Developer Guide

CSE 694

SP 2012

Instructor: Dr. Rajiv Ramnath

Team members:

Yu Qiao, Marty Keegan, Lingchen Xiong

INTRODUCTION 3
STAND-OUT FEATURES 3
DEVELOPMENT NOTES 3
REQUIREMENTS IMPLEMENTED 4
USE CASE DIAGRAM 4
USE CASE #1: USER CREATES A NEW ACCOUNT 5
USE CASE #2: USER LOGS IN 5
USE CASE #3: USER SELECTS THE LEVEL TO PLAY 6
USE CASE #4: USER TURNS ON/OFF THE MUSIC 7
USE CASE #5: USER CHECKS THE VELOCITY/ACCELERATION GRAPH 7
USE CASE #7: USER ADDS WEATHER INFORMATION INTO THE GAME 8
STEM APP REQUIREMENTS 10
DOMAIN MODEL DESIGN 12
ACCOUNT 12
GAME 13
BALL 14
COORDINATE 14
WEATHER 15
TIMER 15
DATABASE SCHEMA 15
ACTIVITY AND VIEW DESIGN 16
SPLASH (ACTIVITY) 16
LOGIN (ACTIVITY) 17
LOGIN (LAYOUT VIEW) 18
CREATENEWACCOUNT (ACTIVITY) 19
CREATE_ACCOUNT (LAYOUT VIEW) 20
MENULIST (ACTIVITY) 20
menu (layout view) 21
SELECTLEVEL (ACTIVITY) 21
SELECT_LEVEL (LAYOUT VIEW) 22
LEVELONE,…, LEVELSIX, FREESTYLE (ACTIVITY) 22
SHOWGRAPH (ACTIVITY) 27
COORDINATES (VIEW CLASS) 28
OPTION (ACTIVITY) 29
OPTION (LAYOUT IEW) 30

HELP (ACTIVITY) 30

HELP (LAYOUT VIEW) 31

OTHER CLASSES 32

DATABASEHELPER 32

WEATHER 32

MYHANDLER 33

TIMER 34

PERFORMANCE ANALYSIS 36

APPENDIX 37

JAVA SOURCE CODE 37

Introduction

The “Real Ball” app is designed to correct the misconceptions students have about velocity and acceleration. The app is game-based and consists of six levels in which the player controls a ball on the screen by tilting the phone back and forth. Each level has a different goal. The goals will involve the player fulfilling certain requirements involving the velocity and the acceleration.

After completing a level, the player can view a graph of their velocity and acceleration over time. This will allow them to better understand what took place from a graphical perspective.

Stand-out Features

1. In “Real Ball”, the motion of the ball is based on physics calculation in order to make the movement realistic, giving the students an intuition about the real world. This app is not only for fun but also for education!
2. After the user plays each level, he or she can view a graph showing the velocity and acceleration of the ball over time. This graph is scaled automatically based on the time the user plays.
3. The user can add weather information into the game based on his or her location. The real world wind data will affect the motion of the ball!

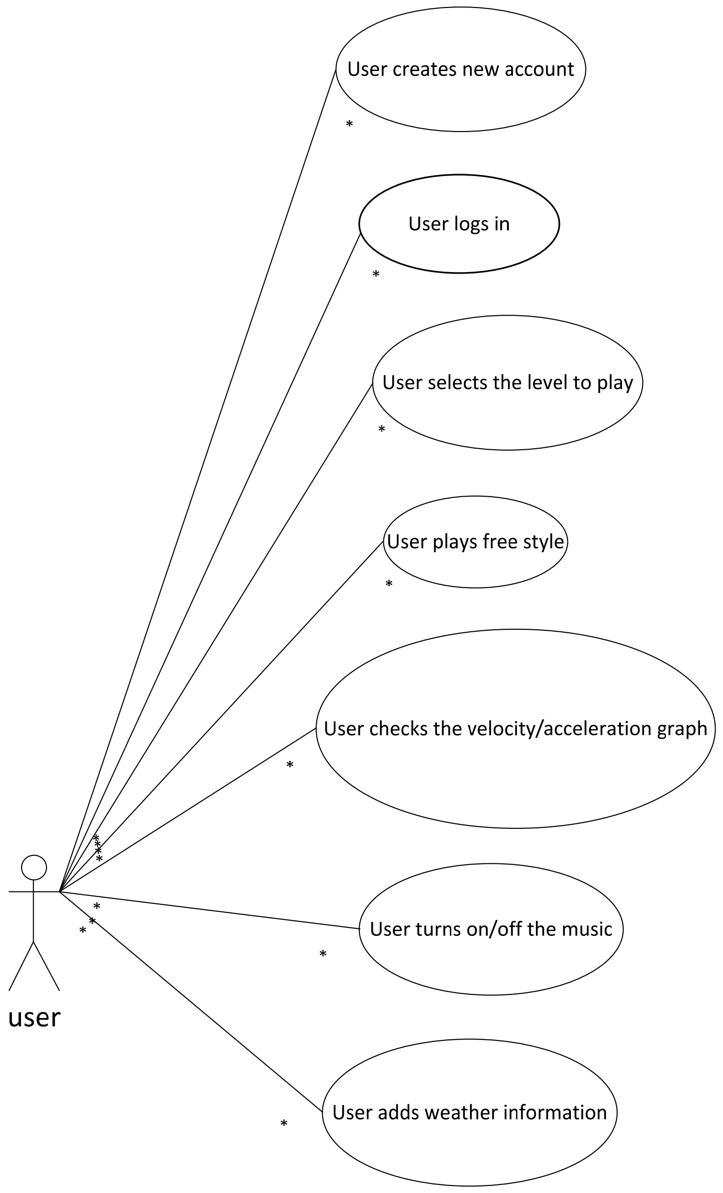
Development notes

- App name: Real Ball
- App version: 1.0
- Minimum SDK version: Google API 10 (Android 2.3.3)
- IDE: Eclipse 3.7.2 (Indigo)
- Test Device: HTC HD2
- Test Device OS: MIUI (Android 2.3.5)

To run the app on you device, you can either install the .apk file of the app directly on you device or import the project into Eclipse and select “Run”->”Run as”->”Android Application”. When using Eclipse to run the app, make sure you have connected your device to your computer.

Requirements implemented

Use case diagram


User creates new account
User logs in
User selects the level to play
User plays free style
User checks the velocity/acceleration graph
User turns on/off the music
User adds weather information
user
*

Use case #1: User creates a new account

Actors

User

Pre-conditions

The user doesn’t have an account for the game.

Post-conditions

The user will have a new account.

Normal flow

1. The user will input the username, password and confirm the password.
2. The user will have a new account for the game.

Alternate flow

If the password doesn’t match the confirm password, the user needs to input them again.

Use case #2: User logs in

Actors

User

Pre-conditions

The user has an account for the game.

Post-conditions

The user will enter the main menu of the game.

Normal flow

1. The user will input his username and password.
2. The user will login the game.

Alternate flow

If there's no record in the database, the user needs to input them again.

Use case #3: User selects the level to play

Actors

User

Pre-conditions

The user has logged into the game.

Post-conditions

The user will stop playing the game

Normal flow

1. The user will select the level to play the game.
2. The user will control the motion of a ball until the ball's motion meets some requirements to win this level.
3. The user will win the game and decide whether he will save the v/a graph of the ball
4. The users will continue to play next level of the game.

Alternate flow

3A3: The user can quit during the game.
3A4: The user can quit instead of continuing to play.

Use case #4: User turns on/off the music

Actors

User

Pre-conditions

The user has logged into the game.

Post-conditions

The device will play or stop playing the music

Normal flow

1. The user will turn on or off the music.

Use case #5: User checks the velocity/acceleration graph

Actors

User

Pre-conditions

The user has logged into the game.

Post-conditions

The user will see the v/a graph.

Normal flow

1. The user will click the “check graph” button.
2. The user will see the v/a he saved before.

Alternate flow

2A2: If the user never saved any graph before, there’s no graph in the system.

Use case #7: User adds weather information into the game

Actors

User

Pre-conditions

The user has logged into the game.

Post-conditions

The user adds weather information into the game.

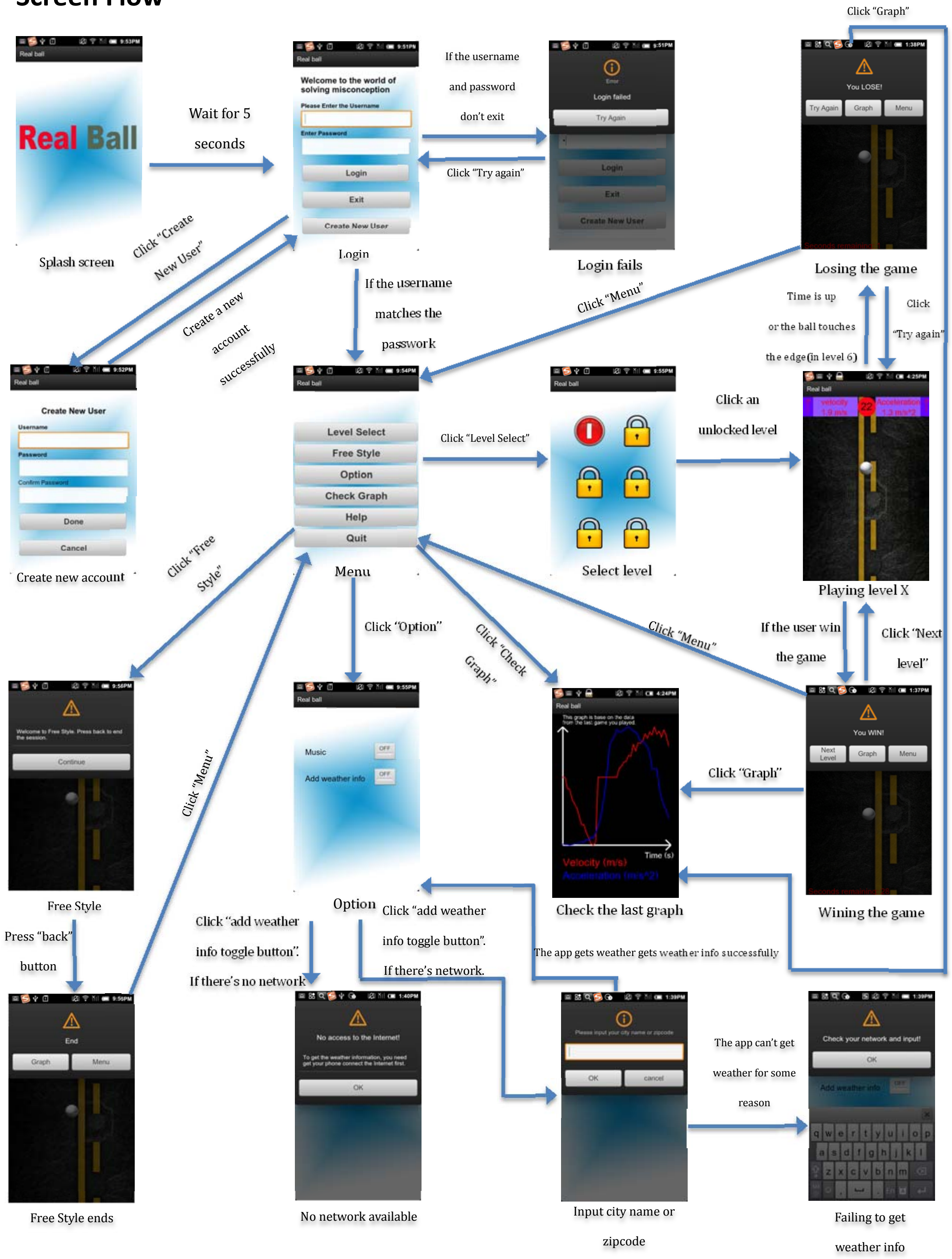
Normal flow

1. The user will be asked to input the Zip code or city name.
2. The app will get the wind speed and add it into the game.

Alternate flow

2A2: If there is not network or the input is not correct, an alert dialog will pop up.

Screen Flow


Real Ball
Splash screen
Wait for 5 seconds
Welcome to the world of solving misconception
Please Enter the Username
Enter Password
Login
Exit
Create New User
Login
If the username and password don't exit
Click "Try again"
Error
Login failed
Try Again
Login fails
Click "Graph"
You LOSE!
Try Again
Graph
Menu
Losing the game
Time is up or the ball touches the edge(in level 6)
Click "Try again"
Click "Create New User"
Create a new account successfully
If the username matches the password
Click "Menu"
Create New User
Username
Password
Confirm Password
Done
Cancel
Create new account
Level Select
Free Style
Option
Check Graph
Help
Quit
Menu
Click "Level Select"
Select level
Click an unlocked level
Playing level X
If the user win the game
Click 'Next level"
Click "Free Style"
Click "Option"
Click "Check Graph"
Click "Menu"
Welcome to Free Style. Press back to end the session.
Continue
Free Style
Press "back" button
Click "Menu"
Music
Add weather info
Option
This graph is base on the data from the last game you played
Velocity (m/s)
Acceleration (m/s^2)
Time (s)
Check the last graph
Click "Graph"
You WIN!
Next Level
Graph
Menu
Wining the game
Click "add weather info toggle button". If there's no network
Click "add weather info toggle button". If there's network.
The app gets weather gets weather info successfully
End
Graph
Menu
Free Style ends
No access to the Internet!
To get the weather information, you need get your phone connect the Internet first.
OK
No network available
Please input your city name or zipcode
OK
cancel
Input city name or zipcode
The app can't get weather for some reason
Check your network and input!
OK
Failing to get weather info

STEM App Requirements

1. Must have a UI

   All activities in our project have a UI such as the login interface. There are several buttons and text words to guide the user to finish the login process. It would be very easy for the user to use this application.

2. Must have a rich enough set of domain objects

   Many domain objects are shown in the diagram and analysis.

3. Must have data that is persistent across sessions

   We have a database called ball.db which uses SQLite3 to store an Account table with the user's information such as username, password, and current level information. The user can create his username and password with a default game level in the CreateAccount activity and then use them in the LogIn activity and each Level activity. Also, at the end of each level, user's information would be updated.

4. Must use one or more external services – such as maps

   We get the weather information by entering ZIP code or City name as the external service. The main weather information we obtain is the wind speed. We have added wind speed in several levels of the game so that the user has to consider wind speed as a factor to win those levels.

5. Must use one or more sensors – GPS, accelerometer, light etc.

   Our application is mainly based on the accelerometer so that the user can move the ball up and down to reach the objective of each level. The speed of the ball is based on how the user does with the phone so the accelerometer can work.

6. Must incorporate techniques for resiliency against crashes, screen orientation changes, application being killed by the OS etc.

   We set portrait orientation with every activity so that the application would not be rotated. In order to improve resiliency against crashes, we rewrite the onPause() and onResume() functions in this application. With these functions, the user can go back the game and resume his game status after encountering some special cases such as a phone call.

7. Must have suitable logging incorporated for debugging purposes

Log cat in eclipse was used for debugging in various places. Several logs were put in various sections to discover the root of problems and troubleshoot them. One example is in LevelOne: We were having issues getting the game to pause properly and getting onCreate, onPause, and onResume to work together correctly. We put a log statement in onResume to see when it was being called. After tracing what was happening we realized we could simply delete the onResume method all together.

Another example is when we were having issues with a level being beaten. We put log statements within if(levelconditions) to see if the level conditions were ever being met.

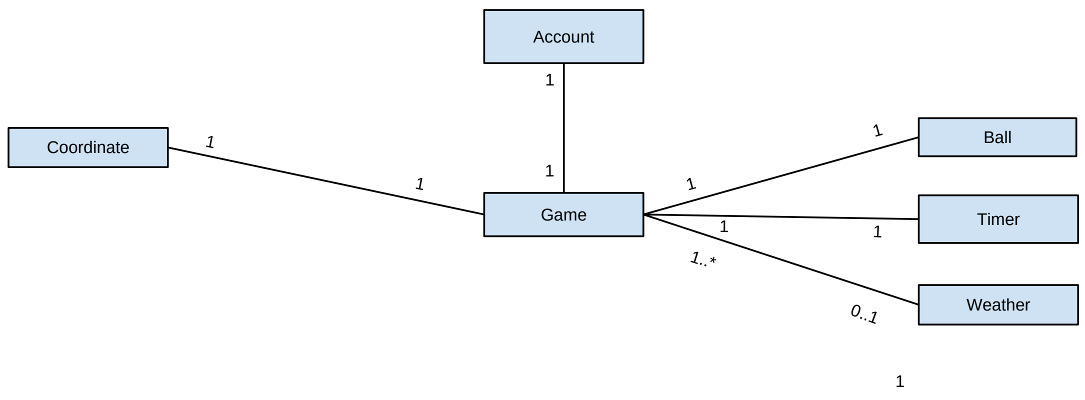
8. Must be profiled and analyzed for performance

See section "Performance analysis".

9. Must be well-designed (as explained in this course) and documented

See diagrams for high level design. See source code for thorough documentation and in depth design.

Domain Model Design



Account

Handles

- Managing the users' accounts
- Allowing user to create a new a account
- Allowing user to login the game

Collaborators

- Game

Implementation class

The account is implemented by three classes: *DatabaseHelper*, *Login* and *CreateNewAccount*. *DababaseHelper* deals with the database directly. It creates new table and provides some public methods for inserting, updating and deleting the records.

Login and *CreateNewAccount* are both activities. They do just what their names describe.

Game

Handles

- Representing a game for user
- Showing the ball and the state (speed and acceleration) of it
- Determining the wining and losing conditions
- Recording the ball's speed and acceleration for drawing the graph

Collaborators

- Ball
- Coordinate
- Timer

Implementation class

There are six levels and a free style level in the game. Each individual level has been implemented as its own class (LevelOne, LevelTwo, etc…) Whether or not a level has been unlocked is decided by the nowLevel variable stored in the Login class. This variable refers to the highest level the user has currently unlocked. Upon beating that level, nowLevel is increased by one. The level classes themselves do not keep track of which level is unlocked. This is kept track of in the Login class and enforced in the SelectLevel class.

Each level has its own unique levelConditions variable which refers to the conditions the user must satisfy in order to complete the level. These conditions are hard coded into each individual level class based on the goal of that level.
All levels share an array for velocity points and an array for acceleration points. These arrays are checked when the user views a graph. They are cleared at the start of any level.

Levels four, five, and six are wind levels. The wind code is added into these levels. There will be a default wind value if the user does not enter in a zip code to get weather data. Otherwise, the wind will be based on the weather data.
The freestyle level is implemented just as the other levels were. It is its own class and can be called from the main menu. There is no goal in this level and it ends when the user presses the back button.

Ball

Handles

- Representing a ball in the game
- Calculating the ball's position, speed and acceleration.

Collaborators

- Game

Implementation class

The ball is implemented by class *Ball.* It is a subclass in every level's class. It represents the ball in the game.

Coordinate

Handles

- Drawing a coordinate system to show velocity-time and acceleration-time of the ball.

Collaborators

- Game

Implementation class

Coordinate is implemented by class *Coordinate* and *ShowGraph*. *Coordinate* extends *View* class. It draws a coordinate system and provides a public method to load data into it and draw the data in the system. *ShowGraph* gets the speed and velocity points from two arrays that are populated in each of the level classes and passes the data of velocity and acceleration into *Coordinate*. These arrays are cleared when a level starts so they will be freshly created based on the current play.

Weather

Handles

- Get weather information via internet

Collaborators

- Game (the motion of the ball is affected by the weather condition)

Implementation class

Weather is implemented by class *Weather* and *Myhandler. Weather* provides public methods to load Zip code or city name into it and make a URL to use Google Weather API to get weather information. The URL returns a XML file and *Weather* uses *Myhandler* to parse the XML file. *Weather* also provides public methods returning the wind speed and direction.

Timer

Handle

- Count down the time

Collaborators

- Game

Implementation class

Timer is implemented by class *Timer.* It extends *android.os.CountDownTimer* and overrides some important methods.

Database schema

<u>id</u>	Username	Password	Level

In this app, there is only one table in the database. It is used for storing the information of users. It has four columns:

- id – Primary key generated automatically.
- Username – The username of the user.
- Password – The password of the user.
- Level – The highest level the user has reached.

Activity and View Design

Splash (activity)

This activity is the starting point of this app. It shows user the name of the app with animation and wait for 5 seconds. Then it will launch *Login* activity.

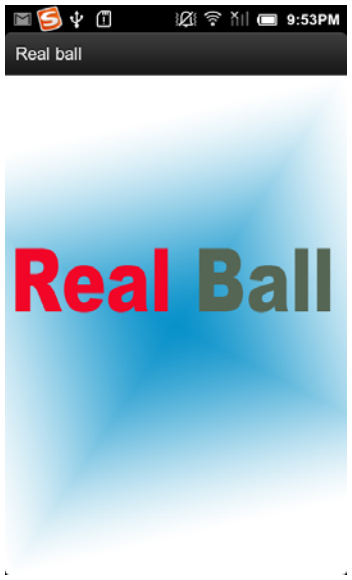
Important fields

- *protected int splashTime = 5000;*
  This is the time that the screen waits for.

Important methods

- *protected void onCreate(Bundle savedInstanceState)*
  This method is called once this activity is launched. This method loads the layout view, loads and starts the animation, and creates a thread to countdown time. When time is up, it will finish itself and launch the *Menulist* activity.

splash (layout view)



This layout view is implemented by XML file. It is a linear layout with blue gradient background. This background is used in several activities. In the linear layout, there is a TextView showing the name of the app. This TextView comes up with animation.

Login (activity)

This activity is launched after the "Splash" activity. User inputs his username and password in order to login the game. User can also click "create new account" to launch a new activity to create a new account.

Important fields

- *private EditText usernameEditableField*
  This field is used to get the string of username.

- *private EditText passwordEditableField*
  This field is used to get the string of password.

- *public static int nowLevel*
  This field stores the highest the current user has reached. It's defined as public static type in order to be shared with other activity.

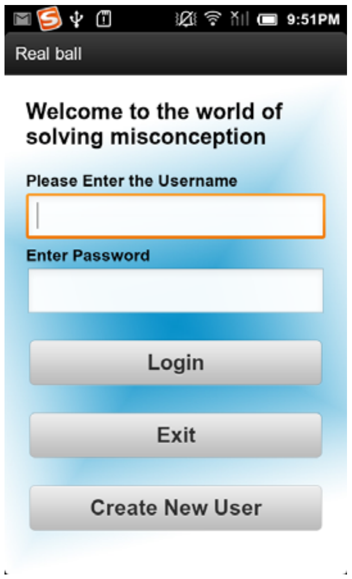
- *public static string nowUserName*
  This field stores the username of current user.

Important methods

- *public void onCreate(Bundle savedInstanceState)*
  This method is called when this activity starts. This method loads the layout view. There are two edit text area displayed for the user to enter the information he needs to login the game.

- *public void checkLogin()*
  This method is used to compare the login information (username and password) that the user enters with the user information stored in the database in order to check whether the user has entered the correct username and password.

- *public void onClick(View)*
  This method is used to listen to which button of the layout has been clicked and then calls the related one.


9:51PM
Real ball
Welcome to the world of solving misconception
Please Enter the Username
Enter Password
Login
Exit
Create New User

login (layout view)

This layout view is implemented by XML file. It is a linear layout with some

TextView, EditText and Button in it.

CreateNewAccount (activity)

This activity is launched when user clicks "Create New User" button in "Login" activity. User inputs username and password and confirms the password in order to create a new account.

Important fields

- *private EditText etUsername*
  This field is used to get the string of username.

- *private EditText etPassword*
  This field is used to get the string of password.

- *private EditText etConfirm*
  This field is used to compare the confirm password with the password in order to make sure that the user enter the correct password.

- *private DatabaseHelper dh*
  This field is used to create a DatabaseHelper object.

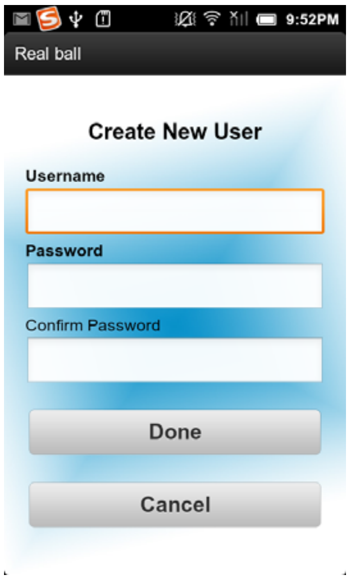
Important methods

- *public void onCreate(Bundle savedInstanceState)*
  This method is called when this activity starts. This method loads the layout view. There are three edit text area displayed for the user to enter the information he wants to create in the database.

- *private void CreateAccount()*
  This method is used to create an account with information of username, password and level of the user. This method can also check whether the user enters valid username or password.

- *public void onClick(View)*
  This method is used to listen to which button of the layout has been clicked and then calls the related one.

create_account (layout view)



This layout view is implemented by XML file. It is pretty same as login's view: a linear layout with some TextView, EditText and Button in it.

Menulist (activity)

This activity is launched when user logins successfully. It shows user the main menu of the app.

Important fields

- *private Button levelSelect*
- *private Button freeStyle*
- *private Button option*
- *private Button checkGragh*
- *private Button help*
- *private Button quit*

These six fields represent the six buttons on the menu's view.

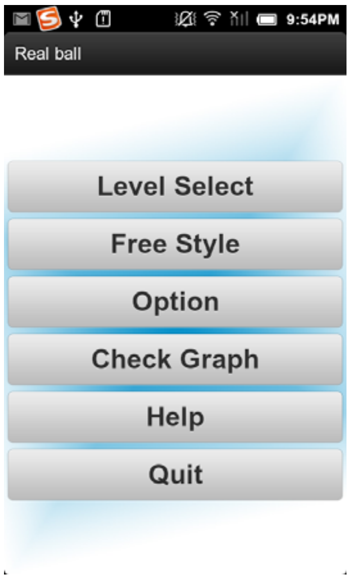
- *public static MediaPlayer mp = new MediaPlayer()*
  This field is the handle of the music player.

Important mathods

- *public void onCreate(Bundle savedInstanceState)*

  This method is called when this activity is launched. The method loads the layout view, stars playing the background music, and sets the on click listener for the buttons. Each button corresponds an activity. This method sends an intent to another activity based on what the user has clicked. Each of the activities that can be called from menu have been implemented as their own class, such as LevelSelect and Help.

menu (layout view)



This layout view is implemented by XML file. This is a linear layout with six Button in it.

SelectLevel (activity)

This activity is launched when user clicks "Level Select" button in "MenuList" activity. It allow user to select levels based on which level the user has reached.

Important fields

- *private final int INVISIBLE = 4*

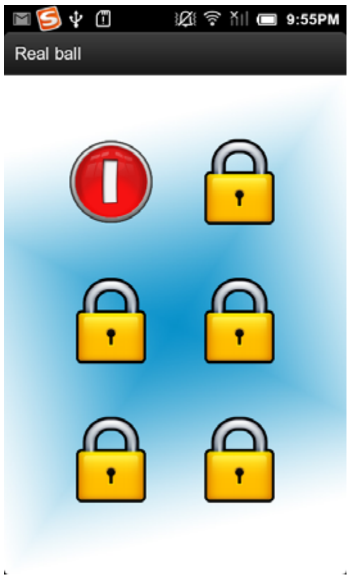
  This is a fixed value passed into an ImageButton's method set it to invisible.

Important methods

- *protected void onCreate(Bundle savedInstanceState)*

  This method is called when the activity is launched. It loads the layout view and the lock icons' and level icons' visibility. For example, if the highest level the user has reached is 1, the icons of levels other than 1 are all hidden and the lock icons are visible.

select_level (layout view)



This layout view is implemented by XML file. It is a relative layout. We use relative layout because it's easy to align the icons.
There are six levels in the game. For every level (except level 1), there are two icons placed in the layout. One is a number icon and the other one is a lock icon. The icons' visibility depends on the highest level the use has reached.

LevelOne,..., LevelSix, Freestyle (activity)

The code for the levels contains subclasses.

Important fields

- *public static ArrayList<Float> speedList*

  This list keeps track of all the speed points over time in order to graph them later.

- *public static ArrayList<Float> accelerationList*

  This list keeps track of all the acceleration points over time in order to graph them later.

Important methods

- *public void onCreate(Bundle savedInstanceState)*

  It is called when the level is created. This method initializes the speed and velocity arrays, gets instances for managers of the services, and begins the level by displaying the pop-up instruction dialog.

- *protected void onPause()*

  This method handles what happens if the player is interrupted during play. It stops the simulation and creates a pop-up pause dialog with a button the user must click in order to resume play.

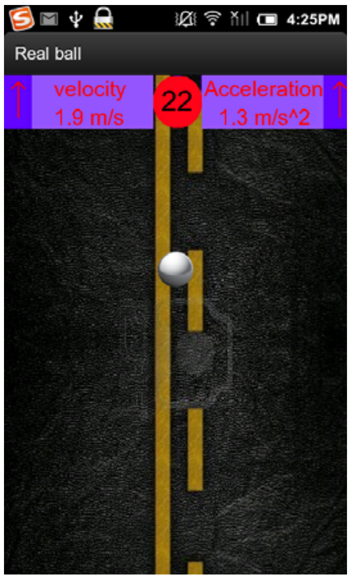
- *public void onBackPressed()*

  This method handles what happens when the user presses the back button on his or her phone. It redirects the user to the main menu.

- *public void finishLevel()*

  This method is called when the level conditions have been met. It creates a pop-up dialog allowing the user to continue to the next level, view the graph, or return to the menu.

Subclass: *class **SimulationView** extends View implements SensorEventListener*



Important methods:

- *private Sensor mAccelerometer*
  The sensor for the accelerometer.

- *private float mXDpi*
- *private float mYDpi*
- *private float mMetersToPixelsX*
- *private float mMetersToPixelsY*
  These are all used for scaling the objects drawn on the screen.

- *private float mXOrigin*
  The x axis origin on the screen.

- *private float mYOrigin*
  The y axis origin on the screen.

- *private float mSensorX*
  It is used in getting changes in the accelerometer.

- *private float mSensorY*
  It is used in getting changes in the accelerometer.

- *private float mHorizontalBound*
  Bound used to represent the horizontal edges of the screen.

- *private float mVerticalBound*
  Bound used to represent the vertical edges of the screen.

- *private final ParticleSystem mParticleSystem*
  The particle system created and used in our simulation.

- *private Boolean levelConditions*

Used to represent the specific conditions of a level. When this is true, the conditions have been met.

Important methods:

- *public void startSimulation()*
  This method simply begins our simulation.

- *public void stopSimulation()*
  This method simply stops our simulation.

- *public SimulationView(Context context)*
  This method takes care of retrieving and scaling the images we need to draw.

- *protected void onSizeChanged(int w, int h, int oldw, int oldh)*
  This method overrides onSizeChanged to make sure it gets the bounds so our ball does not go off screen.

- *public void onSensorChanged(SensorEvent event)*
  This method returns information about how the accelerometer changed (how the phone moved).

- *protected void onDraw(Canvas canvas)*
  This method draws everything on our screen.

Subclass: *class **Particle***

Important fields

- *private float mPosX*
  The x position of the ball (in case we wanted to make the game 2 dimensional in the future).

- *private float mPosY*
  The y position of the ball.

- *private float mAccelX*
  The x acceleration of the ball (in case we wanted to make the game 2 dimensional in the future).

- *private float mAccelY*
  The y acceleration of the ball.

- *private float mLastPosX*
  The previous x position of the ball (in case we wanted to make the game 2 dimensional in the future).

- *private float mLastPosY*

The previous y position of the ball.

- *private float mOneMinusFriction*
  The strength of friction (in case we wanted to add differing frictions in the future).

Important methods

- *public void computePhysics(float sx, float sy, float dT, float dTC)*
  This method is used to compute where the ball should be and compute its acceleration.

- *public void resolveCollisionWithBounds()*
  This method is used to prevent the ball from rolling off screen and instead stop it when it hits a wall.

Subclass: *class* ***ParticleSystem***

Important fields

- *static final int NUM_PARTICLES*
  This field represents the number of particles, allowing us to increase to multiple balls at a time if we wanted in the future.

- *private Particle mBalls[]*
  This simply creates the amount of particles specified in NUM_PARTICLES. Again, for our current purposes this will always create just 1 ball.

Important methods

- *ParticleSystem()*
  This is the constructor. It initializes particles in the system.

- *private void updatePositions(float sx, float sy, long timestamp)*
  This method updates the position of the balls based on a call to computePhysics from the particle class.

- *public void update(float sx, float sy, long now)*
  This method updates the positions of the ball(s) by calling updatePositions. This

method also takes care of calling resolveCollisionWithBounds() if a ball collides with a wall. And this method will take care of different balls colliding with each other in the case of having more than one ball on the screen at a time.

- *public int getParticleCount()*

This method returns the number of particles that are in the system. Again, currently it will always be 1.

- *public float getPosX(int i)*

This gets the x position of the specified particle.

- *public float getPosY(int i)*

This gets the y position of the specified particle.

ShowGraph (activity)

This activity is launched when user clicks “show graph” either in “Menulist” activity or after wining a game.

Important fields

- *private Coordinates graph*

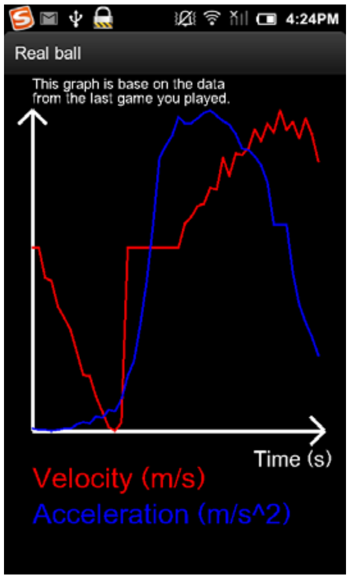
It is a *Coordinates* object.

Important methods

- *protected void onCreate(Bundle savedInstanceState)*

This method is called when the activity is launched. It create a new instance of *Coordinate* and passes the velocity and acceleration into it.

Coordinates (view class)


4:24PM
Real ball
This graph is base on the data from the last game you played.
Time (s)
Velocity (m/s)
Acceleration (m/s^2)

Important fields

- *private final float left = 40f;*
- *private final float right = 450f;*
- *private final float top = 50f;*
- *private final float bottom = 500f;*

These variable is used to determine the position of X-Y axis.

- *private List<Float> time = new ArrayList<Float>();*
  This list stores the X-axis coordinate values of a point.

- *private List<Float> acceleration = new ArrayList<Float>();*
  This list stores the Y-axis coordinate values of an acceleration point.

- *private List<Float> velocity = new ArrayList<Float>();*
  This list stores the Y-axis coordinate values of a velocity point.

Important fields

- *protected void onDraw(Canvas canvas)*

This method draws everything on the screen, including coordinate system, texts, and the data. There are two sets of points. One is for velocity and the other is for acceleration. Every point has coordinate values for both X-axis and Y-axis. X-axis represents time. Y-axis represents the actual value of velocity or acceleration.

- *public void setData(List<Float> v, List<Float> a)*

This method converts two lists – velocity (v) and acceleration (a) – to three lists: *time, velocity* and *acceleration* for drawing the points on the coordinate system*. time* is a list storing the X-axis coordinate values for both acceleration and velocity points. It is based on the number of data in a and v (they have same number of data). *velocity* and *acceleration* are lists storing the Y-axis coordinate values for velocity and acceleration points. They are scaled based on the maximum and minimum value in v and a. The X-axis coordinate value of maximum value in v or a is placed on the top of the coordinate system. The X-axis coordinate value of minimum value in v or a is placed on the bottom of the coordinate system.

After the converting, the method will call *invalidate()* to redraw the whole view.

Option (activity)

This activity is launched when user click “Option” button in “MenuList” activity. It allow user to turn off/on the background music and add/remove weather information in the game.

Important fields

- *private ToggleButton tbMusic, tbWeather*

  These two variables represent the two toggle buttons.

Important methods

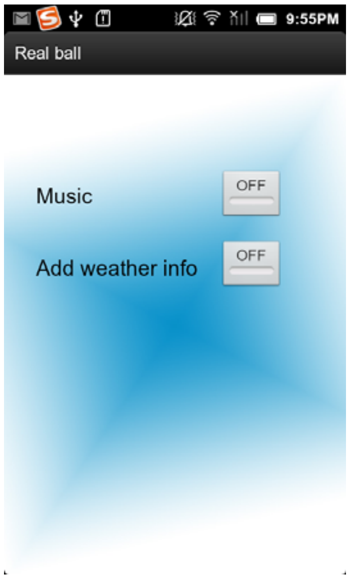
- *protected void onCreate(Bundle savedInstanceState)*

This method is called when the activity is launched. It loads the layout view and sets on checked listeners for the two toggle buttons.

- *private boolean hasNetworkConnection()*

  This method is used for checking whether the device has connected the network (GPRS, 3G, WIFI).

option (layout iew)



This layout view is implemented by XML file. It is a relative layout plus two TextView and two ToggleButton in it.

Help (activity)

This activity is launched when user clicks “Help” button in “MenuList” activity. It shows the description of this app.

Important fields

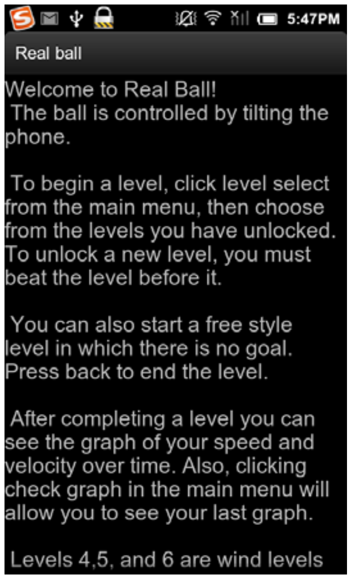
None

Important methods

- *protected void onCreate(Bundle savedInstanceState)*

This method is called when the activity is launched. It loads the layout view.

help (layout view)



Welcome to Real Ball!
 The ball is controlled by tilting the phone.

 To begin a level, click level select from the main menu, then choose from the levels you have unlocked. To unlock a new level, you must beat the level before it.

 You can also start a free style level in which there is no goal. Press back to end the level.

 After completing a level you can see the graph of your speed and velocity over time. Also, clicking check graph in the main menu will allow you to see your last graph.

 Levels 4,5, and 6 are wind levels

This layout view is implemented by XML file. It is a linear layout containing only a TextView.

Other Classes

DatabaseHelper

Important fields

- *private static final string DATABASE_NAME*
  This field is used to set the name of the database.

- *private static final string TABLE_NAME*
  This field is used to set the name of the table.

Important methods

- *public DatabaseHelper(Context context)*
  This is the constructor. This method returns a SQLiteOpenHelper object. With the SQLiteOpenHelper class, the user can use some simple methods to open, close and create database.

- *public long insert(String name, String password)*
  This method is used to insert username, password and level information into the table.

- *public void updateLevel(String username)*
  This method is used to update level information of the user. Each time user finishes one level, his level information is check and his level will increase by one if needed.

- *public void deleteAll()*
  This method is used to delete the table.

- *public List<String> selectAll(String username, String password)*
  This method is used to select the user information from the table.

Weather

This class is used for obtain the weather information.

Important fields

- *private static boolean hasWeather*
  This field represents weather the app has got weather information yet.

- *private static String dir;*
  This field is used to store the wind direction. It can be "W", "E", "N", "S", "NE" and so on.
- *private static String speed;*
  This field is used to store the wind speed. It does not include the unit(mph).

Important methods

- *Weather(String zipcodeOrCity)*
  This is the constructor. It need Zip code or city's name to pass into it. When it is called, it will make an URL inquiry to use Google Weather API. For example, if we want get the weather of Columbus, we will make an URL*: "http://www.google.com/ig/api?weather=Columbus".* If succeed, the inquiry will return a XML file containing the weather condition of Columbus. When this XML file returns, it will be parsed and the wind speed and direction will be retrieved.

- *public static boolean hasWeather()*
  This method returns the value of hasWeather.

- *public static String get_dir()*
  This method returns the value of *dir.*

- *public static String get_speed()*
  This method returns the value of *speed.*

- *public static void allow ()*
  This method sets hasWeather to *"true".*

- *public static void prohibit ()*
  This method sets hasWeather to *"false".*

MyHandler

This class extends org.xml.sax.helpers.DefaultHandler which is SAX parser.

Important fields

- *private String dir*
  This field stores the wind direction parsed from XML.

- *private String speed;*
  This field stores the wind speed parsed from XML.

Important methods

- *public void startElement(String uri, String localName, String qName,Attributes attributes)*
  This is an overridden method. It tells the parser what to do when a start tagis reached.

- *public void endElement(String uri, String localName, String qName)*
  This is an overridden method. It tells the parser what to do when an end tag is reached.

Timer

This class extends android.os.CountDownTimer.

Important fields

- *private long timeRemaining;*
  This field stores the remaining time.

- *private boolean hasFinished;*
  This field represents weather the countdown timer has finished.

Important methods

- *public Timer(long millisInFuture, long countDownInterval)*
  This is the constructor. *millisInFuture (unit: millisecond)* is the total countdown time. *countDownInterval (unit: millisecond)* is the countdown interval time. When the timer stars, after each interval, *onTick()* will be triggered.

- *public void onFinish()*
  This is an overridden method. It is called when time is up. It sets *hasFinished* to "true".
- *public void onTick(long millisUntilFinished)*
  This is an overridden method. It is called after each interval. *millisUntilFinished* (unit: millisecond) is the remaining time. It is set automatically. This methods updates *timeRemaining*.

- *public long getTimeRemaining()*
  This method returns *timeRemaining.*

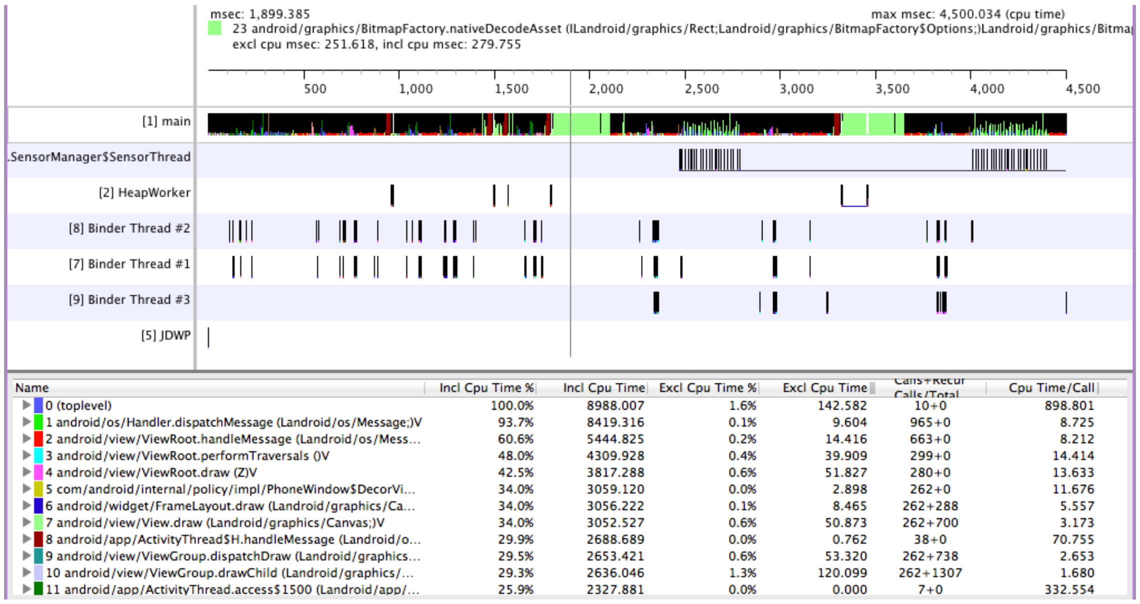
- *public boolean hasFinished()*
  This method returns *hasFinished*.

Performance analysis

We use Profiler in Eclipse to analyze the performance of the app. The steps to do that are shown below:

1. Do: Window->Show View->Other->Devices
2. Start the app.
3. Navigate to start of profiling scenario.
4. Start Method Profiling.
5. Execute several profiling scenario.
6. Stop Method Profiling.

The figure below shows the analysis result.



msec: 1,899.385 max msec: 4,500.034 (cpu time)
🟩 23 android/graphics/BitmapFactory.nativeDecodeAsset (ILandroid/graphics/Rect;Landroid/graphics/BitmapFactory$Options;)Landroid/graphics/Bitma
excl cpu msec: 251.618, incl cpu msec: 279.755

Name	Incl Cpu Time %	Incl Cpu Time	Excl Cpu Time %	Excl Cpu Time	Calls+Recur Calls/Total	Cpu Time/Call
▶ 🟦 0 (toplevel)	100.0%	8988.007	1.6%	142.582	10+0	898.801
▶ 🟩 1 android/os/Handler.dispatchMessage (Landroid/os/Message;)V	93.7%	8419.316	0.1%	9.604	965+0	8.725
▶ 🟥 2 android/view/ViewRoot.handleMessage (Landroid/os/Mess…	60.6%	5444.825	0.2%	14.416	663+0	8.212
▶ 🟦 3 android/view/ViewRoot.performTraversals ()V	48.0%	4309.928	0.4%	39.909	299+0	14.414
▶ 🟪 4 android/view/ViewRoot.draw (Z)V	42.5%	3817.288	0.6%	51.827	280+0	13.633
▶ 🟨 5 com/android/internal/policy/impl/PhoneWindow$DecorVi…	34.0%	3059.120	0.0%	2.898	262+0	11.676
▶ 🟦 6 android/widget/FrameLayout.draw (Landroid/graphics/Ca…	34.0%	3056.222	0.1%	8.465	262+288	5.557
▶ 🟩 7 android/view/View.draw (Landroid/graphics/Canvas;)V	34.0%	3052.527	0.6%	50.873	262+700	3.173
▶ 🟥 8 android/app/ActivityThread$H.handleMessage (Landroid/o…	29.9%	2688.689	0.0%	0.762	38+0	70.755
▶ 🟦 9 android/view/ViewGroup.dispatchDraw (Landroid/graphics…	29.5%	2653.421	0.6%	53.320	262+738	2.653
▶ ⬜ 10 android/view/ViewGroup.drawChild (Landroid/graphics/…	29.3%	2636.046	1.3%	120.099	262+1307	1.680
▶ 🟩 11 android/app/ActivityThread.access$1500 (Landroid/app/…	25.9%	2327.881	0.0%	0.000	7+0	332.554

According to the result, BitmapFactory.nativeDecodeAsset consumes a lot. This method is used to convert the image into bit format. For example, we use this method to convert the ball's image into bit format so that it can be drawn on the view. To minimize the consumption, we use this method convert the image before drawing and store it into a variable. When redrawing the view, we use the variable instead of using the method.

Appendix

Java source code