class);
this.manager.load("data/main/button_start_push_en.png", Texture.class);
this.manager.load("data/main/button_exit_push_en.png", Texture.class);
this.manager.load("data/main/button_back_push_en.png", Texture.class);
this.manager.load("data/main/button_continue_push_en.png",  Texture.class);
this.manager.load("data/main/button_restart_push_en.png", Texture.class);

/**
 * Load texture for help button
 */
this.manager.load("data/main/help.png", Texture.class);

/**
 * Load texture for level button
 */
this.manager.load("data/level/button_easy.png", Texture.class);
this.manager.load("data/level/button_normal.png", Texture.class);
this.manager.load("data/level/button_hard.png", Texture.class);
this.manager.load("data/level/button_easy_push.png", Texture.class);
this.manager.load("data/level/button_normal_push.png", Texture.class);
this.manager.load("data/level/button_hard_push.png", Texture.class);

/**
 * Load textures for level button english lang
 */
this.manager.load("data/level/button_easy_en.png", Texture.class);
this.manager.load("data/level/button_normal_en.png", Texture.class);
this.manager.load("data/level/button_hard_en.png", Texture.class);
this.manager.load("data/level/button_easy_push_en.png", Texture.class);
this.manager.load("data/level/button_normal_push_en.png", Texture.class);
this.manager.load("data/level/button_hard_push_en.png", Texture.class);

/**
 * Load textures for main screen
 */
this.manager.load("data/play/description.png", Texture.class);
this.manager.load("data/play/title.png", Texture.class);

/**
 * Load textures for main screen english lang
 */
this.manager.load("data/play/description_en.png", Texture.class);
this.manager.load("data/play/title_en.png", Texture.class);

/**
 * Load textures for end screen
 */
this.manager.load("data/end/description.png", Texture.class);
this.manager.load("data/end/title.png", Texture.class);
this.manager.load("data/end/your_score.png", Texture.class);
this.manager.load("data/end/max_score.png", Texture.class);

//90

/**
 * Load textures for end screen english lang
 */
this.manager.load("data/end/description_en.png", Texture.class);
this.manager.load("data/end/title_en.png", Texture.class);
this.manager.load("data/end/your_score_en.png", Texture.class);
this.manager.load("data/end/max_score_en.png", Texture.class);

/**
 * Load fonts
 */
this.manager.load("data/font/verdana39.fnt", BitmapFont.class);
this.manager.load("data/font/bitstreamcharter50.fnt", BitmapFont.class);
this.manager.load("data/font/bitstreamcharter60.fnt", BitmapFont.class);
this.manager.load("data/font/bitstreamcharter80.fnt", BitmapFont.class);
this.manager.load("data/font/verdana39.png", Texture.class);
this.manager.load("data/font/bitstreamcharter50.png", Texture.class);
this.manager.load("data/font/bitstreamcharter60.png", Texture.class);
this.manager.load("data/font/bitstreamcharter80.png", Texture.class);

/**
 * Load textures for messages
 */
this.manager.load("data/message/message.png", Texture.class);
this.manager.load("data/message/messagebig.png", Texture.class);

/**
 * Load texture for time
 */
this.manager.load("data/time/time.png", Texture.class);

/**
```

```
 * Load texture for pause
 */
this.manager.load("data/pause/pause.png", Texture.class);

/**
 * Load textures for cards
 */
this.manager.load("data/card/card.png", Texture.class);
this.manager.load("data/card/card_selected.png", Texture.class);
this.manager.load("data/card/card_right.png", Texture.class);

//110

/**
 * Load textures for shapes
 */
String shapes[] = { "pentagon", "star", "circle", "square" };
for (int i = 0; i < shapes.length; ++i) {
    this.manager.load("data/card/" + shapes[i] + "_orange.png",
            Texture.class);
    this.manager.load("data/card/" + shapes[i] + "_yellow.png",
            Texture.class);
    this.manager.load("data/card/" + shapes[i] + "_red.png",
            Texture.class);
    this.manager.load("data/card/" + shapes[i] + "_blue.png",
            Texture.class);
}

/**
 * Load texture for dialog
 */
this.manager.load("data/dialog/dialog.png", Texture.class);

/**
 * Load texture for score
 */
this.manager.load("data/score/score.png", Texture.class);

/**
 * Load textures for math game
 */
this.manager.load("data/calc/c.png", Texture.class);
this.manager.load("data/calc/c_n.png", Texture.class);
this.manager.load("data/calc/c_n_push.png", Texture.class);
this.manager.load("data/calc/a.png", Texture.class);
this.manager.load("data/calc/s.png", Texture.class);
this.manager.load("data/calc/m.png", Texture.class);
this.manager.load("data/calc/d.png", Texture.class);
this.manager.load("data/calc/l.png", Texture.class);
this.manager.load("data/calc/r.png", Texture.class);
this.manager.load("data/calc/e.png", Texture.class);

//138

for (int i = 0; i <= 9; ++i) {
    this.manager.load("data/calc/" + i + ".png", Texture.class);
    this.manager.load("data/calc/" + i + "_n.png", Texture.class);
    this.manager.load("data/calc/" + i + "_n_push.png", Texture.class);
}

/**
 * Load texture and font for visual game
 */
this.manager.load("data/visual/circle.png", Texture.class);
this.manager.load("data/font/bitstreamcharter_visual.fnt",
        BitmapFont.class);

//170

/**
```

```java
 * Load textures for memory and association game
 */
String animals[] = { "card", "bird", "wolf", "cat", "shark", "lion",
        "octopus", "gecko", "egg" };
for (String animal : animals)
    this.manager.load("data/games/memory/cards/" + animal + ".png",
            Texture.class);

/**
 * Load textures for weight game
 *
 */
this.manager.load("data/balances/balance.png", Texture.class);
this.manager.load("data/balances/balance_left.png", Texture.class);
this.manager.load("data/balances/balance_right.png", Texture.class);
for (int i = 1; i <= 6; ++i ){
    this.manager.load("data/balances/weight_" + i + ".png", Texture.class);
}

/**
 * Load textures for games in general
 */
this.manager.load("data/games/memory/title.png", Texture.class);
this.manager.load("data/games/memory/description.png", Texture.class);
this.manager.load("data/games/logic/title.png", Texture.class);
this.manager.load("data/games/logic/description.png", Texture.class);
this.manager.load("data/games/math/title.png", Texture.class);
this.manager.load("data/games/math/description.png", Texture.class);
this.manager.load("data/games/visual/title.png", Texture.class);
this.manager.load("data/games/visual/description.png", Texture.class);
this.manager.load("data/games/sequence/title.png", Texture.class);
this.manager.load("data/games/sequence/description.png", Texture.class);
this.manager.load("data/games/weight/title.png", Texture.class);
this.manager.load("data/games/weight/description.png", Texture.class);

//200

/**
 * Load textures for games in general english lang
 */
//TODO: Add sequence and weight
this.manager.load("data/games/memory/title_en.png", Texture.class);
this.manager.load("data/games/memory/description_en.png", Texture.class);
this.manager.load("data/games/logic/title_en.png", Texture.class);
this.manager.load("data/games/logic/description_en.png", Texture.class);
this.manager.load("data/games/math/title_en.png", Texture.class);
this.manager.load("data/games/math/description_en.png", Texture.class);
this.manager.load("data/games/visual/title_en.png", Texture.class);
this.manager.load("data/games/visual/description_en.png", Texture.class);
this.manager.load("data/games/sequence/title_en.png", Texture.class);
this.manager.load("data/games/sequence/description_en.png", Texture.class);
this.manager.load("data/games/weight/title_en.png", Texture.class);
this.manager.load("data/games/weight/description_en.png", Texture.class);

/**
 * Load textures for max score
 */
this.manager.load("data/games/scoreMax.png", Texture.class);
this.manager.load("data/games/scoreYour.png", Texture.class);

/**
 * Load textures for max score english lang
 */
this.manager.load("data/games/scoreMax_en.png", Texture.class);
this.manager.load("data/games/scoreYour_en.png", Texture.class);

/**
 * Load textures for success and error
 */
this.manager.load("data/games/ok.png", Texture.class);
this.manager.load("data/games/no.png", Texture.class);

/**
 * Load textures for sounds
 */
this.manager.load("data/sounds/sound.png", Texture.class);
this.manager.load("data/sounds/nosound.png", Texture.class);

/**
 * Load textures for score screen
 */
this.manager.load("data/main/button_general.png", Texture.class);
this.manager.load("data/main/button_general_en.png", Texture.class);
this.manager.load("data/main/button_general_push.png", Texture.class);
this.manager.load("data/main/button_general_push_en.png", Texture.class);
for (int i = 0; i <= 9; ++i)
    this.manager.load("data/games/score/" + i + ".png", Texture.class);

//235

/**
 * Load textures for achievements
 */
String games[] = { "math", "memory", "association", "visual", "sequence", "weight"};
String difficulties[] = { "easy", "normal", "hard" };
String leaderboard = "data/leaderboard/";
for (String game : games) {
    for (String difficulty : difficulties) {
        this.manager.load(leaderboard + game + "_" + difficulty
                + ".png", Texture.class);
    }
}
this.manager.load(leaderboard + "play.png", Texture.class);

/**
 * Load textures for streaks
 */
this.manager.load("data/streak/1.png", Texture.class);
this.manager.load("data/streak/2.png", Texture.class);
this.manager.load("data/streak/3.png", Texture.class);
this.manager.load("data/streak/4.png", Texture.class);
this.manager.load("data/streak/5.png", Texture.class);
this.manager.load("data/streak/6.png", Texture.class);
this.manager.load("data/streak/7.png", Texture.class);
this.manager.load("data/streak/8.png", Texture.class);
this.manager.load("data/streak/9.png", Texture.class);

/**
 * Load textures for info
 */
this.manager.load("data/main/button_info.png", Texture.class);
this.manager.load("data/main/button_info_push.png", Texture.class);

/**
 * Load texture for credits
 */
this.manager.load("data/play/credits.png", Texture.class);

/**
 * Load textures for achievements
 */
String achievements = "data/achievements/";
for (String game : games) {
    for (String difficulty : difficulties) {
        this.manager.load(achievements + difficulty + "_" + game + ".png", Texture.class);
        this.manager.load(
                achievements + difficulty + "_" + game + "_achieved.png", Texture.class
        );
    }
}
```

```
    //260
}

/**
 * Load main screen
 */
public void loadScreen() {
    this.dateEnd = new java.util.Date().getTime();
    this.mainScreen = new MainScreen(this);
}

/**
 * Returns the texture selected by the name
 * @param texture
 *            Name of the texture
 * @return the texture selected by the name
 */
public Texture getTexture(String texture) {
    return this.getTexture(texture, false);
}

/**
 * Returns the texture selected by the name and lang
 * @param texture
 *            Name of the texture
 * @param lang
 *            true -> lang active | false -> lang deactivate
 * @return the texture selected by the name and lang
 */
public Texture getTexture(String texture, Boolean lang) {
    Texture t;

    if (lang) {

        if (Utils.langByDefault()) {
            t = this.manager.get("data/" + texture + "_en.png",
                    Texture.class);
            t.setFilter(TextureFilter.Linear, TextureFilter.Linear);
        } else {
            t = this.manager.get("data/" + texture + ".png", Texture.class);
            t.setFilter(TextureFilter.Linear, TextureFilter.Linear);
        }
    } else {
        t = this.manager.get("data/" + texture + ".png", Texture.class);
        t.setFilter(TextureFilter.Linear, TextureFilter.Linear);
    }
    return t;
}

/**
 * Returns the font selected by the name
 * @param font
 *            Name of the font
 * @return the font selected by the name
 */
public BitmapFont getFont(String font) {
    Texture t = this.getTexture("font/" + font);
    BitmapFont bF = new BitmapFont(Gdx.files.internal("data/font/" + font
            + ".fnt"), new TextureRegion(t), false);
    return bF;
}

/**
 * Returns the draw manager
 * @return the draw manager
 */
public SpriteBatch getBatch() {
    return this.batch;
}
```

```
/**
 * Returns the asset manager
 * @return the asset manager
 */
public AssetManager getManager() {
    return this.manager;
}

/** Called when the application is destroyed. Preceded by a call to pause(). */
@Override
public void dispose() {
    super.dispose();
    this.manager.dispose();
    this.batch.dispose();
}

/** Called when the application should render itself. */
@Override
public void render() {
    super.render();
}

/** Called when the application is resized.
 * This can happen at any point during a non-paused state but will never happen
 * before a call to create().
 *
 * @param width the new width in pixels
 * @param height the new height in pixels */
@Override
public void resize(int width, int height) {
    super.resize(width, height);
}

/** Called when the application is paused.
 * An Application is paused before it is destroyed, when a user pressed the Home
 * button on Android or an incoming call happened.
 * On the desktop this will only be called immediately before dispose()
 * is called. */
@Override
public void pause() {
    super.pause();
}

/** Called when the application is resumed from a paused state.
 * On Android this happens when the activity gets focus
 * again. On the desktop this method will never be called. */
@Override
public void resume() {
    super.resume();
}

/**
 * Exit screen
 * @param screen
 *          Screen to exit
 */
public void exitScreen(int screen, int type) {
    if (type == 0) {
        this.main();
    } else {
        if (screen == Screens.MATHSCREEN) {
            this.mathScreen = null;
        }

        this.modeScreen = new ModeScreen(this);
        this.setScreen(this.modeScreen);
    }
}
```

```java
/**
 * Exit screen
 * @param screen
 *          Screen to exit
 * @param type
 *          Mode of game
 */
public void exitScreen(int screen) {
    this.exitScreen(screen, 1);
}

/**
 * Start the mode game
 */
public void play() {
    this.globalScore = 0;
    this.mathScreen = new MathScreen(this, 0);
    this.setScreen(this.mathScreen);
}

/**
 * Function to move from one game to another
 * @param nameChildClass
 *              Name of the class that invokes this function
 * @param scoreNumber
 *              Score obtained
 */
public void nextGame(String nameChildClass, int scoreNumber) {

    this.globalScore += scoreNumber;

    if (nameChildClass.equals("math")) {

        this.mathScreen = null;
        this.logicScreen = new AssociationScreen(this, 0);
        this.setScreen(this.logicScreen);

    } else if (nameChildClass.equals("logic")) {

        this.logicScreen = null;
        this.visualScreen = new VisualScreen(this, 0);
        this.setScreen(this.visualScreen);

    } else if (nameChildClass.equals("visual")) {

        this.visualScreen = null;
        this.memoryScreen = new MemoryScreen(this, 0);
        this.setScreen(this.memoryScreen);

    } else if (nameChildClass.equals("memory")) {

        this.memoryScreen = null;
        this.sequenceScreen = new SequenceScreen(this, 0);
        this.setScreen(this.sequenceScreen);
    } else if (nameChildClass.equals("sequence")) {

        this.sequenceScreen = null;
        this.weightScreen = new WeightScreen(this, 0);
        this.setScreen(this.weightScreen);
    }else if (nameChildClass.equals("weight")) {
        /**
         * In the last game save the score
         */
        //MyPreferences.setGlobalScore(this.globalScore);
        this.databaseFeedYourBrain.insertScore(this.globalScore);
        this.api.createScore(new ApiInterface() {
            @Override
            public void success(Net.HttpResponse response) {

            }
            @Override
            public void failed() {

            }

            @Override
            public void cancelled() {

            }
        }, MyPreferences.getId(), "0", "0", String.valueOf(this.globalScore));


        this.weightScreen = null;
        this.endScreen = new EndScreen(this, this.globalScore);
        this.setScreen(this.endScreen);
    }

}

/**
 * Function to start the application
 */
public void main() {
    this.mainScreen = new MainScreen(this);
    this.setScreen(this.mainScreen);

}

/**
 * Returns the database handler
 * @return the database handler
 */
public DatabaseFeedYourBrain getDatabaseFeedYourBrain() {
    return databaseFeedYourBrain;
}

/**
 * Set the database handler
 * @param databaseFeedYourBrain
 */
public void setDatabaseFeedYourBrain(DatabaseFeedYourBrain databaseFeedYourBrain) {
    this.databaseFeedYourBrain = databaseFeedYourBrain;
}

/**
 * Function that play a sound
 * @param type
 *          Type of sound
 *          type -> 0 SuccessSound
 *          type -> 1 ErrorSound
 */
public void sound(int type) {
    if (MyPreferences.getSound()) {
        Music music = null;
        if (type == FeedYourBrain.SUCCESSSOUND) {
            music = Gdx.audio.newMusic(Gdx.files
                    .internal("data/sounds/success.mp3"));
        } else if (type == FeedYourBrain.ERRORSOUND) {
            Log.log("Error");
            music = Gdx.audio.newMusic(Gdx.files
                    .internal("data/sounds/error.mp3"));
        }
        music.play();
    }
}

/**
 * Static class for represent the games
 * @author juan
```

```
     *
     */
    public static class Screens {
        public static int MATHSCREEN = 1;
        public static int ASSOCIATIONSCREEN = 2;
        public static int MEMORYSCREEN = 3;
        public static int VISUALSCREEN = 4;
        public static int SEQUENCESCREEN = 5;
        public static int WEIGHTSCREEN = 6;
    }
}
```

2.2. Bug

```
package com.juanmartos.games.feedyourbrain.actors.background;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.CircleShape;
import com.badlogic.gdx.physics.box2d.FixtureDef;
import com.badlogic.gdx.physics.box2d.BodyDef.BodyType;
import com.badlogic.gdx.scenes.scene2d.Actor;

/**
 * Bug extends the class Actor and create the view for bugs
 *
 */
public class Bug extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Rigid body
     */
    private Body body;

    /**
     * Body definition holds all the data needed to construct a rigid body
     */
    private BodyDef bodyDef;

    /**
     * Circle shape
     */
    private CircleShape circle;

    /**
     *  A fixture definition is used to create a fixture
     */
    private FixtureDef fixtureDef;

    public Bug(Texture texture) {

        this.texture = texture;

        this.bodyDef = new BodyDef();
        this.bodyDef.type = BodyType.DynamicBody;

        this.circle = new CircleShape();
        int radius = texture.getWidth() / 2;
        this.circle.setRadius(radius);
        this.fixtureDef = new FixtureDef();
        this.fixtureDef.shape = this.circle;
        this.fixtureDef.density = 1.0f;
        this.fixtureDef.friction = 1.0f;
        this.fixtureDef.restitution = 1.0f;

        this.setWidth(this.texture.getWidth());
        this.setHeight(this.texture.getHeight());
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {


        this.body.setAngularDamping(0);
        this.body.setAngularVelocity(0);
        this.setButtonPosition(new Vector2(body.getPosition().x, body
                .getPosition().y));
        batch.draw(this.texture, super.getX(), super.getY());
    }

    public void setButtonPosition(Vector2 position) {
        super.setPosition(position.x - this.getWidth() / 2,
                position.y - this.getHeight() / 2);
    }

    /**
     * Returns the rigid body
     * @return
     */
    public Body getBody() {
        return body;
    }

    /**
     * Set the rigid body
     * @param body
     *          Rigid body
     */
    public void setBody(Body body) {
        this.body = body;
    }

    /**
     * Return the body definition
     * @return
     */
    public BodyDef getBodyDef() {
        return bodyDef;
    }

    /**
     * Set the body definition
     * @param bodyDef
     */
    public void setBodyDef(BodyDef bodyDef) {
        this.bodyDef = bodyDef;
    }
    /**
     * Set the body defintion position
     * @param position
     */
    public void setBodyDefPosition(Vector2 position) {
        this.bodyDef.position.set(position);
        super.setPosition(position.x - (this.getWidth() / 2), position.y
                - (this.getHeight() / 2));
    }

    /**
     * Return the fixture definition
     * @return
     */
    public FixtureDef getFixtureDef() {
        return this.fixtureDef;
    }

    public void setCircleDispose() {
        this.circle.dispose();
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterX() {
        return (int) (super.getX() + this.getWidth() / 2);
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }

}
```

2.3. Card

```
package com.juanmartos.games.feedyourbrain.actors.memory;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;

/**
 * Card extends the class Actor and create the view for cards in memory game
 */
public class Card extends Actor {

    /**
     * Texture head card
     */
    private Texture textureHead;

    /**
     * Texture head tail
     */
    private Texture textureTail;

    /**
     * Type card
     */
    private int type;
```

```
/**
 * Boolean to represent the head
 */
private Boolean head;

/**
 * Boolean to represent the tumble
 */
private Boolean tumble;

/**
 * float to represent the tumbling time
 */
private float TIME = 0.5f;

/**
 * float to control the time of the card
 */
private float time;

/**
 * Constructor parameterized
 * @param textureHead
 *                  Texture head
 * @param textureTail
 *                  Texture tail
 * @param type
 *                  Type card
 */
public Card(Texture textureHead, Texture textureTail, int type) {
    this.textureHead = textureHead;
    this.textureTail = textureTail;

    this.type = type;
    this.head = true;
    this.tumble = false;
    this.time = this.TIME;

    super.setWidth(this.textureHead.getWidth());
    super.setHeight(this.textureHead.getHeight());
    super.setBounds(super.getX(), super.getY(), super.getWidth(),
            super.getHeight());
    super.setTouchable(Touchable.enabled);
}

public Card(Card card) {
    this(card.textureHead, card.textureTail, card.type);
}

/** Draws the actor.
 * The Batch is configured to draw in the parent's coordinate system.
 * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
 * float, float, float, float, float, float, float, float, float)
 * This draw method is convenient to draw a rotated
 * and scaled TextureRegion. begin() has already been called on
 * the Batch. If end() is called to draw without the Batch thenbegin()
 * must be called before the method returns.
 * <p>
 * The default implementation does nothing.
 * @param alpha Should be multiplied with the actor's alpha,
 * allowing a parent's alpha to affect all children. */
@Override
public void draw(Batch batch, float alpha) {

    /**
     *  if tumble
     **/
    if (this.tumble) {
```

```java
            /**
             * width of the image to draw
             */
            float porcentageWidth = (this.time - TIME / 2) * 2;

            /**
             * Check if image it has rotated 90 degrees
             */
            if (this.time <= TIME / 2) {
                porcentageWidth = ((this.TIME - this.time) - TIME / 2) * 2;
            }

            porcentageWidth = porcentageWidth / TIME;

            int offsetX = (int) (super.getWidth() - super.getWidth()
                    * porcentageWidth) / 2;

            if (this.time <= TIME / 2) {
                if (head)
                    batch.draw(this.textureHead, super.getX() + offsetX,
                            super.getY(), super.getWidth() * porcentageWidth,
                            super.getHeight());
                else
                    batch.draw(this.textureTail, super.getX() + offsetX,
                            super.getY(), super.getWidth() * porcentageWidth,
                            super.getHeight());
            } else {
                if (head)
                    batch.draw(this.textureTail, super.getX() + offsetX,
                            super.getY(), super.getWidth() * porcentageWidth,
                            super.getHeight());
                else
                    batch.draw(this.textureHead, super.getX() + offsetX,
                            super.getY(), super.getWidth() * porcentageWidth,
                            super.getHeight());
            }

            return;
        } else {

            if (head)
                batch.draw(this.textureHead, super.getX(), super.getY());
            else
                batch.draw(this.textureTail, super.getX(), super.getY());

        }

    }

    public void tailToHead() {
        this.tumble = true;
        this.time = this.TIME;
        this.head = true;
    }

    /**
     * Update time representing flipping cards
     * If the time we specify zero it will not overturn more
     * @param delta
     *              Time spent in execution
     */
    public void updateTime(float delta) {
        this.time -= delta;

        if (this.time <= 0.0f) {
            this.tumble = false;
        }

    }
    /**
     * Function to check if a card will become
     * @return
     *              True tumble | False no tumble
     */
    public boolean isTumble() {
        return this.tumble;
    }

    /**
     * Set Tumble
     * @param tumble
     */
    public void setTumble(boolean tumble) {

        this.tumble = tumble;

    }

    /**
     * Get Type
     * @return
     *          Type
     */
    public int getType() {
        return this.type;
    }

    /**
     * Set Head
     * @param head
     *          Head
     */
    public void setHead(Boolean head) {
        this.head = head;
    }

}
```

2.4. Description

```
package com.juanmartos.games.feedyourbrain.actors.play;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Description extends the class Actor and create the view for descriptions
 */
public class Description extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor parameterized
     * @param texture
     *              Texture texture
     */
    public Description(Texture texture) {
        this.texture = texture;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());

        float offsetX = (Var.width / 2 - super.getWidth()) / 2;
        float offsetY = (Var.height - super.getHeight()) / 2;

        super.setPosition(offsetX, offsetY);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(texture, super.getX(), super.getY());
    }

}
```

2.5. Description

```
package com.juanmartos.games.feedyourbrain.actors.weight;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.juanmartos.games.feedyourbrain.actors.Weight;

import java.util.ArrayList;

public class Balance extends Actor{

    public static final int EQUAL = 0;
    public static final int LEFT = 1;
    public static final int RIGHT = 2;

    public static final int WIDTH = 500;
    public static final int HEIGHTEQUAL = 273;
    public static final int HEIGHTLEFT = 309;

    public static final int PLATEWIDTH = 164;

    public static final int OFFSET = 79;

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * List of left weights
     */
    private ArrayList<Weight> left;

    /**
     * List of right weights
     */
    private ArrayList<Weight> right;

    private int coordinatesEqual [][][] = {
            {
                    { (PLATEWIDTH - Weight.WIDTH)/2, Balance.HEIGHTEQUAL }
            },
            {
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2, Balance.HEIGHTEQUAL},
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2 + Weight.WIDTH, Balance.HEIGHTEQUAL }
            },
            {
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2, Balance.HEIGHTEQUAL},
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2 + Weight.WIDTH, Balance.HEIGHTEQUAL },
                    { (PLATEWIDTH - Weight.WIDTH)/2, Balance.HEIGHTEQUAL + Weight.HEIGHT }
            }
    };
    private int coordinatesLeft [][][] = {
            {
                    { (PLATEWIDTH - Weight.WIDTH)/2, Balance.HEIGHTLEFT }
            },
            {
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2, Balance.HEIGHTLEFT},
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2 + Weight.WIDTH, Balance.HEIGHTLEFT }
            },
            {
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2, Balance.HEIGHTLEFT},
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2 + Weight.WIDTH, Balance.HEIGHTLEFT},
                    { (PLATEWIDTH - Weight.WIDTH)/2, Balance.HEIGHTLEFT + Weight.HEIGHT }
            }
    };
    private int coordinatesRight [][][] = {
            {
                    { (PLATEWIDTH - Weight.WIDTH)/2, Balance.HEIGHTLEFT }
            },
            {
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2, Balance.HEIGHTLEFT},
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2 + Weight.WIDTH, Balance.HEIGHTLEFT }
            },
            {
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2, Balance.HEIGHTLEFT},
                    { (PLATEWIDTH - (Weight.WIDTH*2))/2 + Weight.WIDTH, Balance.HEIGHTLEFT},
                    { (PLATEWIDTH - Weight.WIDTH)/2, Balance.HEIGHTLEFT + Weight.HEIGHT }
            }
    };

    /**
     * Balance type
     */
    private int type;

    public Balance(Texture texture, int type){

        this.texture = texture;
        this.type = type;

        this.left = new ArrayList<Weight>();
        this.right = new ArrayList<Weight>();
    }

    public void addtoLeft(Weight weight){
        this.left.add(weight);
    }

    public void addtoRight(Weight weight){
        this.right.add(weight);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
```

```
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {

        batch.draw(texture, super.getX(), super.getY());

        int i = 0;

        for (Weight weight:this.left){

            int coordinate[] = this.getCoordinate(this.left.size() - 1, i);

            float x = super.getX() + coordinate[0];
            float y = super.getY() + coordinate[1];

            if(type == Balance.RIGHT){
                y = y - OFFSET;
            }

            batch.draw(weight.getTexture(), x, y);

            ++i;
        }

        i = 0;

        for (Weight weight:this.right){
            int coordinate[] = this.getCoordinate(this.right.size() - 1, i);

            float x = super.getX() + (WIDTH  - Weight.WIDTH - coordinate[0]) ;
            float y = super.getY() + coordinate[1];

            if(type == Balance.LEFT){
                y = y - OFFSET;
            }

            batch.draw(weight.getTexture(),x, y);

            ++i;
        }
    }

    public int[] getCoordinate(int size, int index){

        int coordinate [] = null;

        if(type == Balance.EQUAL){
            return this.coordinatesEqual[size][index];
        }

        if(type == Balance.LEFT){
            return this.coordinatesLeft[size][index];
        }

        if(type == Balance.RIGHT){
            return this.coordinatesRight[size][index];
        }

        return coordinate;

    }
}
```

2.6. Button

```java
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.GlyphLayout;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;

/**
 * Button extends the class Actor and create the view for buttons
 */
public class Button extends Actor {

    /**
     * Render bitmap font
     */
    private BitmapFont font;

    /**
     * Content button
     */
    private String content;

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Type button
     */
    private int type;

    /**
     * Constructor parameterized
     * @param texture
     *              Texture actor
     * @param font
     *              Font actor
     * @param content
     *              Content button
     *
     */
    public Button(Texture texture, BitmapFont font, String content) {
        this.texture = texture;
        this.font = font;
        this.content = content;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);
    }

    /**
     * Constructor parameterized
     * @param texture
     *              Texture actor
     */
    public Button(Texture texture) {
        this.texture = texture;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);
```

```java
}

/** Draws the actor.
 * The Batch is configured to draw in the parent's coordinate system.
 * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
 * float, float, float, float, float, float, float, float, float)
 * This draw method is convenient to draw a rotated
 * and scaled TextureRegion. begin() has already been called on
 * the Batch. If end() is called to draw without the Batch thenbegin()
 * must be called before the method returns.
 * <p>
 * The default implementation does nothing.
 * @param alpha Should be multiplied with the actor's alpha,
 * allowing a parent's alpha to affect all children. */
@Override
public void draw(Batch batch, float alpha) {
    batch.draw(this.texture, super.getX(), super.getY());

    if (this.font != null) {

        GlyphLayout glyphLayout = new GlyphLayout();
        glyphLayout.setText(font, content);

        this.font.setColor(Color.WHITE);
        this.font.draw(batch, content, this.getCenterX() - glyphLayout.width
                / 2, this.getCenterY() + glyphLayout.height / 2);
    }

}

/**
 * Return the center x of actor
 * @return
 */
public int getCenterX() {
    return (int) (super.getX() + this.getWidth() / 2);
}

/**
 * Returns the y coordinate of the center of the actor
 * @return the y coordinate of the center of the actor
 */
public int getCenterY() {
    return (int) (super.getY() + this.getHeight() / 2);
}

/**
 * Set the position of actor
 * @param x
 *          Coordinate x
 * @param y
 *          Coordinate Y
 */
public void setPosition(float x, float y) {
    super.setPosition(x, y);
}

/**
 * Returns the content of button
 * @return the content of button
 */
public String getContent() {
    return this.content;
}

/**
 * Returns the type of button
 * @return the type of button
 */
public int getType() {
        return type;
    }

    /**
     * Set the type of button
     * @param type
     *          Type button
     */
    public void setType(int type) {
        this.type = type;
    }

    /**
     * Set the texture of the actor
     * @param texture
     *              Texture of the actor
     */
    public void setTexture(Texture texture) {
        this.texture = texture;
    }
}
```

2.7. ButtonType

```
package com.juanmartos.games.feedyourbrain.actors;

/**
 * ButtonType provide static variables to represent the types of button.
 * @author juan
 *
 */
public class ButtonType {
    public static final int START = 1;
    public static final int PAUSE = 2;
    public static final int BACK = 3;
    public static final int EXIT = 4;
    public static final int RESTART = 5;
    public static final int RESUME = 6;
    public static final int PLAY = 7;
    public static final int GAMES = 8;
    public static final int SCORE = 9;
    public static final int HELP = 10;
    public static final int MATH = 11;
    public static final int VISUAL = 12;
    public static final int LOGIC = 13;
    public static final int MEMORY = 14;
    public static final int SEQUENCE = 15;
    public static final int WEIGHT = 16;
    public static final int EASY = 17;
    public static final int NORMAL = 18;
    public static final int HARD = 19;
    public static final int GENERAL = 20;
    public static final int INFO = 21;
}
```

2.8. Card

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;

/**
 * Card extends the class Actor and create the view for cards
 */
public class Card extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Number to identify cards
     */
    private int number;

    /**
     * State card
     * No selected -> 0
     * Selected -> 1
     * Correct -> 2
     */
    private int state;

    /**
     * Constructor parameterized
     * @param texture
     *             Texture actor
     * @param position
     *             Positon actor
     * @param number
     *             Number card
     * @param scale
     *             Scale card
     */
    public Card(Texture texture, Vector2 position, int number, float scale) {
        this.state = 0;
        this.texture = texture;
        super.setPosition(position.x, position.y);
        this.number = number;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {


        batch.draw(this.texture, super.getX(), super.getY());
    }

    /**
     * Returns the number of the card
     * @return the number of the card
     */
    public int getNumber() {
        return this.number;
    }

    /**
     * Return the number in format string of the card
     * @return the number in format string of the card
     */
    public String getStringNumber() {
        return String.valueOf(this.number);
    }

    /**
     * Returns the x coordinate of the center of the actor
     * @return the x coordinate of the center of the actor
     */
    public int getCenterX() {
        return (int) (super.getX() + this.getWidth() / 2);
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }

    /**
     * Return the state of the card
     * @return the state of the card
     */
    public int getState() {
        return this.state;
    }

    /**
     * Set the state of the card
     * @param state
     *            State
     */
    public void setState(int state) {
        this.state = state;
    }

    /**
     * Set the texture of the actor
     * @param texture
     *            Texture of the actor
     */
    public void setTexture(Texture texture) {
        this.texture = texture;
    }

}
```

2.9. Character

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;

/**
 * Character extends the class Actor and create the view for characters
 */
public class Character extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Boolean that representing whether the character has done his animation
     */
    private Boolean completed;

    public Character(Texture texture) {
        this.texture = texture;
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }

    /**
     * Return whether the character animation is complete
     * @return whether the character animation is complete
     */
    public Boolean hasCompleted() {
        return completed;
    }

    /**
     * Set whether the animation is complete
     * @param completed
     */
    public void setCompleted(Boolean completed) {
        this.completed = completed;
    }

}
```

2.10. CharacterCalc

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;

/**
 * CharacterCalc extends the class Actor and create the view for characters of calc
 */
public class CharacterCalc extends Actor {

    /**
     * Char to raw
     */
    private char character;

    /**
     * Texture actor
     */
    private Texture texture;

    public CharacterCalc(Texture texture, Vector2 position, char character) {
        this.texture = texture;
        this.character = character;
        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);
        super.setPosition(position.x, position.y);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(texture, super.getX(), super.getY());
    }

    /**
     * Return the character
     * @return the character
     */
    public char getCharacter() {
        return this.character;
    }

    /**
     * Set the texture of the actor
     * @param texture
     *             Texture of the actor
     */
    public void setTexture(Texture texture) {
        this.texture = texture;
    }

    /**
     * Set the position of the actor
     * @param x
     * @param y
     */
    public void setPosition(float x, float y){
        super.setPosition(x, y);
    }
}
```

2.11. Circle

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.GlyphLayout;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.CircleShape;
import com.badlogic.gdx.physics.box2d.FixtureDef;
import com.badlogic.gdx.physics.box2d.BodyDef.BodyType;
import com.badlogic.gdx.scenes.scene2d.Actor;

/**
 * Circle extends the class Actor and create the view for circles
 */
public class Circle extends Actor {

    /**
     * Rigid body
     */
    private Body body;

    /**
     * Body definition holds all the data needed to construct a rigid body
     */
    private BodyDef bodyDef;

    /**
     * Circle shape
     */
    private CircleShape circle;

    /**
     *  A fixture definition is used to create a fixture
     */
    private FixtureDef fixtureDef;

    /**
     * Render bitmap font
     */
    private BitmapFont font;

    /**
     * Number of the circle
     */
    private int number;

    /**
     * Number in string format of the circle
     */
    private String sNumber;

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Consturctor parameterized
     * @param texture
     *             Texture actor
     * @param font
     *             Font actor
     * @param number
     *             Number circle
     */
    public Circle(Texture texture, BitmapFont font, int number) {
        this.texture = texture;
        this.font = font;
        this.number = number;
        this.sNumber = String.valueOf(this.number);

        this.bodyDef = new BodyDef();
        this.bodyDef.type = BodyType.DynamicBody;

        this.circle = new CircleShape();
```

```java
    int radius = texture.getWidth() / 2;
    this.circle.setRadius(radius);

    this.fixtureDef = new FixtureDef();
    this.fixtureDef.shape = this.circle;
    this.fixtureDef.density = 1.0f;
    this.fixtureDef.friction = 1.0f;
    this.fixtureDef.restitution = 1.0f;

    this.setWidth(this.texture.getWidth());
    this.setHeight(this.texture.getHeight());
}

/** Draws the actor.
 * The Batch is configured to draw in the parent's coordinate system.
 * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
 * float, float, float, float, float, float, float, float, float)
 * This draw method is convenient to draw a rotated
 * and scaled TextureRegion. begin() has already been called on
 * the Batch. If end() is called to draw without the Batch thenbegin()
 * must be called before the method returns.
 * <p>
 * The default implementation does nothing.
 * @param alpha Should be multiplied with the actor's alpha,
 * allowing a parent's alpha to affect all children. */
@Override
public void draw(Batch batch, float alpha) {


    this.body.setAngularDamping(0);
    this.body.setAngularVelocity(0);
    this.setButtonPosition(new Vector2(body.getPosition().x, body
            .getPosition().y));
    batch.draw(this.texture, super.getX(), super.getY());

    GlyphLayout glyphLayout = new GlyphLayout();
    glyphLayout.setText(font, this.sNumber);

    this.font.setColor(Color.WHITE);
    this.font.draw(batch, sNumber,
            this.getCenterX() - glyphLayout.width / 2, this.getCenterY()
                    + glyphLayout.height / 2 + 35);

}

/**
 * Set the button position
 * @param position
 */
public void setButtonPosition(Vector2 position) {
    super.setPosition(position.x - this.getWidth() / 2,
            position.y - this.getHeight() / 2);
}

/**
 * Returns the rigid body
 * @return
 */
public Body getBody() {
    return body;
}

/**
 * Set the rigid body
 * @param body
 *            Rigid body
 */
public void setBody(Body body) {
    this.body = body;
}
```

```java
/**
 * Return the body definition
 * @return
 */
public BodyDef getBodyDef() {
    return bodyDef;
}

/**
 * Set the body definition
 * @param bodyDef
 */
public void setBodyDef(BodyDef bodyDef) {
    this.bodyDef = bodyDef;
}

/**
 * Set the body defintion position
 * @param position
 */
public void setBodyDefPosition(Vector2 position) {
    this.bodyDef.position.set(position);
    super.setPosition(position.x - (this.getWidth() / 2), position.y
            - (this.getHeight() / 2));
}

/**
 * Return the fixture definition
 * @return
 */
public FixtureDef getFixtureDef() {
    return this.fixtureDef;
}

/**
 * Return the number of the circle
 * @return the number of the circle
 */
public int getNumber() {
    return number;
}

/**
 * Set the number of the circle
 * @param number
 */
public void setNumber(int number) {
    this.number = number;
}

/**
 * Free memory of body
 */
public void setCircleDispose() {
    this.circle.dispose();
}

/**
 * Returns the x coordinate of the center of the actor
 * @return the x coordinate of the center of the actor
 */
public int getCenterX() {
    return (int) (super.getX() + this.getWidth() / 2);
}

/**
 * Returns the y coordinate of the center of the actor
 * @return the y coordinate of the center of the actor
 */
```

```
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }

}
```

2.12. Help

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Button extends the class Actor and create the view for help button
 */
public class Help extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Type actor
     */
    private int type;

    /**
     * Constructor parameterized
     * @param texture
     *             Texture actor
     */
    public Help(Texture texture) {
        this.texture = texture;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);

        //Calculamos la posicion para la esquina dderecha
        float offsetX = (Var.width - super.getWidth())/2;
        float offsetY = (Var.height - super.getHeight())/2;
        super.setPosition(offsetX, offsetY);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }

    /**
```

```
     * Returns the x coordinate of the center of the actor
     * @return the x coordinate of the center of the actor
     */
    public int getCenterX() {
        return (int) (super.getX() + this.getWidth() / 2);
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }

    /**
     * Return the type actor
     * @return the type actor
     */
    public int getType() {
        return type;
    }

    /**
     * Set the type actor
     * @param type
     *          Type actor
     */
    public void setType(int type) {
        this.type = type;
    }
}
```

2.13. LeaderBoardButton

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;

/**
 * LeaderBoardButton extends the class Actor and create the view for leaderboard button
 */
public class LeaderBoardButton extends Button {

    /**
     * Code that represents the leaderboard that are drawing
     */
    private String code;

    /**
     *
     */
    private String game;

    /**
     *
     */
    private String difficulty;

    /**
     * Constructor parameterized
     * @param texture
     *              Texture actor
     */
    public LeaderBoardButton(Texture texture) {
        super(texture);
    }

    /**
```

```
     * Return the code of the leaderboard
     * @return the code of the leaderboard
     */
    public String getCode() {
        return code;
    }

    /**
     * Set the code of the leaderboard
     * @param code
     *          Code of the leaderboard
     */
    public void setCode(String code) {
        this.code = code;
    }

    public String getGame() {
        return game;
    }

    public void setGame(String game) {
        this.game = game;
    }

    public String getDifficulty() {
        return difficulty;
    }

    public void setDifficulty(String difficulty) {
        this.difficulty = difficulty;
    }
}
```

2.14. Level

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Level extends the class Actor and create the view for level indicator
 */
public class Level extends Actor {

    /**
     * General handler of the application
     */
    private FeedYourBrain feedYourBrain;

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Array of strings for reading textures
     */
    private String textures[];

    /**
     * Constructor parameterized
     * @param feedYourBrain
     *              General handler of the application
     */
    public Level(FeedYourBrain feedYourBrain) {
        this.feedYourBrain = feedYourBrain;

        /**
         * Set the textures
         */
        String textures[] = { "easy", "normal", "hard" };

        this.textures = textures;

        this.configTexture();

        /**
         * Set the dimensions
         */
        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        float x = (Var.width - this.getWidth()) / 2;
        float y = 50;
        super.setBounds(x, y, super.getWidth(), super.getHeight());
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }

    /**
     * Config the texture of the level
     */
    private void configTexture() {
        int index = MyPreferences.getDifficulty();
        this.texture = this.feedYourBrain.getTexture("level/button_"
                + textures[index - 1], true);
    }

}
```

2.15. Message

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.GlyphLayout;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Message extends the class Actor and create the view for messages
 */
public class Message extends Actor {

    /**
     * Render bitmap font
     */
    private BitmapFont font;

    /**
     * String to save the message to represent
     */
    private String message;

    /**
     * float to represent the time the message is displayed
     */
    private float time;

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor parameterized
     * @param texture
     *             Texture actor
     * @param font
     *             Font actor
     *
     */
    public Message(Texture texture, BitmapFont font) {
        this.texture = texture;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);

        float offsetX = (Var.width - super.getWidth()) / 2;
        float offsetY = (Var.height - super.getHeight()) / 2;
        super.setPosition(offsetX, offsetY);
        this.font = font;
    }

    /**
     * Set the content and the time of the message
     * @param message
     *             Message to set
     * @param time
     *             Time to set
     */
    public void setContent(String message, float time) {
        this.message = message;
        this.time = time;
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {


        if (this.time > 0.0f) {
            batch.draw(this.texture, super.getX(), super.getY());

            GlyphLayout glyphLayout = new GlyphLayout();
            glyphLayout.setText(font, this.message);

            this.font.draw(batch, message, this.getCenterX() - glyphLayout.width
                    / 2, this.getCenterY() + glyphLayout.height / 2);
        }
    }

    /**
     * Return the time of the message
     * @return the time of the message
     */
    public float getTime() {
        return this.time;
    }

    /**
     * Set the time of the message
     * @param time
     *           Time of the message
     */
    public void setTime(float time) {
        this.time = this.time - time;
    }

    /**
     * Returns the x coordinate of the center of the actor
     * @return the x coordinate of the center of the actor
     */
    public int getCenterX() {
        return (int) (super.getX() + this.getWidth() / 2);
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }

}
```

2.16. Message

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.GlyphLayout;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Message extends the class Actor and create the view for messages
 */
public class MessageLog extends Actor {

    /**
     * Render bitmap font
     */
    private BitmapFont font;

    /**
     * String to save the message to represent
 */
private String message;

/**
 * float to represent the time the message is displayed
 */
private float time;

/**
 * Texture actor
 */
private Texture texture;

/**
 * Constructor parameterized
 * @param texture
 *              Texture actor
 * @param font
 *              Font actor
 *
 */
public MessageLog(Texture texture, BitmapFont font) {
    this.texture = texture;

    super.setWidth(this.texture.getWidth());
    super.setHeight(this.texture.getHeight());
    super.setBounds(super.getX(), super.getY(), super.getWidth(),
            super.getHeight());
    super.setTouchable(Touchable.enabled);

    float offsetX = (Var.width - super.getWidth()) / 2;
    float offsetY = (Var.height - super.getHeight()) / 2;
    super.setPosition(offsetX, offsetY);
    this.font = font;
}

/**
 * Set the content and the time of the message
 * @param message
 *              Message to set
 * @param time
 *              Time to set
 */
public void setContent(String message, float time) {
    this.message = message;
    this.time = time;
}

/** Draws the actor.
 * The Batch is configured to draw in the parent's coordinate system.
 * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
 * float, float, float, float, float, float, float, float, float)
 * This draw method is convenient to draw a rotated
 * and scaled TextureRegion. begin() has already been called on
 * the Batch. If end() is called to draw without the Batch thenbegin()
 * must be called before the method returns.
 * <p>
 * The default implementation does nothing.
 * @param alpha Should be multiplied with the actor's alpha,
 * allowing a parent's alpha to affect all children. */
@Override
public void draw(Batch batch, float alpha) {
    batch.draw(this.texture, super.getX(), super.getY());
    GlyphLayout glyphLayout = new GlyphLayout();
    glyphLayout.setText(font, this.message);

    this.font.draw(
            batch,
            message,
            this.getCenterX() - glyphLayout.width / 2,
                this.getCenterY() + glyphLayout.height / 2
        );
    }

    /**
     * Return the time of the message
     * @return the time of the message
     */
    public float getTime() {
        return this.time;
    }

    /**
     * Set the time of the message
     * @param time
     *          Time of the message
     */
    public void setTime(float time) {
        this.time = this.time - time;
    }

    /**
     * Returns the x coordinate of the center of the actor
     * @return the x coordinate of the center of the actor
     */
    public int getCenterX() {
        return (int) (super.getX() + this.getWidth() / 2);
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }

    /**
     * Set position
     * @param x
     *          Coordinate x
     * @param y
     *          Coordinate y
     */
    public void setPosition(float x, float y) {
        super.setPosition((Var.width - super.getWidth()) / 2, y);
    }

}
```

2.17. No

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * No extends the class Actor and create the view for error
 */
public class No extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;
```

```
/**
 * Time it takes to float away the actor
 */
private float time;

/**
 * Constructor parameterized
 * @param texture
 *              Texture actor
 */
public No(Texture texture) {
    this.texture = texture;

    super.setWidth(this.texture.getWidth());
    super.setHeight(this.texture.getHeight());
    super.setBounds(super.getX(), super.getY(), super.getWidth(),
            super.getHeight());
    super.setTouchable(Touchable.enabled);

    /**
     * Calculate the position of left corner
     */
    float offsetX = Var.width - this.getWidth() - this.getWidth() / 2;
    float offsetY = 0 + this.getHeight() / 2;
    super.setPosition(offsetX, offsetY);
    this.time = 0.0f;
}

/** Draws the actor.
 * The Batch is configured to draw in the parent's coordinate system.
 * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
 * float, float, float, float, float, float, float, float, float)
 * This draw method is convenient to draw a rotated
 * and scaled TextureRegion. begin() has already been called on
 * the Batch. If end() is called to draw without the Batch thenbegin()
 * must be called before the method returns.
 * <p>
 * The default implementation does nothing.
 * @param alpha Should be multiplied with the actor's alpha,
 * allowing a parent's alpha to affect all children. */
@Override
public void draw(Batch batch, float alpha) {
    alpha = (float) Math.round(this.time * 10) / 10;

    if (this.time >= 0.0f) {
        super.draw(batch, alpha);
        batch.setColor(this.getColor().r, this.getColor().g,
                this.getColor().b, alpha);
        batch.draw(texture, super.getX(), super.getY());
        super.draw(batch, 1);
        batch.setColor(this.getColor().r, this.getColor().g,
                this.getColor().b, 1);

    }
}

/** Updates the actor based on time. Typically this is called each frame by act(float).
 * <p>
 * The default implementation calls act(float) on each action and removes actions that are complete.
 * @param delta Time in seconds since the last frame. */
@Override
public void act(float delta) {
    this.time -= delta;
}

/**
 * Reload the time of the actor
 */
public void reload() {
        this.time = 1.0f;
    }
}
```

2.18. Ok

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;

/**
 * Ok extends the class Actor and create the view for success
 */
public class Ok extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /** Tiempo que tarda en desaparacer el actor **/
    private float time;

    /**
     * Constructor parameterized
     * @param texture
     *             Texture actor
     *
     */
    public Ok(Texture texture) {
        this.texture = texture;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);

        //Calculamos la posicion para la esquina inferior izquierda
        float offsetX = 0 + this.getWidth() / 2;
        float offsetY = 0 + this.getHeight() / 2;
        super.setPosition(offsetX, offsetY);
        this.time = 0.0f;
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {


        alpha = (float) Math.round(this.time * 10) / 10;

        if (this.time >= 0.0f) {
            super.draw(batch, alpha);
            batch.setColor(this.getColor().r, this.getColor().g,
                    this.getColor().b, alpha);
            batch.draw(texture, super.getX(), super.getY());
            super.draw(batch, 1);
            batch.setColor(this.getColor().r, this.getColor().g,
                    this.getColor().b, 1);

        }
    }

    /** Updates the actor based on time.
     * Typically this is called each frame by act(float).
     * <p>
     * The default implementation calls act(float) on each action
     * and removes actions that are complete.
     * @param delta Time in seconds since the last frame. */
    @Override
    public void act(float delta) {
        this.time -= delta;
    }

    /**
     * Reload the time of the actor
     */
    public void reload() {
        this.time = 1.0f;
    }

}
```

2.19. Pause

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Pause extends the class Actor and create the view for pause
 */
public class Pause extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Type of the actor
     */
    private int type;

    /**
     * Constructor parameterized
     * @param texture
     *            Texture actor
     */
    public Pause(Texture texture) {
        this.texture = texture;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);

        //Calculamos la posicion para la esquina dderecha
        float offsetX = Var.width - super.getWidth() - super.getWidth() / 10;
        float offsetY = Var.height - super.getHeight() - super.getHeight() / 10;
        super.setPosition(offsetX, offsetY);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }

    /**
     * Returns the x coordinate of the center of the actor
     * @return the x coordinate of the center of the actor
     */
    public int getCenterX() {
        return (int) (super.getX() + this.getWidth() / 2);
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }

    /**
     * Return the type actor
     * @return the type actor
     */
    public int getType() {
        return type;
    }

    /**
     * Set the type actor
     * @param type
     *          Type actor
     */
    public void setType(int type) {
        this.type = type;
    }
}
```

2.20. Score

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.GlyphLayout;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;
```

```
/**
 * Score extends the class Actor and create the view for score
 */
public class Score extends Actor {

    /**
     * Render bitmap font
     */
    private BitmapFont font;

    /**
     * Content of the score
     */
    private String content;

    /**
     * Number of score
     */
    public int score;

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Level number
     */
    private int level;

    /**
     * Name of the class that instance this class
     */
    private int className;

    /**
     * Number to control the streak
     */
    private int streak;

    /**
     * Constructor parameterized
     * @param texture
     *             Texture actor
     * @param font
     *             Font actor
     *
     */
    public Score(Texture texture, BitmapFont font) {
        this.texture = texture;
        this.font = font;
        this.score = 0;
        this.content = String.valueOf(this.score);

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);

        /**
         * Calculate the top left corner
         */
        float offsetX = (Var.width - this.getWidth()) / 2;
        float offsetY = Var.height - super.getHeight() - super.getHeight() / 6;
        super.setPosition(offsetX, offsetY);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
```

```java
 * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
 * float, float, float, float, float, float, float, float, float)
 * This draw method is convenient to draw a rotated
 * and scaled TextureRegion. begin() has already been called on
 * the Batch. If end() is called to draw without the Batch thenbegin()
 * must be called before the method returns.
 * <p>
 * The default implementation does nothing.
 * @param alpha Should be multiplied with the actor's alpha,
 * allowing a parent's alpha to affect all children. */
@Override
public void draw(Batch batch, float alpha) {
    batch.draw(this.texture, super.getX(), super.getY());

    GlyphLayout glyphLayout = new GlyphLayout();
    glyphLayout.setText(font, content);

    this.font.draw(batch, content,
            this.getCenterX() - glyphLayout.width / 2, this.getCenterY()
                    + glyphLayout.height / 2);

}

/**
 * Function to increase the score
 */
public void increase() {
    /** Fix **/
    if (this.streak == 0) {
        this.streak = 1;
    }
    this.score += (Utils.getScore(this.className, this.level) * this.streak);
    this.content = String.valueOf(score);
}

/**
 * Function to clear the score
 */
public void clear() {
    this.content = "0";
    this.score = 0;
}

/**
 * Returns the x coordinate of the center of the actor
 * @return the x coordinate of the center of the actor
 */
public int getCenterX() {
    return (int) (super.getX() + this.getWidth() / 2);
}

/**
 * Returns the y coordinate of the center of the actor
 * @return the y coordinate of the center of the actor
 */
public int getCenterY() {
    return (int) (super.getY() + this.getHeight() / 2);
}

/**
 * Return the score
 * @return the level
 */
public int getScore() {
    return this.score;
}

/**
 * Return the level
 * @return the level
```

```
     */
    public int getLevel() {
        return level;
    }

    /**
     * Set level
     * @param level
     *          Level
     */
    public void setLevel(int level) {
        this.level = level;
    }

    /**
     * Return the name of the class
     * @return
     */
    public int getClassName() {
        return className;
    }

    /**
     * Set the name of the class
     * @param className
     */
    public void setClassName(int className) {
        this.className = className;
    }

    /**
     * Set the streak
     * @param streak
     *          Streak
     */
    public void setStreak(int streak) {
        this.streak = streak;
    }
}
```

2.21. Shape

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;

/**
 * Shape extends the class Actor and create the view for shapes
 */
public class Shape extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor parameterized
     * @param texture
     *              Texture Actor
     * @param position
     *              Position Actor
     */
    public Shape(Texture texture, Vector2 position) {
        this.texture = texture;
        super.setPosition(position.x, position.y);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }

}
```

2.22. Sound

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.InputListener;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Sound extends the class Actor and create the view for sound
 */
public class Sound extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * General handler of the application
     */
    private FeedYourBrain feedyourbrain;

    /**
     * Constructor parameterized
     * @param feedYourBrain
     *             General handler of the application
     */
    public Sound(FeedYourBrain feedyourbrain) {
        this.feedyourbrain = feedyourbrain;
        this.configTexture();

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);

        /**
         * Calculate the top right corner
         */
        float offsetX = Var.width - super.getWidth() - super.getWidth() / 10;
        float offsetY = super.getHeight() / 2;
        super.setPosition(offsetX, offsetY);

        super.addListener(new InputListener() {
            public boolean touchDown(InputEvent event, float x, float y,
                    int pointer, int button) {
                return true;
            }

            public void touchUp(InputEvent event, float x, float y,
                    int pointer, int button) {
                MyPreferences.changeSound();
            }
        });
    }

    private void configTexture() {

        if (MyPreferences.getSound()) {
            this.texture = feedyourbrain.getTexture("sounds/sound");
        } else {
            this.texture = feedyourbrain.getTexture("sounds/nosound");
        }

    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        this.configTexture();
        batch.draw(this.texture, super.getX(), super.getY());
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterX() {
        return (int) (super.getX() + this.getWidth() / 2);
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }
}
```

2.23. Streak

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
```

```java
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.InputListener;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Streak extends the class Actor and create the view for streak
 */
public class Streak extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * General handler of the application
     */
    private FeedYourBrain feedyourbrain;

    /**
     * Number to control the streak
     */
    private int streak;

    /**
     * Constructor parameterized
     * @param feedYourBrain
     *             General handler of the application
     */
    public Streak(FeedYourBrain feedyourbrain) {
        this.feedyourbrain = feedyourbrain;

        this.streak = 1;

        this.configTexture();

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);

        /**
         * Calculate the top right corner
         */
        float offsetX = Var.width - super.getWidth() - super.getWidth() / 10;
        float offsetY = Var.height / 2 + super.getHeight() / 2;
        super.setPosition(offsetX, offsetY);

        super.addListener(new InputListener() {
            public boolean touchDown(InputEvent event, float x, float y,
                    int pointer, int button) {
                return true;
            }

            public void touchUp(InputEvent event, float x, float y,
                    int pointer, int button) {
                MyPreferences.changeSound();
            }
        });
    }

    /**
     * Config the texture of the actor
     */
    private void configTexture() {
        this.texture = feedyourbrain.getTexture("streak/" + this.streak);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {


        this.configTexture();
        batch.draw(this.texture, super.getX(), super.getY());
    }

    /**
     * Returns the x coordinate of the center of the actor
     * @return the x coordinate of the center of the actor
     */
    public int getCenterX() {
        return (int) (super.getX() + this.getWidth() / 2);
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }

    /**
     * Update the streak
     * @param update
     */
    public void update(int update) {

        int streakAux = (int) Math.ceil(update / 5) + 1;

        if (streakAux >= 9) {
            this.streak = 9;
        } else {
            this.streak = streakAux;
        }
    }

    /**
     * Return the streak
     * @return the streak
     */
    public int getStreak() {
        return this.streak;
    }

    /**
     * Set streak
     * @param streak
     */
    public void setStreak(int streak) { this.streak = streak; }
}
```

2.24. Time

```java
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.GlyphLayout;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Time extends the class Actor and create the view for time
 */
public class Time extends Actor {

    /**
     * Render bitmap font
     */
    private BitmapFont font;

    /**
     * String to save the content of the time
     */
    private String content;

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor parameterized
     * @param texture
     *             Texture actor
     * @param font
     *             Font actor
     *
     */
    public Time(Texture texture, BitmapFont font) {
        this.texture = texture;
        this.font = font;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(),
                super.getHeight());
        super.setTouchable(Touchable.enabled);

        //Calculamos la posicion para la esquina izquierda
        float offsetX = 0.0f + this.getWidth() / 10;
        float offsetY = Var.height - super.getHeight() - this.getHeight() / 10;
        super.setPosition(offsetX, offsetY);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());

        GlyphLayout glyphLayout = new GlyphLayout();
        glyphLayout.setText(font, content);

        this.font.draw(batch, content,
                this.getCenterX() - glyphLayout.width / 2, this.getCenterY()
                        + glyphLayout.height / 2);

    }

    /**
     * Returns the x coordinate of the center of the actor
     * @return the x coordinate of the center of the actor
     */
    public int getCenterX() {
        return (int) (super.getX() + this.getWidth() / 2);
    }

    /**
     * Returns the y coordinate of the center of the actor
     * @return the y coordinate of the center of the actor
     */
    public int getCenterY() {
        return (int) (super.getY() + this.getHeight() / 2);
    }

    /**
     * Set the content of the time in seconds
     * @param seconds
     */
    public void setContent(float seconds) {
        this.content = String.valueOf((int) seconds);
    }

}
```

2.25. Time

```
package com.juanmartos.games.feedyourbrain.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Touchable;

public class Weight extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    private String letter;

    private int amount;

    public static final int WIDTH = 147;
    public static final int HEIGHT = 147;

    public Weight(Texture texture, int amount, String letter){
        this.texture = texture;
        this.amount = amount;
        this.letter = letter;

        super.setWidth(this.texture.getWidth());
        super.setHeight(this.texture.getHeight());
        super.setBounds(super.getX(), super.getY(), super.getWidth(), super.getHeight());
        super.setTouchable(Touchable.enabled);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(texture, super.getX(), super.getY());
    }

    public Texture getTexture() {
        return texture;
    }

    public void setTexture(Texture texture) {
        this.texture = texture;
    }

    public String getLetter() {
        return letter;
    }
}
```

2.26. AbstractScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import java.util.Random;

import com.badlogic.gdx.Application;
import com.badlogic.gdx.Game;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.Box2DDebugRenderer;
import com.badlogic.gdx.physics.box2d.World;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.background.Bug;
import com.juanmartos.games.feedyourbrain.utils.CreateWorld;
import com.juanmartos.games.feedyourbrain.utils.Log;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * AbstractScreen implements the interface Screen and create the background for play and mode screens
 */
public class AbstractScreen implements Screen {

    /**
     * General handler of the application
     */
    protected FeedYourBrain feedyourbrain;

    /**
     * Handler textures
     */
    protected SpriteBatch batch;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Variable to represent the world where we draw the objects to apply physical rules
     */
    private World world;

    /**
     * Virtual world for test
     */
    private Box2DDebugRenderer debugRenderer;

    /**
     * Camara representing where we see the virtual world
     */
    private OrthographicCamera cam;

    /**
     * Variable to represent test mode
     */
    private boolean debugMode;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                  General handler of the application
     */
    public AbstractScreen(FeedYourBrain feedyourbrain) {
        this.feedyourbrain = feedyourbrain;
        this.batch = feedyourbrain.getBatch();

        this.stage = new Stage(new FitViewport(Var.width, Var.height));

        this.loadInputProcessor();

        this.createBugs();
    }

    /**
     * Function to load the input processor
     */
    private void loadInputProcessor() {
        InputMultiplexer multiplexer = new InputMultiplexer();
        multiplexer.addProcessor(this.stage);

        Gdx.input.setInputProcessor(multiplexer);
    }

    /**
     * Function to generate animation in background
     */
    public void createBugs() {

        world = CreateWorld.createWorld();

        int nBugs = 10;
        int nDifferentBugs = 6;

        Random random = new Random();

        for (int i = 0; i < nBugs; ++i) {

            int nBug = random.nextInt(nDifferentBugs) + 1;
            Texture texture = this.feedyourbrain.getTexture("background/bug"
                    + nBug);

            Bug bug = new Bug(texture);

            Vector2 position = new Vector2(random.nextInt(Var.width),
                    random.nextInt(Var.height));

            bug.setBodyDefPosition(position);

            Body body = world.createBody(bug.getBodyDef());
            body.createFixture(bug.getFixtureDef());
            bug.setCircleDispose();

            int x = 100;
            int y = 100;

            if (random.nextInt(2) == 1)
                x = x * (-1);

            if (random.nextInt(2) == 1)
                y = y * (-1);

            body.setLinearVelocity(x, y);
            body.setUserData(bug);
            bug.setBody(body);

            this.stage.addActor(bug);

        }
    }


    /**
     * Called when this screen should release all resources.
     */
    @Override
    public void dispose() {
        // TODO Auto-generated method stub

    }

    /**
     * Called when this screen is no longer the current screen for a Game.
     */
    @Override
    public void hide() {
        // TODO Auto-generated method stub

    }

    /** Called when the application is paused.
     * An Application is paused before it is destroyed, when a user pressed the Home
     * button on Android or an incoming call happened.
     * On the desktop this will only be called immediately before dispose()
     * is called. */
    @Override
    public void pause() {
        // TODO Auto-generated method stub

    }

    /**
     * Called when the screen should render itself.
     * @param delta The time in seconds since the last render.
     */
    @Override
    public void render(float arg0) {
        this.stage.draw();

        this.world.step(1 / 60f, 6, 2);

        if (this.debugMode) {
            this.debugRenderer.render(this.world, this.cam.combined);
        }

    }

    /** Called when the {@link Application} is resized.
     * This can happen at any point during a non-paused state but will never happen
     * before a call to {@link #create()}.
     *
     * @param width the new width in pixels
     * @param height the new height in pixels */
    @Override
    public void resize(int arg0, int arg1) {
        // TODO Auto-generated method stub

    }

    /** Called when the {@link Application} is resumed from a paused state.
     * On Android this happens when the activity gets focus
     * again. On the desktop this method will never be called. */
    @Override
    public void resume() {
        stage = new Stage(new FitViewport(Var.width, Var.height));
        // TODO Auto-generated method stub

    }

    /** Called when this screen becomes the current screen for a {@link Game}. */
    @Override
    public void show() {
        // TODO Auto-generated method stub
    }

}
```

2.27. AbstractScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.Net;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.Array;
import com.badlogic.gdx.utils.JsonReader;
import com.badlogic.gdx.utils.JsonValue;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.LeaderBoardButton;
import com.juanmartos.games.feedyourbrain.model.Achievement;
import com.juanmartos.games.feedyourbrain.utils.ApiInterface;
import com.juanmartos.games.feedyourbrain.utils.Log;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;
```

```
/**
 * AchievementScreen extends the class AbstractScreen and create the view for achievement screen
 */
public class AchievementScreen extends AbstractScreen {

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Back button
     */
    private Button backButton;

    /**
     * Achievements
     */
    private Array<Achievement> achievements;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                  General handler of the application
     */
    public AchievementScreen(FeedYourBrain feedyourbrain) {
        super(feedyourbrain);

        this.achievements = new Array<Achievement>();

        this.stage = new Stage(new FitViewport(Var.width, Var.height));

        //Capturamos la pulsacion en el boton atras
        Gdx.input.setCatchBackKey(true);

        this.loadInputProcessor();

        this.getAchievements();
    }

    /**
     * Function to load the input processor
     */
    private void loadInputProcessor() {
        InputMultiplexer multiplexer = new InputMultiplexer();
        multiplexer.addProcessor(this.stage);

        Gdx.input.setInputProcessor(multiplexer);

    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
                Var.blueBackground, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        super.render(delta);
        this.stage.draw();
    }

    private void getAchievements()
    {
        this.feedyourbrain.api.getAchievements(new ApiInterface() {
            @Override
            public void success(Net.HttpResponse httpResponse) {

                try {
```

```
                String response = httpResponse.getResultAsString();

                JsonValue achievementsJson = new JsonReader().parse(response);

                for (JsonValue achievementJson : achievementsJson.iterator())
                {
                    Achievement achievement = new Achievement();

                    achievement.udid = achievementJson.getString("udid");

                    achievements.add(achievement);
                }

            } catch (Exception e) {
                Log.log(e.getLocalizedMessage());
            }

            Gdx.app.postRunnable(new Runnable() {
                @Override
                public void run() {
                    loadView();
                }
            });
        }

        @Override
        public void failed() {

        }

        @Override
        public void cancelled() {

        }
    }, MyPreferences.getId());
}

/**
 * Function to load views
 */
private void loadView() {

    /**
     * Width and height
     */
    int width = Var.width;
    int height = Var.height;

    /**
     * Set the width and height of the button
     */
    float buttonWidth = 200;
    float buttonHeight = 200;

    float buttonOffset = 50.0f;

    int nButtonsX = 6;
    int nButtonsY = 3;

    /**
     * Get the offset x
     */
    float offsetX = (width - buttonWidth * nButtonsX - buttonOffset
            * nButtonsX) / 2;

    /**
     * Get the offset y
     */
    float offsetY = (height - buttonHeight * nButtonsY - buttonOffset
            * nButtonsY) / 2;
    float x = offsetX;
    float y = offsetY;

    /**
     * Load the achievements
     */
    String games[] = { "visual", "association", "memory", "math", "weight", "sequence" };
    String difficulties[] = { "hard", "normal", "easy" };

    ButtonClickListener buttonClickListener = new ButtonClickListener();

    for (String difficulty : difficulties) {
        x = offsetX;
        for (String game : games) {
            String achieved = "";
            for (Achievement achievement:this.achievements) {
                if (achievement.udid.equals(difficulty + "_" + game)) {
                    achieved = "_achieved";
                }
            }

            Texture texture = this.feedyourbrain.getTexture("achievements/"
                    + difficulty + "_" + game + achieved);
            LeaderBoardButton leaderBoardButton = new LeaderBoardButton(
                    texture);
            leaderBoardButton.setPosition(x, y);
            //leaderBoardButton.addListener(buttonClickListener);
            //leaderBoardButton.setCode(Utils.getCode(game, difficulty));
            //leaderBoardButton.setGame(game);
            //leaderBoardButton.setDifficulty(difficulty);
            this.stage.addActor(leaderBoardButton);
            x += buttonOffset + buttonWidth;
        }
        y += buttonOffset + buttonHeight;
    }

    /**
     * Implemented the back button
     */
    Texture buttonTextureBack = this.feedyourbrain
            .getTexture("main/button_back_mini");
    Vector2 position = Utils.getTopRightCorner(buttonTextureBack);
    this.backButton = new Button(buttonTextureBack);
    this.backButton.setPosition(position.x, position.y);
    this.backButton.setType(ButtonType.EXIT);
    this.backButton.addListener(buttonClickListener);

    this.stage.addActor(this.backButton);
}

/**
 * Listener for all the level buttons
 */
private class ButtonClickListener extends ActorGestureListener {
    @Override
    public void touchUp(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();

            if (type == ButtonType.EXIT) {
                feedyourbrain.leaderBoardScreen = null;
                feedyourbrain.scoreSelectScreen = new ScoreSelectScreen(
                        feedyourbrain);
                feedyourbrain.setScreen(feedyourbrain.scoreSelectScreen);
            }
```

```
                }
            }

            @Override
            public void touchDown(InputEvent event, float x, float y, int pointer,
                    int button) {
                Actor actor = event.getListenerActor();

                if (actor instanceof Button) {
                    Button btn = (Button) actor;
                    int type = btn.getType();
                    Texture texture = null;

                    if (type == ButtonType.PLAY) {
                        texture = feedyourbrain.getTexture("main/button_play_push");
                    } else if (type == ButtonType.GAMES) {
                        texture = feedyourbrain
                                .getTexture("main/button_games_push");
                    } else if (type == ButtonType.EXIT) {
                        texture = feedyourbrain
                                .getTexture("main/button_exit_mini_push");
                    }

                    if (texture != null) {
                        btn.setTexture(texture);
                    }

                }
            }
        }

}
```

2.28. AssociationScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Card;
import com.juanmartos.games.feedyourbrain.actors.Shape;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

import java.util.ArrayList;
import java.util.Random;

/**
 * AssociationScreen extends the class GameScreen,
 * create the view for association game and manages methods for controlling the game
 */
public class AssociationScreen extends GameScreen {

    /**
     * Stage
     */
    private Stage stage;

    /**
 * Actors for this stage
 */
private ArrayList<Actor> actors;

/**
 * List of cards
 */
private ArrayList<Card> cards;

/**
 * List of shaes
 */
private ArrayList<Shape> shapes;

/**
 * List of repeat numbers
 */
private ArrayList<Integer> numbersRepeats;

/**
 * Selected card
 */
private Card cardSelected;

/**
 * Variable to control the number of couples are successful
 */
private int nCoupleHit;

/**
 * Number of hit
 */
private int nHit;

/**
 * Number of streak
 */
private int streak;

/**
 * Array shapes
 */
private String shapesNames[] = { "circle_red", "circle_blue",
        "circle_yellow", "circle_orange", "square_red", "square_blue",
        "square_yellow", "square_orange", "pentagon_red", "pentagon_blue",
        "pentagon_yellow", "pentagon_orange", "star_red", "star_blue",
        "star_yellow", "star_orange" };

/**
 * Constructor parameterized
 * @param feedyourbrain
 *                 General handler of the application
 * @param type
 *                 Type game
 */
public AssociationScreen(FeedYourBrain fyb, int type) {

    super(fyb, type);

    this.nHit = 0;
    this.streak = 0;

    this.stage = new Stage(new FitViewport(Var.width, Var.height));
    this.actors = new ArrayList<Actor>();

    super.loadInputProcessor(this.stage);

    /**
     * Catch the back key
     */
```

```
        Gdx.input.setCatchBackKey(true);

        /**
         * Catch the menu key
         */
        Gdx.input.setCatchMenuKey(true);

        /**
         * Load the default view
         */
        super.loadView();

        /**
         * Load custom view
         */
        this.loadCustomView();

        if (Utils.langByDefault()) {
            super.message.setContent("SEARCH FOR A COUPLE", 1.5f);
        } else {
            super.message.setContent("BUSCA UNA PAREJA", 1.5f);
        }

        this.nCoupleHit = 0;

        this.cards = new ArrayList<Card>();
        this.shapes = new ArrayList<Shape>();

        this.cardSelected = null;

        /**
         * if type 0 -> start
         */
        if (type == 0) {
            this.start();
        }

        /**
         * Send the type of game
         */
        super.gameStage.getScore().setClassName(
                FeedYourBrain.Screens.VISUALSCREEN);
    }

    /**
     * Function to load custom view
     */
    private void loadCustomView() {
        // TODO Auto-generated method stub

    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
                Var.blueBackground, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        super.render(delta);
        if (this.onGame(delta)) { //Dentro del juego

            this.gameStage.draw();
            this.stage.draw();

        }

        delta = Math.min(0.06f, delta);
```

```java
}

/**
 * Function to generate cards
 * @param difficulty
 *                  Game difficulty
 */
public void generateCards(int difficulty) {

    /**
     * Update streak and score
     */
    super.gameStage.getScore().setLevel(difficulty);

    int filas = 0;
    int columnas = 0;
    int parejas = 0;

    if (difficulty == 1) {
        filas = 2;
        columnas = 3;
        parejas = 1;
    } else if (difficulty == 2) {
        filas = 3;
        columnas = 3;
        parejas = 2;
    } else if (difficulty == 3) {
        filas = 3;
        columnas = 5;
        parejas = 4;
    }

    this.shapes.clear();
    this.cards.clear();

    Texture textureCard = super.feedyourbrain.getTexture("card/card");

    float width = textureCard.getWidth();
    float height = textureCard.getHeight();

    /**
     * Random to generate number
     */
    Random randomGenerator = new Random();

    /**
     * Card aux
     */
    Card card;

    /**
     * Block offset
     */
    int separacion = 5;

    /**
     * Array numbers
     */
    ArrayList<Integer> numbers;

    /**
     * Number aux
     */
    int number;

    /**
     * Couple count
     */
    int contadorParejas = 0;
```

```java
/**
 * Number re
 */
int numerosRestantes;

/**
 * Coordinate x
 */
float x = (Var.width - ((columnas * width) + ((columnas - 1) * separacion))) / 2;
x = (int) x;

/**
 * Coordinate y
 */
float y = (Var.height - ((filas * height) + ((filas - 1) * separacion))) / 2;
y = (int) y;

int acc = 0;

boolean set = true;

numbers = new ArrayList<Integer>();
numbersRepeats = new ArrayList<Integer>();

for (int i = 0; i < columnas * filas; ++i) {
    do {
        number = randomGenerator.nextInt(16);
        if (numbers.contains(number)) {
            if (contadorParejas < parejas
                    && !numbersRepeats.contains(number)) {
                contadorParejas++;
                numbersRepeats.add(number);
                set = false;
            }
        } else {
            set = false;
        }
        numerosRestantes = ((columnas * filas) - numbers.size()) - 1;
        if (numerosRestantes < (parejas - contadorParejas)) {
            set = true;
        }
    } while (set);
    numbers.add(number);
    set = true;

    if (Gdx.graphics.getWidth() < 1000) {
        card = new Card(textureCard, new Vector2(x, y), number,
                0.5f);
    } else {
        card = new Card(textureCard, new Vector2(x, y), number,
                1);
    }

    Texture textureShape = super.feedyourbrain.getTexture("card/"
            + this.shapesNames[card.getNumber()]);

    int offsetX = (textureCard.getWidth() - textureShape.getWidth()) / 2;
    int offsetY = (textureCard.getHeight() - textureShape.getHeight()) / 2;

    Shape shape = new Shape(textureShape, new Vector2(x + offsetX, y
            + offsetY));
    CardClickListener cardClickListener = new CardClickListener();
    card.addListener(cardClickListener);

    /**
     * Configurate coordinates
     */
    x = x + width + separacion;
    if (acc == columnas - 1) {
        y = y + height + separacion;
```

```
                x = (Var.width - ((columnas * width) + ((columnas - 1) * separacion))) / 2;

                acc = 0;
            } else {
                acc++;
            }

            /**
             * Add to list
             */
            cards.add(card);
            this.shapes.add(shape);

            this.actors.add(card);
            this.actors.add(shape);

        }

        this.nCoupleHit = parejas;
    }

    /**
     * Function to check card
     * @param card
     *          Card card
     */
    public void checkCard(Card card) {

        Boolean generateCard = false;

        if (card.getState() != 2) {
            card.setState(1);
            card.setTexture(super.feedyourbrain
                    .getTexture("card/card_selected"));
            if (this.cardSelected == null) {
                this.cardSelected = card;
            } else {
                if (this.cardSelected.getNumber() == card.getNumber()
                        && this.cardSelected != card) {
                    card.setTexture(super.feedyourbrain
                            .getTexture("card/card_right"));
                    this.cardSelected.setTexture(super.feedyourbrain
                            .getTexture("card/card_right"));
                    card.setState(2);
                    this.cardSelected.setState(2);
                    this.cardSelected = null;
                    generateCard = true;
                    --this.nCoupleHit;
                } else {
                    this.streak = 0;
                    super.noIncrease();
                    card.setTexture(super.feedyourbrain.getTexture("card/card"));
                    this.cardSelected.setTexture(super.feedyourbrain
                            .getTexture("card/card"));
                    card.setState(0);
                    this.cardSelected.setState(0);
                    this.cardSelected = null;
                }
            }
        }

        if (generateCard && nCoupleHit == 0) {
            super.increase();
            this.nHit++;
            this.streak++;

            /**
             * Del actor and shapes
             */
            for (Actor actor : this.cards)
                actor.remove();

            for (Actor actor : this.shapes)
                actor.remove();

            this.actors.clear();

            /** Only if we are in the main gameplay update the difficulty **/
            if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {

                if (this.nHit == 6) {
                    if (Utils.langByDefault()) {
                        super.message.setContent("SEARCH FOR TWO COUPLE", 1.5f);
                    } else {
                        this.message.setContent("BUSCA DOS PAREJAS", 1.5f);
                    }
                } else if (this.nHit == 11) {
                    if (Utils.langByDefault()) {
                        super.message
                                .setContent("SEARCH FOR FOUR COUPLE", 1.5f);
                    } else {
                        this.message.setContent("BUSCA CUATRO PAREJAS", 1.5f);
                    }
                }

                if (this.nHit <= 5)
                    this.generateCards(1);
                else if (this.nHit <= 10)
                    this.generateCards(2);
                else if (this.nHit <= 15)
                    this.generateCards(3);
                else
                    this.generateCards(3);
            } else {
                this.generateCards(MyPreferences.getDifficulty());
            }

            for (Actor actor : this.cards)
                this.stage.addActor(actor);

            for (Actor actor : this.shapes)
                this.stage.addActor(actor);
        }

        /**
         * Send the streak
         */
        super.gameStage.getStreak().update(this.streak);
        super.gameStage.getScore().setStreak(
                super.gameStage.getStreak().getStreak());
    }

    /**
     * Punctuation function to restore based on the gameplay we're playing
     */
    private void setDifficulty() {
        if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
            if (Utils.langByDefault()) {
                super.message.setContent("SEARCH FOR A COUPLE", 1.5f);
            } else {
                super.message.setContent("BUSCA UNA PAREJA", 1.5f);
            }
            this.generateCards(1);
        } else if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMODE) {
            if (MyPreferences.getDifficulty() == 1) {
                if (Utils.langByDefault()) {
                    super.message.setContent("SEARCH FOR A COUPLE", 1.5f);
                } else {
                    super.message.setContent("BUSCA UNA PAREJA", 1.5f);
                }
            } else if (MyPreferences.getDifficulty() == 2) {
                if (Utils.langByDefault()) {
                    super.message.setContent("SEARCH TWO A COUPLE", 1.5f);
                } else {
                    super.message.setContent("BUSCA DOS PAREJAS", 1.5f);
                }
            } else if (MyPreferences.getDifficulty() == 3) {
                if (Utils.langByDefault()) {
                    super.message.setContent("SEARCH FOR FOUR COUPLE", 1.5f);
                } else {
                    super.message.setContent("BUSCA CUATRO PAREJAS", 1.5f);
                }
            }
            this.generateCards(MyPreferences.getDifficulty());
        }
    }

    /**
     * Start
     */
    @Override
    public void start() {
        super.start();

        this.setDifficulty();

        for (Actor actor : this.actors)
            this.stage.addActor(actor);
    }

    /** Called when the application is paused.
     * An Application is paused before it is destroyed, when a user pressed the Home
     * button on Android or an incoming call happened.
     * On the desktop this will only be called immediately before dispose()
     * is called. */
    @Override
    public void pause() {

        //Guardamos la pantalla en el estado actual
        for (Actor actor : this.stage.getActors())
            this.actors.add(actor);

        super.pause();

    }


    /** Called when the {@link Application} is resumed from a paused state.
     * On Android this happens when the activity gets focus
     * again. On the desktop this method will never be called. */
    @Override
    public void resume() {
        super.resume();

        for (Actor actor : this.actors)
            this.stage.addActor(actor);

        this.actors.clear();
    }

    /**
     * Restart
     */
    @Override
    public void restart() {
        super.restart();

        /**
         * Delete cards and shapes
         */
        for (Actor actor : this.cards)
            actor.remove();

        for (Actor actor : this.shapes)
            actor.remove();

        /**
         * Clear actors
         */
        this.actors.clear();

        this.setDifficulty();

        for (Actor actor : this.cards)
            this.stage.addActor(actor);

        for (Actor actor : this.shapes)
            this.stage.addActor(actor);

        this.streak = 0;
        this.nHit = 0;
    }

    @Override
    public void exit() {
        super.exit();

        this.feedyourbrain.exitScreen(FeedYourBrain.Screens.ASSOCIATIONSCREEN,
                getType());
    }

    /**
     * Listener for all the level buttons.
     */
    private class CardClickListener extends ActorGestureListener {
        @Override
        public void touchUp(InputEvent event, float x, float y, int pointer,
                int button) {

            Actor actor = event.getListenerActor();
            Card card = (Card) actor;

            checkCard(card);

        }
    }

}
```

2.29. AssociationScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.Help;
import com.juanmartos.games.feedyourbrain.utils.Utils;
```

```java
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * HelpScreen extends the class AbstractScreen,
 * implements the interface InputProcessor and create the view for help screen
 */
public class CreditsScreen extends AbstractScreen implements InputProcessor{

    /**
     * Help actor
     */
    private Help help;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Button back
     */
    private Button backButton;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                   General handler of the application
     */
    public CreditsScreen(FeedYourBrain feedyourbrain) {
        super(feedyourbrain);

        /**
         * Catch the back key
         */
        Gdx.input.setCatchBackKey(true);

        this.stage = new Stage(new FitViewport(Var.width, Var.height));

        this.loadInputProcessor();

        this.loadView();
    }

    /**
     * Function to load the input processor
     */
    private void loadInputProcessor() {
        InputMultiplexer multiplexer = new InputMultiplexer();
        multiplexer.addProcessor(this);
        multiplexer.addProcessor(this.stage);

        Gdx.input.setInputProcessor(multiplexer);

    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
                Var.blueBackground, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        super.render(delta);
        this.stage.draw();
    }

    /**
     * Function to load views
     */
```

```java
private void loadView() {

    Texture buttonTextureBack = this.feedyourbrain
            .getTexture("main/button_back_mini");

    Vector2 position = Utils.getTopRightCorner(buttonTextureBack);
    this.backButton = new Button(buttonTextureBack);
    this.backButton.setPosition(position.x, position.y);
    this.backButton.setType(ButtonType.EXIT);

    /**
     * Create listener for all buttons
     */
    ButtonClickListener buttonClickListener = new ButtonClickListener();

    this.backButton.addListener(buttonClickListener);

    this.stage.addActor(this.backButton);

    Texture textureHelp = this.feedyourbrain.getTexture("play/credits");
    this.help = new Help(textureHelp);
    this.stage.addActor(this.help);
}


/** Called when a key was pressed
 *
 * @param keycode one of the constants in {@link Input.Keys}
 * @return whether the input was processed */
@Override
public boolean keyDown(int keycode) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when a key was released
 *
 * @param keycode one of the constants in {@link Input.Keys}
 * @return whether the input was processed */
@Override
public boolean keyUp(int keycode) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when a key was typed
 *
 * @param character The character
 * @return whether the input was processed */
@Override
public boolean keyTyped(char character) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when the screen was touched or a mouse button was pressed.
 * The button parameter will be {@link Buttons#LEFT} on
 * Android and iOS.
 * @param screenX The x coordinate, origin is in the upper left corner
 * @param screenY The y coordinate, origin is in the upper left corner
 * @param pointer the pointer for the event.
 * @param button the button
 * @return whether the input was processed */
@Override
public boolean touchDown(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
```

```
}


/** Called when a finger was lifted or a mouse button was released.
 * The button parameter will be {@link Buttons#LEFT} on Android
 * and iOS.
 * @param pointer the pointer for the event.
 * @param button the button
 * @return whether the input was processed */
@Override
public boolean touchUp(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when a finger or the mouse was dragged.
 * @param pointer the pointer for the event.
 * @return whether the input was processed */
@Override
public boolean touchDragged(int screenX, int screenY, int pointer) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when the mouse was moved without any buttons being pressed.
 * Will not be called on either Android or iOS.
 * @return whether the input was processed */
@Override
public boolean mouseMoved(int screenX, int screenY) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when the mouse wheel was scrolled. Will not be called on either Android or iOS.
 * @param amount the scroll amount, -1 or 1 depending on the direction the wheel was scrolled.
 * @return whether the input was processed. */
@Override
public boolean scrolled(int amount) {
    // TODO Auto-generated method stub
    return false;
}

/**
 * Listener for all the level buttons
 */
private class ButtonClickListener extends ActorGestureListener {
    @Override
    public void touchUp(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();

            if (type == ButtonType.EXIT) {
                feedyourbrain.creditsScreen = null;
                feedyourbrain.helpScreen = new HelpScreen(feedyourbrain);
                feedyourbrain.setScreen(feedyourbrain.helpScreen);
            }

        }
    }

    @Override
    public void touchDown(InputEvent event, float x, float y, int pointer,
            int button) {
```

```
            Actor actor = event.getListenerActor();

            if (actor instanceof Button) {
                Button btn = (Button) actor;
                int type = btn.getType();
                Texture texture = null;

                if (type == ButtonType.EXIT) {
                    texture = feedyourbrain.getTexture("main/button_back_mini_push");
                }

                if (texture != null) {
                    btn.setTexture(texture);
                }

            }
        }
    }

}
```

2.30. EndScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.Input.Buttons;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.stages.Score;
import com.juanmartos.games.feedyourbrain.stages.actors.MaxScore;
import com.juanmartos.games.feedyourbrain.stages.actors.Title;
import com.juanmartos.games.feedyourbrain.stages.actors.YourScore;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * EndScreen extends the class AbstractScreen,
 * implements the interface InputProcessor and create the view for end screen
 */
public class EndScreen extends AbstractScreen implements InputProcessor {

    /**
     * Score obtained
     */
    private int globalScore;

    /**
     * Actor yourScore
     */
    private YourScore yourScore;

    /**
     * Actor maxScore
     */
    private MaxScore maxScore;

    /**
     * Button to restart
     */
    private Button buttonRestart;

    /**
     * Button to exit
     */
    private Button buttonExit;

    /**
     * Actor title
     */
    private Title title;

    /**
     * Button listener
     */
    private ButtonClickListener buttonClickListener;

    /**
     * Actor score left
     */
    private Score scoreLeft;

    /**
     * Actor score right
     */
    private Score scoreRight;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                 General handler of the application
     * @param globalScore
     *                 Global Score
     */
    public EndScreen(FeedYourBrain feedyourbrain, int globalScore) {
        super(feedyourbrain);

        this.globalScore = globalScore;

        //TODO: Score
        //feedyourbrain.actionResolver.submitScoreGPGS(Utils.LEADERBOARDPLAY, this.globalScore);

        this.stage = new Stage(new FitViewport(Var.width, Var.height));

        this.buttonClickListener = new ButtonClickListener();

        /**
         * Catch the back key
         */
        Gdx.input.setCatchBackKey(true);

        this.loadInputProcessor();

        this.loadView();
    }

    /**
     * Function to load the input processor
     */
    private void loadInputProcessor() {
        InputMultiplexer multiplexer = new InputMultiplexer();
        multiplexer.addProcessor(this);
        multiplexer.addProcessor(this.stage);
```

```java
        Gdx.input.setInputProcessor(multiplexer);

    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
                Var.blueBackground, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        super.render(delta);
        this.stage.draw();

    }

    /**
     * Function to load views
     */
    private void loadView() {

        /**
         * Create your score
         */
        Texture textureYourScore = this.feedyourbrain.getTexture(
                "end/your_score", true);
        this.yourScore = new YourScore(textureYourScore);

        /**
         * Create max score
         */
        Texture textureMaxScore = this.feedyourbrain.getTexture(
                "end/max_score", true);
        this.maxScore = new MaxScore(textureMaxScore);

        /**
         * Create buttons
         */
        Texture textureButtonRestart = this.feedyourbrain.getTexture(
                "main/button_restart", true);
        Texture textureButtonExit = this.feedyourbrain.getTexture(
                "main/button_exit", true);

        /**
         * Create title
         */
        this.title = new Title(this.feedyourbrain.getTexture("end/title"));

        /**
         * Create textures
         */
        Texture textures[] = new Texture[10];
        for (int i = 0; i <= 9; ++i)
            textures[i] = this.feedyourbrain.getTexture("games/score/" + i);

        /**
         * Create score left
         */
        this.scoreLeft = new Score(this.feedyourbrain, textures, globalScore);

        /**
         * Create score right
         */
        this.scoreRight = new Score(
                this.feedyourbrain,
                textures,
                this.feedyourbrain.getDatabaseFeedYourBrain().getMaxScore()
        );
        this.scoreRight.setSide(true);
        int nButtons = 2;
        int x = 0;
        int y = 0;
        int offsetX = (Var.width - textureButtonExit.getWidth() * nButtons)
                / (nButtons + 1);
        int offsetY = (int) (Var.height * 0.10);

        x = offsetX;
        y = offsetY;

        this.buttonRestart = new Button(textureButtonRestart);
        this.buttonRestart.setPosition(x, y);
        this.buttonRestart.setType(ButtonType.RESTART);
        this.buttonRestart.addListener(this.buttonClickListener);

        x = x + textureButtonExit.getWidth() + offsetX;

        this.buttonExit = new Button(textureButtonExit);
        this.buttonExit.setPosition(x, y);
        this.buttonExit.setType(ButtonType.EXIT);
        this.buttonExit.addListener(this.buttonClickListener);

        this.stage.addActor(this.yourScore);
        this.stage.addActor(this.maxScore);
        this.stage.addActor(this.buttonRestart);
        this.stage.addActor(this.title);
        this.stage.addActor(this.buttonExit);
        this.stage.addActor(this.scoreLeft);
        this.stage.addActor(this.scoreRight);
    }

    /**
     * Exit screen
     */
    public void exit() {
        super.feedyourbrain.main();
    }

    /** Called when a key was pressed
     *
     * @param keycode one of the constants in {@link Input.Keys}
     * @return whether the input was processed */
    @Override
    public boolean keyDown(int keycode) {
        // TODO Auto-generated method stub
        return false;
    }

    /** Called when a key was released
     *
     * @param keycode one of the constants in {@link Input.Keys}
     * @return whether the input was processed */
    @Override
    public boolean keyUp(int keycode) {
        // TODO Auto-generated method stub
        return false;
    }

    /** Called when a key was typed
     *
     * @param character The character
     * @return whether the input was processed */
    @Override
    public boolean keyTyped(char character) {
        // TODO Auto-generated method stub
        return false;
    }

    /** Called when the screen was touched or a mouse button was pressed.
```

```
 * The button parameter will be {@link Buttons#LEFT} on
 * Android and iOS.
 * @param screenX The x coordinate, origin is in the upper left corner
 * @param screenY The y coordinate, origin is in the upper left corner
 * @param pointer the pointer for the event.
 * @param button the button
 * @return whether the input was processed */
@Override
public boolean touchDown(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when a finger was lifted or a mouse button was released.
 * The button parameter will be {@link Buttons#LEFT} on Android
 * and iOS.
 * @param pointer the pointer for the event.
 * @param button the button
 * @return whether the input was processed */
@Override
public boolean touchUp(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when a finger or the mouse was dragged.
 * @param pointer the pointer for the event.
 * @return whether the input was processed */
@Override
public boolean touchDragged(int screenX, int screenY, int pointer) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when the mouse was moved without any buttons being pressed.
 * Will not be called on either Android or iOS.
 * @return whether the input was processed */
@Override
public boolean mouseMoved(int screenX, int screenY) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when the mouse wheel was scrolled. Will not be called on either Android or iOS.
 * @param amount the scroll amount, -1 or 1 depending on the direction the wheel was scrolled.
 * @return whether the input was processed. */
@Override
public boolean scrolled(int amount) {
    // TODO Auto-generated method stub
    return false;
}

/**
 * Listener for all the level buttons
 */
private class ButtonClickListener extends ActorGestureListener {
    @Override
    public void touchUp(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();

            if (type == ButtonType.RESTART) {
                feedyourbrain.play();
            } else if (type == ButtonType.EXIT) {
                exit();
            }
            }
        }

        @Override
        public void touchDown(InputEvent event, float x, float y, int pointer,
                int button) {
            Actor actor = event.getListenerActor();

            if (actor instanceof Button) {
                Button btn = (Button) actor;
                int type = btn.getType();
                Texture texture = null;

                if (type == ButtonType.RESTART) {
                    texture = feedyourbrain.getTexture(
                            "main/button_restart_push", true);
                } else if (type == ButtonType.EXIT) {
                    texture = feedyourbrain.getTexture("main/button_exit_push",
                            true);
                }

                if (texture != null) {
                    btn.setTexture(texture);
                }

            }
        }
    }
}
```

2.31. GameScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.Message;
import com.juanmartos.games.feedyourbrain.actors.Pause;
import com.juanmartos.games.feedyourbrain.actors.Score;
import com.juanmartos.games.feedyourbrain.actors.Time;
import com.juanmartos.games.feedyourbrain.stages.EndStage;
import com.juanmartos.games.feedyourbrain.stages.GameStage;
import com.juanmartos.games.feedyourbrain.stages.PauseStage;
import com.juanmartos.games.feedyourbrain.stages.StartStage;
import com.juanmartos.games.feedyourbrain.utils.Log;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * GameScreen implements the interface Screen and manages all stages of a game
 */
public class GameScreen implements Screen {

    /**
     * General handler of the application
     */
    protected FeedYourBrain feedyourbrain;

    /**
 * Start Stage
 */
protected StartStage startStage;

/**
 * Message stage
 */
protected Stage messageStage;

/**
 * Static stage
 */
protected Stage stage;

/**
 * Game stage
 */
protected GameStage gameStage;

/**
 * Pause stage
 */
protected PauseStage pauseStage;

/**
 * Final stage
 */
protected EndStage endStage;

/**
 * Variable to show messages
 */
protected Message message;

/**
 * State 0 => No start, 1 => Playing, 2 => Pause
 **/
protected int state;

/**
 * Boolean to check when finalized game
 */
protected boolean end;

/**
 * Variable to save multiple input
 */
private InputMultiplexer multiplexer;

/**
 *  Type 0 => Main | 1 => Mode
 **/
private int type;

/**
 * Constructor parameterized
 * @param feedyourbrain
 *                  General handler of the application
 * @param type
 *                  Type game
 */
public GameScreen(FeedYourBrain feedyourbrain, int type) {

    this.feedyourbrain = feedyourbrain;
    this.messageStage = new Stage(new FitViewport(Var.width, Var.height));
    this.startStage = new StartStage(this.feedyourbrain,
            new StageClickListener(), this.getNameChildClass());
    this.gameStage = new GameStage(this.feedyourbrain,
            new StageClickListener(), type);
    this.pauseStage = new PauseStage(this.feedyourbrain,
                new StageClickListener());
        this.endStage = new EndStage(this.feedyourbrain,
                new StageClickListener(), this.getNameChildClass());

        /**
         * Type of game
         */
        this.type = type;

        /**
         * No started
         */
        this.state = 0;

        /**
         * Game unfinished
         */
        this.end = false;

    }

    /**
     * Function to load the input processor in the stage
     * @param stage
     *
     */
    public void loadInputProcessor(Stage stage) {

        this.stage = stage;

        multiplexer = new InputMultiplexer();
        multiplexer.addProcessor(this.startStage);
        multiplexer.addProcessor(this.stage);
        multiplexer.addProcessor(this.gameStage);
        multiplexer.addProcessor(this.pauseStage);
        multiplexer.addProcessor(this.endStage);
        Gdx.input.setInputProcessor(multiplexer);
    }

    /**
     * Function to load the views
     */
    public void loadView() {
        this.message = new Message(
                this.feedyourbrain.getTexture("message/message"),
                this.feedyourbrain.getFont("bitstreamcharter60"));
        this.messageStage.addActor(this.message);
    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        if (this.onStart()) {
            this.startStage.draw();
        } else if (this.onMessage()) {
            this.message.setTime(delta);
            this.messageStage.draw();
        } else if (this.onPause()) {
            this.pauseStage.draw();
        } else if (this.onEnd()) {
            this.finish();
            this.endStage.message.setTime(delta);
            this.endStage.draw();
        }

    }

    /**
     * Start
```

```
 */
public void start() {

    /** Limpiamos primero los actores **/
    this.startStage.remove();

    //Limpiamos la pantalla de comienzo
    this.startStage.actors.clear();

    this.startStage.clear();

    this.state = 1;
}

/** Called when the application is paused.
 * An Application is paused before it is destroyed, when a user pressed the Home
 * button on Android or an incoming call happened.
 * On the desktop this will only be called immediately before dispose()
 * is called. */
@Override
public void pause() {
    //Limpiamos la pantalla
    this.stage.clear();

    //Cargamos la pantalla de pausa
    this.pauseStage.loadActors();

    this.state = 2;
}

/**
 * Restart
 */
public void restart() {
    // Restart streak


    this.end = false;

    this.endStage.remove();

    /**
     * Delete pause
     */
    this.pauseStage.clear();

    this.gameStage.restart();

    this.state = 1;

    /** if play mode **/
    /**
     * if type 0 -> start
     */
    if (type == 0) {
        this.feedyourbrain.playScreen = new PlayScreen(this.feedyourbrain);
        this.feedyourbrain.setScreen(this.feedyourbrain.playScreen);
    }
}

/** Called when the {@link Application} is resumed from a paused state.
 * On Android this happens when the activity gets focus
 * again. On the desktop this method will never be called. */
@Override
public void resume() {
    /**
     * Delete pause
     */
    this.pauseStage.clear();
```

```
        this.state = 1;
    }

    /**
     * Exit
     */
    public void exit() {
        /**
         * if type 0 -> start
         */
        if (type == 0) {
            Log.log("Entramos");
            this.feedyourbrain.main();
        }
    }

    /**
     * Finish the game
     */
    public void finish() {

        if (!this.end) {

            this.stage.clear();

            if (this.getType() == 0) {
                this.endStage.remove();
                this.feedyourbrain.nextGame(this.getNameChildClass(), this.getScoreNumber());

            } else {
                this.saveScore();
                //TODO: Ver que recursos podemos liberar

                this.endStage.sendScore(this.gameStage.score.score);
                this.endStage.loadActors();
                this.end = true;
            }
        }
    }

    /**
     * Function to store the punctuation obtained in this game
     */
    private void saveScore() {

        //TODO: Revisar esto
//      if (this.getNameChildClass().equals("math")) {
//          MyPreferences.setMathScore(this.getScoreNumber());
//      } else if (this.getNameChildClass().equals("logic")) {
//          MyPreferences.setLogicScore(this.getScoreNumber());
//      } else if (this.getNameChildClass().equals("visual")) {
//          MyPreferences.setVisualScore(this.getScoreNumber());
//      } else if (this.getNameChildClass().equals("memory")) {
//          MyPreferences.setMemoryScore(this.getScoreNumber());
//      }

    }

    /**
     * Function that checks whether we are in the "Game"
     * @param Delta time in each iteration drawn
     * @return True -> If you're in the game | false -> If we are not in the game
     */
    public boolean onGame(float delta) {
        if (this.gameStage.seconds > 0 && this.state == 1 && !this.onMessage()) {
            this.gameStage.update(delta);
            return true;
        } else {
            return false;
        }
```

```
}

/**
 * Function that checks if we are in the "Start"
 * @return True -> If we are on the screen to start
 * false -> If we are not on the screen to start
 **/
public boolean onStart() {
    return (this.state == 0) ? true : false;
}

/**
 * Function that checks if we are in the "Message"
 * @return True -> If we are in the message screen
 * false -> If we are not on screen message
 **/
public boolean onMessage() {
    return (this.message.getTime() >= 0.0f) ? true : false;
}

/**
 * Function that checks if we are in the "Pause"
 * @return True -> If we are in the pause screen
 * false -> If we are not in the pause screen
 **/
public boolean onPause() {
    return (this.state == 2) ? true : false;
}

/**
 * Function that checks if we are in the "Finished"
 * @return True -> If we are in the order screen
 * false -> If we are not on the screen order
 **/
public boolean onEnd() {
    return (this.gameStage.seconds <= 0.0f) ? true : false;
}

/**
 * Increase score
 */
public void increase() {
    this.gameStage.score.increase();
    this.gameStage.ok();
    this.feedyourbrain.sound(FeedYourBrain.SUCCESSSOUND);
}

/**
 * No increase score
 */
public void noIncrease() {
    this.gameStage.no();
    this.feedyourbrain.sound(FeedYourBrain.ERRORSOUND);
}

/**
 * Return score
 * @return score
 */
public int getScoreNumber() {
    return this.gameStage.score.getScore();
}

/**
 * Return time
 * @return Time
 */
public Time getTime() {
    return this.gameStage.getTime();
```

```
}

/**
 * Return actor score
 * @return
 */
public Score getScore() {
    return this.gameStage.getScore();
}

/**
 * Return actor pause
 * @return
 */
public Pause getPause() {
    return this.gameStage.getPause();
}

public String getNameChildClass() {

    if (this instanceof MemoryScreen) {
        return "memory";
    } else if (this instanceof MathScreen) {
        return "math";
    } else if (this instanceof AssociationScreen) {
        return "logic";
    } else if (this instanceof VisualScreen) {
        return "visual";
    } else if (this instanceof SequenceScreen) {
        return "sequence";
    } else if (this instanceof WeightScreen) {
        return "weight";
    }

    return "memory";

}

/**
 * Return type
 * @return type
 */
public int getType() {
    return this.type;
}

/**
 * Listener for all the level buttons
 */
private class StageClickListener extends ActorGestureListener {
    @Override
    public void touchUp(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();
        int type = -1;

        Button btn = null;

        if (actor instanceof Button) {
            btn = (Button) event.getListenerActor();
            type = btn.getType();
        } else if (actor instanceof Pause) {
            Pause pause = (Pause) event.getListenerActor();
            type = pause.getType();
        }

        Texture texture = feedyourbrain.getTexture("main/button_exit");

        if (type == ButtonType.BACK) {
            texture = feedyourbrain.getTexture("main/button_back", true);
            exit();
        } else if (type == ButtonType.START) {
            texture = feedyourbrain.getTexture("main/button_start", true);
            start();
        } else if (type == ButtonType.EXIT) {
            texture = feedyourbrain.getTexture("main/button_exit", true);
            exit();
        } else if (type == ButtonType.PAUSE) {
            texture = feedyourbrain.getTexture("main/button_exit", true);
            pause();
        } else if (type == ButtonType.RESUME) {
            texture = feedyourbrain.getTexture("main/button_continue", true);
            resume();
        } else if (type == ButtonType.RESTART) {
            System.out.println("RESTART");
            texture = feedyourbrain.getTexture("main/button_restart", true);
            restart();
        }

        if (actor instanceof Button) {
            btn.setTexture(texture);
        }

    }

    @Override
    public void touchDown(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();
        int type = -1;

        if (actor instanceof Button) {
            Button btn = (Button) event.getListenerActor();
            type = btn.getType();

            Texture texture = feedyourbrain.getTexture(
                    "main/button_exit_push", true);

            if (type == ButtonType.BACK) {
                texture = feedyourbrain.getTexture("main/button_back_push",
                        true);
                Log.log("Volver");
            } else if (type == ButtonType.START) {
                texture = feedyourbrain.getTexture(
                        "main/button_start_push", true);
                Log.log("Empezar");
            } else if (type == ButtonType.EXIT) {
                texture = feedyourbrain.getTexture("main/button_exit_push",
                        true);
                Log.log("Salir");
            } else if (type == ButtonType.PAUSE) {
                texture = feedyourbrain
                        .getTexture("main/button_exit", true);
                Log.log("Parar");
            } else if (type == ButtonType.RESUME) {
                texture = feedyourbrain.getTexture(
                        "main/button_continue_push", true);
                Log.log("Continuar");
            } else if (type == ButtonType.RESTART) {
                texture = feedyourbrain.getTexture(
                        "main/button_restart_push", true);
                Log.log("Reiniciar");
            }

            btn.setTexture(texture);
        }
    }
}

/**
```

```
     * Called when this screen should release all resources.
     */
    @Override
    public void dispose() {
        // TODO Auto-generated method stub

    }

    /**
     * Called when this screen is no longer the current screen for a Game.
     */
    @Override
    public void hide() {
        // TODO Auto-generated method stub

    }


    /** Called when the {@link Application} is resized.
     * This can happen at any point during a non-paused state but will never happen
     * before a call to {@link #create()}.
     *
     * @param width the new width in pixels
     * @param height the new height in pixels */
    @Override
    public void resize(int arg0, int arg1) {
        // TODO Auto-generated method stub

    }

    /** Called when this screen becomes the current screen for a {@link Game}. */
    @Override
    public void show() {
        // TODO Auto-generated method stub

    }
}
```

2.32. HelpScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.Help;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * HelpScreen extends the class AbstractScreen,
 * implements the interface InputProcessor and create the view for help screen
 */
public class HelpScreen extends AbstractScreen implements InputProcessor{

    /**
     * Help actor
     */
```

```
private Help help;

/**
 * Stage
 */
private Stage stage;

/**
 * Button back
 */
private Button backButton;

/**
 * Button back
 */
private Button creditsButton;

/**
 * Constructor parameterized
 * @param feedyourbrain
 *                  General handler of the application
 */
public HelpScreen(FeedYourBrain feedyourbrain) {
    super(feedyourbrain);

    /**
     * Catch the back key
     */
    Gdx.input.setCatchBackKey(true);

    this.stage = new Stage(new FitViewport(Var.width, Var.height));

    this.loadInputProcessor();

    this.loadView();
}

/**
 * Function to load the input processor
 */
private void loadInputProcessor() {
    InputMultiplexer multiplexer = new InputMultiplexer();
    multiplexer.addProcessor(this);
    multiplexer.addProcessor(this.stage);

    Gdx.input.setInputProcessor(multiplexer);

}

/** Called when the screen should render itself.
 * @param delta The time in seconds since the last render. */
@Override
public void render(float delta) {
    Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
            Var.blueBackground, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    super.render(delta);
    this.stage.draw();
}

/**
 * Function to load views
 */
private void loadView() {

    Texture buttonTextureBack = this.feedyourbrain
            .getTexture("main/button_back_mini");

    Vector2 position = Utils.getTopRightCorner(buttonTextureBack);
```

```
        this.backButton = new Button(buttonTextureBack);
        this.backButton.setPosition(position.x, position.y);
        this.backButton.setType(ButtonType.EXIT);

        Texture buttonTextureInfo = this.feedyourbrain
                .getTexture("main/button_info");

        Vector2 positionInfo = Utils.getBottomRightCorner(buttonTextureInfo);
        this.creditsButton = new Button(buttonTextureInfo);
        this.creditsButton.setPosition(positionInfo.x, positionInfo.y);
        this.creditsButton.setType(ButtonType.INFO);

        /**
         * Create listener for all buttons
         */
        ButtonClickListener buttonClickListener = new ButtonClickListener();

        this.backButton.addListener(buttonClickListener);
        this.creditsButton.addListener(buttonClickListener);

        this.stage.addActor(this.backButton);
        this.stage.addActor(this.creditsButton);

        Texture textureHelp = this.feedyourbrain.getTexture("main/help");
        this.help = new Help(textureHelp);
        this.stage.addActor(this.help);
    }


    /** Called when a key was pressed
     *
     * @param keycode one of the constants in {@link Input.Keys}
     * @return whether the input was processed */
    @Override
    public boolean keyDown(int keycode) {
        // TODO Auto-generated method stub
        return false;
    }


    /** Called when a key was released
     *
     * @param keycode one of the constants in {@link Input.Keys}
     * @return whether the input was processed */
    @Override
    public boolean keyUp(int keycode) {
        // TODO Auto-generated method stub
        return false;
    }


    /** Called when a key was typed
     *
     * @param character The character
     * @return whether the input was processed */
    @Override
    public boolean keyTyped(char character) {
        // TODO Auto-generated method stub
        return false;
    }


    /** Called when the screen was touched or a mouse button was pressed.
     * The button parameter will be {@link Buttons#LEFT} on
     * Android and iOS.
     * @param screenX The x coordinate, origin is in the upper left corner
     * @param screenY The y coordinate, origin is in the upper left corner
     * @param pointer the pointer for the event.
     * @param button the button
     * @return whether the input was processed */
```

```java
@Override
public boolean touchDown(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when a finger was lifted or a mouse button was released.
 * The button parameter will be {@link Buttons#LEFT} on Android
 * and iOS.
 * @param pointer the pointer for the event.
 * @param button the button
 * @return whether the input was processed */
@Override
public boolean touchUp(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when a finger or the mouse was dragged.
 * @param pointer the pointer for the event.
 * @return whether the input was processed */
@Override
public boolean touchDragged(int screenX, int screenY, int pointer) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when the mouse was moved without any buttons being pressed.
 * Will not be called on either Android or iOS.
 * @return whether the input was processed */
@Override
public boolean mouseMoved(int screenX, int screenY) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when the mouse wheel was scrolled. Will not be called on either Android or iOS.
 * @param amount the scroll amount, -1 or 1 depending on the direction the wheel was scrolled.
 * @return whether the input was processed. */
@Override
public boolean scrolled(int amount) {
    // TODO Auto-generated method stub
    return false;
}

/**
 * Listener for all the level buttons
 */
private class ButtonClickListener extends ActorGestureListener {
    @Override
    public void touchUp(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();

            if (type == ButtonType.EXIT) {
                feedyourbrain.helpScreen = null;
                feedyourbrain.main();
            }else if (type == ButtonType.INFO) {
                feedyourbrain.helpScreen = null;
                feedyourbrain.creditsScreen = new CreditsScreen(feedyourbrain);
                feedyourbrain.setScreen(feedyourbrain.creditsScreen);
            }
            }
        }

        @Override
        public void touchDown(InputEvent event, float x, float y, int pointer,
                int button) {
            Actor actor = event.getListenerActor();

            if (actor instanceof Button) {
                Button btn = (Button) actor;
                int type = btn.getType();
                Texture texture = null;

                if (type == ButtonType.EXIT) {
                    texture = feedyourbrain.getTexture("main/button_back_mini_push");
                }else if (type == ButtonType.INFO) {
                    texture = feedyourbrain.getTexture("main/button_info_push");
                }

                if (texture != null) {
                    btn.setTexture(texture);
                }

            }
        }
    }

}
```

2.33. LeaderBoardScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.Net;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.Array;
import com.badlogic.gdx.utils.JsonReader;
import com.badlogic.gdx.utils.JsonValue;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.model.Score;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.LeaderBoardButton;
import com.juanmartos.games.feedyourbrain.actors.MessageLog;
import com.juanmartos.games.feedyourbrain.utils.ApiInterface;
import com.juanmartos.games.feedyourbrain.utils.Log;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

import java.util.ArrayList;

/**
 * LeaderBoardScreen extends the class AbstractScreen and create the view for leaderboard screen
 */
public class LeaderBoardGameScreen extends AbstractScreen {

    /**
     * Stage
 */
private Stage stage;

/**
 * Back button
 */
private Button backButton;

private Array<Score> scores;

private Score userScore;

private String game;

private String difficulty;

private Boolean isMain;

/**
 * Constructor parameterized
 * @param feedyourbrain
 *                  General handler of the application
 */
public LeaderBoardGameScreen(
        FeedYourBrain feedyourbrain, String game, String difficulty, boolean isMain
) {
    super(feedyourbrain);

    this.scores = new Array<Score>();
    this.userScore = new Score();
    this.isMain = isMain;

    if (this.isMain) {
        this.game = "0";
        this.difficulty = "0";
    } else {
        this.game = String.valueOf(Utils.getGame(game) + 1);
        this.difficulty = String.valueOf(Utils.getDifficulty(difficulty) + 1);
    }

    this.stage = new Stage(new FitViewport(Var.width, Var.height));

    //Capturamos la pulsacion en el boton atras
    Gdx.input.setCatchBackKey(true);

    this.loadInputProcessor();

    this.getScore();

    //this.loadView();
}

/**
 * Function to load the input processor
 */
private void loadInputProcessor() {
    InputMultiplexer multiplexer = new InputMultiplexer();
    multiplexer.addProcessor(this.stage);

    Gdx.input.setInputProcessor(multiplexer);

}

/** Called when the screen should render itself.
 * @param delta The time in seconds since the last render. */
@Override
public void render(float delta) {
    Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
            Var.blueBackground, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
    super.render(delta);
    this.stage.draw();
}

private void getScore()
{
    this.feedyourbrain.api.getScore(new ApiInterface() {
        @Override
        public void success(Net.HttpResponse httpResponse) {

            try {
                String response = httpResponse.getResultAsString();

                JsonValue json = new JsonReader().parse(response);

                JsonValue scoresJson = json.get("scores");

                for (JsonValue scoreJson : scoresJson.iterator())
                {
                    Score score = new Score();

                    score.index = scoreJson.getString("index");
                    score.name = scoreJson.getString("name");
                    score.amount = scoreJson.getString("amount");
                    scores.add(score);
                }

                // Load user score
                JsonValue userScoreJson = json.get("user_score");

                userScore.index = userScoreJson.getString("index");
                userScore.name = userScoreJson.getString("name");
                userScore.amount = userScoreJson.getString("amount");

            } catch (Exception e) {
                Log.log(e.getLocalizedMessage());
            }

            Gdx.app.postRunnable(new Runnable() {
                @Override
                public void run() {
                    loadView();
                }
            });
        }

        @Override
        public void failed() {

        }

        @Override
        public void cancelled() {

        }
    }, MyPreferences.getId(), game, difficulty);
}

/**
 * Function to load views
 */
private void loadView() {

    ArrayList<MessageLog> messageLogs = new ArrayList<MessageLog>();

    float y = Var.height - (Var.height * .3f);
    float offsetY = 150;

    for (Score score:this.scores) {
            MessageLog messageLog = new MessageLog(
                    this.feedyourbrain.getTexture("message/messagebig"),
                    this.feedyourbrain.getFont("bitstreamcharter50")
            );

            String content = score.index + " . " + score.amount + " " + score.name;

            messageLog.setContent(content, 1.0f);
            messageLog.setPosition(0, y);
            messageLogs.add(messageLog);
            y = y - offsetY;
        }


        for(MessageLog messageLog : messageLogs) {
            this.stage.addActor(messageLog);
        }

        y = y - 50;

        MessageLog messageLog = new MessageLog(
                this.feedyourbrain.getTexture("message/messagebig"),
                this.feedyourbrain.getFont("bitstreamcharter50")
        );

        String content = ((userScore.index != null) ? userScore.index : "-") +
                " . " + ((userScore.amount != null) ? userScore.amount : "0") + " Your score";
        messageLog.setContent(content, 1.0f);
        messageLog.setPosition(0, y);
        this.stage.addActor(messageLog);

        ButtonClickListener buttonClickListener = new ButtonClickListener();

        /**
         * Implemented the back button
         */
        Texture buttonTextureBack = this.feedyourbrain
                .getTexture("main/button_back_mini");
        Vector2 position = Utils.getTopRightCorner(buttonTextureBack);
        this.backButton = new Button(buttonTextureBack);
        this.backButton.setPosition(position.x, position.y);
        this.backButton.setType(ButtonType.EXIT);
        this.backButton.addListener(buttonClickListener);

        this.stage.addActor(this.backButton);
    }

    /**
     * Listener for all the level buttons
     */
    private class ButtonClickListener extends ActorGestureListener {
        @Override
        public void touchUp(InputEvent event, float x, float y, int pointer,
                int button) {
            Actor actor = event.getListenerActor();

            if (actor instanceof LeaderBoardButton) {
                LeaderBoardButton leaderBoardButton = (LeaderBoardButton) actor;
                String code = leaderBoardButton.getCode();
                //feedyourbrain.actionResolver.getLeaderboardGPGS(code);
                //stage.clear();
            } else if (actor instanceof Button) {
                Button btn = (Button) actor;
                int type = btn.getType();

                if (type == ButtonType.EXIT) {
                    feedyourbrain.leaderBoardGameScreen = null;

                    if (isMain) {
                        feedyourbrain.scoreSelectScreen = new ScoreSelectScreen(feedyourbrain);
                        feedyourbrain.setScreen(feedyourbrain.scoreSelectScreen);
                    } else {
                        feedyourbrain.leaderBoardScreen = new LeaderBoardScreen(feedyourbrain);
                        feedyourbrain.setScreen(feedyourbrain.leaderBoardScreen);
                    }
                }

            }
        }

        @Override
        public void touchDown(InputEvent event, float x, float y, int pointer,
                int button) {
            Actor actor = event.getListenerActor();

            if (actor instanceof Button) {
                Button btn = (Button) actor;
                int type = btn.getType();
                Texture texture = null;

                if (type == ButtonType.PLAY) {
                    texture = feedyourbrain.getTexture("main/button_play_push");
                } else if (type == ButtonType.GAMES) {
                    texture = feedyourbrain
                            .getTexture("main/button_games_push");
                } else if (type == ButtonType.EXIT) {
                    texture = feedyourbrain
                            .getTexture("main/button_exit_mini_push");
                }

                if (texture != null) {
                    btn.setTexture(texture);
                }

            }
        }
    }

}
```

2.34. LeaderBoardScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.LeaderBoardButton;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * LeaderBoardScreen extends the class AbstractScreen and create the view for leaderboard screen
 */
public class LeaderBoardScreen extends AbstractScreen {

    /**
     * Stage
     */
```

```
private Stage stage;

/**
 * Back button
 */
private Button backButton;

/**
 * Constructor parameterized
 * @param feedyourbrain
 *                  General handler of the application
 */
public LeaderBoardScreen(FeedYourBrain feedyourbrain) {
    super(feedyourbrain);

    this.stage = new Stage(new FitViewport(Var.width, Var.height));

    //Capturamos la pulsacion en el boton atras
    Gdx.input.setCatchBackKey(true);

    this.loadInputProcessor();

    this.loadView();
}

/**
 * Function to load the input processor
 */
private void loadInputProcessor() {
    InputMultiplexer multiplexer = new InputMultiplexer();
    multiplexer.addProcessor(this.stage);

    Gdx.input.setInputProcessor(multiplexer);

}

/** Called when the screen should render itself.
 * @param delta The time in seconds since the last render. */
@Override
public void render(float delta) {
    Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
            Var.blueBackground, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    super.render(delta);
    this.stage.draw();
}

/**
 * Function to load views
 */
private void loadView() {

    /**
     * Width and height
     */
    int width = Var.width;
    int height = Var.height;

    /**
     * Set the width and height of the button
     */
    float buttonWidth = 200;
    float buttonHeight = 200;

    float buttonOffset = 50.0f;

    int nButtonsX = 6;
    int nButtonsY = 3;
    /**
     * Get the offset x
     */
    float offsetX = (width - buttonWidth * nButtonsX - buttonOffset
            * nButtonsX) / 2;

    /**
     * Get the offset y
     */
    float offsetY = (height - buttonHeight * nButtonsY - buttonOffset
            * nButtonsY) / 2;

    float x = offsetX;
    float y = offsetY;

    /**
     * Load the achievements
     */
    String games[] = { "visual", "association", "memory", "math", "weight", "sequence" };
    String difficulties[] = { "hard", "normal", "easy" };

    ButtonClickListener buttonClickListener = new ButtonClickListener();

    for (String difficulty : difficulties) {
        x = offsetX;
        for (String game : games) {
            Texture texture = this.feedyourbrain.getTexture("leaderboard/"
                    + game + "_" + difficulty);
            LeaderBoardButton leaderBoardButton = new LeaderBoardButton(
                    texture);
            leaderBoardButton.setPosition(x, y);
            leaderBoardButton.addListener(buttonClickListener);
            leaderBoardButton.setCode(Utils.getCode(game, difficulty));
            leaderBoardButton.setGame(game);
            leaderBoardButton.setDifficulty(difficulty);
            this.stage.addActor(leaderBoardButton);
            x += buttonOffset + buttonWidth;
        }
        y += buttonOffset + buttonHeight;
    }

    /**
     * Implemented the back button
     */
    Texture buttonTextureBack = this.feedyourbrain
            .getTexture("main/button_back_mini");
    Vector2 position = Utils.getTopRightCorner(buttonTextureBack);
    this.backButton = new Button(buttonTextureBack);
    this.backButton.setPosition(position.x, position.y);
    this.backButton.setType(ButtonType.EXIT);
    this.backButton.addListener(buttonClickListener);

    this.stage.addActor(this.backButton);
}

/**
 * Listener for all the level buttons
 */
private class ButtonClickListener extends ActorGestureListener {
    @Override
    public void touchUp(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof LeaderBoardButton) {
            LeaderBoardButton leaderBoardButton = (LeaderBoardButton) actor;
            String code = leaderBoardButton.getCode();
            String game = leaderBoardButton.getGame();
            String difficulty = leaderBoardButton.getDifficulty();
            feedyourbrain.leaderBoardScreen = null;
```

```
                feedyourbrain.leaderBoardGameScreen
                        = new LeaderBoardGameScreen(feedyourbrain, game, difficulty, false);
                feedyourbrain.setScreen(feedyourbrain.leaderBoardGameScreen);

                //feedyourbrain.actionResolver.getLeaderboardGPGS(code);
                //stage.clear();
            } else if (actor instanceof Button) {
                Button btn = (Button) actor;
                int type = btn.getType();

                if (type == ButtonType.EXIT) {
                    feedyourbrain.leaderBoardScreen = null;
                    feedyourbrain.scoreSelectScreen = new ScoreSelectScreen(
                            feedyourbrain);
                    feedyourbrain.setScreen(feedyourbrain.scoreSelectScreen);
                }

            }
        }

        @Override
        public void touchDown(InputEvent event, float x, float y, int pointer,
                int button) {
            Actor actor = event.getListenerActor();

            if (actor instanceof Button) {
                Button btn = (Button) actor;
                int type = btn.getType();
                Texture texture = null;

                if (type == ButtonType.PLAY) {
                    texture = feedyourbrain.getTexture("main/button_play_push");
                } else if (type == ButtonType.GAMES) {
                    texture = feedyourbrain
                            .getTexture("main/button_games_push");
                } else if (type == ButtonType.EXIT) {
                    texture = feedyourbrain
                            .getTexture("main/button_exit_mini_push");
                }

                if (texture != null) {
                    btn.setTexture(texture);
                }

            }
        }
    }

}
```

2.35. LevelScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
```

```java
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * LevelScreen extends the class AbstractScreen and create the view for level screen
 */
public class LevelScreen extends AbstractScreen{

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                  General handler of the application
     */
    public LevelScreen(FeedYourBrain feedyourbrain) {
        super(feedyourbrain);

        this.stage = new Stage(new FitViewport(Var.width, Var.height));

        /**
         * Catch the back key
         */
        Gdx.input.setCatchBackKey(true);

        this.loadInputProcessor();

        this.loadView();
    }

    /**
     * Function to load the input processor
     */
    private void loadInputProcessor() {
        InputMultiplexer multiplexer = new InputMultiplexer();
        multiplexer.addProcessor(this.stage);

        Gdx.input.setInputProcessor(multiplexer);
    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
                Var.blueBackground, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        super.render(delta);
        this.stage.draw();
    }

    /**
     * Function to load views
     */
    private void loadView() {

        /**
         * Width and height screens
         */
        int width = Var.width;
        int height = Var.height;

        /**
         * Width and height button
         */
        float buttonWidth = 800;
        float buttonHeight = 133.33f;

        /**
         * Get the offset x
         */
        float offsetX = (width - buttonWidth) / 2;

        /**
         * Get the offset y
         */
        int nButtons = 3;
        float offsetY = (height - buttonHeight * nButtons) / (nButtons + 1);

        float x = offsetX;
        float y = offsetY;

        int typeButtons[] = { ButtonType.EASY, ButtonType.NORMAL,
                ButtonType.HARD };
        String nameButtons[] = { "easy", "normal", "hard" };

        ButtonClickListener buttonClickListener = new ButtonClickListener();

        for (int i = nButtons - 1; i >= 0; --i) {
            Texture texture = this.feedyourbrain.getTexture("level/button_"
                    + nameButtons[i], true);
            Button button = new Button(texture);
            button.setPosition(x, y);
            button.setType(typeButtons[i]);
            button.addListener(buttonClickListener);
            y += offsetY + buttonHeight;
            this.stage.addActor(button);
        }

        /**
         * Add back button
         */
        Texture buttonTextureBack = this.feedyourbrain
                .getTexture("main/button_back_mini");

        Vector2 position = Utils.getTopRightCorner(buttonTextureBack);
        Button backButton = new Button(buttonTextureBack);
        backButton.setPosition(position.x, position.y);
        backButton.setType(ButtonType.EXIT);
        backButton.addListener(buttonClickListener);

        this.stage.addActor(backButton);
    }

    /**
     * Function to go to mode screen
     */
    public void goToMode() {
        super.feedyourbrain.mainScreen = null;
        super.feedyourbrain.modeScreen = new ModeScreen(super.feedyourbrain);
        super.feedyourbrain.setScreen(super.feedyourbrain.modeScreen);
    }

    /**
     * Listener for all the level buttons
     */
    private class ButtonClickListener extends ActorGestureListener {
        @Override
        public void touchUp(InputEvent event, float x, float y, int pointer,
                int button) {
            Actor actor = event.getListenerActor();

            if (actor instanceof Button) {
                Button btn = (Button) actor;
                int type = btn.getType();
                if (type == ButtonType.EASY) {
                    MyPreferences.setDifficulty(MyPreferences.GAMEPLAYEASY);
                } else if (type == ButtonType.NORMAL) {
                    MyPreferences.setDifficulty(MyPreferences.GAMEPLAYNORMAL);
                } else if (type == ButtonType.HARD) {
                    MyPreferences.setDifficulty(MyPreferences.GAMEPLAYHARD);
                } else if (type == ButtonType.EXIT) {
                    System.out.println("Exit");
                    feedyourbrain.levelScreen = null;
                    feedyourbrain.main();
                    return;
                }

                MyPreferences.setGamePlay(MyPreferences.GAMEPLAYMODE);

                goToMode();
            }
        }

        @Override
        public void touchDown(InputEvent event, float x, float y, int pointer,
                int button) {
            Actor actor = event.getListenerActor();

            if (actor instanceof Button) {
                Button btn = (Button) actor;
                int type = btn.getType();
                String nameButton = "";

                if (type == ButtonType.EASY) {
                    nameButton = "level/button_easy";
                } else if (type == ButtonType.NORMAL) {
                    nameButton = "level/button_normal";
                } else if (type == ButtonType.HARD) {
                    nameButton = "level/button_hard";
                } else if (type == ButtonType.EXIT) {
                    nameButton = "main/button_back_mini";
                }

                if (!nameButton.equals("")) {
                    if (nameButton.equals("main/button_back_mini")) {
                        btn.setTexture(feedyourbrain.getTexture(nameButton
                                + "_push"));
                    } else {
                        btn.setTexture(feedyourbrain.getTexture(nameButton
                                + "_push", true));
                    }
                }

            }
        }
    }

}
```

2.36. LoadingScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import java.util.ArrayList;
import java.util.Random;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.scenes.scene2d.Action;
```

```java
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Character;
import com.juanmartos.games.feedyourbrain.actors.Shape;
import com.juanmartos.games.feedyourbrain.actors.loading.Background;
import com.juanmartos.games.feedyourbrain.actors.loading.Bar;
import com.juanmartos.games.feedyourbrain.utils.Log;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * LoadingScreen implements the interface Screen, create the view and load all the assets
 */
public class LoadingScreen implements Screen {

    /**
     * String to draw in the loading screen
     */
    private String loading = "FEEDYOURBRAIN";

    /**
     * Stage
     */
    protected Stage stage;

    /**
     * Variable to control the last character
     */
    protected boolean load;

    /**
     * General handler of the application
     */
    private FeedYourBrain feedYourBrain;

    /**
     * List of characters to draw
     */
    private ArrayList<Character> characters;

    private Bar bar;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                  General handler of the application
     */
    public LoadingScreen(FeedYourBrain feedyourbrain) {
        this.feedYourBrain = feedyourbrain;
        this.stage = new Stage(new FitViewport(Var.width, Var.height));
        this.load = false;
        this.characters = new ArrayList<Character>();
    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground, Var.blueBackground, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
        this.stage.act(Gdx.graphics.getDeltaTime());
        /**
         * If load all the images
         */
        if (this.feedYourBrain.getManager().update()) {

            Boolean completed = true;

            for (Character character : this.characters) {
                if (!character.hasCompleted()) {
                    completed = false;
                }
            }

            if (completed) {
                this.feedYourBrain.loadScreen();
                this.feedYourBrain.setScreen(this.feedYourBrain.mainScreen);
            }
        }

        this.stage.draw();

        /**
         * if load the last image
         */
        if (this.feedYourBrain.getManager().isLoaded("data/loading/bar_up.png",
                Texture.class)
                && !this.load) {
            loadView();
            this.load = true;
        }

        if (this.bar != null) {
            ShapeRenderer shapeRenderer = new ShapeRenderer();
            shapeRenderer.begin(ShapeRenderer.ShapeType.Filled);
            shapeRenderer.setColor(Color.BLACK);
            shapeRenderer.rect(Var.width * 0.05f, Var.height * 0.3f, Var.width * 0.9f, 25);
            shapeRenderer.end();

            shapeRenderer.begin(ShapeRenderer.ShapeType.Filled);
            shapeRenderer.setColor(Color.GREEN);
            shapeRenderer.rect(Var.width * 0.05f, Var.height * 0.3f, this.getProgress(), 25);
            shapeRenderer.end();
        }
    }

    public void loadView() {
        Texture textureBackground =  this.feedYourBrain.getTexture("loading/background");
        Background background = new Background(textureBackground);

        this.bar = new Bar(this.feedYourBrain.getManager());

        this.stage.addActor(background);
        //this.stage.addActor(bar);
    }

    private int getProgress() {

        int progress
                = (int)(Var.width * 0.9 * this.feedYourBrain.getManager().getLoadedAssets()) / 260;

        if (progress > Var.width * 0.9) {
            progress = (int) (Var.width * 0.9);
        }

        return progress;
    }

    @Override
    public void resize(int width, int height) {
        // TODO Auto-generated method stub

    }

    /** Called when this screen becomes the current screen for a {@link Game}. */
    @Override
    public void show() {
        // TODO Auto-generated method stub
    }

    /**
     * Called when this screen is no longer the current screen for a Game.
     */
    @Override
    public void hide() {
        // TODO Auto-generated method stub

    }


    /** Called when the application is paused.
     * An Application is paused before it is destroyed, when a user pressed the Home
     * button on Android or an incoming call happened.
     * On the desktop this will only be called immediately before dispose()
     * is called. */
    @Override
    public void pause() {
        // TODO Auto-generated method stub

    }


    /** Called when the {@link Application} is resumed from a paused state.
     * On Android this happens when the activity gets focus
     * again. On the desktop this method will never be called. */
    @Override
    public void resume() {
        // TODO Auto-generated method stub

    }

    /**
     * Called when this screen should release all resources.
     */
    @Override
    public void dispose() {
        // TODO Auto-generated method stub

    }

}
```

2.37. MainScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.Input.Buttons;
import com.badlogic.gdx.Input.Keys;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.Message;
import com.juanmartos.games.feedyourbrain.actors.MessageLog;
import com.juanmartos.games.feedyourbrain.actors.Sound;
import com.juanmartos.games.feedyourbrain.utils.Log;
```

```java
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * MainScreen extends the class AbstractScreen,
 * implements the interface InputProcessor and create the view for main screen
 */
public class MainScreen extends AbstractScreen implements InputProcessor {
    /**
     * Button to access main mode
     */
    private Button mainButton;

    /**
     * Button to access games mode
     */
    private Button modeButton;

    /**
     * Button to access score screen
     */
    private Button scoreButton;

    /**
     * Button to access help screen
     */
    private Button helpButton;

    /**
     * Button to exit app
     */
    private Button exitButton;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Variable to show the sound
     */
    protected Sound sound;

    /**
     * Message to show
     */
    protected Message message;

    protected MessageLog log;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                  General handler of the application
     */
    public MainScreen(FeedYourBrain feedyourbrain) {
        super(feedyourbrain);

        /*if (!feedyourbrain.actionResolver.getSignedInGPGS())
            feedyourbrain.actionResolver.loginGPGS();*/

        this.stage = new Stage(new FitViewport(Var.width, Var.height));
        this.message = new Message(
                this.feedyourbrain.getTexture("message/messagebig"),
                this.feedyourbrain.getFont("bitstreamcharter50"));

        this.log = new MessageLog(
                this.feedyourbrain.getTexture("message/message"),
                this.feedyourbrain.getFont("bitstreamcharter50"));
        /**
         * Catch the back key
         */
        Gdx.input.setCatchBackKey(true);

        this.loadInputProcessor();

        this.loadView();
    }

    /**
     * Function to load the input processor
     */
    private void loadInputProcessor() {
        InputMultiplexer multiplexer = new InputMultiplexer();
        multiplexer.addProcessor(this);
        multiplexer.addProcessor(this.stage);

        Gdx.input.setInputProcessor(multiplexer);

    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
                Var.blueBackground, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        super.render(delta);
        this.stage.draw();
        this.message.setTime(delta);
    }

    /**
     * Function to load views
     */
    private void loadView() {

        /**
         * Width and height screen
         */
        int width = Var.width;
        int height = Var.height;

        /**
         * Width and height buttos
         */
        float buttonWidth = 800;
        float buttonHeight = 133.33f;

        /**
         * Get the offset x
         */
        float offsetX = (width - buttonWidth) / 2;

        /**
         * Get the offset y
         */
        float offsetY = (height - buttonHeight * 4) / 5;

        float x = offsetX;
        float y = offsetY;

        Texture buttonTextureHelp = this.feedyourbrain.getTexture(
                "main/button_help", true);
        Texture buttonTextureScore = this.feedyourbrain.getTexture(
                "main/button_score", true);
        Texture buttonTextureGames = this.feedyourbrain.getTexture(
                "main/button_games", true);
        Texture buttonTextureGame = this.feedyourbrain.getTexture(
                "main/button_play", true);
        Texture buttonTextureExit = this.feedyourbrain
                .getTexture("main/button_exit_mini");

        BitmapFont buttonFont = this.feedyourbrain
                .getFont("bitstreamcharter50");
        this.helpButton = new Button(buttonTextureHelp, buttonFont, "");
        this.helpButton.setPosition(x, y);
        this.helpButton.setType(ButtonType.HELP);

        y += offsetY + buttonHeight;
        this.scoreButton = new Button(buttonTextureScore, buttonFont, "");
        this.scoreButton.setPosition(x, y);
        this.scoreButton.setType(ButtonType.SCORE);

        y += offsetY + buttonHeight;
        this.modeButton = new Button(buttonTextureGames, buttonFont, "");
        this.modeButton.setPosition(x, y);
        this.modeButton.setType(ButtonType.GAMES);

        y += offsetY + buttonHeight;
        this.mainButton = new Button(buttonTextureGame, buttonFont, "");
        this.mainButton.setPosition(x, y);
        this.mainButton.setType(ButtonType.PLAY);

        Vector2 position = Utils.getTopRightCorner(buttonTextureExit);
        this.exitButton = new Button(buttonTextureExit);
        this.exitButton.setPosition(position.x, position.y);
        this.exitButton.setType(ButtonType.EXIT);

        ButtonClickListener buttonClickListener = new ButtonClickListener();

        this.helpButton.addListener(buttonClickListener);
        this.scoreButton.addListener(buttonClickListener);
        this.modeButton.addListener(buttonClickListener);
        this.mainButton.addListener(buttonClickListener);
        this.exitButton.addListener(buttonClickListener);

        this.sound = new Sound(this.feedyourbrain);

        this.stage.addActor(this.sound);
        this.stage.addActor(this.helpButton);
        this.stage.addActor(this.scoreButton);
        this.stage.addActor(this.modeButton);
        this.stage.addActor(this.mainButton);
        this.stage.addActor(this.exitButton);
    }

    private void log() {
        String content = "User: " + MyPreferences.getId();
        this.log.setContent(content, 1.0f);
        this.stage.addActor(this.log);
    }

    /**
     * Function to go games mode
     */
    private void goToLevel() {
        super.feedyourbrain.mainScreen = null;
        super.feedyourbrain.levelScreen = new LevelScreen(super.feedyourbrain);
        super.feedyourbrain.setScreen(super.feedyourbrain.levelScreen);
    }

    /**
     * Function to go play mode
     */
    private void goToPlay() {
        super.feedyourbrain.mainScreen = null;
        MyPreferences.setGamePlay(MyPreferences.GAMEPLAYMAIN);
        super.feedyourbrain.playScreen = new PlayScreen(super.feedyourbrain);
        super.feedyourbrain.setScreen(super.feedyourbrain.playScreen);
    }

    /**
     * Function to go to score screen
     */
    private void goToScore() {

        /**
         * Check internet connection
         */
        if(!this.feedyourbrain.api.checkStatus()){
            if (Utils.langByDefault()) {
                this.message.setContent("To visit the scores you need a connection", 2);
            }else{
                this.message.setContent(
                        "Para ver las puntuaciones necesita estar conectado a internet", 2
                );
            }
            this.stage.addActor(this.message);
        }else{
            Log.log("Internet");
            super.feedyourbrain.mainScreen = null;
            super.feedyourbrain.scoreSelectScreen = new ScoreSelectScreen(
                    super.feedyourbrain);
            super.feedyourbrain.setScreen(super.feedyourbrain.scoreSelectScreen);
        }


    }

    /**
     * Function to go to help screen
     */
    private void goToHelp() {
        super.feedyourbrain.helpScreen = null;
        super.feedyourbrain.helpScreen = new HelpScreen(
                super.feedyourbrain);
        super.feedyourbrain.setScreen(super.feedyourbrain.helpScreen);
    }

    /** Called when a key was pressed
     *
     * @param keycode one of the constants in {@link Input.Keys}
     * @return whether the input was processed */
    @Override
    public boolean keyDown(int keycode) {
        return false;
    }

    /** Called when a key was typed
     *
     * @param character The character
     * @return whether the input was processed */
    @Override
    public boolean keyTyped(char character) {
        // TODO Auto-generated method stub
        return false;
    }

    /** Called when a key was released
     *
     * @param keycode one of the constants in {@link Input.Keys}
     * @return whether the input was processed */
    @Override
    public boolean keyUp(int keycode) {
        if (keycode == Keys.BACK) {
```

```java
            Gdx.app.exit();
            return true;
        }
        return false;
    }

    /** Called when a finger was lifted or a mouse button was released.
     * The button parameter will be {@link Buttons#LEFT} on Android
     * and iOS.
     * @param pointer the pointer for the event.
     * @param button the button
     * @return whether the input was processed */
    @Override
    public boolean touchUp(int screenX, int screenY, int pointer, int button) {
        // TODO Auto-generated method stub
        return false;
    }

    /** Called when a finger or the mouse was dragged.
     * @param pointer the pointer for the event.
     * @return whether the input was processed */
    @Override
    public boolean touchDragged(int screenX, int screenY, int pointer) {
        // TODO Auto-generated method stub
        return false;
    }

    /** Called when the mouse was moved without any buttons being pressed.
     * Will not be called on either Android or iOS.
     * @return whether the input was processed */
    @Override
    public boolean mouseMoved(int screenX, int screenY) {
        // TODO Auto-generated method stub
        return false;
    }

    /** Called when the mouse wheel was scrolled. Will not be called on either Android or iOS.
     * @param amount the scroll amount, -1 or 1 depending on the direction the wheel was scrolled.
     * @return whether the input was processed. */
    @Override
    public boolean scrolled(int amount) {
        // TODO Auto-generated method stub
        return false;
    }

    /** Called when the screen was touched or a mouse button was pressed.
     * The button parameter will be {@link Buttons#LEFT} on
     * Android and iOS.
     * @param screenX The x coordinate, origin is in the upper left corner
     * @param screenY The y coordinate, origin is in the upper left corner
     * @param pointer the pointer for the event.
     * @param button the button
     * @return whether the input was processed */
    @Override
    public boolean touchDown(int screenX, int screenY, int pointer, int button) {
        // TODO Auto-generated method stub
        return false;
    }

    /**
     * Listener for all the level buttons
     */
    private class ButtonClickListener extends ActorGestureListener {
        @Override
        public void touchUp(InputEvent event, float x, float y, int pointer,
                int button) {
            Actor actor = event.getListenerActor();

            if (actor instanceof Button) {
                Button btn = (Button) actor;
            int type = btn.getType();

            if (type == ButtonType.PLAY) {
                goToPlay();
            } else if (type == ButtonType.GAMES) {
                goToLevel();
            } else if (type == ButtonType.SCORE) {
                goToScore();
            } else if (type == ButtonType.HELP) {
                goToHelp();
            } else if (type == ButtonType.EXIT) {
                Gdx.app.exit();
            }

            Texture texture = null;

            if (type == ButtonType.PLAY) {
                texture = feedyourbrain.getTexture("main/button_play",
                        true);
            } else if (type == ButtonType.GAMES) {
                texture = feedyourbrain.getTexture(
                        "main/button_games", true);
            } else if (type == ButtonType.SCORE) {
                texture = feedyourbrain.getTexture(
                        "main/button_score", true);
            } else if (type == ButtonType.HELP) {
                texture = feedyourbrain.getTexture("main/button_help",
                        true);
            } else if (type == ButtonType.EXIT) {
                texture = feedyourbrain
                        .getTexture("main/button_exit_mini");
            }

            if (texture != null) {
                btn.setTexture(texture);
            }

        }
    }

    @Override
    public void touchDown(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();
            Texture texture = null;

            if (type == ButtonType.PLAY) {
                texture = feedyourbrain.getTexture("main/button_play_push",
                        true);
            } else if (type == ButtonType.GAMES) {
                texture = feedyourbrain.getTexture(
                        "main/button_games_push", true);
            } else if (type == ButtonType.SCORE) {
                texture = feedyourbrain.getTexture(
                        "main/button_score_push", true);
            } else if (type == ButtonType.HELP) {
                texture = feedyourbrain.getTexture("main/button_help_push",
                        true);
            } else if (type == ButtonType.EXIT) {
                texture = feedyourbrain
                        .getTexture("main/button_exit_mini_push");
            }

            if (texture != null) {
                btn.setTexture(texture);
            }
            }
        }
    }
}
```

2.38. MathScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import java.util.ArrayList;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.CharacterCalc;
import com.juanmartos.games.feedyourbrain.utils.Formulate;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * MathScreen extends the class GameScreen,
 * create the view for math game and manages methods for controlling the game
 */
public class MathScreen extends GameScreen {

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Actors for this stage
     */
    private ArrayList<Actor> actors;

    /**
     * List of characters for calculators
     */
    private ArrayList<CharacterCalc> charactersCalc;

    /**
     * Formulate
     */
    private Formulate formulate;

    /**
     * Number hit
     */
    private int nHit;

    /**
     * Number streak
     */
    private int streak;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                  General handler of the application
     */
```

```
public MathScreen(FeedYourBrain fyb, int type) {

    super(fyb, type);

    this.nHit = 0;
    this.streak = 0;

    this.stage = new Stage(new FitViewport(Var.width, Var.height));
    this.actors = new ArrayList<Actor>();

    super.loadInputProcessor(this.stage);

    /**
     * Catch the back key
     */
    Gdx.input.setCatchBackKey(true);

    /**
     * Catch the menu key
     */
    Gdx.input.setCatchMenuKey(true);

    /**
     * Load defaults views
     */
    super.loadView();

    this.charactersCalc = new ArrayList<CharacterCalc>();
    this.formulate = new Formulate(super.feedyourbrain, this.stage);

    this.loadCustomView();
    if (Utils.langByDefault()) {
        super.message.setContent("Make operations", 2.0f);
    } else {
        super.message.setContent("Realiza las operaciones", 2.0f);
    }

    /**
     * if type 0 -> start
     */
    if (type == 0) {
        this.start();
    }

    /**
     * Send the type of game
     */
    super.gameStage.getScore().setClassName(
            FeedYourBrain.Screens.MATHSCREEN);
}

/**
 * Function to load custom view
 */
private void loadCustomView() {
    float width = 300.0f;
    float height = 200.0f;

    int separation = 10;

    float offsetX = (Var.width - width * 4 + separation * 3) / 2;
    float offsetY = Var.height * 0.01f;

    float x = offsetX;
    float y = offsetY;

    CharacterCalc characterCalc1 = new CharacterCalc(
            super.feedyourbrain.getTexture("calc/1_n"), new Vector2(x, y),
            '1');
    x = x + width + separation;
```

```
        CharacterCalc characterCalc2 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/2_n"), new Vector2(x, y),
                '2');
        x = x + width + separation;
        CharacterCalc characterCalc3 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/3_n"), new Vector2(x, y),
                '3');
        x = x + width + separation;
        CharacterCalc characterCalcC = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/c_n"), new Vector2(x, y),
                'c');
        x = offsetX;
        y = y + height + separation;
        CharacterCalc characterCalc4 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/4_n"), new Vector2(x, y),
                '4');
        x = x + width + separation;
        CharacterCalc characterCalc5 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/5_n"), new Vector2(x, y),
                '5');
        x = x + width + separation;
        CharacterCalc characterCalc6 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/6_n"), new Vector2(x, y),
                '6');
        x = offsetX;
        y = y + height + separation;
        CharacterCalc characterCalc7 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/7_n"), new Vector2(x, y),
                '7');
        x = x + width + separation;
        CharacterCalc characterCalc8 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/8_n"), new Vector2(x, y),
                '8');
        x = x + width + separation;
        CharacterCalc characterCalc9 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/9_n"), new Vector2(x, y),
                '9');
        x = x + width + separation;
        CharacterCalc characterCalc0 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/0_n"), new Vector2(x, y),
                '0');

        this.charactersCalc.add(characterCalc0);
        this.charactersCalc.add(characterCalcC);
        this.charactersCalc.add(characterCalc1);
        this.charactersCalc.add(characterCalc2);
        this.charactersCalc.add(characterCalc3);
        this.charactersCalc.add(characterCalc4);
        this.charactersCalc.add(characterCalc5);
        this.charactersCalc.add(characterCalc6);
        this.charactersCalc.add(characterCalc7);
        this.charactersCalc.add(characterCalc8);
        this.charactersCalc.add(characterCalc9);

        /**
         * Implements listener
         */
        CharacterClickListener characterClickListener = new CharacterClickListener();

        for (CharacterCalc characterCalc : this.charactersCalc) {
            characterCalc.addListener(characterClickListener);
        }
    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
                Var.blueBackground, 1);
```

```java
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    super.render(delta);
    if (this.onGame(delta)) {

        this.gameStage.draw();
        this.stage.draw();

        int result = this.formulate.checkResult();

        if (result == 1) {
            this.nHit++;
            this.streak++;

            this.increase();

            /**
             *  Only if we are in the main gameplay update the difficulty
             **/
            if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
                if (this.nHit == 10) {
                    if (Utils.langByDefault()) {
                        this.message.setContent("Difficulty 2", 1.5f);
                    } else {
                        this.message.setContent("Dificultad 2", 1.5f);
                    }
                    super.gameStage.getScore().setLevel(2);
                    this.formulate.setDifficulty(2);
                } else if (this.nHit == 20) {
                    if (Utils.langByDefault()) {
                        this.message.setContent("Difficulty 3", 1.5f);
                    } else {
                        this.message.setContent("Dificultad 3", 1.5f);
                    }
                    super.gameStage.getScore().setLevel(3);
                    this.formulate.setDifficulty(3);
                }
            }

            this.formulate.generateFormulate();

        } else if (result == 2) {
            this.noIncrease();
            this.streak = 0;
            this.formulate.generateFormulate();
        }

        /**
         * Send streak
         */
        super.gameStage.getStreak().update(this.streak);
        super.gameStage.getScore().setStreak(
                super.gameStage.getStreak().getStreak());
    }

    delta = Math.min(0.06f, delta);

}

/**
 * Punctuation function to restore based on the gameplay we're playing
 */
private void setDifficulty() {
    if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
        super.gameStage.getScore().setLevel(1);
        this.formulate.setDifficulty(1);
    } else if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMODE) {
        super.gameStage.getScore().setLevel(MyPreferences.getDifficulty());
        this.formulate.setDifficulty(MyPreferences.getDifficulty());
    }
```

```
}

@Override
public void start() {
    super.start();

    /**
     * Draw the screen game
     */
    for (CharacterCalc characterCalc : this.charactersCalc) {
        this.stage.addActor(characterCalc);
    }

    this.setDifficulty();

    this.formulate.generateFormulate();

}

/** Called when the application is paused.
 * An Application is paused before it is destroyed, when a user pressed the Home
 * button on Android or an incoming call happened.
 * On the desktop this will only be called immediately before dispose()
 * is called. */
@Override
public void pause() {

    //Guardamos la pantalla en el estado actual
    for (Actor actor : this.stage.getActors())
        this.actors.add(actor);

    super.pause();

}

/** Called when the {@link Application} is resumed from a paused state.
 * On Android this happens when the activity gets focus
 * again. On the desktop this method will never be called. */
@Override
public void resume() {
    super.resume();

    for (Actor actor : this.actors)
        this.stage.addActor(actor);

    this.actors.clear();
}

@Override
public void restart() {
    super.restart();

    /**
     * Clear
     */
    this.actors.clear();

    /**
     * Draw game screen
     */
    for (CharacterCalc characterCalc : this.charactersCalc) {
        this.stage.addActor(characterCalc);
    }

    this.setDifficulty();

    this.formulate.generateFormulate();

    if (Utils.langByDefault()) {
        super.message.setContent("Make operations", 2.0f);
        } else {
            super.message.setContent("Realiza las operaciones", 2.0f);
        }

        this.streak = 0;
        this.nHit = 0;
    }

    @Override
    public void exit() {
        super.exit();

        this.feedyourbrain.exitScreen(FeedYourBrain.Screens.MATHSCREEN,
                getType());
    }

    /**
     * Listener for all the level buttons.
     */
    private class CharacterClickListener extends ActorGestureListener {
        @Override
        public void touchUp(InputEvent event, float x, float y, int pointer,
                int button) {

            Actor actor = event.getListenerActor();
            CharacterCalc characterCalc = (CharacterCalc) actor;

            if (characterCalc.getCharacter() == 'c') {
                noIncrease();
                formulate.generateFormulate();
            } else {
                formulate.addCharacter(characterCalc.getCharacter());
            }

            Texture texture = feedyourbrain.getTexture("calc/"
                    + characterCalc.getCharacter() + "_n");
            characterCalc.setTexture(texture);
        }

        @Override
        public void touchDown(InputEvent event, float x, float y, int pointer,
                int button) {
            Actor actor = event.getListenerActor();
            CharacterCalc characterCalc = (CharacterCalc) actor;

            Texture texture = feedyourbrain.getTexture("calc/"
                    + characterCalc.getCharacter() + "_n_push");
            characterCalc.setTexture(texture);

        }
    }

}
```

2.39. MemoryScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import java.util.ArrayList;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.memory.Card;
```

```java
import com.juanmartos.games.feedyourbrain.structures.MemoryStructure;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * MemoryScreen extends the class GameScreen,
 * create the view for memory game and manages methods for controlling the game
 */
public class MemoryScreen extends GameScreen {

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Actors for this stage
     */
    private ArrayList<Actor> actors;

    /**
     * Structure aux for memory game
     */
    private MemoryStructure memoryStructure;

    /**
     * Number of hit
     */
    private int nHit;

    /**
     * Number of streak
     */
    private int streak;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                  General handler of the application
     */
    public MemoryScreen(FeedYourBrain fyb, int type) {

        super(fyb, type);

        this.nHit = 0;
        this.streak = 0;

        this.stage = new Stage(new FitViewport(Var.width, Var.height));
        this.actors = new ArrayList<Actor>();

        CardClickListener cardClickListener = new CardClickListener();

        super.loadInputProcessor(this.stage);

        /**
         * Catch the back key
         */
        Gdx.input.setCatchBackKey(true);

        /**
         * Catch the menu key
         */
        Gdx.input.setCatchMenuKey(true);

        /**
         * Load the default view
         */
        super.loadView();
        /**
         * Load the custom view
         */
        this.loadCustomView();

        if (Utils.langByDefault()) {
            super.message.setContent("Memorize the cards", 2.0f);
        } else {
            super.message.setContent("Memoriza las cartas", 2.0f);
        }

        this.memoryStructure = new MemoryStructure(feedyourbrain, this.stage,
                cardClickListener);
        this.setDifficulty();
        this.memoryStructure.generate();


        /**
         * if type 0 -> start
         */
        if (type == 0) {
            this.start();
        }

        /**
         * Send the type of game
         */
        super.gameStage.getScore().setClassName(
                FeedYourBrain.Screens.MEMORYSCREEN);
    }

    /**
     * Function to load custom view
     */
    private void loadCustomView() {
        // TODO Auto-generated method stub

    }

    /** Called when the screen should render itself.
     * @param delta The time in seconds since the last render. */
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
                Var.blueBackground, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        super.render(delta);
        if (this.onGame(delta)) { //Dentro del juego

            this.gameStage.draw();
            this.stage.draw();

            if (this.memoryStructure.isHead()) {
                this.memoryStructure.updateTime(delta);
            }
            if (this.memoryStructure.isTumble()) {
                this.memoryStructure.updateTimeForTumble(delta);
            }
            if (this.memoryStructure.isChange()) {
                this.memoryStructure.updateTimeForChange(delta);
            }
        }

        delta = Math.min(0.06f, delta);

    }

    /**
     * Funcion para chequear la carta que hemos pulsado
```

```
     * @param card
     */
    private void checkCard(Card card) {

        int check = memoryStructure.checkCard(card);

        if (check == 0) {
            this.noIncrease();
            this.streak = 0;
            this.memoryStructure.tailToHeadAllCards();
            this.memoryStructure.setChange(true);
        } else if (check == 2) {
            this.increase();
            this.streak++;
            this.nHit++;

            /** Only if we are in the main gameplay update the difficulty **/
            if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
                if (this.nHit >= 11) {
                    super.gameStage.getScore().setLevel(3);
                    this.memoryStructure.setDifficulty(3);
                } else if (this.nHit >= 6) {
                    super.gameStage.getScore().setLevel(2);
                    this.memoryStructure.setDifficulty(2);
                }
            }

            this.memoryStructure.setChange(true);
        }

        /**
         * Send streak
         */
        super.gameStage.getStreak().update(this.streak);
        super.gameStage.getScore().setStreak(
                super.gameStage.getStreak().getStreak());
    }

    /**
     * Punctuation function to restore based on the gameplay we're playing
     */
    private void setDifficulty() {
        if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
            super.gameStage.getScore().setLevel(1);
            this.memoryStructure.setDifficulty(1);
        } else if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMODE) {
            super.gameStage.getScore().setLevel(MyPreferences.getDifficulty());
            this.memoryStructure.setDifficulty(MyPreferences.getDifficulty());
        }
    }

    @Override
    public void start() {
        super.start();

        this.setDifficulty();

        this.memoryStructure.loadActors();
    }

    /** Called when the application is paused.
     * An Application is paused before it is destroyed, when a user pressed the Home
     * button on Android or an incoming call happened.
     * On the desktop this will only be called immediately before dispose()
     * is called. */
    @Override
    public void pause() {

        //Guardamos la pantalla en el estado actual
        for (Actor actor : this.stage.getActors())
```

```
            this.actors.add(actor);

        super.pause();

    }

    /** Called when the {@link Application} is resumed from a paused state.
     * On Android this happens when the activity gets focus
     * again. On the desktop this method will never be called. */
    @Override
    public void resume() {
        super.resume();

        for (Actor actor : this.actors)
            this.stage.addActor(actor);

        this.actors.clear();
    }

    @Override
    public void restart() {
        super.restart();

        /**
         * Clear
         */
        this.actors.clear();

        this.setDifficulty();
        this.memoryStructure.setChange(true);

        if (Utils.langByDefault()) {
            super.message.setContent("Memorize the cards", 2.0f);
        } else {
            super.message.setContent("Memoriza las cartas", 2.0f);
        }

        this.streak = 0;
        this.nHit = 0;
    }

    @Override
    public void exit() {
        super.exit();

        this.feedyourbrain.exitScreen(FeedYourBrain.Screens.MEMORYSCREEN,
                getType());
    }

    /**
     * Listener for all the level buttons.
     */
    private class CardClickListener extends ActorGestureListener {
        @Override
        public void touchUp(InputEvent event, float x, float y, int pointer,
                int button) {

            Actor actor = event.getListenerActor();
            Card card = (Card) actor;
            checkCard(card);

        }
    }

}
```

2.40. ModeScreen

```
package com.juanmartos.games.feedyourbrain.screen;
```

```
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.Input.Buttons;
import com.badlogic.gdx.Input.Keys;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.Level;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * ModeScreen extends the class AbstractScreen,
 * implements the interface InputProcessor and create the view for mode screen
 */
public class ModeScreen extends AbstractScreen implements InputProcessor {
    /**
     * Button to access Logic Game
     */
    private Button logicButton;

    /**
     * Button to access Visual Game
     */
    private Button visualButton;

    /**
     * Button to access Math Game
     */
    private Button mathButton;

    /**
     * Button to access Memory Game
     */
    private Button memoryButton;

    /**
     * Button to access Sequence Game
     */
    private Button sequenceButton;

    /**
     * Button to access Weight Game
     */
    private Button weightButton;

    /**
     * Button to back
     */
    private Button backButton;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                   General handler of the application
     */
    public ModeScreen(FeedYourBrain feedyourbrain) {
        super(feedyourbrain);

        /**
         * Catch the back key
         */
        Gdx.input.setCatchBackKey(true);

        this.stage = new Stage(new FitViewport(Var.width, Var.height));

        this.loadInputProcessor();

        this.loadView();
    }

    /**
     * Function to load the input processor
     */
    private void loadInputProcessor() {
        InputMultiplexer multiplexer = new InputMultiplexer();
        multiplexer.addProcessor(this);
        multiplexer.addProcessor(this.stage);

        Gdx.input.setInputProcessor(multiplexer);

    }

    /**
     * Function to load views
     */
    private void loadView() {

        /**
         * Width and height screen
         */
        int width = Var.width;
        int height = (int)(Var.height * 0.8);

        /**
         * Width and height button
         */
        float buttonWidth = 800;
        float buttonHeight = 133.33f;

        /**
         * Get the offset x
         */
        float offsetX = (width - buttonWidth * 2) / 3;

        /**
         * Get the offset y
         */
        float offsetY = (height - buttonHeight * 3) / 4;

        float x = offsetX;
        float y = (float)(offsetY + (Var.height * 0.2)/2);

        Texture buttonTextureMath = this.feedyourbrain.getTexture(
                "main/button_math", true);
        Texture buttonTextureMemory = this.feedyourbrain.getTexture(
                "main/button_memory", true);
        Texture buttonTextureLogic = this.feedyourbrain.getTexture(
                "main/button_logic", true);
        Texture buttonTextureVisual = this.feedyourbrain.getTexture(
                "main/button_visual", true);
        Texture buttonTextureSequence = this.feedyourbrain.getTexture(
                "main/button_sequence", true);
        Texture buttonTextureWeight = this.feedyourbrain.getTexture(
                    "main/button_weight", true);
            Texture buttonTextureBack = this.feedyourbrain
                    .getTexture("main/button_back_mini");
            BitmapFont buttonFont = this.feedyourbrain
                    .getFont("bitstreamcharter50");

            this.visualButton = new Button(buttonTextureVisual, buttonFont, "");
            this.visualButton.setPosition(x, y);
            this.visualButton.setType(ButtonType.VISUAL);

            x += offsetX + buttonWidth;
            this.logicButton = new Button(buttonTextureLogic, buttonFont, "");
            this.logicButton.setPosition(x, y);
            this.logicButton.setType(ButtonType.LOGIC);

            x = offsetX;
            y += offsetY + buttonHeight;
            this.mathButton = new Button(buttonTextureMath, buttonFont, "");
            this.mathButton.setPosition(x, y);
            this.mathButton.setType(ButtonType.MATH);

            x += offsetX + buttonWidth;
            this.memoryButton = new Button(buttonTextureMemory, buttonFont, "");
            this.memoryButton.setPosition(x, y);
            this.memoryButton.setType(ButtonType.MEMORY);

            x = offsetX;
            y += offsetY + buttonHeight;
            this.sequenceButton = new Button(buttonTextureSequence, buttonFont, "");
            this.sequenceButton.setPosition(x, y);
            this.sequenceButton.setType(ButtonType.SEQUENCE);

            x += offsetX + buttonWidth;
            this.weightButton = new Button(buttonTextureWeight, buttonFont, "");
            this.weightButton.setPosition(x, y);
            this.weightButton.setType(ButtonType.WEIGHT);

            Vector2 position = Utils.getTopRightCorner(buttonTextureBack);
            this.backButton = new Button(buttonTextureBack);
            this.backButton.setPosition(position.x, position.y);
            this.backButton.setType(ButtonType.EXIT);

            /**
             * Add image level
             */
            Level level = new Level(this.feedyourbrain);

            /**
             * Create listener for buttons
             */
            ButtonClickListener buttonClickListener = new ButtonClickListener();

            this.visualButton.addListener(buttonClickListener);
            this.logicButton.addListener(buttonClickListener);
            this.mathButton.addListener(buttonClickListener);
            this.memoryButton.addListener(buttonClickListener);
            this.sequenceButton.addListener(buttonClickListener);
            this.weightButton.addListener(buttonClickListener);
            this.backButton.addListener(buttonClickListener);

            this.stage.addActor(this.visualButton);
            this.stage.addActor(this.logicButton);
            this.stage.addActor(this.mathButton);
            this.stage.addActor(this.memoryButton);
            this.stage.addActor(this.sequenceButton);
            this.stage.addActor(this.weightButton);
            this.stage.addActor(this.backButton);
            this.stage.addActor(level);

    }
```

```java
/** Called when the screen should render itself.
 * @param delta The time in seconds since the last render. */
@Override
public void render(float delta) {
    Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
            Var.blueBackground, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    super.render(delta);
    this.stage.draw();
}

/**
 * Function to go to Visual Game
 */
private void goToVisual() {
    super.feedyourbrain.visualScreen = new VisualScreen(
            super.feedyourbrain, 1);
    super.feedyourbrain.setScreen(super.feedyourbrain.visualScreen);
}

/**
 * Function to go to Logic Game
 */
private void goToLogic() {
    super.feedyourbrain.associationScreen = new AssociationScreen(
            super.feedyourbrain, 1);
    super.feedyourbrain.setScreen(super.feedyourbrain.associationScreen);
}

/**
 * Function to go to Math Game
 */
private void goToMath() {
    super.feedyourbrain.mathScreen = new MathScreen(super.feedyourbrain, 1);
    super.feedyourbrain.setScreen(super.feedyourbrain.mathScreen);
}

/**
 * Function to go to Memory Game
 */
private void goToMemory() {
    super.feedyourbrain.memoryScreen = new MemoryScreen(
            super.feedyourbrain, 1);
    super.feedyourbrain.setScreen(super.feedyourbrain.memoryScreen);
}

/**
 * Function to go to Sequence Game
 */
private void goToSequence() {
    super.feedyourbrain.sequenceScreen = new SequenceScreen(
            super.feedyourbrain, 1);
    super.feedyourbrain.setScreen(super.feedyourbrain.sequenceScreen);
}

/**
 * Function to go to Weight Game
 */
private void goToWeight() {
    super.feedyourbrain.weightScreen = new WeightScreen(
            super.feedyourbrain, 1);
    super.feedyourbrain.setScreen(super.feedyourbrain.weightScreen);
}

/** Called when a key was pressed
 *
 * @param keycode one of the constants in {@link Input.Keys}
 * @return whether the input was processed */
```

```
@Override
public boolean keyDown(int keycode) {
    return false;
}

/** Called when a key was typed
 *
 * @param character The character
 * @return whether the input was processed */
@Override
public boolean keyTyped(char character) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when a key was released
 *
 * @param keycode one of the constants in {@link Input.Keys}
 * @return whether the input was processed */
@Override
public boolean keyUp(int keycode) {
    if (keycode == Keys.BACK) {
        Gdx.app.exit();
        return true;
    }
    return false;
}

/** Called when a finger was lifted or a mouse button was released.
 * The button parameter will be {@link Buttons#LEFT} on Android
 * and iOS.
 * @param pointer the pointer for the event.
 * @param button the button
 * @return whether the input was processed */
@Override
public boolean touchUp(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when a finger or the mouse was dragged.
 * @param pointer the pointer for the event.
 * @return whether the input was processed */
@Override
public boolean touchDragged(int screenX, int screenY, int pointer) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when the mouse was moved without any buttons being pressed.
 * Will not be called on either Android or iOS.
 * @return whether the input was processed */
@Override
public boolean mouseMoved(int screenX, int screenY) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when the mouse wheel was scrolled. Will not be called on either Android or iOS.
 * @param amount the scroll amount, -1 or 1 depending on the direction the wheel was scrolled.
 * @return whether the input was processed. */
@Override
public boolean scrolled(int amount) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when the screen was touched or a mouse button was pressed.
 * The button parameter will be {@link Buttons#LEFT} on
 * Android and iOS.
 * @param screenX The x coordinate, origin is in the upper left corner
 * @param screenY The y coordinate, origin is in the upper left corner
 * @param pointer the pointer for the event.
 * @param button the button
 * @return whether the input was processed */
@Override
public boolean touchDown(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
}


/**
 * Listener for all the level buttons
 */
private class ButtonClickListener extends ActorGestureListener {
    @Override
    public void touchUp(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();

            if (type == ButtonType.MATH) {
                goToMath();
            } else if (type == ButtonType.LOGIC) {
                goToLogic();
            } else if (type == ButtonType.VISUAL) {
                goToVisual();
            } else if (type == ButtonType.MEMORY) {
                goToMemory();
            }else if (type == ButtonType.SEQUENCE) {
                goToSequence();
            }else if (type == ButtonType.WEIGHT) {
                goToWeight();
            } else if (type == ButtonType.EXIT) {
                feedyourbrain.modeScreen = null;
                feedyourbrain.levelScreen = new LevelScreen(feedyourbrain);
                feedyourbrain.setScreen(feedyourbrain.levelScreen);
            }

        }
    }

    @Override
    public void touchDown(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();
            Texture texture = null;

            if (type == ButtonType.MATH) {
                texture = feedyourbrain.getTexture("main/button_math_push",
                        true);
            } else if (type == ButtonType.LOGIC) {
                texture = feedyourbrain.getTexture(
                        "main/button_logic_push", true);
            } else if (type == ButtonType.VISUAL) {
                texture = feedyourbrain.getTexture(
                        "main/button_visual_push", true);
            } else if (type == ButtonType.MEMORY) {
                texture = feedyourbrain.getTexture(
                        "main/button_memory_push", true);
            }else if (type == ButtonType.SEQUENCE) {
                texture = feedyourbrain.getTexture(
                        "main/button_sequence_push", true);
            } else if (type == ButtonType.WEIGHT) {
                texture = feedyourbrain.getTexture(
                        "main/button_weight_push", true);
            } else if (type == ButtonType.EXIT) {
                texture = feedyourbrain.getTexture("main/button_back_mini_push");
            }

            if (texture != null) {
                btn.setTexture(texture);
            }

        }
    }
  }
}
```

2.41. PlayScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.play.Description;
import com.juanmartos.games.feedyourbrain.stages.Score;
import com.juanmartos.games.feedyourbrain.stages.actors.MaxScore;
import com.juanmartos.games.feedyourbrain.stages.actors.Title;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * PlayScreen extends the class AbstractScreen,
 * implements the interface InputProcessor and create the view for play screen
 */
public class PlayScreen extends AbstractScreen implements InputProcessor {

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Actor description
     */
    private Description description;

    /**
     * Actor title
     */
    private Title title;

    /**
     * Button to start game
     */
    private Button buttonStart;

    /**
     * Button to back
     */
```

```java
private Button buttonBack;

/**
 * Actor maxScore
 */
private MaxScore maxScore;

/**
 * Actor score
 */
private Score scoreRight;

/**
 * Listener buttons
 */
ButtonClickListener buttonClickListener;

/**
 * Constructor parameterized
 * @param feedyourbrain
 *                  General handler of the application
 */
public PlayScreen(FeedYourBrain feedyourbrain) {
    super(feedyourbrain);

    this.stage = new Stage(new FitViewport(Var.width, Var.height));

    this.buttonClickListener = new ButtonClickListener();

    /**
     * Catch the back key
     */
    Gdx.input.setCatchBackKey(true);

    this.loadInputProcessor();

    this.loadView();
}

/**
 * Function to load the input processor
 */
private void loadInputProcessor() {
    InputMultiplexer multiplexer = new InputMultiplexer();
    multiplexer.addProcessor(this);
    multiplexer.addProcessor(this.stage);

    Gdx.input.setInputProcessor(multiplexer);

}

/** Called when the screen should render itself.
 * @param delta The time in seconds since the last render. */
@Override
public void render(float delta) {
    Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
            Var.blueBackground, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    super.render(delta);
    this.stage.draw();
}

/**
 * Function to load views
 */
private void loadView() {

    /**
     * Create description
 */
Texture textureDescription = this.feedyourbrain.getTexture(
        "play/description", true);
this.description = new Description(textureDescription);

/**
 * Create buttons
 */
Texture textureButtonStart = this.feedyourbrain.getTexture(
        "main/button_start", true);
Texture textureButtonBack = this.feedyourbrain.getTexture(
        "main/button_back", true);

/**
 * Create your score
 */
Texture textureMaxScore = this.feedyourbrain.getTexture(
        "end/max_score", true);
this.maxScore = new MaxScore(textureMaxScore);

/**
 * Create title
 */
this.title = new Title(
        this.feedyourbrain.getTexture("play/title", true));

/**
 * Create textures
 */
Texture textures[] = new Texture[10];
for (int i = 0; i <= 9; ++i)
    textures[i] = this.feedyourbrain.getTexture("games/score/" + i);

/**
 * Create score
 */
//TODO: Score
this.scoreRight = new Score(
        this.feedyourbrain,
        textures,
        this.feedyourbrain.getDatabaseFeedYourBrain().getMaxScore()
);
//this.scoreRight = new Score(this.feedyourbrain, textures, 0);
this.scoreRight.setSide(true);

int nButtons = 2;
int x = 0;
int y = 0;
int offsetX = (Var.width - textureButtonBack.getWidth() * nButtons)
        / (nButtons + 1);
int offsetY = (int) (Var.height * 0.10);

x = offsetX;
y = offsetY;

this.buttonStart = new Button(textureButtonStart);
this.buttonStart.setPosition(x, y);
this.buttonStart.setType(ButtonType.START);
this.buttonStart.addListener(this.buttonClickListener);

x = x + textureButtonBack.getWidth() + offsetX;

this.buttonBack = new Button(textureButtonBack);
this.buttonBack.setPosition(x, y);
this.buttonBack.setType(ButtonType.BACK);
this.buttonBack.addListener(this.buttonClickListener);

this.stage.addActor(this.title);
this.stage.addActor(this.description);
this.stage.addActor(this.buttonStart);
```

```java
    this.stage.addActor(this.buttonBack);
    this.stage.addActor(this.scoreRight);
    this.stage.addActor(this.maxScore);

}

/**
 * Function to start game
 */
private void start() {
    this.feedyourbrain.play();
}

/**
 * Function to return to main screen
 */
private void back() {
    this.feedyourbrain.main();
}

/** Called when a key was pressed
 *
 * @param keycode one of the constants in {@link Input.Keys}
 * @return whether the input was processed */
@Override
public boolean keyDown(int keycode) {
    // TODO Auto-generated method stub
    return false;
}

/** Called when a key was released
 *
 * @param keycode one of the constants in {@link Input.Keys}
 * @return whether the input was processed */
@Override
public boolean keyUp(int keycode) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when a key was typed
 *
 * @param character The character
 * @return whether the input was processed */
@Override
public boolean keyTyped(char character) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when the screen was touched or a mouse button was pressed.
 * The button parameter will be {@link Buttons#LEFT} on
 * Android and iOS.
 * @param screenX The x coordinate, origin is in the upper left corner
 * @param screenY The y coordinate, origin is in the upper left corner
 * @param pointer the pointer for the event.
 * @param button the button
 * @return whether the input was processed */
@Override
public boolean touchDown(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when a finger was lifted or a mouse button was released.
 * The button parameter will be {@link Buttons#LEFT} on Android
 * and iOS.
 * @param pointer the pointer for the event.
 * @param button the button
 * @return whether the input was processed */
@Override
public boolean touchUp(int screenX, int screenY, int pointer, int button) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when a finger or the mouse was dragged.
 * @param pointer the pointer for the event.
 * @return whether the input was processed */
@Override
public boolean touchDragged(int screenX, int screenY, int pointer) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when the mouse was moved without any buttons being pressed.
 * Will not be called on either Android or iOS.
 * @return whether the input was processed */
@Override
public boolean mouseMoved(int screenX, int screenY) {
    // TODO Auto-generated method stub
    return false;
}


/** Called when the mouse wheel was scrolled. Will not be called on either Android or iOS.
 * @param amount the scroll amount, -1 or 1 depending on the direction the wheel was scrolled.
 * @return whether the input was processed. */
@Override
public boolean scrolled(int amount) {
    // TODO Auto-generated method stub
    return false;
}

/**
 * Listener for all the level buttons
 */
private class ButtonClickListener extends ActorGestureListener {
    @Override
    public void touchUp(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();

            if (type == ButtonType.START) {
                start();
            } else if (type == ButtonType.BACK) {
                back();
            }

        }
    }

    @Override
    public void touchDown(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();
            Texture texture = null;
```

```
                if (type == ButtonType.START) {
                    texture = feedyourbrain.getTexture(
                            "main/button_start_push", true);
                } else if (type == ButtonType.BACK) {
                    texture = feedyourbrain.getTexture("main/button_back_push",
                            true);
                }

                if (texture != null) {
                    btn.setTexture(texture);
                }

            }
        }
    }
}
```

2.42. ScoreSelectScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputMultiplexer;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * ScoreSelectScreen extends the class AbstractScreen and create the view for score select screen
 */
public class ScoreSelectScreen extends AbstractScreen {
    /**
     * Button to access score of main mode
     */
    private Button mainButton;

    /**
     * Button to access score of games mode
     */
    private Button modeButton;

    /**
     * Button to access to show the achievements
     */
    private Button generalButton;

    /**
     * Button to back
     */
    private Button exitButton;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Constructor parameterized
 * @param feedyourbrain
 *                    General handler of the application
 */
public ScoreSelectScreen(FeedYourBrain feedyourbrain) {
    super(feedyourbrain);

    this.stage = new Stage(new FitViewport(Var.width, Var.height));

    /**
     * Catch the back key
     */
    Gdx.input.setCatchBackKey(true);

    this.loadInputProcessor();

    this.loadView();
}

/**
 * Function to load the input processor
 */
private void loadInputProcessor() {
    InputMultiplexer multiplexer = new InputMultiplexer();
    multiplexer.addProcessor(this.stage);

    Gdx.input.setInputProcessor(multiplexer);

}

/** Called when the screen should render itself.
 * @param delta The time in seconds since the last render. */
@Override
public void render(float delta) {
    Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
            Var.blueBackground, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    super.render(delta);
    this.stage.draw();
}

/**
 * Function to load views
 */
private void loadView() {

    /**
     * Width and height screen
     */
    int width = Var.width;
    int height = Var.height;

    /**
     * Width and height button
     */
    float buttonWidth = 800;
    float buttonHeight = 133.33f;

    /**
     * Get the offset x
     */
    float offsetX = (width - buttonWidth) / 2;

    /**
     * Get the offset y
     */
    float offsetY = (height - buttonHeight * 3) / 4;

    float x = offsetX;
    float y = offsetY;
    Texture buttonTextureGames = this.feedyourbrain.getTexture(
            "main/button_games", true);
    Texture buttonTextureGame = this.feedyourbrain.getTexture(
            "main/button_play", true);
    Texture buttonTextureGeneral = this.feedyourbrain.getTexture(
            "main/button_general", true);
    Texture buttonTextureExit = this.feedyourbrain
            .getTexture("main/button_back_mini");

    this.generalButton = new Button(buttonTextureGeneral);
    this.generalButton.setPosition(x, y);
    this.generalButton.setType(ButtonType.GENERAL);

    y += offsetY + buttonHeight;
    this.modeButton = new Button(buttonTextureGames);
    this.modeButton.setPosition(x, y);
    this.modeButton.setType(ButtonType.GAMES);

    y += offsetY + buttonHeight;
    this.mainButton = new Button(buttonTextureGame);
    this.mainButton.setPosition(x, y);
    this.mainButton.setType(ButtonType.PLAY);

    Vector2 position = Utils.getTopRightCorner(buttonTextureExit);
    this.exitButton = new Button(buttonTextureExit);
    this.exitButton.setPosition(position.x, position.y);
    this.exitButton.setType(ButtonType.BACK);

    ButtonClickListener buttonClickListener = new ButtonClickListener();

    this.generalButton.addListener(buttonClickListener);
    this.modeButton.addListener(buttonClickListener);
    this.mainButton.addListener(buttonClickListener);
    this.exitButton.addListener(buttonClickListener);

    this.stage.addActor(this.generalButton);
    this.stage.addActor(this.modeButton);
    this.stage.addActor(this.mainButton);
    this.stage.addActor(this.exitButton);
}

/**
 * Listener for all the level buttons
 */
private class ButtonClickListener extends ActorGestureListener {
    @Override
    public void touchUp(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();
            Texture texture = null;

            if (type == ButtonType.PLAY) {
                texture = feedyourbrain.getTexture("main/button_play", true);
                feedyourbrain.leaderBoardGameScreen
                        = new LeaderBoardGameScreen(feedyourbrain, "", "", true);
                feedyourbrain.setScreen(feedyourbrain.leaderBoardGameScreen);
            } else if (type == ButtonType.GAMES) {
                feedyourbrain.scoreSelectScreen = null;
                feedyourbrain.leaderBoardScreen = new LeaderBoardScreen(
                        feedyourbrain);
                feedyourbrain.setScreen(feedyourbrain.leaderBoardScreen);
                texture = feedyourbrain.getTexture("main/button_games",
                        true);
            } else if (type == ButtonType.GENERAL) {
                feedyourbrain.scoreSelectScreen = null;
                feedyourbrain.achievementScreen = new AchievementScreen(
                        feedyourbrain);
                feedyourbrain.setScreen(feedyourbrain.achievementScreen);
                texture = feedyourbrain.getTexture("main/button_general",
                        true);
            } else if (type == ButtonType.BACK) {
                feedyourbrain.scoreSelectScreen = null;
                feedyourbrain.main();
                texture = feedyourbrain.getTexture("main/button_back_mini");
            }

            if (texture != null) {
                btn.setTexture(texture);
            }
        }
    }

    @Override
    public void touchDown(InputEvent event, float x, float y, int pointer,
            int button) {
        Actor actor = event.getListenerActor();

        if (actor instanceof Button) {
            Button btn = (Button) actor;
            int type = btn.getType();
            Texture texture = null;

            if (type == ButtonType.PLAY) {
                texture = feedyourbrain.getTexture("main/button_play_push", true);
            } else if (type == ButtonType.GAMES) {
                texture = feedyourbrain.getTexture(
                        "main/button_games_push", true);
            } else if (type == ButtonType.GENERAL) {
                texture = feedyourbrain.getTexture(
                        "main/button_general_push", true);
            } else if (type == ButtonType.BACK) {
                texture = feedyourbrain
                        .getTexture("main/button_back_mini_push");
            }

            if (texture != null) {
                btn.setTexture(texture);
            }

        }
    }
  }
}
```

2.43. ScoreSelectScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.Touchable;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.CharacterCalc;
import com.juanmartos.games.feedyourbrain.utils.Sequence;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;
```

```java
import java.util.ArrayList;

public class SequenceScreen extends GameScreen{

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Actors for this stage
     */
    private ArrayList<Actor> actors;

    /**
     * List of characters for calculators
     */
    private ArrayList<CharacterCalc> charactersCalc;

    /**
     * Formulate
     */
    private Sequence sequence;

    /**
     * Number hit
     */
    private int nHit;

    /**
     * Number streak
     */
    private int streak;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                  General handler of the application
     */
    public SequenceScreen(FeedYourBrain fyb, int type) {

        super(fyb, type);

        this.nHit = 0;
        this.streak = 0;

        this.stage = new Stage(new FitViewport(Var.width, Var.height));
        this.actors = new ArrayList<Actor>();

        super.loadInputProcessor(this.stage);

        /**
         * Catch the back key
         */
        Gdx.input.setCatchBackKey(true);

        /**
         * Catch the menu key
         */
        Gdx.input.setCatchMenuKey(true);

        /**
         * Load defaults views
         */
        super.loadView();

        this.charactersCalc = new ArrayList<CharacterCalc>();
        this.sequence = new Sequence(super.feedyourbrain, this.stage);
        this.loadCustomView();
        if (Utils.langByDefault()) {
            super.message.setContent("Repeat sequence", 2.0f);
        } else {
            super.message.setContent("Repite la secuencia", 2.0f);
        }

        /**
         * if type 0 -> start
         */
        if (type == 0) {
            this.start();
        }

        /**
         * Send the type of game
         */
        super.gameStage.getScore().setClassName(
                FeedYourBrain.Screens.SEQUENCESCREEN);
    }

    /**
     * Function to load custom view
     */
    private void loadCustomView() {

        float width = 300.0f;
        float height = 200.0f;

        int separation = 10;

        float offsetX = (Var.width - width * 3 + separation * 2) / 2;
        float offsetY = Var.height * 0.01f;

        float x = offsetX;
        float y = offsetY;

        CharacterCalc characterCalc1 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/1_n"), new Vector2(x, y),
                '1');
        x = x + width + separation;
        CharacterCalc characterCalc2 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/2_n"), new Vector2(x, y),
                '2');
        x = x + width + separation;
        CharacterCalc characterCalc3 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/3_n"), new Vector2(x, y),
                '3');
        x = offsetX;
        y = y + height + separation;
        CharacterCalc characterCalc4 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/4_n"), new Vector2(x, y),
                '4');
        x = x + width + separation;
        CharacterCalc characterCalc5 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/5_n"), new Vector2(x, y),
                '5');
        x = x + width + separation;
        CharacterCalc characterCalc6 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/6_n"), new Vector2(x, y),
                '6');
        x = offsetX;
        y = y + height + separation;
        CharacterCalc characterCalc7 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/7_n"), new Vector2(x, y),
                '7');
        x = x + width + separation;
        CharacterCalc characterCalc8 = new CharacterCalc(
                super.feedyourbrain.getTexture("calc/8_n"), new Vector2(x, y),
                '8');
    x = x + width + separation;
    CharacterCalc characterCalc9 = new CharacterCalc(
            super.feedyourbrain.getTexture("calc/9_n"), new Vector2(x, y),
            '9');

    this.charactersCalc.add(characterCalc1);
    this.charactersCalc.add(characterCalc2);
    this.charactersCalc.add(characterCalc3);
    this.charactersCalc.add(characterCalc4);
    this.charactersCalc.add(characterCalc5);
    this.charactersCalc.add(characterCalc6);
    this.charactersCalc.add(characterCalc7);
    this.charactersCalc.add(characterCalc8);
    this.charactersCalc.add(characterCalc9);

    /**
     * Implements listener
     */
    SequenceScreen.CharacterClickListener characterClickListener
            = new SequenceScreen.CharacterClickListener();

    for (CharacterCalc characterCalc : this.charactersCalc) {
        characterCalc.setTouchable(Touchable.disabled);
        characterCalc.setVisible(false);
        characterCalc.addListener(characterClickListener);
    }
}

/** Called when the screen should render itself.
 * @param delta The time in seconds since the last render. */
@Override
public void render(float delta) {
    Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
            Var.blueBackground, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    super.render(delta);
    if (this.onGame(delta)) {

        this.gameStage.draw();
        this.stage.draw();

        //Update time sequence
        if (this.sequence.getState() == 0){
            this.sequence.updateTime(delta);
        }

        int result = this.sequence.checkResult();

        if (result == 1) {
            this.nHit++;
            this.streak++;

            this.increase();

            /**
             *  Only if we are in the main gameplay update the difficulty
             **/
            if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
                if (this.nHit == 10) {
                    if (Utils.langByDefault()) {
                        this.message.setContent("Difficulty 2", 1.5f);
                    } else {
                        this.message.setContent("Dificultad 2", 1.5f);
                    }
                    super.gameStage.getScore().setLevel(2);
                    this.sequence.setDifficulty(2);
                } else if (this.nHit == 20) {
                    if (Utils.langByDefault()) {
                        this.message.setContent("Difficulty 3", 1.5f);
                    } else {
                        this.message.setContent("Dificultad 3", 1.5f);
                    }
                    super.gameStage.getScore().setLevel(3);
                    this.sequence.setDifficulty(3);
                }
            }

            this.sequence.generate();

        } else if (result == 2) {
            this.noIncrease();
            this.streak = 0;
            this.sequence.generate();
        }

        if(sequence.getState() == 0){
            for(CharacterCalc characterCalc:this.charactersCalc){
                characterCalc.setTouchable(Touchable.disabled);
                characterCalc.setVisible(false);
            }
        }else if (sequence.getState() == 1){
            for(CharacterCalc characterCalc:this.charactersCalc){
                characterCalc.setTouchable(Touchable.enabled);
                characterCalc.setVisible(true);
            }
        }

        /**
         * Send streak
         */
        super.gameStage.getStreak().update(this.streak);
        super.gameStage.getScore().setStreak(
                super.gameStage.getStreak().getStreak());
    }

    delta = Math.min(0.06f, delta);

}

/**
 * Punctuation function to restore based on the gameplay we're playing
 */
private void setDifficulty() {
    if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
        super.gameStage.getScore().setLevel(1);
        this.sequence.setDifficulty(1);
    } else if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMODE) {
        super.gameStage.getScore().setLevel(MyPreferences.getDifficulty());
        this.sequence.setDifficulty(MyPreferences.getDifficulty());
    }
}

@Override
public void start() {
    super.start();

    /**
     * Draw the screen game
     */

    for (CharacterCalc characterCalc : this.charactersCalc){
        this.stage.addActor(characterCalc);
    }

    this.setDifficulty();

    this.sequence.generate();

}
```

```java
/** Called when the application is paused.
 * An Application is paused before it is destroyed, when a user pressed the Home
 * button on Android or an incoming call happened.
 * On the desktop this will only be called immediately before dispose()
 * is called. */
@Override
public void pause() {

    //Guardamos la pantalla en el estado actual
    for (Actor actor : this.stage.getActors())
        this.actors.add(actor);

    super.pause();

}

/** Called when the {@link Application} is resumed from a paused state.
 * On Android this happens when the activity gets focus
 * again. On the desktop this method will never be called. */
@Override
public void resume() {
    super.resume();

    for (Actor actor : this.actors)
        this.stage.addActor(actor);

    this.actors.clear();
}

@Override
public void restart() {
    super.restart();

    /**
     * Clear
     */
    this.actors.clear();

    /**
     * Draw game screen
     */
    for (CharacterCalc characterCalc : this.charactersCalc) {
        this.stage.addActor(characterCalc);
    }

    this.setDifficulty();

    this.sequence.generate();

    if (Utils.langByDefault()) {
        super.message.setContent("Repeat sequence", 2.0f);
    } else {
        super.message.setContent("Repite la secuencia", 2.0f);
    }

    this.streak = 0;
    this.nHit = 0;
}

@Override
public void exit() {
    super.exit();

    this.feedyourbrain.exitScreen(FeedYourBrain.Screens.SEQUENCESCREEN,
            getType());
}

/**
 * Listener for all the level buttons.
```

```
     */
    private class CharacterClickListener extends ActorGestureListener {
        @Override
        public void touchUp(InputEvent event, float x, float y, int pointer,
                            int button) {

            Actor actor = event.getListenerActor();
            CharacterCalc characterCalc = (CharacterCalc) actor;

            if (characterCalc.getCharacter() == 'c') {
                noIncrease();
                sequence.generate();
            } else {
                sequence.addCharacter(characterCalc.getCharacter());
            }

            Texture texture = feedyourbrain.getTexture("calc/"
                    + characterCalc.getCharacter() + "_n");
            characterCalc.setTexture(texture);
        }

        @Override
        public void touchDown(InputEvent event, float x, float y, int pointer,
                              int button) {
            Actor actor = event.getListenerActor();
            CharacterCalc characterCalc = (CharacterCalc) actor;

            Texture texture = feedyourbrain.getTexture("calc/"
                    + characterCalc.getCharacter() + "_n_push");
            characterCalc.setTexture(texture);

        }
    }
}
```

2.44. VisualScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.Box2DDebugRenderer;
import com.badlogic.gdx.physics.box2d.CircleShape;
import com.badlogic.gdx.physics.box2d.PolygonShape;
import com.badlogic.gdx.physics.box2d.World;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Circle;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Operation;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

import java.util.ArrayList;

/**
 * VisualScreen extends the class GameScreen,
 * create the view for visual game and manages methods for controlling the game
 */
public class VisualScreen extends GameScreen {
```

```
/**
 * Stage
 */
private Stage stage;

/**
 * Actors for this stage
 */
private ArrayList<Actor> actors;

/** Variable para representar el mundo donde dibujamos
    los objetos para aplicarle reglas fisicas **/
/**
 * Variable to represent the world where we draw the objects to apply physical rules
 */
private World world;

/**
 * Virtual world for test
 */
private Box2DDebugRenderer debugRenderer;

/**
 * Camara representing where we see the virtual world
 */
private OrthographicCamera cam;

/**
 * Operation handle for the visual game
 */
private Operation operation;

/**
 * Number hit
 */
private int nHit;

/**
 * Number streak
 */
private int streak;

/**
 * Variable to control test mode
 */
private boolean debugMode;

/**
 * Constructor parameterized
 * @param feedyourbrain
 *                  General handler of the application
 * @param type
 *                  Game type
 */
public VisualScreen(FeedYourBrain fyb, int type) {

    super(fyb, type);

    this.nHit = 0;
    this.streak = 0;

    this.stage = new Stage(new FitViewport(Var.width, Var.height));
    this.actors = new ArrayList<Actor>();

    super.loadInputProcessor(this.stage);

    /**
     * Catch the back key
     */
    Gdx.input.setCatchBackKey(true);
```

```
        /**
         * Catch the menu key
         */
        Gdx.input.setCatchMenuKey(true);

        /**
         * Load default view
         */
        super.loadView();

        this.loadCustomView();

        this.debugMode = false;

        if (Utils.langByDefault()) {
            super.message.setContent("Ordered from major to minor", 2.0f);
        } else {
            super.message.setContent("Ordena de menor a mayor", 2.0f);
        }

        CircleClickListener circleClickListener = new CircleClickListener();

        this.operation = new Operation(super.feedyourbrain, circleClickListener);
        this.operation.setWidth((int) Var.width);
        this.operation.setHeight((int) Var.height);

        /**
         * if type 0 -> start
         */
        if (type == 0) {
            this.start();
        }

        /**
         * Send the type of game
         */
        super.gameStage.getScore().setClassName(
                FeedYourBrain.Screens.VISUALSCREEN);
    }

    /**
     * Function to load custom view
     */
    private void loadCustomView() {
        float width = Var.width;
        float height = Var.height;
        float offsetX = 0;
        float offsetY = 0;

        if (this.debugMode) {
            offsetX = width / 2;
            offsetY = height / 2;
        }

        /**
         * Initialize world
         */
        this.world = new World(new Vector2(0, 0), true);

        /**
         * Initialize test
         */
        this.debugRenderer = new Box2DDebugRenderer();

        /**
         * Initialize cam
         */
        this.cam = new OrthographicCamera(Gdx.graphics.getWidth(),
                Gdx.graphics.getHeight());
```

```
BodyDef downBodyDef;
Body downBody;
PolygonShape downBox;

BodyDef leftBodyDef;
Body leftBody;
PolygonShape leftBox;

BodyDef rightBodyDef;
Body rightBody;
PolygonShape rightBox;

BodyDef upBodyDef;
Body upBody;
PolygonShape upBox;

/**
 * Initialize rigid bodies
 */
downBodyDef = new BodyDef();
leftBodyDef = new BodyDef();
rightBodyDef = new BodyDef();
upBodyDef = new BodyDef();

/**
 * Initialize shapes
 */
downBox = new PolygonShape();
leftBox = new PolygonShape();
rightBox = new PolygonShape();
upBox = new PolygonShape();

downBodyDef.position.set(new Vector2(width / 2 - offsetX, 0 - offsetY));
downBody = this.world.createBody(downBodyDef);

downBox.setAsBox(width / 2, 1.f);
downBody.createFixture(downBox, 0.0f);
downBox.dispose();

leftBodyDef.position
        .set(new Vector2(0 - offsetX, height / 2 - offsetY));
leftBody = this.world.createBody(leftBodyDef);

leftBox.setAsBox(1.0f, height / 2);
leftBody.createFixture(leftBox, 0.0f);
leftBox.dispose();

rightBodyDef.position.set(new Vector2(width - offsetX, height / 2
        - offsetY));
rightBody = this.world.createBody(rightBodyDef);

rightBox.setAsBox(0.0f, height);
rightBody.createFixture(rightBox, 0.0f);
rightBox.dispose();

upBodyDef.position.set(new Vector2(width / 2 - offsetX, height
        - offsetY));
upBody = this.world.createBody(upBodyDef);

upBox.setAsBox(width / 2, 0.0f);
upBody.createFixture(upBox, 0.0f);
upBox.dispose();

//Creamos el cuerpo fisico para el reloj
BodyDef timeBodyDef;
Body timeBody;
CircleShape timeBox;

//Reservamos memoria para los cuerpos fisicos
```

```
    timeBodyDef = new BodyDef();
    //Reservamos memoria para las formas fisicas
    timeBox = new CircleShape();

    timeBodyDef.position.set(super.getTime().getCenterX() - offsetX, super
            .getTime().getCenterY() - offsetY);
    timeBody = this.world.createBody(timeBodyDef);

    timeBox.setRadius(super.getTime().getWidth() / 2);
    timeBody.createFixture(timeBox, 0.0f);
    timeBox.dispose();

    BodyDef scoreBodyDef;
    Body scoreBody;
    PolygonShape scoreBox;


    scoreBodyDef = new BodyDef();
    scoreBox = new PolygonShape();

    scoreBodyDef.position.set(super.getScore().getCenterX() - offsetX,
            super.getScore().getCenterY() - offsetY);
    scoreBody = this.world.createBody(scoreBodyDef);

    scoreBox.setAsBox(super.getScore().getWidth() / 2, super.getScore()
            .getHeight() / 2);
    scoreBody.createFixture(scoreBox, 0.0f);
    scoreBox.dispose();

    BodyDef pauseBodyDef;
    Body pauseBody;
    PolygonShape pauseBox;

    pauseBodyDef = new BodyDef();
    pauseBox = new PolygonShape();

    pauseBodyDef.position.set(super.getPause().getCenterX() - offsetX,
            super.getPause().getCenterY() - offsetY);
    pauseBody = this.world.createBody(pauseBodyDef);

    pauseBox.setAsBox(super.getPause().getWidth() / 2, super.getPause()
            .getHeight() / 2);
    pauseBody.createFixture(pauseBox, 0.0f);
    pauseBox.dispose();
}

/** Called when the screen should render itself.
 * @param delta The time in seconds since the last render. */
@Override
public void render(float delta) {
    Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
            Var.blueBackground, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    super.render(delta);
    if (this.onGame(delta)) { //Dentro del juego

        this.gameStage.draw();
        this.stage.draw();

        this.world.step(1 / 60f, 6, 2);

        if (this.debugMode) {
            this.debugRenderer.render(this.world, this.cam.combined);
        }

    }

    delta = Math.min(0.06f, delta);
```

```java
}

/**
 * Function to check the circle down
 * @param circle
 */
private void checkCircle(Circle circle) {

    int exit = operation.check(circle);

    /**
     * Delete
     */
    if (exit == 0) {
        this.noIncrease();
        this.streak = 0;
        this.operation.clearBodies(this.world);
        for (Actor actor : this.operation.getCircles())
            actor.remove();
        this.actors.clear();
        this.operation.getNumbers().clear();
    } else {
        if (exit == 2) {
            super.increase();
            this.nHit++;
            this.streak++;

            if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
                if (this.nHit == 6) {
                    super.gameStage.getScore().setLevel(2);
                    this.operation.setDifficulty(2);
                    if (Utils.langByDefault()) {
                        this.message.setContent("Difficulty 2", 1.5f);
                    } else {
                        this.message.setContent("Dificultad 2", 1.5f);
                    }
                } else if (this.nHit == 11) {
                    super.gameStage.getScore().setLevel(3);
                    this.operation.setDifficulty(3);
                    if (Utils.langByDefault()) {
                        this.message.setContent("Difficulty 3", 1.5f);
                    } else {
                        this.message.setContent("Dificultad 3", 1.5f);
                    }
                }
            }
        }
        circle.remove();
        world.destroyBody(circle.getBody());
        this.operation.getCircles().remove(circle);
        this.operation.getNumbers().remove(0);
    }

    /** Mandamos la racha **/
    super.gameStage.getStreak().update(this.streak);
    super.gameStage.getScore().setStreak(
            super.gameStage.getStreak().getStreak());

    if (operation.getNumbers().size() == 0) {
        operation.setOperation(this.world);

        //Cargamos nuestros actores
        for (Actor actor : this.operation.getCircles())
            this.stage.addActor(actor);
    }

}

/**
 * Punctuation function to restore based on the gameplay we're playing
 */
private void setDifficulty() {
    if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
        super.gameStage.getScore().setLevel(1);
        this.operation.setDifficulty(1);
    } else if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMODE) {
        super.gameStage.getScore().setLevel(MyPreferences.getDifficulty());
        this.operation.setDifficulty(MyPreferences.getDifficulty());
    }
}

/**
 * Start game
 */
@Override
public void start() {
    super.start();

    this.setDifficulty();

    /**
     * Set the operation after difficulty
     */
    this.operation.setOperation(this.world);

    /**
     * Load actors
     */
    for (Actor actor : this.operation.getCircles())
        this.stage.addActor(actor);
}

/** Called when the application is paused.
 * An Application is paused before it is destroyed, when a user pressed the Home
 * button on Android or an incoming call happened.
 * On the desktop this will only be called immediately before dispose()
 * is called. */
@Override
public void pause() {
    //Guardamos la pantalla en el estado actual
    for (Actor actor : this.stage.getActors())
        this.actors.add(actor);

    super.pause();
}

/** Called when the {@link Application} is resumed from a paused state.
 * On Android this happens when the activity gets focus
 * again. On the desktop this method will never be called. */
@Override
public void resume() {
    super.resume();

    for (Actor actor : this.actors)
        this.stage.addActor(actor);

    this.actors.clear();
}

@Override
public void restart() {
    super.restart();

    for (Actor actor : this.actors)
        actor.remove();

    /**
     * Clear
     */
    this.actors.clear();
```

```
        this.setDifficulty();
        this.operation.setOperation(this.world);
        if (Utils.langByDefault()) {
            super.message.setContent("Ordered from major to minor", 2.0f);
        } else {
            super.message.setContent("Ordena de menor a mayor", 2.0f);
        }

        for (Circle circle : this.operation.getCircles())
            this.stage.addActor(circle);

        this.streak = 0;
        this.nHit = 0;
    }

    @Override
    public void exit() {
        super.exit();

        this.feedyourbrain.exitScreen(FeedYourBrain.Screens.VISUALSCREEN,
                getType());
    }

    /**
     * Listener for all the level buttons.
     */
    private class CircleClickListener extends ActorGestureListener {
        @Override
        public void touchUp(InputEvent event, float x, float y, int pointer,
                int button) {

            Actor actor = event.getListenerActor();
            Circle circle = (Circle) actor;

            checkCircle(circle);

        }
    }

}
```

2.45. VisualScreen

```
package com.juanmartos.games.feedyourbrain.screen;

import java.util.ArrayList;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Weight;
import com.juanmartos.games.feedyourbrain.actors.memory.Card;
import com.juanmartos.games.feedyourbrain.structures.WeightStructure;
import com.juanmartos.games.feedyourbrain.utils.Log;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * WeightScreen extends the class GameScreen,
 * create the view for memory game and manages methods for controlling the game
 */
public class WeightScreen extends GameScreen {
```

```
/**
 * Stage
 */
private Stage stage;

/**
 * Actors for this stage
 */
private ArrayList<Actor> actors;

/**
 * Structure aux for weight game
 */
private WeightStructure weightStructure;

/**
 * Number of hit
 */
private int nHit;

/**
 * Number of streak
 */
private int streak;

/**
 * Constructor parameterized
 * @param FeedYourBrain feedyourbrain
 *                  General handler of the application
 */
public WeightScreen(FeedYourBrain fyb, int type) {

    super(fyb, type);

    this.nHit = 0;
    this.streak = 0;

    this.stage = new Stage(new FitViewport(Var.width, Var.height));
    this.actors = new ArrayList<Actor>();

    WeightClickListener weightClickListener = new WeightClickListener();

    super.loadInputProcessor(this.stage);

    /**
     * Catch the back key
     */
    Gdx.input.setCatchBackKey(true);

    /**
     * Catch the menu key
     */
    Gdx.input.setCatchMenuKey(true);

    /**
     * Load the default view
     */
    super.loadView();

    /**
     * Load the custom view
     */
    this.loadCustomView();

    if (Utils.langByDefault()) {
        super.message.setContent("Select the heaviest", 2.0f);
    } else {
        super.message.setContent("Selecciona el mas pesado", 2.0f);
    }
    this.weightStructure = new WeightStructure(feedyourbrain, this.stage,
            weightClickListener);
    this.setDifficulty();
    this.weightStructure.generate();


    /**
     * if type 0 -> start
     */
    if (type == 0) {
        this.start();
    }

    /**
     * Send the type of game
     */
    super.gameStage.getScore().setClassName(
            FeedYourBrain.Screens.MEMORYSCREEN);
}

/**
 * Function to load custom view
 */
private void loadCustomView() {
    // TODO Auto-generated method stub

}

/** Called when the screen should render itself.
 * @param delta The time in seconds since the last render. */
@Override
public void render(float delta) {
    Gdx.gl.glClearColor(Var.redBackground, Var.greenBackground,
            Var.blueBackground, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    super.render(delta);
    if (this.onGame(delta)) { //Dentro del juego

        this.gameStage.draw();
        this.stage.draw();
    }

    delta = Math.min(0.06f, delta);

}

/**
 * Punctuation function to restore based on the gameplay we're playing
 */
private void setDifficulty() {
    if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
        super.gameStage.getScore().setLevel(1);
        this.weightStructure.setDifficulty(1);
    } else if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMODE) {
        super.gameStage.getScore().setLevel(MyPreferences.getDifficulty());
        this.weightStructure.setDifficulty(MyPreferences.getDifficulty());
    }
}

private void result(boolean result){

    if(result){
        this.increase();
        this.streak++;
        this.nHit++;

        /** Only if we are in the main gameplay update the difficulty **/
        if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
            if (this.nHit >= 11) {
                super.gameStage.getScore().setLevel(3);
                this.weightStructure.setDifficulty(3);
            } else if (this.nHit >= 6) {
                super.gameStage.getScore().setLevel(2);
                this.weightStructure.setDifficulty(2);
            }
        }
    }else{
        this.noIncrease();
        this.streak = 0;
    }

    this.weightStructure.generate();
    this.weightStructure.loadActors();
}

@Override
public void start() {
    super.start();

    this.setDifficulty();

    this.weightStructure.loadActors();
}

/** Called when the application is paused.
 * An Application is paused before it is destroyed, when a user pressed the Home
 * button on Android or an incoming call happened.
 * On the desktop this will only be called immediately before dispose()
 * is called. */
@Override
public void pause() {

    //Guardamos la pantalla en el estado actual
    for (Actor actor : this.stage.getActors())
        this.actors.add(actor);

    super.pause();

}

/** Called when the {@link Application} is resumed from a paused state.
 * On Android this happens when the activity gets focus
 * again. On the desktop this method will never be called. */
@Override
public void resume() {
    super.resume();

    for (Actor actor : this.actors)
        this.stage.addActor(actor);

    this.actors.clear();
}

@Override
public void restart() {
    super.restart();

    /**
     * Clear
     */
    this.actors.clear();

    this.setDifficulty();

    this.weightStructure.generate();
    this.weightStructure.loadActors();

    if (Utils.langByDefault()) {
            super.message.setContent("Selecciona el mas pesado", 2.0f);
        } else {
            super.message.setContent("Select the heaviest", 2.0f);
        }

        this.streak = 0;
        this.nHit = 0;
    }

    @Override
    public void exit() {
        super.exit();

        this.feedyourbrain.exitScreen(FeedYourBrain.Screens.WEIGHTSCREEN,
                getType());
    }

    /**
     * Listener for all the level buttons.
     */
    private class WeightClickListener extends ActorGestureListener {
        @Override
        public void touchUp(InputEvent event, float x, float y, int pointer,
                            int button) {

            Actor actor = event.getListenerActor();
            Weight weight = (Weight) actor;

            result(weightStructure.checkWeight(weight));
        }
    }

}
```

2.46. MaxScore

```
package com.juanmartos.games.feedyourbrain.stages.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * MaxScore extends the class Actor and create the view for max score
 */
public class MaxScore extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor StartGameTitle
     * @param texture
     *                 Texture a dibujar
     */
    public MaxScore(Texture texture) {
        this.texture = texture;

        this.loadPosition();
    }

    /**
     * Function to load position of this actor
     */
    private void loadPosition() {
        int width = Var.width / 2;
        int height = Var.height;

        int x = 0;
        int y = 0;

        x = width + (width - texture.getWidth()) / 2;
        y = (int) ((height * 0.6f));

        this.setPosition(x, y);

    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }
}
```

2.47. StartGameBackground

```
package com.juanmartos.games.feedyourbrain.stages.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;

/**
 * StartGameBackground extends the class Actor and create the view for start game
 */
public class StartGameBackground extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor StartGameBackground
     * @param texture
     *                  Texture a dibujar
     */
    public StartGameBackground(Texture texture) {
        this.texture = texture;
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }
}
```

2.48. StartGameDescription

```
package com.juanmartos.games.feedyourbrain.stages.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;

/**
 * StartGameDescription extends the class Actor and create the view for description in game
 */
public class StartGameDescription extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor StartGameDescription
     * @param texture
     *                  Texture a dibujar
     */
    public StartGameDescription(Texture texture) {
        this.texture = texture;
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }
}
```

2.49. StartGameScoreMax

```
package com.juanmartos.games.feedyourbrain.stages.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;

/**
 * StartGameScoreMax extends the class Actor and create the view for score max in start game
 */
public class StartGameScoreMax extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor StartGameScoreMax
     * @param texture
     *                  Texture a dibujar
     */
    public StartGameScoreMax(Texture texture) {
        this.texture = texture;
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }
}
```

2.50. StartGameTitle

```
package com.juanmartos.games.feedyourbrain.stages.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;

/**
 * StartGameTitle extends the class Actor and create the view for title in game
 */
public class StartGameTitle extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor StartGameTitle
     * @param texture
     *                  Texture a dibujar
     */
    public StartGameTitle(Texture texture) {
        this.texture = texture;
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }
}
```

2.51. Title

```
package com.juanmartos.games.feedyourbrain.stages.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Title extends the class Actor and create the view for title
 */
public class Title extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor StartGameTitle
     * @param texture
     *                  Texture a dibujar
     */
    public Title(Texture texture) {
        this.texture = texture;

        this.loadPosition();
    }

    /**
     * Function to load position of this actor
     */
    private void loadPosition() {

        int width = Var.width;
        int height = Var.height;

        int x = 0;
        int y = 0;

        x = (width - texture.getWidth()) / 2;
        y = (int) ((height * 0.75f) + (height * 0.25f - texture.getHeight()) / 2);

        this.setPosition(x, y);

    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }
}
```

2.52. YourScore

```
package com.juanmartos.games.feedyourbrain.stages.actors;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * YourScore extends the class Actor and create the view for your score
 */
public class YourScore extends Actor {

    /**
     * Texture actor
     */
    private Texture texture;

    /**
     * Constructor StartGameTitle
     * @param texture
     *                  Texture a dibujar
     */
    public YourScore(Texture texture) {
        this.texture = texture;

        this.loadPosition();
    }

    /**
     * Function to load position of this actor
     */
    private void loadPosition() {

        int width = Var.width / 2;
        int height = Var.height;

        int x = 0;
        int y = 0;

        x = (width - texture.getWidth()) / 2;
        y = (int) ((height * 0.6f));

        this.setPosition(x, y);

    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {
        batch.draw(this.texture, super.getX(), super.getY());
    }
}
```

2.53. EndStage

```java
package com.juanmartos.games.feedyourbrain.stages;

import com.badlogic.gdx.Net;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.Message;
import com.juanmartos.games.feedyourbrain.stages.actors.StartGameTitle;
import com.juanmartos.games.feedyourbrain.utils.ApiInterface;
import com.juanmartos.games.feedyourbrain.utils.DatabaseFeedYourBrain;
import com.juanmartos.games.feedyourbrain.utils.Log;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Utils;
import com.juanmartos.games.feedyourbrain.utils.Var;

import java.util.ArrayList;

/**
 * EndStage extends the class Stage and create the stage for end screen
 */
public class EndStage extends Stage {

    /**
     * General handler of the application
     */
    private FeedYourBrain fyb;

    /**
     * List of actors
     */
    public ArrayList<Actor> actors;

    /**
     * Listener click actor
     */
    private ActorGestureListener actorGestureListener;

    /**
     * Name children class
     */
    private String nameChildClass;

    /**
     * Actor start title
     */
    private StartGameTitle startGameTitle;

    /**
     * Actor max score title
     */
    private StartGameTitle startGameScoreMax;

    /**
     * Actor your score title
     */
    private StartGameTitle startGameScoreYour;

    /**
     * Button to restart
     */
    private Button buttonRestart;
```

```java
/**
 * Button to back
 */
private Button buttonBack;

/**
 * Score max
 */
private Score scoreMax;

/**
 * Score your
 */
private Score scoreYour;

public Message message;

/**
 * Constructor parameterized
 * @param fyb
 *             General handler of the application
 * @param actorGestureListener
 *             Listener for the actos
 * @param nameChildClass
 *             Name of the children class
 */
public EndStage(FeedYourBrain fyb,
        ActorGestureListener actorGestureListener, String nameChildClass) {
    super(new FitViewport(Var.width, Var.height));

    /**
     * Set FeedYourBrain
     */
    this.fyb = fyb;

    /**
     * Set listener
     */
    this.actorGestureListener = actorGestureListener;

    /**
     * Set name child class
     */
    this.nameChildClass = nameChildClass;

    this.actors = new ArrayList<Actor>();

    this.loadView();
}

/**
 * Function to load view
 */
private void loadView() {

    int width = Var.width;
    int height = Var.height;

    int x = 0;
    int y = 0;
    int offsetX = 0;
    int offsetY = 0;

    Texture textureStartGameTitle = this.fyb.getTexture("games/"
            + this.nameChildClass + "/title", true);
    Texture textureStartGameDescription = this.fyb.getTexture("games/"
            + this.nameChildClass + "/description", true);
    Texture textureStartGameScoreMax = this.fyb.getTexture(
            "games/scoreMax", true);
    Texture textureStartGameScoreYour = this.fyb.getTexture(
        "games/scoreYour", true);
Texture textureButtonBack = this.fyb.getTexture("main/button_back",
        true);
Texture textureButtonRestart = this.fyb.getTexture(
        "main/button_restart", true);
BitmapFont fontButton = this.fyb.getFont("bitstreamcharter50");

x = (width - textureStartGameTitle.getWidth()) / 2;
y = (int) ((height * 0.75f) + (height * 0.25f - textureStartGameTitle
        .getHeight()) / 2);

this.startGameTitle = new StartGameTitle(textureStartGameTitle);
this.startGameTitle.setPosition(x, y);

x = (int) ((width * 0.5f - textureStartGameDescription.getWidth()) / 2);
y = (int) ((height - textureStartGameDescription.getHeight()) / 2);

x = (int) ((width * 0.5f) + (width * 0.5f - textureStartGameScoreMax
        .getWidth()) / 2);
y = (int) (height * 0.60);

this.startGameScoreMax = new StartGameTitle(textureStartGameScoreMax);
this.startGameScoreMax.setPosition(x, y);

x = (int) ((width * 0.5f) - (textureStartGameScoreYour.getWidth())) / 2;
y = (int) (height * 0.60);

this.startGameScoreYour = new StartGameTitle(textureStartGameScoreYour);
this.startGameScoreYour.setPosition(x, y);

int nButtons = 2;
x = 0;
y = 0;
offsetX = (width - textureButtonBack.getWidth() * nButtons)
        / (nButtons + 1);
offsetY = (int) (height * 0.10);

x = offsetX;
y = offsetY;

this.buttonRestart = new Button(textureButtonRestart, fontButton, "");
this.buttonRestart.setPosition(x, y);
this.buttonRestart.setType(ButtonType.RESTART);
this.buttonRestart.addListener(this.actorGestureListener);

x = x + textureButtonBack.getWidth() + offsetX;

this.buttonBack = new Button(textureButtonBack, fontButton, "");
this.buttonBack.setPosition(x, y);
this.buttonBack.setType(ButtonType.BACK);
this.buttonBack.addListener(this.actorGestureListener);

Texture textures[] = new Texture[10];
for (int i = 0; i <= 9; ++i)
    textures[i] = this.fyb.getTexture("games/score/" + i);

this.scoreMax = new Score(this.fyb, textures, this.nameChildClass);
this.actors.add(this.scoreMax);
this.addActor(this.scoreMax);

this.scoreYour = new Score(this.fyb, textures, 0);
this.actors.add(this.scoreYour);
this.addActor(this.scoreYour);

this.actors.add(this.startGameTitle);
this.actors.add(this.startGameScoreMax);
this.actors.add(this.startGameScoreYour);
this.actors.add(this.buttonRestart);
this.actors.add(this.buttonBack);
```

```
    this.message = new Message(
            this.fyb.getTexture("message/message"),
            this.fyb.getFont("bitstreamcharter60"));

    if (Utils.langByDefault()) {
        this.message.setContent("You' have got an achievement!", 2.0f);
    } else {
        this.message.setContent("Has conseguido un logro", 2.0f);
    }
}

/**
 * Function to remove the actors in this stage
 */
public void remove() {
    this.startGameTitle.remove();
    this.startGameScoreMax.remove();
    this.startGameScoreYour.remove();
    this.buttonBack.remove();
    this.buttonRestart.remove();
    this.scoreMax.remove();
    this.scoreYour.remove();
}

/**
 * Function to load actors
 */
public void loadActors() {
    for (Actor actor : this.actors)
        this.addActor(actor);
}

/**
 * Function to set score
 * @param score
 */
public void sendScore(int score) {
    this.scoreYour.setScore(score);
    DatabaseFeedYourBrain database = this.fyb.getDatabaseFeedYourBrain();

    int game = 0;

    int difficulty = MyPreferences.getDifficulty();

    if (this.nameChildClass.equals("math")) {
        game = DatabaseFeedYourBrain.MATH;
        if (score >= 100000 && difficulty == MyPreferences.GAMEPLAYHARD) {
            this.setAchievements(Utils.STREAKMATHHARD);
        } else if (score >= 10000
                && difficulty == MyPreferences.GAMEPLAYNORMAL) {
            this.setAchievements(Utils.STREAKMATHNORMAL);
        } else if (score >= 1000
                && difficulty == MyPreferences.GAMEPLAYEASY) {
            this.setAchievements(Utils.STREAKMATHEASY);
        }
    } else if (this.nameChildClass.equals("logic")) {
        game = DatabaseFeedYourBrain.ASSOCIATION;
        if (score >= 100000 && difficulty == MyPreferences.GAMEPLAYHARD) {
            this.setAchievements(Utils.STREAKASSOCIATIONHARD);
        } else if (score >= 10000
                && difficulty == MyPreferences.GAMEPLAYNORMAL) {
            this.setAchievements(Utils.STREAKASSOCIATIONNORMAL);
        } else if (score >= 1000
                && difficulty == MyPreferences.GAMEPLAYEASY) {
            this.setAchievements(Utils.STREAKASSOCIATIONEASY);
        }
    } else if (this.nameChildClass.equals("visual")) {
        game = DatabaseFeedYourBrain.VISUAL;
        if (score >= 100000 && difficulty == MyPreferences.GAMEPLAYHARD) {
            this.setAchievements(Utils.STREAKVISUALHARD);
    } else if (score >= 10000
            && difficulty == MyPreferences.GAMEPLAYNORMAL) {
        this.setAchievements(Utils.STREAKVISUALNORMAL);
    } else if (score >= 1000
            && difficulty == MyPreferences.GAMEPLAYEASY) {
        this.setAchievements(Utils.STREAKVISUALEASY);
    }
} else if (this.nameChildClass.equals("memory")) {
    game = DatabaseFeedYourBrain.MEMORY;
    if (score >= 100000 && difficulty == MyPreferences.GAMEPLAYHARD) {
        this.setAchievements(Utils.STREAKMEMORYHARD);
    } else if (score >= 10000
            && difficulty == MyPreferences.GAMEPLAYNORMAL) {
        this.setAchievements(Utils.STREAKMEMORYNORMAL);
    } else if (score >= 1000 && difficulty == MyPreferences.GAMEPLAYEASY) {
        this.setAchievements(Utils.STREAKMEMORYEASY);
    }
}else if (this.nameChildClass.equals("sequence")) {
    game = DatabaseFeedYourBrain.SEQUENCE;
    if (score >= 100000 && difficulty == MyPreferences.GAMEPLAYHARD) {
        this.setAchievements(Utils.STREAKSEQUENCEHARD);
    } else if (score >= 10000
            && difficulty == MyPreferences.GAMEPLAYNORMAL) {
        this.setAchievements(Utils.STREAKSEQUENCENORMAL);
    } else if (score >= 1000 && difficulty == MyPreferences.GAMEPLAYEASY) {
        this.setAchievements(Utils.STREAKSEQUENCEEASY);
    }
}else if (this.nameChildClass.equals("weight")) {
    game = DatabaseFeedYourBrain.WEIGHT;
    if (score >= 100000 && difficulty == MyPreferences.GAMEPLAYHARD) {
        this.setAchievements(Utils.STREAKWEIGHTHARD);
    } else if (score >= 10000
            && difficulty == MyPreferences.GAMEPLAYNORMAL) {
        this.setAchievements(Utils.STREAKWEIGHTNORMAL);
    } else if (score >= 1000 && difficulty == MyPreferences.GAMEPLAYEASY) {
        this.setAchievements(Utils.STREAKWEIGHTEASY);
    }
}

/**
 * Set the score to Google Play Games
 */
//String marker = Utils.getMarker(this.nameChildClass, difficulty);
//TODO: GPGS
//this.fyb.actionResolver.submitScoreGPGS(marker, score);

Log.log("Send score");
Log.log(String.valueOf("Game: " + MyPreferences.getGamePlay()));

if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMAIN) {
    fyb.api.createScore(new ApiInterface() {
        @Override
        public void success(Net.HttpResponse response) {

        }

        @Override
        public void failed() {

        }

        @Override
        public void cancelled() {

        }
    }, MyPreferences.getId(), "0", "0", String.valueOf(score));
    database.insertScore(score);
} else if (MyPreferences.getGamePlay() == MyPreferences.GAMEPLAYMODE) {
    fyb.api.createScore(new ApiInterface() {
        @Override
```

```
                public void success(Net.HttpResponse response) {

                }

                @Override
                public void failed() {

                }

                @Override
                public void cancelled() {

                }
            }, MyPreferences.getId(), String.valueOf(game),
                    String.valueOf(MyPreferences.getDifficulty()), String.valueOf(score));
            database.insertScoreMode(score, game, MyPreferences.getDifficulty());
        }

        //TODO: Score
        int maxScore = database.getMaxScoreMode(game, MyPreferences.getDifficulty());
        //int maxScore = 0;
        this.scoreMax.setScore(maxScore);
    }

    /**
     * Function to set the achievements to Google Play Games
     * @param code
     *              Code of the achievements
     */
    public void setAchievements(String code) {
        fyb.api.setAchievements(MyPreferences.getId(), code);
        this.actors.add(this.message);
    }

}
```

2.54. GameStage

```
package com.juanmartos.games.feedyourbrain.stages;

import java.util.ArrayList;

import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.actors.No;
import com.juanmartos.games.feedyourbrain.actors.Ok;
import com.juanmartos.games.feedyourbrain.actors.Pause;
import com.juanmartos.games.feedyourbrain.actors.Score;
import com.juanmartos.games.feedyourbrain.actors.Sound;
import com.juanmartos.games.feedyourbrain.actors.Streak;
import com.juanmartos.games.feedyourbrain.actors.Time;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * GameStage extends the class Stage and create the stage for game screen
 */
public class GameStage extends Stage {

    /**
     * General handler of the application
     */
    private FeedYourBrain fyb;

    /**
     * List of actors
```

```java
 */
public ArrayList<Actor> actors;

/**
 * Listener click actor
 */
private ActorGestureListener actorGestureListener;

/**
 * Variable to show the time
 */
protected Time time;

/**
 * Variable to control the time
 */
public float seconds;

/**
 * Actor pause
 */
protected Pause pause;

/**
 * Actor sound
 */
protected Sound sound;

/**
 * Variable to show the streal
 */
protected Streak streak;

/**
 * Score
 */
public Score score;

/**
 * Stick OK
 */
protected Ok ok;

/**
 * Stick No
 */
protected No no;

/**
 * Time Global
 */
private static float TIME = Var.TIME;

/**
 * Constructor parameterized
 * @param fyb
 *              General handler of the application
 * @param actorGestureListener
 *              Listener for the actos
 * @param nameChildClass
 *              Name of the children class
 */
public GameStage(FeedYourBrain fyb,
        ActorGestureListener actorGestureListener, int type) {
    super(new FitViewport(Var.width, Var.height));

    /**
     * Set FeedYourBrain
     */
    this.fyb = fyb;
```

```
        /**
         * Set listener
         */
        this.actorGestureListener = actorGestureListener;

        this.actors = new ArrayList<Actor>();

        this.loadView();
    }

    /**
     * Function to load view
     */
    private void loadView() {

        this.score = new Score(this.fyb.getTexture("score/score"),
                this.fyb.getFont("bitstreamcharter60"));

        this.time = new Time(this.fyb.getTexture("time/time"),
                this.fyb.getFont("bitstreamcharter50"));

        this.restart();

        this.time.setContent(this.seconds);

        this.pause = new Pause(this.fyb.getTexture("pause/pause"));
        this.pause.setType(ButtonType.PAUSE);
        this.pause.addListener(this.actorGestureListener);

        this.sound = new Sound(this.fyb);

        this.streak = new Streak(this.fyb);

        this.actors.add(this.score);
        this.actors.add(this.time);
        this.actors.add(this.sound);
        this.actors.add(this.streak);

        this.actors.add(this.pause);

        this.loadActors();

        this.ok = new Ok(this.fyb.getTexture("games/ok"));
        this.no = new No(this.fyb.getTexture("games/no"));
        this.addActor(this.ok);
        this.addActor(this.no);
    }

    /**
     * Function to load actors
     */
    protected void loadActors() {
        for (Actor actor : this.actors)
            this.addActor(actor);

        this.actors.clear();
    }

    /**
     * Update time
     * @param delta
     */
    public void update(float delta) {
        this.seconds -= delta;
        this.time.setContent(this.seconds);
        this.ok.act(delta);
        this.no.act(delta);
    }
    /**
     * Restart game
     */
    public void restart() {

        if (this.getStreak() != null) {
            this.getStreak().setStreak(1);
        }

        this.seconds = GameStage.TIME;
        this.score.clear();
    }

    /**
     * Return Time
     * @return Time
     */
    public Time getTime() {
        return this.time;
    }

    /**
     * Return Score
     * @return Score
     */
    public Score getScore() {
        return this.score;
    }

    /**
     * Return Pause
     * @return Pause
     */
    public Pause getPause() {
        return this.pause;
    }

    /**
     * Return Streak
     * @return
     */
    public Streak getStreak() {
        return this.streak;
    }

    /**
     * Function to show a success
     */
    public void ok() {
        // TODO Auto-generated method stub
        this.ok.reload();
    }

    /**
     * Function to show a error
     */
    public void no() {
        // TODO Auto-generated method stub
        this.no.reload();

    }

}
```

2.55. PauseStage

```
package com.juanmartos.games.feedyourbrain.stages;

import java.util.ArrayList;
```

```java
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * PauseStage extends the class Stage and create the stage for pause screen
 */
public class PauseStage extends Stage {

    /**
     * General handler of the application
     */
    private FeedYourBrain fyb;

    /**
     * List of actors
     */
    public ArrayList<Actor> actors;

    /**
     * Listener click actor
     */
    private ActorGestureListener actorGestureListener;

    /**
     * Constructor parameterized
     * @param fyb
     *              General handler of the application
     * @param actorGestureListener
     *              Listener for the actos
     * @param nameChildClass
     *              Name of the children class
     */
    public PauseStage(FeedYourBrain fyb,
            ActorGestureListener actorGestureListener) {
        super(new FitViewport(Var.width, Var.height));

        //Asignamos FYB
        this.fyb = fyb;

        //Asignamos el listener
        this.actorGestureListener = actorGestureListener;

        //Guardamos memoria para los actores
        this.actors = new ArrayList<Actor>();

        this.loadView();
    }

    /**
     * Function to load default view
     */
    private void loadView() {

        Texture textureButtonContinue = this.fyb.getTexture(
                "main/button_continue", true);
        Texture textureButtonRestart = this.fyb.getTexture(
                "main/button_restart", true);
        Texture textureButtonExit = this.fyb.getTexture("main/button_exit",
                true);
        BitmapFont fontButton = this.fyb.getFont("bitstreamcharter50");
        int x = 0;
        int y = 0;
        int height = Var.height;
        int width = Var.width;
        int buttonWidth = 800;
        int buttonHeight = 133;
        int nButtons = 3;

        int offsetX = 0;
        int offsetY = 0;

        offsetX = (width - buttonWidth) / 2;
        offsetY = (height - buttonHeight * nButtons) / (nButtons + 1);

        x = offsetX;
        y = offsetY;

        Button buttonExit = new Button(textureButtonExit, fontButton, "");
        buttonExit.setPosition(x, y);
        buttonExit.setType(ButtonType.EXIT);
        buttonExit.addListener(this.actorGestureListener);

        y += offsetY + buttonHeight;

        Button buttonRestart = new Button(textureButtonRestart, fontButton, "");
        buttonRestart.setPosition(x, y);
        buttonRestart.setType(ButtonType.RESTART);
        buttonRestart.addListener(this.actorGestureListener);

        y += offsetY + buttonHeight;

        Button buttonContinue = new Button(textureButtonContinue, fontButton,
                "");
        buttonContinue.setPosition(x, y);
        buttonContinue.setType(ButtonType.RESUME);
        buttonContinue.addListener(this.actorGestureListener);

        this.actors.add(buttonExit);
        this.actors.add(buttonContinue);
        this.actors.add(buttonRestart);

        //this.loadActors();
    }

    /**
     * Function to load actors
     */
    public void loadActors() {
        for (Actor actor : this.actors)
            this.addActor(actor);
    }

}
```

2.56. Score

```
package com.juanmartos.games.feedyourbrain.stages;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.utils.DatabaseFeedYourBrain;
import com.juanmartos.games.feedyourbrain.utils.MyPreferences;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * Score extends the class Actor and create the view for score in stages
 */
```

```java
public class Score extends Actor {

    /**
     * General handler of the application
     */
    private FeedYourBrain feedYourBrain;

    /**
     * Array textures
     */
    private Texture[] textureNumbers;

    /**
     * Name children class
     */
    private String nameChilds;

    /**
     * Score
     */
    private String score;

    /** It represents the side which show the score **/
    /** False -> Left | True -> Right **/
    private boolean side;

    /**
     *
     * @param feedYourBrain
     *                     General handler of the application
     * @param textureNumbers
     *                     Texture numbers
     * @param nameChilds
     *                     Name of the child class
     */
    public Score(FeedYourBrain feedYourBrain, Texture[] textureNumbers,
            String nameChilds) {
        this.feedYourBrain = feedYourBrain;
        this.textureNumbers = textureNumbers;
        this.nameChilds = nameChilds;
        this.score = this.getMaxScore();
        this.side = true;
    }

    /**
     *
     * @param feedYourBrain
     *                     General handler of the application
     * @param textureNumbers
     *                     Texture numbers
     * @param score
     *                     Score to draw
     */
    public Score(FeedYourBrain feedYourBrain, Texture[] textureNumbers,
            int score) {
        this.feedYourBrain = feedYourBrain;
        this.textureNumbers = textureNumbers;
        this.score = String.valueOf(score);
        this.side = false;
    }

    /**
     * Return the max score
     * @return the max score
     */
    private String getMaxScore() {

        int score = 0;
        int game = 0;
        DatabaseFeedYourBrain database = this.feedYourBrain
                .getDatabaseFeedYourBrain();

        if (this.nameChilds.equals("math")) {
            game = DatabaseFeedYourBrain.MATH;
        } else if (this.nameChilds.equals("logic")) {
            game = DatabaseFeedYourBrain.ASSOCIATION;
        } else if (this.nameChilds.equals("visual")) {
            game = DatabaseFeedYourBrain.VISUAL;
        } else if (this.nameChilds.equals("memory")) {
            game = DatabaseFeedYourBrain.MEMORY;
        }else if (this.nameChilds.equals("sequence")) {
            game = DatabaseFeedYourBrain.SEQUENCE;
        } else if (this.nameChilds.equals("weight")) {
            game = DatabaseFeedYourBrain.WEIGHT;
        } else {
            score = database.getMaxScore();
            return String.valueOf(score);
        }

        score = database.getMaxScoreMode(game, MyPreferences.getDifficulty());

        return String.valueOf(score);
    }

    /** Draws the actor.
     * The Batch is configured to draw in the parent's coordinate system.
     * draw(com.badlogic.gdx.graphics.g2d.TextureRegion,
     * float, float, float, float, float, float, float, float, float)
     * This draw method is convenient to draw a rotated
     * and scaled TextureRegion. begin() has already been called on
     * the Batch. If end() is called to draw without the Batch thenbegin()
     * must be called before the method returns.
     * <p>
     * The default implementation does nothing.
     * @param alpha Should be multiplied with the actor's alpha,
     * allowing a parent's alpha to affect all children. */
    @Override
    public void draw(Batch batch, float alpha) {

        int width = Var.width;
        int height = Var.height;
        int x = 0;
        int y = 0;
        int offsetX = 0;
        int offsetY = 0;
        int textureWidth = this.textureNumbers[0].getWidth();

        if (this.side) {
            offsetX = width / 2
                    + (width / 2 - textureWidth * this.score.length()) / 2;
        } else {
            offsetX = (width / 2 - textureWidth * this.score.length()) / 2;
        }

        offsetY = (int) (height * 0.25f);
        x = offsetX;
        y = offsetY;

        for (int i = 0; i < this.score.length(); ++i) {
            int index = this.score.charAt(i) - 48;
            batch.draw(this.textureNumbers[index], x, y);
            x = x + textureWidth;
        }
    }

    /**
     * Set score
     * @param score
     */
```

```java
    public void setScore(int score) {
        // TODO Auto-generated method stub
        this.score = String.valueOf(score);

    }

    /**
     * Set side
     * @param side
     */
    public void setSide(boolean side) {
        this.side = side;
    }
}
```

2.57. StartStage

```java
package com.juanmartos.games.feedyourbrain.stages;

import java.util.ArrayList;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Button;
import com.juanmartos.games.feedyourbrain.actors.ButtonType;
import com.juanmartos.games.feedyourbrain.stages.actors.StartGameDescription;
import com.juanmartos.games.feedyourbrain.stages.actors.StartGameTitle;
import com.juanmartos.games.feedyourbrain.utils.Var;

/**
 * StartStage extends the class Stage and create the stage for stage screen
 */
public class StartStage extends Stage {

    /**
     * General handler of the application
     */
    private FeedYourBrain fyb;

    /**
     * List of actors
     */
    public ArrayList<Actor> actors;

    /**
     * Listener click actor
     */
    private ActorGestureListener actorGestureListener;

    /**
     * Name children class
     */
    private String nameChildClass;

    /**
     * Actor start title
     */
    private StartGameTitle startGameTitle;

    /**
     * Actor start description
     */
    private StartGameDescription startGameDescription;
```

```
/**
 * Actor max score title
 */
private StartGameTitle startGameScoreMax;

/**
 * Button to start
 */
private Button buttonStart;

/**
 * Button to back
 */
private Button buttonBack;

/**
 * Score
 */
private Score score;

/**
 *
 * @param feedYourBrain
 *                      General handler of the application
 * @param textureNumbers
 *                      Texture numbers
 * @param nameChilds
 *                      Name of the child class
 */
public StartStage(FeedYourBrain fyb,
        ActorGestureListener actorGestureListener, String nameChildClass) {
    super(new FitViewport(Var.width, Var.height));

    //Asignamos FYB
    this.fyb = fyb;

    //Asignamos el listener
    this.actorGestureListener = actorGestureListener;

    /** Asignamos el nombre de la clase **/
    this.nameChildClass = nameChildClass;

    //Guardamos memoria para los actores
    this.actors = new ArrayList<Actor>();

    this.loadView();
}

/**
 * Function to load view
 */
private void loadView() {

    int width = Var.width;
    int height = Var.height;

    int x = 0;
    int y = 0;
    int offsetX = 0;
    int offsetY = 0;

    Texture textureStartGameTitle = this.fyb.getTexture("games/"
            + this.nameChildClass + "/title", true);
    Texture textureStartGameDescription = this.fyb.getTexture("games/"
            + this.nameChildClass + "/description", true);
    Texture textureStartGameScoreMax = this.fyb.getTexture(
            "games/scoreMax", true);
    Texture textureButtonBack = this.fyb.getTexture("main/button_back",
            true);
    Texture textureButtonStart = this.fyb.getTexture("main/button_start",
            true);
    BitmapFont fontButton = this.fyb.getFont("bitstreamcharter50");

    x = (width - textureStartGameTitle.getWidth()) / 2;
    y = (int) ((height * 0.75f) + (height * 0.25f - textureStartGameTitle
            .getHeight()) / 2);

    this.startGameTitle = new StartGameTitle(textureStartGameTitle);
    this.startGameTitle.setPosition(x, y);

    x = (int) ((width * 0.5f - textureStartGameDescription.getWidth()) / 2);
    y = (int) ((height - textureStartGameDescription.getHeight()) / 2);

    this.startGameDescription = new StartGameDescription(
            textureStartGameDescription);
    this.startGameDescription.setPosition(x, y);

    x = (int) ((width * 0.5f) + (width * 0.5f - textureStartGameScoreMax
            .getWidth()) / 2);
    y = (int) (height * 0.60);

    this.startGameScoreMax = new StartGameTitle(textureStartGameScoreMax);
    this.startGameScoreMax.setPosition(x, y);

    int nButtons = 2;
    x = 0;
    y = 0;
    offsetX = (width - textureButtonBack.getWidth() * nButtons)
            / (nButtons + 1);
    offsetY = (int) (height * 0.10);

    x = offsetX;
    y = offsetY;

    this.buttonStart = new Button(textureButtonStart, fontButton, "");
    this.buttonStart.setPosition(x, y);
    this.buttonStart.setType(ButtonType.START);
    this.buttonStart.addListener(this.actorGestureListener);

    x = x + textureButtonBack.getWidth() + offsetX;

    this.buttonBack = new Button(textureButtonBack, fontButton, "");
    this.buttonBack.setPosition(x, y);
    this.buttonBack.setType(ButtonType.BACK);
    this.buttonBack.addListener(this.actorGestureListener);

    Texture textures[] = new Texture[10];
    for (int i = 0; i <= 9; ++i)
        textures[i] = this.fyb.getTexture("games/score/" + i);

    this.score = new Score(this.fyb, textures, this.nameChildClass);

    this.actors.add(this.startGameTitle);
    this.actors.add(this.startGameDescription);
    this.actors.add(this.startGameScoreMax);
    this.actors.add(this.buttonStart);
    this.actors.add(this.buttonBack);
    this.actors.add(this.score);

    this.loadActors();
}

/**
 * Function to load actors
 */
protected void loadActors() {
    for (Actor actor : this.actors)
        this.addActor(actor);
}
/**
 * Function to remove the actos of this stage
 */
public void remove() {
    this.startGameTitle.remove();
    this.startGameDescription.remove();
    this.startGameScoreMax.remove();
    this.buttonBack.remove();
    this.buttonStart.remove();
    this.buttonStart.clear();
    this.buttonBack.clear();
    this.score.remove();
}

}
```

2.58. MemoryStructure

```
package com.juanmartos.games.feedyourbrain.structures;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.badlogic.gdx.utils.TimeUtils;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.memory.Card;
import com.juanmartos.games.feedyourbrain.utils.Var;

import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashSet;
import java.util.Random;
import java.util.Set;

/**
 * MemoryStructure provides methods for control to operations in memory game
 */
public class MemoryStructure {

    /**
     * General handler of the application
     */
    private FeedYourBrain fyb;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * Actor listener
     */
    private ActorGestureListener actorGestureListener;

    /**
     * List of cards to show up
     */
    private ArrayList<Card> cardsUp;

    /**
     * List of card to show down
     */
    private ArrayList<Card> cardsDown;

    /**
     * List of card to find
     */
    private ArrayList<Card> cardsToFind;
```

```
/**
 * List of types of cards
 */
private ArrayList<Integer> types;

/**
 * Difficulty
 */
private int difficulty;

/**
 * Time to show the cards
 */
private float time;

/**
 * Variable to control the head of the cards
 */
private Boolean heads;

/**
 * Variable to control the tumble of the cards
 */
private Boolean tumble;

/**
 * Variable to control when change the sctructure
 */
private Boolean changeStructure;

/**
 * Time to change structure
 */
private float timeStructure;

/**
 * Array of names of types
 */
private String animals[] = { "cat", "shark", "lion", "octopus", "egg",
        "gecko" };

/**
 * Constructor parameterized
 * @param fyb
 *              General handler of the application
 * @param stage
 *              Stage
 * @param actorGestureListener
 *              Actor listener
 */
public MemoryStructure(FeedYourBrain fyb, Stage stage,
        ActorGestureListener actorGestureListener) {

    this.fyb = fyb;
    this.stage = stage;
    this.actorGestureListener = actorGestureListener;

    this.cardsUp = new ArrayList<Card>();
    this.cardsDown = new ArrayList<Card>();
    this.cardsToFind = new ArrayList<Card>();

    this.types = new ArrayList<Integer>();

    this.difficulty = 1;

    this.changeStructure = false;
    this.timeStructure = 0.75f;
}

/**
```

```
 * Function to generate a secuence of cards
 */
public void generate() {

    /**
     * Clear cards
     */
    this.cardsDown.clear();
    this.cardsToFind.clear();
    this.cardsUp.clear();

    /** UpdateTime **/
    this.time = .75f;

    this.heads = true;
    this.tumble = false;

    /**
     * Get width and height of the cards
     */
    Texture textureTail = this.fyb.getTexture("games/memory/cards/card");
    int widthCard = textureTail.getWidth();
    int heightCard = textureTail.getHeight();

    /**
     * Get width and height of the screen
     */
    int width = Var.width;
    int height = Var.height;

    /**
     * Number of cards to generate
     */
    int nCards = 0;

    nCards = this.getNCards();

    int x = 0;
    int y = 0;
    int offsetX = 0;
    int offsetY = 0;

    /**
     * Get the offset x
     */
    offsetX = (width - (nCards * widthCard)) / (nCards + 1);

    /**
     * Get the offset y
     * This block is on the upper half of the screen
     */
    offsetY = (int) (height * 0.75 - heightCard / 2);

    /**
     * Set offsets
     */
    x = offsetX;
    y = offsetY;

    for (int i = 0; i < nCards; ++i) {

        int type = this.getRandomType();

        Texture textureHead = this.getTextureBy(type);
        Card card = new Card(textureHead, textureTail, type);
        card.setPosition(x, y);

        x = x + widthCard + offsetX;

        this.cardsUp.add(card);
    }

    /**
     * Check if the card are the same
     */
    Card cardUp= this.cardsUp.get(0);

    for (int i = 1; i < this.cardsUp.size(); ++i) {
        if (cardUp.getType() == this.cardsUp.get(i).getType()) {
            cardUp = this.cardsUp.get(i);
        } else {
            return;
        }
    }

    /**
     * Generate
     */
    this.generate();

}

/**
 * Return the number of cards to generate
 * @return number of cards
 */
private int getNCards() {

    if (this.difficulty == 1) {
        return 3;
    } else if (this.difficulty == 2) {
        return 4;
    } else if (this.difficulty == 3) {
        return 5;
    }

    return 3;
}

/**
 * Function to randomly choose a card type
 * @return
 *          Tipo de carta
 */
public int getRandomType() {
    Random random = new Random();
    int nTypes = this.animals.length;
    int type = (random.nextInt(nTypes));
    return type;
}

/**
 * Function to select the texture based on the type of letter
 * @param type
 *              Type of card
 * @return
 *              Texture
 */
private Texture getTextureBy(int type) {

    String animal = this.animals[type];

    Texture texture = this.fyb.getTexture("games/memory/cards/" + animal);

    return texture;
}

/**
 * Function to load the actors on stage
 */
public void loadActors() {
    for (Card card : this.cardsUp)
        this.stage.addActor(card);
}

/**
 * Return true if we see the faces of the cards
 * @return
 *          True => Head | False => Cross
 */
public Boolean isHead() {
    return this.heads;
}

/**
 * Update time representing unturned cards
 * If the time we got to zero specify that no more update
 * And proceed to flip the cards
 * @param delta
 *              Tiempo que ha pasado en ejecucion
 */
public void updateTime(float delta) {
    this.time -= delta;

    if (this.time <= 0.0f) {
        this.heads = false;

        this.tumbleCard();
    }

}

/**
 * Function to assign the cards to tumble
 */
private void tumbleCard() {

    int nCardsHide = this.getNCardsHide();

    /**
     * Shuffle listing
     */
    long seed = TimeUtils.nanoTime();
    Collections.shuffle(this.cardsUp, new Random(seed));

    /**
     * Clear types
     */
    this.types.clear();

    for (int i = 0; i < nCardsHide; ++i) {
        Card card = this.cardsUp.get(i);
        card.setTumble(true);
        card.setHead(false);
        this.types.add(card.getType());

        this.cardsToFind.add(card);

    }

    /**
     * We are looking array of types
     * We added a type that does not exist
     */
    for (int i = 0; i < this.animals.length; ++i) {

        Boolean find = false;

        for (int type : this.types) {
            if (type == i) {
                find = true;
            }
        }

        if (find == false) {
            /**
             * Add types
             */
            this.types.add(i);
            break;
        }
    }

    /**
     * Generate cards down
     */
    this.generateDown();

    this.tumble = true;

}

private void generateDown() {

    /**
     * Get width and height of the cards
     */
    Texture textureTail = this.fyb.getTexture("games/memory/cards/card");
    int widthCard = textureTail.getWidth();
    int heightCard = textureTail.getHeight();

    /**
     * Get width and height of the screen
     */
    int width = Var.width;
    int height = Var.height;

    /**
     * Number of cards to generate
     */
    int nCards = 0;

    nCards = this.getNCards();

    int x = 0;
    int y = 0;
    int offsetX = 0;
    int offsetY = 0;

    /**
     * Delete the repeat types
     */
    Set<Integer> linkedHashSet = new LinkedHashSet<Integer>();
    linkedHashSet.addAll(types);
    types.clear();
    types.addAll(linkedHashSet);

    nCards = types.size();

    /**
     * Get the offset x
     */
    offsetX = (width - (nCards * widthCard)) / (nCards + 1);

    /**
     * Get the offset y
     * This first block of cards was found in the lower half of the screen
     */
    offsetY = (int) (height * 0.25 - heightCard / 2);
```

```
        /**
         * Set offsets
         */
        x = offsetX;
        y = offsetY;

        /**
         * Shuffle listing
         */
        long seed = TimeUtils.nanoTime();
        Collections.shuffle(types, new Random(seed));

        /**
         * Create down buttos
         */
        for (int type : types) {
            Texture textureHead = this.getTextureBy(type);
            Card card = new Card(textureHead, textureTail, type);
            card.setPosition(x, y);
            card.addListener(this.actorGestureListener);

            x = x + widthCard + offsetX;

            this.cardsDown.add(card);
        }

        this.loadActorsDown();
    }

    /**
     * Function to load actors of this stage
     */
    public void loadActorsDown() {
        for (Card card : this.cardsDown)
            this.stage.addActor(card);
    }

    /**
     * Return true if we have to tumble the cards
     * @return
     *          True => Tumble | False => No Tumble
     */
    public boolean isTumble() {
        return this.tumble;
    }

    /**
     * Update time representing flipping cards
     * If the time we specify zero it will not overturn more
     * @param delta
     *              Time spent in execution
     */
    public void updateTimeForTumble(float delta) {
        for (Card card : this.cardsUp) {
            if (card.isTumble())
                card.updateTime(delta);
        }
    }

    /**
     * Returns the number of cards that are hidden on the basis of difficulty
     * @return
     *              Number of cards that are hidden
     */
    private int getNCardsHide() {

        if (this.difficulty == 1) {
            return 1;
        } else if (this.difficulty == 2) {
```

```
        return 2;
    } else if (this.difficulty == 3) {
        return 3;
    }

    return 1;
}

/**
 * Function to check the letter that we have pressed
 * @param cardToFind
 *                  Card to check
 * @return
 *          0 -> Card no find
 *          1 -> Card find
 *          2 -> Last card find
 */
public int checkCard(Card cardToFind) {

    int check = 0;

    for (Card card : this.cardsToFind) {

        if (card.getType() == cardToFind.getType()) {
            card.tailToHead();
            check = 1;
            this.cardsToFind.remove(card);
            break;
        }

    }

    if (check == 1 && this.cardsToFind.size() == 0) {
        return 2;
    }

    return check;

}

/**
 * Function to clear the current structure and assign a new
 */
public void clear() {
    for (Card card : this.cardsDown)
        card.remove();

    for (Card card : this.cardsUp)
        card.remove();

    this.generate();
    this.loadActors();
}

/**
* Update time representing flipping cards
* If the time we specify zero it will not overturn more
 * @param delta
 *              Time spent in execution
 */
public void updateTimeForChange(float delta) {
    this.timeStructure -= delta;

    if (this.timeStructure <= 0.0f) {
        this.clear();
        this.timeStructure = 0.75f;
        this.setChange(false);
    }
}
```

```java
    /**
     * Function to turn all the cards when we make a judgment
     */
    public void tailToHeadAllCards() {
        for (Card card : this.cardsToFind) {
            card.tailToHead();
        }

    }

    /**
     * Set change structure
     * @param change
     */
    public void setChange(boolean change) {
        this.changeStructure = change;
    }

    /**
     * Function to check if structure change
     * @return
     */
    public boolean isChange() {
        return this.changeStructure;
    }

    /**
     * Function to set difficulty
     * @param difficulty
     *                  difficulty
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    /**
     * Static class to represent the types of letter
     * @author juan
     *
     */
    public static class CardType {
        public static int BIRD = 7;
        public static int WOLF = 8;
        public static int CAT = 1;
        public static int SHARK = 2;
        public static int LION = 3;
        public static int OCTOPUS = 4;
        public static int EGG = 5;
        public static int GECKO = 6;
    }

}
```

2.59. MemoryStructure

```java
package com.juanmartos.games.feedyourbrain.structures;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Weight;
import com.juanmartos.games.feedyourbrain.actors.weight.Balance;
import com.juanmartos.games.feedyourbrain.utils.Log;
import com.juanmartos.games.feedyourbrain.utils.Var;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Random;

/**
 * WeightStructure provides methods for control to operations in memory game
 */
public class WeightStructure {

    /**
     * General handler of the application
     */
    private FeedYourBrain fyb;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * List of balances to show up
     */
    private ArrayList<Balance> balances;

    /**
     * List of weights to show up
     */
    private ArrayList<Weight> weights;

    /**
     * List of weights to show up
     */
    private ArrayList<Weight> weightsDown;

    /**
     * Letter to find
     */
    private String letter;

    /**
     * Actor listener
     */
    private ActorGestureListener actorGestureListener;

    /**
     * Difficulty
     */
    private int difficulty;

    /**
     * Time to show the cards
     */
    private float time;

    /**
     * Array of names of types
     */
    private String weightsTextures[]
            = { "weight_1", "weight_2", "weight_3", "weight_4", "weight_5", "weight_6" };

    /**
     * Constructor parameterized
     * @param fyb
     *             General handler of the application
     * @param stage
     *             Stage
     * @param actorGestureListener
     *             Actor listener
     */
    public WeightStructure(FeedYourBrain fyb, Stage stage,
                           ActorGestureListener actorGestureListener) {
```

```
        this.fyb = fyb;
        this.stage = stage;
        this.actorGestureListener = actorGestureListener;

        this.weights = new ArrayList<Weight>();
        this.weightsDown = new ArrayList<Weight>();
        this.balances = new ArrayList<Balance>();

        this.difficulty = 1;
    }

    /**
     * Function to generate a weight structure
     */
    public void generate() {

        /**
         * Clear balances and weights
         */
        for(Balance balance:this.balances){
            balance.remove();
        }
        for(Weight weight:this.weights){
            weight.remove();
        }
        for(Weight weight:this.weightsDown){
            weight.remove();
        }
        this.balances.clear();
        this.weights.clear();
        this.weightsDown.clear();

        this.shuffleWeights();

        /** UpdateTime **/
        this.time = .75f;

        Random random = new Random();

        if(this.difficulty == 1){ //Test case

            String weightsS [] = {"A", "B", "C"};
            int weightsI [] = {1, 1, 2};
            int nWeights = 3;
            int option = random.nextInt(3);

            if (option == 0){
                int nBalances = 2;
                int balances [] = {Balance.EQUAL, Balance.LEFT};
                String balancesOptions [][] = {{"A", "B"}, {"B", "C"}};

                this.letter = "C";

                this.load(balances, nBalances, nWeights, weightsI, weightsS, balancesOptions);
            }else if (option == 1){
                int nBalances = 2;
                int balances [] = {Balance.EQUAL, Balance.RIGHT};
                String balancesOptions [][] = {{"A", "BB"}, {"A", "CB"}};

                this.letter = "A";

                this.load(balances, nBalances, nWeights, weightsI, weightsS, balancesOptions);
            }else if (option == 2){
                int nBalances = 2;
                int balances [] = {Balance.LEFT, Balance.RIGHT};
                String balancesOptions [][] = {{"B", "C"}, {"A", "C"}};

                this.letter = "A";

                this.load(balances, nBalances, nWeights, weightsI, weightsS, balancesOptions);
            }
        } else if(this.difficulty == 2){ //Test case

            String weightsS [] = {"A", "B", "C"};
            int weightsI [] = {1, 2, 3};
            int nWeights = 3;

            int option = random.nextInt(3);

            if (option == 0){
                int nBalances = 3;
                int balances [] = {Balance.LEFT, Balance.RIGHT, Balance.LEFT};
                String balancesOptions [][] = {{"A", "B"}, {"C", "A"}, {"B", "C"}};

                this.letter = "C";

                this.load(balances, nBalances, nWeights, weightsI, weightsS, balancesOptions);
            }else if (option == 1){
                int nBalances = 2;
                int balances [] = {Balance.EQUAL, Balance.EQUAL};
                String balancesOptions [][] = {{"AB", "C"}, {"AB", "AB"}};

                this.letter = "C";

                this.load(balances, nBalances, nWeights, weightsI, weightsS, balancesOptions);
            }else if (option == 2){
                int nBalances = 3;
                int balances [] = {Balance.LEFT, Balance.EQUAL, Balance.RIGHT};
                String balancesOptions [][] = {{"A", "B"}, {"AAA", "C"}, {"C", "B"}};

                this.letter = "C";

                this.load(balances, nBalances, nWeights, weightsI, weightsS, balancesOptions);
            }
        } else if(this.difficulty == 3){ //Test case

            String weightsS [] = {"A", "B", "C", "D"};
            int weightsI [] = {1, 2, 2, 3};
            int nWeights = 4;
            int option = random.nextInt(3);

            if (option == 0){
                int nBalances = 3;
                int balances [] = {Balance.LEFT, Balance.EQUAL, Balance.LEFT};
                String balancesOptions [][] = {{"B", "D"}, {"BB", "CC"}, {"AA", "CC"}};

                this.letter = "D";

                this.load(balances, nBalances, nWeights, weightsI, weightsS, balancesOptions);
            }else if (option == 1){
                int nBalances = 3;
                int balances [] = {Balance.LEFT, Balance.RIGHT, Balance.EQUAL};
                String balancesOptions [][] = {{"AB", "D"}, {"ABD", "C"}, {"A", "B"}};

                this.letter = "D";

                this.load(balances, nBalances, nWeights, weightsI, weightsS, balancesOptions);
            }else if (option == 2){
                int nBalances = 3;
                int balances [] = {Balance.EQUAL, Balance.EQUAL, Balance.EQUAL};
                String balancesOptions [][] = {{"D", "AB"}, {"D", "AC"}, {"BA", "CA"}};

                this.letter = "D";

                this.load(balances, nBalances, nWeights, weightsI, weightsS, balancesOptions);
            }
        }
    }

    public void load(
```

```java
        int balances [],
        int nBalances,
        int nWeights,
        int [] weightsI,
        String weightsS [],
        String balancesOptions [][]
){
    /**
     * Get width and height of the screen
     */
    int width = Var.width;
    int height = Var.height;

    int x = 0;
    int y = 0;
    int offsetX = 0;
    int offsetY = 0;

    /**
     * Get the offset x
     */
    offsetX = (width - (nBalances * Balance.WIDTH) ) / (nWeights + 1);

    /**
     * Set offsets
     */
    x = offsetX;
    y = (int) (height * 0.45 - Balance.HEIGHTEQUAL / 2);

    for(int i = 0; i < weightsI.length; i++){

        Texture textureWeight = this.fyb.getTexture("balances/" + weightsTextures[i]);

        Weight weight = new Weight(textureWeight, weightsI[i], weightsS[i]);

        weight.addListener(this.actorGestureListener);

        this.weights.add(weight);
        this.weightsDown.add(weight);
    }

    for (int i = 0; i < nBalances; ++i) {

        Balance balance = new Balance(this.getTextureBy(balances[i]), balances[i]);

        String options[] = balancesOptions[i];

        String optionLeft = options[0];
        String optionRight = options[1];

        for (int o = 0; o < optionLeft.length(); ++o){
            balance.addtoLeft(this.findWeightByLetter((String.valueOf(optionLeft.charAt(o)))));
        }

        for (int o = 0; o < optionRight.length(); ++o){
            balance.addtoRight(
                    this.findWeightByLetter((String.valueOf(optionRight.charAt(o))))
            );
        }

        this.balances.add(balance);
    }

    Collections.shuffle(this.balances);

    for(Balance balance:this.balances){
        balance.setPosition(x, y);
        x = x + Balance.WIDTH + offsetX;
    }
    //Draw weights
    offsetX = (width - (this.weightsDown.size() * Weight.WIDTH) )
            / (this.weightsDown.size() + 1);

    x = offsetX;
    y = (int) (height * 0.1);

    Collections.shuffle(this.weightsDown);

    for (Weight weight:this.weightsDown){
        weight.setPosition(x, y);
        x = x + Weight.WIDTH + offsetX;
    }
}

/**
 *
 * @param letter
 * @return
 */
private Weight findWeightByLetter(String letter){

    for(Weight weight:this.weights){

        if (weight.getLetter().equals(letter))
            return weight;

    }

    return this.weights.get(0);
}

/**
 * Function to select the texture based on the type of letter
 * @param type
 *              Type of card
 * @return
 *              Texture
 */
private Texture getTextureBy(int type) {

    String name = "balance";

    if (type == Balance.LEFT){
        name = "balance_left";
    } else if (type == Balance.RIGHT){
        name = "balance_right";
    }

    return this.fyb.getTexture("balances/" + name);
}

/**
 * Function to load the actors on stage
 */
public void loadActors() {
    for (Balance balance : this.balances){
        this.stage.addActor(balance);
    }
    for (Weight weight:this.weightsDown){
        this.stage.addActor(weight);
    }
}

/**
 * Update time representing unturned cards
 * If the time we got to zero specify that no more update
 * And proceed to flip the cards
 * @param delta
 *              Tiempo que ha pasado en ejecucion
```

```
     */
    public void updateTime(float delta) {
        this.time -= delta;

        if (this.time <= 0.0f) {
            //this.heads = false;

            //this.tumbleCard();
        }

    }

    /**
     * CheckWeight
     * @param weightToFind
     * @return
     */
    public boolean checkWeight(Weight weightToFind) {

        Log.log("L1: " + weightToFind.getLetter());
        Log.log("L2: " + this.letter);

        if(weightToFind.getLetter().equals(this.letter)){
            Log.log("True");
            return true;
        }

        Log.log("False");
        return false;
    }

    public void shuffleWeights(){
        Collections.shuffle(Arrays.asList(this.weightsTextures));
    }

    /**
     * Function to set difficulty
     * @param difficulty
     *                  difficulty
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

}
```

2.60. ActionResolver

```
package com.juanmartos.games.feedyourbrain.utils;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Net;
import com.badlogic.gdx.net.HttpParametersUtils;
import com.badlogic.gdx.utils.JsonReader;
import com.badlogic.gdx.utils.JsonValue;

import java.util.HashMap;
import java.util.Map;

public class Api {

    public static String host = "http://backend.fyb.juanmartoscaceres.com";
    public static String version = "v1";
    public static String language = "es";

    public void createUser(ApiInterface apiInterface)
    {
        Log.log("Create user");
        String url = host + "/" + version + "/" + language + "/users";
        Net.HttpRequest httpGet = new Net.HttpRequest(Net.HttpMethods.POST);
        httpGet.setUrl(url);

        Gdx.net.sendHttpRequest (httpGet, new Net.HttpResponseListener() {
            public void handleHttpResponse(Net.HttpResponse httpResponse) {
                Log.log("http create user success");
                JsonValue json = new JsonReader().parse(httpResponse.getResultAsString());
                String udid = json.getString("udid");
                String name = json.getString("name");
                MyPreferences.setId(udid);
                //do stuff here based on response
            }

            public void failed(Throwable t) {
                Log.log("http create user failed");
                //do stuff here based on the failed attempt
            }

            @Override
            public void cancelled() {
                Log.log("canceled");
            }
        });
    }

    public void login(String udid, final ApiInterface apiInterface)
    {
        String url = host + "/" + version + "/" + language + "/users/" + udid;

        Log.log("Login url");
        Log.log(url);

        Net.HttpRequest httpGet = new Net.HttpRequest(Net.HttpMethods.GET);
        httpGet.setUrl(url);

        Gdx.net.sendHttpRequest (httpGet, new Net.HttpResponseListener() {
            public void handleHttpResponse(Net.HttpResponse httpResponse) {
                Log.log("Http login success");
                JsonValue json = new JsonReader().parse(httpResponse.getResultAsString());

                String udid = json.getString("udid");
                Log.log(udid);
                apiInterface.success(httpResponse);
            }

            public void failed(Throwable t) {
                Log.log("Http login failed");
                Log.log(t.getLocalizedMessage());
                apiInterface.failed();
                //do stuff here based on the failed attempt
            }

            @Override
            public void cancelled() {
                Log.log("canceled");
                apiInterface.cancelled();
            }
        });
    }

    public void createScore(
            final ApiInterface apiInterface, String udid, String game, String level, String amount
    )
    {
        Map parameters = new HashMap();
        parameters.put("udid", udid);
        parameters.put("game", game);
        parameters.put("level", level);
        parameters.put("amount", amount);
        Log.log("Create score");
        String url = host + "/" + version + "/" + language + "/scores";

        Net.HttpRequest httpGet = new Net.HttpRequest(Net.HttpMethods.POST);
        httpGet.setHeader("Content-Type", "application/x-www-form-urlencoded");
        httpGet.setUrl(url);
        httpGet.setContent(HttpParametersUtils.convertHttpParameters(parameters));

        Gdx.net.sendHttpRequest (httpGet, new Net.HttpResponseListener() {
            public void handleHttpResponse(Net.HttpResponse httpResponse) {
                Log.log("http create score success");
            }

            public void failed(Throwable t) {
                Log.log("http create score failed");
                //do stuff here based on the failed attempt
            }

            @Override
            public void cancelled() {
                Log.log("http create score cancelled");
            }
        });
    }

    public void getScore(
            final ApiInterface apiInterface, String udid, String game, String difficulty
    )
    {
        String url = host + "/" + version + "/" + language
                + "/scores?udid=" + udid + "&game=" + game + "&level=" + difficulty;

        Log.log(url);

        Net.HttpRequest httpGet = new Net.HttpRequest(Net.HttpMethods.GET);
        httpGet.setUrl(url);

        Gdx.net.sendHttpRequest (httpGet, new Net.HttpResponseListener() {
            public void handleHttpResponse(Net.HttpResponse httpResponse) {
                apiInterface.success(httpResponse);
            }

            public void failed(Throwable t) {
                Log.log("Http get score failed");
                Log.log(t.getLocalizedMessage());
                apiInterface.failed();
                //do stuff here based on the failed attempt
            }

            @Override
            public void cancelled() {
                Log.log("canceled");
                apiInterface.cancelled();
            }
        });
    }

    public void getAchievements(final ApiInterface apiInterface, String udid)
    {
        String url = host + "/" + version + "/" + language + "/achievements?udid=" + udid;

        Log.log(url);

        Net.HttpRequest httpGet = new Net.HttpRequest(Net.HttpMethods.GET);
        httpGet.setUrl(url);

        Gdx.net.sendHttpRequest (httpGet, new Net.HttpResponseListener() {
            public void handleHttpResponse(Net.HttpResponse httpResponse) {
                Log.log("http get achievements success");
```

```
                apiInterface.success(httpResponse);
            }

            public void failed(Throwable t) {
                Log.log("http get achievements failed");
                Log.log(t.getLocalizedMessage());
                apiInterface.failed();
            }

            @Override
            public void cancelled() {
                Log.log("http get achievements cancelled");
                apiInterface.cancelled();
            }
        });
    }

    public void setAchievements(String udid, String code)
    {
        Map parameters = new HashMap();
        parameters.put("user_udid", udid);
        parameters.put("achievement_udid", code);

        String url = host + "/" + version + "/" + language + "/achievements";

        Net.HttpRequest httpGet = new Net.HttpRequest(Net.HttpMethods.POST);
        httpGet.setHeader("Content-Type", "application/x-www-form-urlencoded");
        httpGet.setUrl(url);
        httpGet.setContent(HttpParametersUtils.convertHttpParameters(parameters));

        Gdx.net.sendHttpRequest (httpGet, new Net.HttpResponseListener() {
            public void handleHttpResponse(Net.HttpResponse httpResponse) {
                Log.log("http create achievement success");
            }

            public void failed(Throwable t) {
                Log.log("http create achievement failed");
                //do stuff here based on the failed attempt
            }

            @Override
            public void cancelled() {
                Log.log("http create achievement cancelled");
            }
        });
    }

    /**
     * Function to check api status
     * @return
     */
    public boolean checkStatus()
    {
        //TODO: Implement
        return true;
    }


}
```

2.61. ActionResolver

```
package com.juanmartos.games.feedyourbrain.utils;

import com.badlogic.gdx.Net;

public interface ApiInterface {

    void success(Net.HttpResponse response);
```

```java
    void failed();

    void cancelled();

}
```

2.62. CreateWorld

```java
package com.juanmartos.games.feedyourbrain.utils;

import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.BodyDef;
import com.badlogic.gdx.physics.box2d.PolygonShape;
import com.badlogic.gdx.physics.box2d.World;

/**
 * Static function to generate world
 */
public class CreateWorld {

    /**
     * Static function to generate a world
     * @return World world
     */
    public static World createWorld() {

        float width = Var.width;
        float height = Var.height;
        float offsetX = 0;
        float offsetY = 0;

        World world = new World(new Vector2(0, 0), true);

        BodyDef downBodyDef;
        Body downBody;
        PolygonShape downBox;

        BodyDef leftBodyDef;
        Body leftBody;
        PolygonShape leftBox;

        BodyDef rightBodyDef;
        Body rightBody;
        PolygonShape rightBox;

        BodyDef upBodyDef;
        Body upBody;
        PolygonShape upBox;

        downBodyDef = new BodyDef();
        leftBodyDef = new BodyDef();
        rightBodyDef = new BodyDef();
        upBodyDef = new BodyDef();

        downBox = new PolygonShape();
        leftBox = new PolygonShape();
        rightBox = new PolygonShape();
        upBox = new PolygonShape();

        downBodyDef.position.set(new Vector2(width / 2 - offsetX, 0 - offsetY));
        downBody = world.createBody(downBodyDef);

        downBox.setAsBox(width / 2, 1.f);
        downBody.createFixture(downBox, 0.0f);
        downBox.dispose();

        leftBodyDef.position
```

```
                .set(new Vector2(0 - offsetX, height / 2 - offsetY));
        leftBody = world.createBody(leftBodyDef);

        leftBox.setAsBox(1.0f, height / 2);
        leftBody.createFixture(leftBox, 0.0f);
        leftBox.dispose();

        rightBodyDef.position.set(new Vector2(width - offsetX, height / 2
                - offsetY));
        rightBody = world.createBody(rightBodyDef);

        rightBox.setAsBox(0.0f, height);
        rightBody.createFixture(rightBox, 0.0f);
        rightBox.dispose();

        upBodyDef.position.set(new Vector2(width / 2 - offsetX, height
                - offsetY));
        upBody = world.createBody(upBodyDef);

        upBox.setAsBox(width / 2, 0.0f);
        upBody.createFixture(upBox, 0.0f);
        upBox.dispose();

        return world;

    }

}
```

2.63. Formulate

```
package com.juanmartos.games.feedyourbrain.utils;

import java.util.ArrayList;
import java.util.Random;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.CharacterCalc;

/**
 * Formulate provides methods for control to operations in math game
 */
public class Formulate {

    /**
     * General handler of the application
     */
    private FeedYourBrain feedyourbrain;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * List of characters for operation
     */
    private ArrayList<CharacterCalc> characters;

    /**
     * Variable to save the formulate
     */
    private String formulate;

    /**
     * Formulate result in number
     */
    private int iResult;

    /**
     * Formulate result in string
     */
    private String sResult;

    /**
     * Difficulty
     */
    private int difficulty;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *              General handler of the application
     * @param stage
     *              Stage
     */
    public Formulate(FeedYourBrain feedyourbrain, Stage stage) {

        this.feedyourbrain = feedyourbrain;
        this.stage = stage;
        this.characters = new ArrayList<CharacterCalc>();
        this.formulate = "";
        this.iResult = 0;
        this.sResult = "";
        this.setDifficulty(0);
    }

    /**
     * Function to generate formulate
     */
    public void generateFormulate() {

        this.iResult = 0;
        this.sResult = "";

        for (CharacterCalc characterCalc : this.characters) {
            characterCalc.remove();
        }

        this.characters.clear();
        this.formulate = "";

        Random random = new Random();

        int type = random.nextInt(4);

        if (difficulty == 1) {

            if (type == 0) {
                int a = random.nextInt(10);
                int b = random.nextInt(10);
                this.formulate = a + "a" + b + "e";
                this.iResult = (a + b);

            } else if (type == 1) {
                int a = random.nextInt(10);
                int b = random.nextInt(10);
                if (a > b) {
                    this.formulate = a + "s" + b + "e";
                    this.iResult = (a - b);
                } else {
                    this.formulate = b + "s" + a + "e";
                    this.iResult = (b - a);
                }
            } else if (type == 2) {
                int a = random.nextInt(10);
```

```
                int b = random.nextInt(10);
                this.formulate = a + "m" + b + "e";
                this.iResult = (a * b);
            } else if (type == 3) {
                int a = random.nextInt(10) + 1;
                int b = random.nextInt(10) + 1;

                int result = a * b;
                this.formulate = result + "d" + a + "e";
                this.iResult = result / a;

            }

        } else if (difficulty == 2) {

            if (type == 0) {
                int a = random.nextInt(20);
                int b = random.nextInt(20);
                this.formulate = a + "a" + b + "e";
                this.iResult = (a + b);

            } else if (type == 1) {
                int a = random.nextInt(20);
                int b = random.nextInt(20);
                if (a > b) {
                    this.formulate = a + "s" + b + "e";
                    this.iResult = (a - b);
                } else {
                    this.formulate = b + "s" + a + "e";
                    this.iResult = (b - a);
                }
            } else if (type == 2) {
                int a = random.nextInt(10);
                int b = random.nextInt(10);
                int c = random.nextInt(10);
                if (c > a * b) {
                    this.formulate = a + "m" + b + "a" + c + "e";
                    this.iResult = (a * b + c);
                } else {
                    this.formulate = a + "m" + b + "s" + c + "e";
                    this.iResult = (a * b - c);
                }

            } else if (type == 3) {
                int a = random.nextInt(10) + 1;
                int b = random.nextInt(10) + 1;
                int result = a * b;
                int c = random.nextInt(9);

                if(c == a){
                    c++;
                }

                if (c > result / a) {
                    this.formulate = result + "d" + a + "a" + c + "e";
                    this.iResult = result / a + c;
                } else {
                    this.formulate = result + "d" + a + "s" + c + "e";
                    this.iResult = result / a - c;
                }

            }

        } else if (difficulty == 3) {
            if (type == 0) {
                int a = random.nextInt(30);
                int b = random.nextInt(30);
                this.formulate = a + "a" + b + "e";
                this.iResult = (a + b);
            } else if (type == 1) {
                int a = random.nextInt(30);
                int b = random.nextInt(30);
                if (a > b) {
                    this.formulate = a + "s" + b + "e";
                    this.iResult = (a - b);
                } else {
                    this.formulate = b + "s" + a + "e";
                    this.iResult = (b - a);
                }
            } else if (type == 2) {
                int a = random.nextInt(10);
                int b = random.nextInt(10);
                int c = random.nextInt(20);
                if (c > a * b) {
                    this.formulate = a + "m" + b + "a" + c + "e";
                    this.iResult = (a * b + c);
                } else {
                    this.formulate = a + "m" + b + "s" + c + "e";
                    this.iResult = (a * b - c);
                }

            } else if (type == 3) {
                int a = random.nextInt(10) + 1;
                int b = random.nextInt(10) + 1;
                int result = a * b;
                int c = random.nextInt(19);

                if(c == a){
                    c++;
                }

                if (c > result / a) {
                    this.formulate = result + "d" + a + "a" + c + "e";
                    this.iResult = result / a + c;
                } else {
                    this.formulate = result + "d" + a + "s" + c + "e";
                    this.iResult = result / a - c;
                }

            }
        }

        float width = 150;

        Vector2 position;
        Texture texture;
        int separation = 10;
        float offsetX = Var.width * 0.10f;
        float offsetY = Var.height * 0.65f;
        float x = offsetX;
        float y = offsetY;

        for (char character : this.formulate.toCharArray()) {
            position = new Vector2(x, y);
            texture = this.feedyourbrain.getTexture("calc/" + character);
            final CharacterCalc characterCalc = new CharacterCalc(texture,
                    position, character);
            this.characters.add(characterCalc);
            this.stage.addActor(characterCalc);
            x = x + width + separation;
        }

    }

    /**
     * Function to add character
     * @param character
     */
    public void addCharacter(char character) {
```

```java
        //Sacamos las dimensiones de las cartas en base a la pantalla
        float width = 150;
        //float height = Assets.getCharacterHeight(this.feedyourbrain.getWidth());
        int separation = 10;

        int lastIndex = this.characters.size() - 1;
        CharacterCalc lastCharacter = this.characters.get(lastIndex);
        float x = lastCharacter.getX() + width + separation;
        float y = lastCharacter.getY();

        CharacterCalc characterCalc;
        Vector2 position;
        Texture texture;
        position = new Vector2(x, y);
        texture = this.feedyourbrain.getTexture("calc/" + character);
        characterCalc = new CharacterCalc(texture, position, character);
        this.stage.addActor(characterCalc);
        this.characters.add(characterCalc);

        sResult = sResult + character;
    }

    /**
     * Function to check what state the result we are
     * 0 -> No result
     * 1 -> Success result
     * 2 -> Error result
     * @return
     */
    public int checkResult() {

        if (this.sResult.length() > 0) {
            if (Integer.valueOf(this.sResult) == this.iResult) {
                this.sResult = "";
                return 1;
            } else if (this.sResult.length() >= String.valueOf(this.iResult)
                    .length()) {
                return 2;
            }
        }

        return 0;
    }

    /**
     * Set the formulate
     * @param formulate
     */
    public void setFormulate(String formulate) {
        this.formulate = formulate;
    }

    /**
     * Return difficulty
     * @return difficulty
     */
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * Set difficulty
     * @param difficulty
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }
}
```

2.64. Interface

```java
package com.juanmartos.games.feedyourbrain.utils;

public abstract interface Interface {

    void checkDialog();

}
```

2.65. Log

```java
package com.juanmartos.games.feedyourbrain.utils;

import com.badlogic.gdx.Gdx;

/**
 * Log provide a method for print log
 */
public class Log {

    /**
     * Name of the package of the string
     */
    public static String tag = "com.juanmartos.games.feedyourbrain";

    /**
     * Static function to log
     * @param content
     *              Content to print
     */
    public static void log(String content) {
        Gdx.app.log(tag, content);
    }

}
```

2.66. MyPreferences

```java
package com.juanmartos.games.feedyourbrain.utils;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Preferences;

/**
 * MyPreferences provide a method for control the internal variables in application
 */
public class MyPreferences {

    /**
     *  Variable punctuation to control the play mode
     */
    public static String globalScore = "globalScore";

    /**
     * Variable to control the game
     */
    public static String gamePlay = "gamePlay";

    /**
     * Variable to control the difficulty
     */
    public static String difficulty = "difficulty";

    /**
     * Variable to control the sound
 */
public static String sound = "sound";

/**
 * Variable to control the id
 */
public static String id = "id";

/**
 * Variable to represent the game mode
 */
public static int GAMEPLAYMAIN = 0;

/**
 * Variable to represent the game games
 */
public static int GAMEPLAYMODE = 1;

/**
 * Variable to represent the easy difficulty
 */
public static int GAMEPLAYEASY = 1;

/**
 * Variable to represent the normal difficulty
 */
public static int GAMEPLAYNORMAL = 2;

/**
 * Variable to represent the hard difficulty
 */
public static int GAMEPLAYHARD = 3;

/**
 * Function to send the global punctuation
 * @param GlobalScore
 */
public static void setGlobalScore(int globalScore) {
    if (globalScore > MyPreferences.getGlobalScore()) {
        Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
        String value = String.valueOf(globalScore);
        prefs.putString(MyPreferences.globalScore, value);
        prefs.flush();
    }
}

/**
 * Function to query the global punctuation
 * @return GlobalScore overall Punctuation
 */
public static int getGlobalScore() {
    Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
    String value = prefs.getString(MyPreferences.globalScore, "0");
    return Integer.valueOf(value);
}

/**
 * Function to send the game mode
 * @param gamePlay
 */
public static void setGamePlay(int gamePlay) {
    Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
    String value = String.valueOf(gamePlay);
    prefs.putString(MyPreferences.gamePlay, value);
    prefs.flush();
}

/**
 * Return the game mode
 * @return the game mode
 */
public static int getGamePlay() {
    Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
    String value = prefs.getString(MyPreferences.gamePlay, "0");
    return Integer.valueOf(value);
}

/**
 * Function to set difficulty
 * @param difficulty
 */
public static void setDifficulty(int difficulty) {
    Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
    String value = String.valueOf(difficulty);
    prefs.putString(MyPreferences.difficulty, value);
    prefs.flush();
}

/**
 * Return difficulty
 * @return difficulty
 */
public static int getDifficulty() {
    Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
    String value = prefs.getString(MyPreferences.difficulty, "1");
    return Integer.valueOf(value);
}

/**
 * Function to set sound
 * @param sound
 */
public static void setSound(Boolean sound) {
    Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
    String value = String.valueOf(sound);
    prefs.putString(MyPreferences.sound, value);
    prefs.flush();
}

/**
 * Return sound
 * @return sound
 */
public static Boolean getSound() {
    Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
    String value = prefs.getString(MyPreferences.sound, "1");
    return Boolean.valueOf(value);
}


public static void setId(String id) {
    Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
    String value = String.valueOf(id);
    prefs.putString(MyPreferences.id, value);
    prefs.flush();
}


public static String getId() {
    Preferences prefs = Gdx.app.getPreferences(Var.myPackage);
    return prefs.getString(MyPreferences.id, "");
}

/**
 * Function to change sound
 */
public static void changeSound() {
    if (MyPreferences.getSound()) {
        MyPreferences.setSound(false);
    } else {
```

```
                MyPreferences.setSound(true);
            }
        }
}
```

2.67. Operation

```
package com.juanmartos.games.feedyourbrain.utils;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.physics.box2d.Body;
import com.badlogic.gdx.physics.box2d.World;
import com.badlogic.gdx.scenes.scene2d.utils.ActorGestureListener;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.Circle;

/**
 * Operation provides methods for control to operations in visual game
 */
public class Operation {

    /**
     * List of circles
     */
    private ArrayList<Circle> circles;

    /**
     * List of numbers
     */
    public ArrayList<Integer> numbers;

    /**
     * Variable to represent difficulty
     */
    private int difficulty;

    /**
     * Width screen
     */
    private int width;

    /**
     * Height screen
     */
    private int height;

    /**
     * General handler of the application
     */
    private FeedYourBrain feedyourbrain;

    /** Listener Circle Listener**/
    private ActorGestureListener actorGestureListener;

    /**
     * Constructor parameterized
     * @param feedyourbrain
     *                  General handler of the application
     * @param actorGestureListener
     *                  Actor listener
     */
    public Operation(FeedYourBrain feedyourbrain,
            ActorGestureListener actorGestureListener) {

        this.feedyourbrain = feedyourbrain;
    this.actorGestureListener = actorGestureListener;

    this.circles = new ArrayList<Circle>();
    this.numbers = new ArrayList<Integer>();
    this.difficulty = 1;
}

/**
 * Function to generate and set the operation to world
 * @param world
 */
public void setOperation(World world) {

    this.circles.clear();
    this.numbers.clear();

    /**
     * Array of index
     */
    int[] indexNumbers;

    /**
     * Circle aux
     */
    Circle circle;

    /**
     * Variable to generate number
     */
    Random random = new Random();

    /**
     * Body aux
     */
    Body bodyAux;

    int numbers = 0;

    /**
     * Range of numbers to difficulty
     */
    int range = 40;

    if (this.difficulty == 1) {
        numbers = 3;
        range = 40;
    } else if (this.difficulty == 2) {
        numbers = 3;
        range = 50;
    } else if (this.difficulty == 3) {
        numbers = 4;
        range = 99;
    }

    indexNumbers = new int[numbers];

    int number = 0;

    /**
     * Variable to control if repeat any number
     */
    boolean repeat;

    do {

        repeat = false;

        for (int i = 0; i < numbers; ++i) {

            /**
```

```java
             * Create a random number
             */
            number = random.nextInt(range);

            /**
             * Add any negative number
             */
            if (random.nextInt(7) == 1 && this.difficulty != 1) {
                number = number * (-1);
            }

            indexNumbers[i] = number;

        }

        Arrays.sort(indexNumbers);

        for (int i = 1; i < indexNumbers.length; i++) {
            if (indexNumbers[i] == indexNumbers[i - 1]) {
                repeat = true;
                break;
            }
        }

    } while (repeat);

    ArrayList<Vector2> points = this.generatePoints(numbers);

    for (int i = 0; i < numbers; ++i) {

        number = indexNumbers[i];

        circle = new Circle(this.feedyourbrain.getTexture("visual/circle"),
                this.feedyourbrain.getFont("bitstreamcharter60"), number);

        circle.addListener(this.actorGestureListener);

        circle.setBodyDefPosition(points.get(i));

        bodyAux = null;

        /**
         * Create a body in world
         */
        bodyAux = world.createBody(circle.getBodyDef());
        bodyAux.setAngularDamping(0);
        bodyAux.setAngularVelocity(0);

        bodyAux.createFixture(circle.getFixtureDef());
        circle.setCircleDispose();

        int x;
        int y;

        x = 100;
        y = 100;

        if (random.nextInt(2) == 1)
            x = x * (-1);

        if (random.nextInt(2) == 1)
            y = y * (-1);

        bodyAux.setLinearVelocity(x, y);
        bodyAux.setUserData(circle);
        circle.setBody(bodyAux);
        this.numbers.add(number);
        this.circles.add(circle);

    }
```

```java
}

/**
 * Return the list circles
 * @return the list circles
 */
public ArrayList<Circle> getCircles() {
    return this.circles;
}

/**
 * Return the difficulty
 * @return difficulty
 */
public int getDifficulty() {
    return difficulty;
}

/**
 * Set difficulty
 * @param difficulty
 */
public void setDifficulty(int difficulty) {
    this.difficulty = difficulty;
}

/**
 * Return width
 * @return width
 */
public int getWidth() {
    return width;
}

/**
 * Set width
 * @param width
 */
public void setWidth(int width) {
    this.width = width;
}

/**
 * Return height
 * @return height
 */
public int getHeight() {
    return height;
}

/**
 * Set height
 * @param height
 */
public void setHeight(int height) {
    this.height = height;
}

/**
 * Function that checks the number of the circle we clicked
 *
 * @param number
 *                  Number to check
 *
 * @return number
 *                              0 - Error
 *                              1 - Success result
 *                              2 - Success result and finished operation
 */
public int check(Circle circle) {
    int number = circle.getNumber();

    int numberExpected = this.numbers.get(0);

    if (numberExpected == number) {
        if (this.numbers.size() == 1)
            return 2;

        return 1;

    } else {
        return 0;
    }

}

/**
 * Function to clear bodies in world
 * @param world World
 */
public void clearBodies(World world) {
    for (Circle circle : this.circles) {
        world.destroyBody(circle.getBody());
    }
}

/**
 * Return the list numbers
 * @return the list numbers
 */
public ArrayList<Integer> getNumbers() {
    // TODO Auto-generated method stub
    return this.numbers;
}

/**
 * Function to generate points within the screen with a minimum distance of 110px
 * @param nPoints Number of points
 * @return Array points
 */
public ArrayList<Vector2> generatePoints(int nPoints) {

    int offsetX = 200;
    int offsetY = 200;

    int width = Var.width;
    int height = Var.height;

    ArrayList<Vector2> points = new ArrayList<Vector2>();
    boolean ok = true;

    int cont = 0;

    Random random = new Random();

    while (ok) {

        cont++;

        ok = false;
        points.clear();

        for (int i = 0; i < nPoints; ++i) {

            Vector2 point = new Vector2();
            point.x = random.nextInt(width - offsetX);
            //point.x -= offsetX/2;

            point.y = random.nextInt(height - offsetY);
```

```java
                //point.y -= offsetY/2;

                points.add(point);

            }

            for (int i = 0; i < nPoints; ++i) {

                for (int j = i + 1; j < nPoints; ++j) {

                    double distance = this.getDistance(points.get(i),
                            points.get(j));
                    if (distance < 100.0f) {
                        ok = true;
                    }
                }

            }
        }

        return points;

    }

    /**
     * Function to generate the distance between two points
     * @param point1
     * @param point2
     * @return
     */
    public double getDistance(Vector2 point1, Vector2 point2) {
        return Math.sqrt((point1.x - point2.x) * (point1.x - point2.x)
                + (point1.y - point2.y) * (point1.y - point2.y));
    }

}
```

2.68. Operation

```java
package com.juanmartos.games.feedyourbrain.utils;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;
import com.juanmartos.games.feedyourbrain.actors.CharacterCalc;

import java.util.ArrayList;
import java.util.Random;

public class Sequence {

    /**
     * General handler of the application
     */
    private FeedYourBrain feedyourbrain;

    /**
     * Stage
     */
    private Stage stage;

    /**
     * List of characters in sequence
     */
    private ArrayList<CharacterCalc> charactersSequence;

    /**
     * List of characters input to find sequence
```

```java
 */
private ArrayList<CharacterCalc> charactersInput;

/**
 * Variable to save the formulate
 */
private String formulate;

/**
 * Formulate result in number
 */
private int iResult;

/**
 * Formulate result in string
 */
private String sResult;

/**
 * Difficulty
 */
private int difficulty;

/**
 * State for control of sequence/input
 * 0 -> Sequence
 * 1 -> Input
 */
private int state;

/**
 * Control the time of sequence
 */
private float time;

/**
 * Constructor parameterized
 * @param feedyourbrain
 *              General handler of the application
 * @param stage
 *              Stage
 */
public Sequence(FeedYourBrain feedyourbrain, Stage stage) {

    this.feedyourbrain = feedyourbrain;
    this.stage = stage;

    this.charactersSequence = new ArrayList<CharacterCalc>();
    this.charactersInput = new ArrayList<CharacterCalc>();

    this.formulate = "";
    this.iResult = 0;
    this.sResult = "";

    this.setDifficulty(0);
}

/**
 * Function to generate sequence
 */
public void generate() {

    this.state = 0;

    this.iResult = 0;
    this.sResult = "";

    for (CharacterCalc characterCalc : this.charactersInput) {
        characterCalc.remove();
    }
```

```java
        this.charactersInput.clear();
        this.formulate = "";

        Random random = new Random();

        int length = 0;

        if (difficulty == 1) {
            this.time = 1f;
            length = (random.nextInt(2) + 3);
        } else if (difficulty == 2) {
            this.time = 1.5f;
            length = (random.nextInt(2) + 5);
        } else if (difficulty == 3) {
            this.time = 2f;
            length = (random.nextInt(2) + 7);
        }

        this.formulate = this.generateNumber(length);
        this.iResult = Integer.valueOf(this.formulate);

        float width = 150;

        Vector2 position;
        Texture texture;
        int separation = 10;
        float offsetX = (Var.width - (length * 150) - (separation * (length - 1))) / 2;
        float offsetY = Var.height * 0.65f;
        float x = offsetX;
        float y = offsetY;

        for (char character : this.formulate.toCharArray()) {
            position = new Vector2(x, y);
            texture = this.feedyourbrain.getTexture("calc/" + character);
            final CharacterCalc characterCalc = new CharacterCalc(texture,
                    position, character);
            this.charactersSequence.add(characterCalc);
            this.stage.addActor(characterCalc);
            x = x + width + separation;
        }

    }

    public String generateNumber(int length){

        String number = "";

        Random random = new Random();

        for(int i = 0; i < length; ++i){
            number = number + (random.nextInt(9) + 1);
        }

        return number;
    }

    /**
     * Function to add character
     * @param character
     */
    public void addCharacter(char character) {

        if (this.state == 1){

            //Sacamos las dimensiones de las cartas en base a la pantalla
            float width = 150;
            //float height = Assets.getCharacterHeight(this.feedyourbrain.getWidth());
            int separation = 10;
        int lastIndex = this.charactersInput.size() - 1;

        float x = 0;
        float y = Var.height * 0.65f;

        if(lastIndex > -1){
            CharacterCalc lastCharacter = this.charactersInput.get(lastIndex);
            x = lastCharacter.getX() + width + separation;
            y = lastCharacter.getY();
        }

        CharacterCalc characterCalc;
        Vector2 position;
        Texture texture;
        position = new Vector2(x, y);
        texture = this.feedyourbrain.getTexture("calc/" + character);
        characterCalc = new CharacterCalc(texture, position, character);
        this.stage.addActor(characterCalc);
        this.charactersInput.add(characterCalc);

        sResult = sResult + character;


        // Set new position
        x = (Var.width - (this.charactersInput.size() * width)
                - ((this.charactersInput.size() - 1) * separation) ) / 2;

        for (CharacterCalc characterInput:this.charactersInput){

            characterInput.setPosition(x, y);

            x = x + width + separation;
        }

    }
}

/**
 * Function to check what state the result we are
 * 0 -> No result
 * 1 -> Success result
 * 2 -> Error result
 * @return
 */
public int checkResult() {

    if (this.sResult.length() > 0) {
        if (Integer.valueOf(this.sResult) == this.iResult) {
            this.sResult = "";
            return 1;
        } else {

            for (int i = 0; i < this.sResult.length(); ++i){

                if (this.formulate.charAt(i) != this.sResult.charAt(i)){
                    return 2;
                }
            }
        }
    }

    return 0;
}

public void updateTime(float delta) {
    this.time -= delta;

    if (this.time <= 0.0f) {
        this.state = 1;
        this.cleanSequence();
```

```java
        }

    }

    public void cleanSequence(){

        for (CharacterCalc characterCalc : this.charactersSequence) {
            characterCalc.remove();
        }

        this.charactersSequence.clear();
    }

    /**
     * Set the formulate
     * @param formulate
     */
    public void setFormulate(String formulate) {
        this.formulate = formulate;
    }

    /**
     * Return difficulty
     * @return difficulty
     */
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * Set difficulty
     * @param difficulty
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    public int getState() {return state;}

    public void setState(int state) {this.state = state;}

}
```

2.69. Utils

```java
package com.juanmartos.games.feedyourbrain.utils;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;
import com.juanmartos.games.feedyourbrain.FeedYourBrain;

/**
 * Utils provides methods to perform generic actions
 * @author juan
 *
 */
public class Utils {

    /** Code Google Play Games for table games **/
    public static String LEADERBOARDPLAY = "CgkI3ceDxNYIEAIQEg";

    /**
     * Achievement easy math
     */
    public static String STREAKMATHEASY = "easy_math";

    /**
     * Achievement normal math
     */
```

```
public static String STREAKMATHNORMAL = "normal_math";

/**
 * Achievement hard math
 */
public static String STREAKMATHHARD = "hard_math";

/**
 * Achievement easy memory
 */
public static String STREAKMEMORYEASY = "easy_memory";

/**
 * Achievement normal memory
 */
public static String STREAKMEMORYNORMAL = "normal_memory";

/**
 * Achievement hard memory
 */
public static String STREAKMEMORYHARD = "hard_memory";

/**
 * Achievement easy visual
 */
public static String STREAKVISUALEASY = "easy_visual";

/**
 * Achievement normal visual
 */
public static String STREAKVISUALNORMAL = "normal_visual";

/**
 * Achievement hard visual
 */
public static String STREAKVISUALHARD = "hard_visual";

/**
 * Achievement easy association
 */
public static String STREAKASSOCIATIONEASY = "easy_association";

/**
 * Achievement normal association
 */
public static String STREAKASSOCIATIONNORMAL = "normal_association";

/**
 * Achievement hard association
 */
public static String STREAKASSOCIATIONHARD = "hard_association";

/**
 * Achievement easy sequence
 */
public static String STREAKSEQUENCEEASY = "easy_sequence";

/**
 * Achievement normal sequence
 */
public static String STREAKSEQUENCENORMAL = "normal_sequence";

/**
 * Achievement hard sequence
 */
public static String STREAKSEQUENCEHARD = "hard_sequence";

/**
 * Achievement easy weight
 */
```

```
public static String STREAKWEIGHTEASY = "easy_weight";

/**
 * Achievement normal weight
 */
public static String STREAKWEIGHTNORMAL = "normal_weight";

/**
 * Achievement hard weight
 */
public static String STREAKWEIGHTHARD = "hard_weight";

/**
 * Function to get the top right corner based on texture
 * @param texture
 * @return vector2 position of top right corner
 */
public static Vector2 getTopRightCorner(Texture texture) {

    int width = Var.width;
    int height = Var.height;

    float x = width - texture.getWidth() * 1.5f;
    float y = height - texture.getHeight() * 1.5f;

    return new Vector2(x, y);
}

/**
 * Function to get the bottom right corner based on texture
 * @param texture
 * @return vector2 position of top right corner
 */
public static Vector2 getBottomRightCorner(Texture texture) {

    int width = Var.width;
    int height = 0;

    float x = width - texture.getWidth() * 1.5f;
    float y = height + texture.getHeight() * 1.5f;

    return new Vector2(x, y);
}

/**
 * Function to get the top right corner based on texture
 * @param texture
 * @return vector2 position of top left corner
 */
public static Vector2 getTopLeftCorner(Texture texture) {

    int width = 0;
    int height = Var.height;

    float x = width + texture.getWidth() * 1.5f;
    float y = height - texture.getHeight() * 1.5f;

    return new Vector2(x, y);
}

/**
 * Function to get the bottom right corner based on texture
 * @param texture
 * @return vector2 position of top left corner
 */
public static Vector2 getBottomLeftCorner(Texture texture) {

    int width = 0;
    int height = 0;
```

```java
        float x = width + texture.getWidth() * 1.5f;
        float y = height + texture.getHeight() * 1.5f;

        return new Vector2(x, y);
    }

    /**
     * Get code of game based on game and difficulty
     * @param game
     *             Game
     * @param difficulty
     *             Difficulty
     * @return Code game
     */
    public static String getCode(String game, String difficulty) {
        return "code";
    }

    /**
     * Return game in format number
     * @param game
     *             Game in string number
     * @return game in format number
     */
    public static int getGame(String game) {
        if (game.equals("math"))
            return 0;
        if (game.equals("memory"))
            return 1;
        if (game.equals("association"))
            return 3;
        if (game.equals("logic"))
            return 2;
        if (game.equals("visual"))
            return 2;
        if (game.equals("sequence"))
            return 4;
        if (game.equals("weight"))
            return 5;
        return 0;
    }

    /**
     * Return difficulty in format number
     * @param difficulty
     *             Difficulty in string number
     * @return difficulty in format number
     */
    public static int getDifficulty(String difficulty) {
        if (difficulty.equals("easy"))
            return 0;
        if (difficulty.equals("normal"))
            return 1;
        if (difficulty.equals("hard"))
            return 2;
        return 0;
    }

    /**
     * Return the lang of device
     * @return the lang of device
     */
    public static boolean langByDefault() {
        String langS = java.util.Locale.getDefault().toString();

        if (langS.equals("es_ES")) {
            return false;
        }

        return true;
    }

    /**
     * Return the score based on name game and difficulty
     * @param className
     *                  Name of the game
     * @param difficulty
     *                  Difficulty
     * @return
     */
    public static int getScore(int className, int difficulty) {

        int score = 0;

        if (difficulty == 1) {
            score = 10;
        } else if (difficulty == 2) {
            score = 100;
        } else if (difficulty == 3) {
            score = 1000;
        }

        if (className == FeedYourBrain.Screens.MEMORYSCREEN) {
            score = score * 2;
        }

        return score;
    }
}
```

2.70. Var

```
package com.juanmartos.games.feedyourbrain.utils;

/**
 * Var is a static class for represent static variables used in application
 */
public class Var {

    /** Representation red color app **/
    public static float redBackground = 0.0f / 255.0f;

    /** Representation green color app **/
    public static float greenBackground = 127.0f / 255.0f;

    /** Representation blue color app **/
    public static float blueBackground = 127.0f / 255.0f;

    /** Width optimized game **/
    public static int width = 1920;

    /** Game optimized high **/
    public static int height = 1200;

    /** Name of package **/
    public static String myPackage = "com.juanmartos.games.feedyourbrain";

    /** Time game **/
    public static float TIME = 60;
}
```