In The God Name



**Software Graduation Project**

*Prepared by:*

Maysoon Ashayer & Muna Sholi

*Submitted to*

Dr.Luai Malhis, Dr.Raed Al-Qadi and Miss.Haya Samaanh

A sample Report Submitted in Partial Fulfillment of the Requirements of Principles ofSoftware Graduation Project

Department of Computer Engineering

Faculty of Engineering

An-Najah National University

*Monday, December17, 2012*

**Table of Contents:**

[Chapter 1 Introduction 5](#_Toc343423826)

[1.1 Overview 5](#_Toc343423827)

[1.2 Existing problems 5](#_Toc343423828)

[1.3 Motivation for Carrying out the Project 5](#_Toc343423829)

[1.4 Main Contribution 6](#_Toc343423830)

[1.5 Report Organization 6](#_Toc343423831)

[Chapter 2 Methodology 6](#_Toc343423832)

[Chapter 3 Requirements and analysis 8](#_Toc343423833)

[3.1 Functional Requirements: 8](#_Toc343423834)

[3.2 Non-Functional Requirements: 9](#_Toc343423835)

[Chapter 4 Background 9](#_Toc343423836)

[4.1 Game components: 9](#_Toc343423837)

[4.2 AndEngine: 10](#_Toc343423838)

[4.3 Box2d Physics Engine: 14](#_Toc343423839)

[Chapter 5 Bubble Shooter Game 16](#_Toc343423840)

[Chapter 6 Shoot Game 19](#_Toc343423841)

[Chapter 7 Pool Game 23](#_Toc343423842)

[Chapter 8 Conclusion 26](#_Toc343423843)

[Chapter 9 References 27](#_Toc343423844)

[Chapter 10 Appendix 28](#_Toc343423845)

**List of Figures:**

[Figure 1: Pop-up menu in bubble shooter game 13](#_Toc343380091)

[Figure 2: Static menu in bubble shooter game 13](#_Toc343380092)

[Figure 3: flowchart of main functionality of Bubble Shooter game 17](#_Toc343380093)

[Figure 4: Bubble shooter game 19](#_Toc343380094)

[Figure 5: flowchart shows how to play Shoot game 20](#_Toc343380095)

[Figure 6: First Level in the game 21](#_Toc343380096)

[Figure 7: Second Level in shoot game 22](#_Toc343380097)

[Figure 8: Third Level in shoot game 22](#_Toc343380098)

[Figure 9: Billiard Game Scene 23](#_Toc343380099)

[Figure 10: flowchart of pool game 24](#_Toc343380100)

**Acknowledgements**

I would like to thank my project supervisors, Dr. Raed AL-Qadi, Dr.LuaiMalhis, Dr.HayaSamaaneh for providing an awful amount of guidance and help us in our project.

**Abstract**

Games are the killer applications for smartphones today. Almost everyone likes to play games of some sort. ”Android users spent an average of 9.3 hours per month playing games on their phones” (1).

Creating mobile games can be a very profitable business. It’s very difficult to predict which games will be hits, but a quick scan of Android Market shows that hundreds of thousands of users have downloaded certain games. We would also like to point out that access to games on Android devices cheaper than others and there are many of them free and is due to that these applications built with open source language.

Many engines used for the construction of these games. In this project has been building games using AndEngine due to appropriate to the requirements and needs of the games to be manufactured with Engine, and it contain Box2d physics as an extension.

In this project we need to develop 3 games, Bubble shooter, Shoot game and Billiard game.

# Introduction

## 1.1 Overview

Games are the killer applications for Smartphone’s today. Creating mobile games can be a very profitable business. It’s very difficult to predict which games will be hits, but a quick scan of Android Market shows that hundreds of thousands of users have downloaded certain games.

Now, after we decide the idea of our project, it’s time to decide what we will use to accomplish this idea and How we implement what is going on in our head and turn it into something tangible.

After Research we found “Corona SDK”, this is used to make android games, but we found the money problem in this library.

After that we look for COCOS-2D, and we read about it but we didn’t find much information because it is mainly used in IOS application.

The last thing we found it is AndEngine, and we diced to use it in our project, also we decide to use Box2d physics engine extension in our project.

Before we create a new game, we want to think about which types of games exist, as well as which types work well on mobile devices and which don’t. We also want to take a look at the components that are common to all phone games.

## 1.2 Existing problems

The main topic of our project is “bundle of android games”, So, How turn the images and ideas to game, game meaning sprits& drawing + won & fail + animation +goals +music +text +physics and the most important thing is idea. So what ideas that will apply?Is what we have available sufficient to supported? Is there demand?Is there a possibility to develop these ideas regardless of whether they are an idea for a graduation project?The idea of ​​applying physics engine you will benefit or will it be a node in this area?

Several things must be taken into account for the completion of the idea.

## 1.3 Motivation for Carrying out the Project

The main motive for this project is that the main idea oriented Mobile, one of the most common applications at this time and we went to the games section, because these applications a very large turnout by all ages**.**

The possibility of development where possible and also a great addition to that we have never produced anything in this area, was an opportunity for self-education and achievement as a graduation project.

## 1.4 Main Contribution

This Project contributes the following:

* This project was instrumental in the development of recreational games.
* Use the AndEngine addition to extensions and Box2d physics also in the development of game.
* Employment of the equations of motion, speed and force and employment to the movement of objects within the game and also certain limits.

## 1.5 Report Organization

The report is organized as follows. In the first chapteris displayeda general introductionon the subjectwhich will bereviewed by theproject, in addition to displayingthe main problems and the project idea.  
The methodology that are detailed in Chapter tow. In the third chapter the first game we start in it is Bubble shooter in details.

And in the chapter four the second game is shoot game.

And in the chapter five the third game is pool game.  
Finally,   
Conclusion and final results are summarized in chapter seven.

# Methodology

When we decide to make games we start to search what engines or libraries support android games, we found some of these such as Corona SDK, COCOS-2Das mentioned before in Overview, but we had problems with them, Corona needs money and no enough tutorials for COCOS-2D in android most of them were about IOS, so we finally use Andengine library that supports Physics engine, but we spent a lot of time to found the suitable version of Andengine and Box2d can be used and support all features we need and this belongs to something in

AndEngine called GLES1 and GLES2 (blog), after asking and searching we found that the new devices use Android OS works with GLES2. After that, about month and half, we searched for simple games are made using AndEngine and Box2D, we learned a little bit but were not enough, we started making games for us, we used some of the things that we have learned, but with modifications suitable with our own game, some of the things, for example, we have to follow the equations of motion and Newton's laws to employ them in our games.

We read about making games using Andengine both from books or Andengine and Box2D forums until we have formed the basic principles and we started our own games making.

**How to start coding?**

We need to download the android requirements: (2)

* Android SDK
* Android SDK Platform Components
* Android SDK Tools
* ADT Plugins for Eclipse
* Classic Eclipse
* Oracle/Sun Java Development Kit ( JDK).

Then download the JAR files for AndEngine and Extensions for Box2d.

After we download the JAR files that we need it, we must take care that the versions we have is true. (3)

And we have two versions from AndEngine.jar, GLES 1 and GLES2, the second one is more efficient than before. And take care about the SDK version, it is must be Compatible, so we use Android SDK 2.2.

1-In Eclipse, create a new Android project, during that we select the SDK 2.2 that we want.

2-Create folder in the project Folder, name it “Libs”, Import the .jar file into your project’s lib folder.

Then in the Eclipse Project Explorer, right-click on the .jar file, and select Build

Path > Add to Build Path.

Also we need another thing called “armeabi”.

1-Download the shared library libandenginephysicsbox2dextension.sofrom the armeabi subfolder to your development system.

2. Create a new first-level folder under your Eclipse project named libs (right click on project in Project Explorer, select New > Folder). Create a subfolder under libs called armeabi.

3. Import the shared library into the armeabi subfolder.

4. You do not have to add the shared library to the build path-it’ll get sucked in automatically.

We build the first application using physics AndEngine, the application idea is simple and it is just to touch the screen to add objects and this is this is objects founds in physical world and move by the movement of the mobile(Accelerometer Listener).

**Test the Game on Actual Devices**

You need to test your game on the actual devices on which users will run it. The emulators that come with the Android SDK are great, but emulation on a PC o using a mouse for touch just isn’t the real thing. If you’re short of Android devices and need to do unit testing using the emulator, that’s fine.

I leave an Android device connected to my development machine and always test and debug on a real Android device.

However, one Android device isn’t nearly enough. Devices can be vastly different, running different versions of Android, with different processors, including some with graphics processors and some without; some devices with large, high-resolution screens and others with small, low-resolution screens; and some with only single-touch capabilities and others with advanced multi-touch support. You need to test your game on as many Android devices as you can reach. The good news is that, unlike with some other platforms such as iOS , you can side load your game to as many Android devices as you want to before committing it to an app store.

# Requirements and analysis

## 3.1 Functional Requirements:

Concentrated primary task of the project is to use physics engine employed in the investigation of mathematical equations serve games, where this type of engines gives the game more Action & fun.

* So, the first requirement is calculate mathematical equations hits the ball a certain direction is determined by touching the screen and the original location of the ball on the Scene.
* The second requirement is move the ball to build on ballistics physical equations where moving a certain angle and direction of this movement depends also on the amount of tensile played by the player.
* The third requirement is move the ball on a smooth surface, taking into account the friction, flexibility and density to that surface. Account the equations of motion of the ball on the move stick in a certain direction and intensity.

In a brief the main goals of the games is interesting and note the effect of physics engine games.

## 3.2 Non-Functional Requirements:

* User friendly: games can be played by any user easily and it is suitable for all ages due to their design not complex, and they are containing menus and help.
* Availability: games are available any time. Any time users like to play he installs it from Play store.
* Portable and **Platform**: there are able to run on any device use android operating system like series of Samsung, HTC, LG phones.
* Security: any user want to play one of our games, he download the apk file without source code.

# Background

### 4.1 Game components:

The general components that we need to work into the game and implement in the code. (Rogers)

**Opening (Splash) Screen**

To maximize performance as the game is being played, the graphics needed for a game level are often loaded before the level is started. During the loading process, which can take several seconds, you don’t want to leave the user with a blank screen, so you display a splash screen. It lets the user know that the game is working as it should. Splash screens are optional, but we’ll include one in our game, just to show how it’s done.

**Menu Screen**

Once the game is ready to run, we’ll need a place for the user to enter various options. This is typically done with a graphical menu screen that presents the options and either implements the option or calls another screen (such as Help) to do so.

**Scene**

It is contain all the objects in the game, and any object we need to show we attach them to the scene. And the update handlers will register on it also. itis a setting where some action takes place over a constrainedperiod of time.

**Music**

For most of us, music has strong emotional influence. Background music is very important for setting the mood of your game, and helping with the transitions between parts of the game.

**Time**

Most games will incorporate time—either clock time (scoring completion of a puzzle based on the time taken to solve the puzzle) or playing against moves makes in real time. We need time also for events ordering and delays.

**Obstacles**

Obstacles are used in different ways in different games. In many games, the player is trying to achieve some goal, and obstacles are thrown in the player’s path.

**Levels**

Players can start with easy challenges and gradually ramp up to higher challenges as their game-playing skills and experience improve. Levels are a proven way to achieve this effect

### 4.2 AndEngine:

AndEngine is a game engine library that makes it easier to write two-dimensional games for Android devices. Nicolas Gramlich led the effort to create AndEngine and wrote much of its code. The project is open source, so you are encouraged to go to the project website and join in the development effort for AndEngine.

We could write our own functions in Java, using the Android APIs to implement the components of a game.

AndEngine comes as a .jar file that is, a Java archive. We will see how to use that archive with our Android SDK project when we start our game coding later.

**AndEngine Game Concepts**

The game is like a movie, and the game engine includes concepts that are analogous to those involved in making a movie. (Rogers)

**Camera**

The “camera” of the game determines the view of the game that is presented to players. It is very much like a movie camera in two-dimensional space. The camera can pan and zoom across the scene to change the view presented. The panning and zooming can either be under the player’s control or be driven programmatically.

**Scene**

A game, like a movie, consists of a series of scenes where the action takes place. In a movie, the scenes are edited together in a fixed way. In games, the sequence of scenes is driven by the play of the game. Games are like movies edited on the fly.

**Sprite**

Sprites are the visual representation of the actors in our movie, whether those actors are people or objects. Sprites can be animated or not, but they often move about the scene during the course of game play. Sprite textures are often loaded from one large image that comprises a collection of sprite images, called a sprite sheet.

**Entity**

In AndEngine, entities are just about anything that’s drawn to the screen. Sprites are entities, as are tiles, geometric shapes, and lines drawn on the screen. All entities have properties, such as color, rotation, scale and position that can be changed by.

**Texture**

A texture is 2D, generally bitmapped graphic that can be applied to objects to give them, well, texture. Textures define the way entities look, and much of the OpenGL graphics environment is built around the use of textures.

**Texture Region**

A texture defines a complete bitmap graphic, and a texture region defines a subset of that region. We’ll talk a lot about performance optimizations of 2D graphics later, and using texture regions to map small pieces of a large combined bitmap is one of the key tricks used to create these optimizations.

**Engine**

An engine runs a scene. It takes care of letting animations and modifiers know when to update the presented graphics, coordinates the actual drawing, handles user input events (touch, keys, sensors), and generally manages the progress of the game. The engine is a lot like the producer/director of our movie, telling everyone what they need to do.

**BaseGameActivity**

This class, which extends the Android Activity class, will be the basis of each scene in our game. BaseGameActivity does all the work common to all scenes, setting up the game engine, conforming to the Android Activity Lifecycle requirements, and enabling sensors.

**Physics Connector**

AndEngine includes basic physics capabilities in the base engine, but the Box2D physics engine expands greatly on those capabilities. We connect AndEngine objects with Box2D through a physics connector. If your game doesn’t use Box2D physics, you won’t have a physics connector.

**Menus in AndEngine**

Menus are a special type of Scene in AndEngine. They’re special because they arrange text or graphics in an ordered list, and they accept touch inputs from the player to select one of the items on the list. They also provide for animation of the menu items as the menu is being displayed.

We created a text menu that lists the menu items, and we saw the default behavior for text menus. We also created a pop-up graphical menu with a few options, and showed how its creation is both similar to and different from the creation of a text menu.



Figure 1: Pop-up menu in bubble shooter game

****

Figure 2: Static menu in bubble shooter game

**Box2D Physics Engine**

AndEngine includes the open-source JBox2D port of the Box2D physics engine. It can be used to realistically simulate the interaction of physical objects in the following ways (among others):

* Simulation of the physics of rigid bodies
* Stable stacking
* Gravity

### 4.3 Box2d Physics Engine:

In game development, “physics” is the collection of effects that mimic the physics of the real world. Physics don’t enter into every type of game. For example, we generally won’t need physics if we are developing a board game. Other games are almost entirely physics based, with missiles of various kinds following natural arcs, crashing into piles of objects that then break or fall down under the influence of gravity.

a very complete physics engine for use in our games is available as an extension to AndEngine.

This is the main things that we used from BOX2d Physics AndEngineExtention: (Rogers)

**Units**

The physics simulation that Box2D performs is sensitive to the units we use for mass, velocity, and other quantities. Box2D is tuned for MKS (meter–kilogram–second) units, and is designed to simulate “normal-sized” objects as they interact.

**World**

Box2D enables us to build virtual worlds where bodies behave much like the physical objects we encounter in real life. A Box2D world consists of bodies, fixtures, joints, and constraints that add up to a physical simulation of that world.

**Rigid Body**

Bodies are the basic simulation objects in Box2D. They have only a few characteristics of their own, but can take on more complex physical attributes through association with shapes, fixtures, and constraints, as described later in this section. The primary restriction that Box2D places on the bodies is that they are all rigid bodies, meaning their shapes never become distorted. To quote from the Box2D Manual, a rigid body is“A chunk of matter that is so strong that the distance between any two bits of matter on the chunk is completely constant. They are hard like a diamond”.

Bodies have a type, which can be any of three values:

1-Static: Bodies that are normally fixed in place. The user can move these bodies, but they are not moved as part of the physics simulation. Static bodies act as though they have infinite mass (represented by a mass of zero when we create them), and they collide only with dynamic bodies.

2-Dynamic: The moving, fully simulated bodies in a physics world. Dynamic bodies always have a finite, non-zero mass, and can collide with static, kinematic, and dynamic bodies. If you try to set the mass of a dynamic body to zero, it is automatically reset to 1 kilogram.

3-Kinematic: Bodies that move only by virtue of their velocity. Kinematic bodies are not part of the physics simulation, and don’t respond to forces. They also behave as though they have infinite mass, and collide only with dynamic bodies.

**Shape**

Box2D supports two basic shapes that approximate the shape of two real object namely, a circle and a polygon. When we create a Box2D body, we normally associate a shape with it through a fixture (discussed next). When Box2D is simulating the physics of bodies interacting, it uses the shape of the body to determine when collisions occur.

**Sensor**

Perhaps your game needs a body that detects collisions, but doesn’t respond to them. Sensors fill that need, and we’ll see that you can declare anybody to be a sensor.

**Setting up Box2D**

The pattern for building a physics world and starting the simulation includes the following steps:

1. Create a PhysicsWorld, using that class’s constructor. In the constructor, you canoptionally create a gravitational acceleration vector.

2. Create the static objects in the simulation, which might include a floor, somewalls, and a ceiling, to keep objects from flying off out of screen range. We’ll dothis in two steps:

* Create the shapes for the objects (usually Sprites).
* Create the bodies, attaching the bodies to our world, and attaching shapes tothe bodies through an appropriate fixture.

3. Attach the shapes to the AndEngine Scene so they will be displayed.

4. Connect the Sprites to the Physics with PhysicsConnectors.

5. Register the Box2D PhysicsWorld as an UpdateHandler for our Scene, so it can update Sprite positions.

# Bubble Shooter Game

**Idea (in general)**

The idea of Bubble Shooter is the same as always: fire a bubble and try to hit at least two touching bubbles of the same color to burst them There are two different game modes in Bubble Shooter: **Arcade** and**Puzzle**. In Arcade mode you need to clear all the descending bubbles to continue, while in puzzle mode you need to clear all the static bubbles on a level to progress to the next.

our game is Arcade mode, there is a fixed time to finish the level while the bubbles descending if time is up and you didn’t clean the bubbles then you will lose else you will pass the level and go to next level ,there is 5 levels in this game and can be developed in the future as levels ascending the hardness of level will increase such as adding another color to bubbles and decreasing the time required to finish the level .for future expanding we can put some special bubbles such as stones, glass, bombs, fire, etc.

Graphically, Bubble Shooter is cute and colorful, though sweet soundtrack.

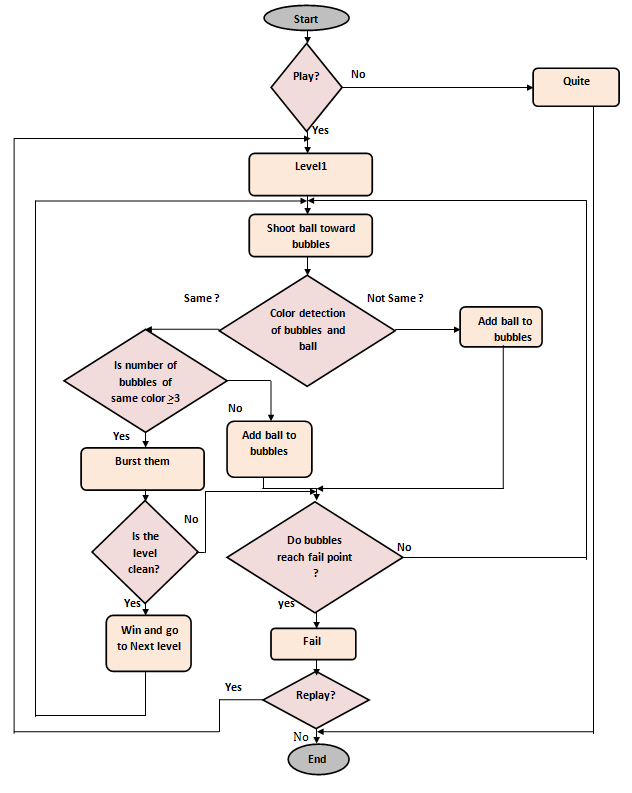


Figure 3: flowchart of main functionality of Bubble Shooter game

**Problems and solutions:**

Many problems we faced during the development of this game such as

1. The direction of the ball released, We have traced the equations of motion to determine the right direction, take the initial place of ball and take the position of mouse where it touched the screen of the emulator then and according to this coordination we give the ball linear velocity but the problem was whenwe shoot it from top to bottom it works correctly but we need to shoot it from bottom to top and there was a problem in the direction.After several attempts and review equations we found the solution,the problem was that the coordination of the screen of the emulator start from (0,0) at up right and increased as we go down so in first case the location of ball on top and the location of the touch is down then the distance will be positive and the direction true ,but in the second case we have to opposite the sign of X direction and modification in the Y direction.
2. Color detection of bubbles that should be burst after shooting the ball. the problem was not only the bubbles that have the same color are burst but also the neighbors
3. Descent of bubbles for a specific distance and a certain period of time and coincided with the release of the ball took a considerable length of time to complete the game
4. Detect the scene if empty to go to win page. We cannot found a method to check if scene is empty, so we store all bubbles in a linked list and delete each bubble that exploded and then when linked list is empty then the screen will be empty.

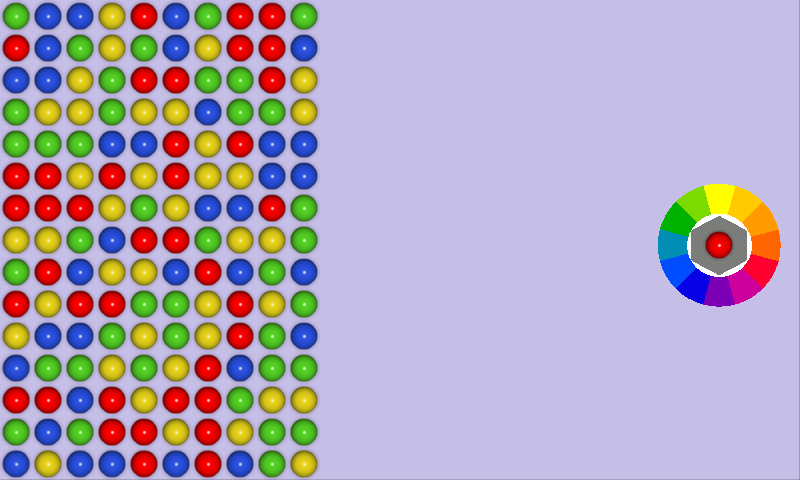


Figure 4: Bubble shooter game

# Shoot Game

**Idea:**

The idea of this game is based on the principle straighten the ball on the goal and dropping it, like famous Angry Birdsgame. To play this game you have to pull the ball strongly by certain distance that you want to reach and the angle at which you will throw out. If you drop all the targets will pass the stage you can try 3 times (you have 3 balls to drop targets). Each level differs from other levels either in the number of targets or the location of the target.

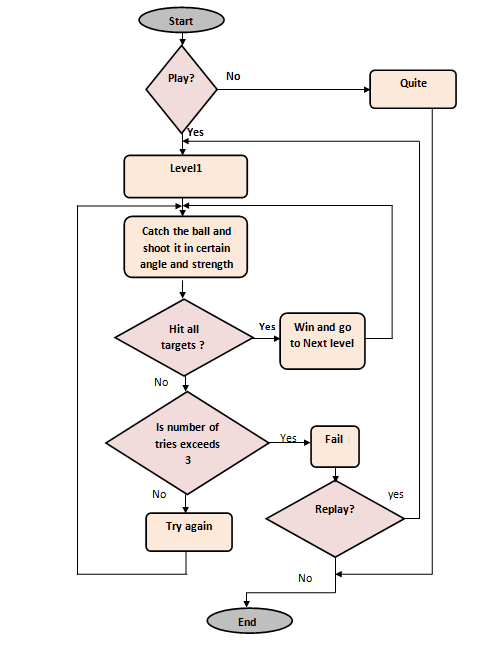


Figure 5: flowchart shows how to play Shoot game

**Problems and solutions:**

The main problem in this game is how to shoot the projectile in the correct direction with correct angle based on amount of tensile played by the player and also the direction in which he wants during the checkout process.

To solve this problem we need to do equations, and these equations based on the initial position and the final one when we pull the ball on the scene.

The second thing, we want to check if the ball came to be in place and hit the target.

To solve this we add an update handler to the scene and detect if the ball collide with target.



Figure 6: First Level in the game



Figure 7: Second Level in shoot game

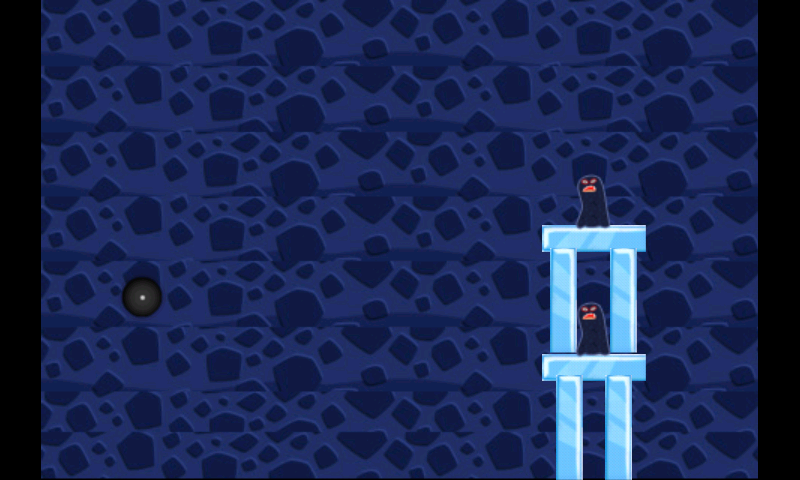


Figure 8: Third Level in shoot game

# Pool Game

**Idea**

It is simply a billiard game, the idea of this game is to focus a stick on a white ball in a specific angle to shoot the ball in a given force that chosen by user then this ball will collides with the other colored balls that ordered on the table in a triangle shape and this leads to move balls in random directions. The aim is fall the colored balls in the holes of table and you must leave the black ball for last.

If the white ball falls in the hole it will give foul and the level will be loaded again.

The game contains three levels, the first level consist of just black ball and this should be dropped in any hole using the white ball and the stick, then in next levels the number of colored balls will increased in addition to black one.



Figure 9: Billiard Game Scene

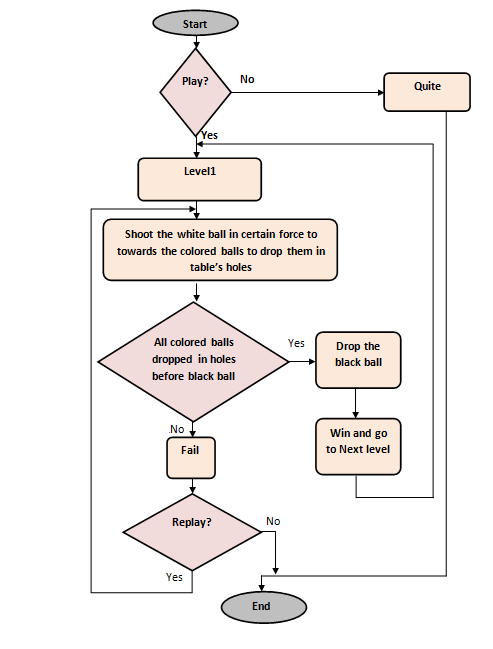


Figure 10: flowchart of pool game

**Problems and solutions:**

1. How do we make the stick move by the movement of the finger remains constant head toward the white ball. We use setRotate() function but with this function we cannot control the point at which the stick will rotate, it takes the center point as default point ”No parameters for rotation point”. We searched a lot about solutions and then we found something called RevoluteJoint() it takes two bodies and a connection line between them, the first body will rotate around the second body and the two ends of the connection line attached to the centers of these two bodies, we tried to consider the white ball as the first body, create a hidden body as body two and consider the connection line as a stick but the problem was the white ball must be a static body, if it is a dynamic body it will move and rotate without touch, and will not affected by physics world and collisions. Then we try to make a static body over the white ball to solve this problem it does not make sense the system always crashes so finally we use a kinematic body and this solution was the best of them.
2. The detection if a ball is matched a hole in order to drop it and delete it from balls, when we delete the ball the program crashes , but we can solve this problem after many tries.
3. When we need to shoot the ball after the first time, we need to detect if it’s velocity = 0, to re-shoot it another time, so we need to recreate the connection line (stick) near the white ball.
4. We need to detect that the last ball is black, so every shoot we check the balls color dropped in the holes.

# Conclusion

Any game start with idea, then it is developed by the time. After decide the idea the search begins how to convert this idea to real game, and then the business is established while the development on it increased.

To start making any game you must have the main concept about what will you use, such as the type of the game, idea, language, programs used to make it. Then start your work.

# References

(n.d.). Retrieved from http://androinica.com/2011/07/android-users-love-games-according-to-nielsen-data-stats/

(n.d.). Retrieved from http://developer.android.com/tools/device.html

(n.d.). Retrieved from http://www.andengine.org/

(n.d.). Retrieved from http://www.andengine.org/blog/

Rogers, R. In R. Rogers, *Learning Android Game Programming.*

Other references used during coding:

* http://code.google.com/p/andengine/
* The AndEngine community forum: http://www.andengine.org/forums/
* The AndEngine wiki: <http://wiki.andengine.org/AndEngine>
* <https://github.com/nicolasgramlich/AndEngine>
* https://github.com/nicolasgramlich/AndEngine

The source code repository for examples: http://code.google.com/p/andengineexamples/

AndEngine Forums:

* http://www.andengine.org/forums/

# Appendix

**Appendix A:**

This part of code used in bubble shooter game for determine the motion of the ball released depend on the point on the scene where the player touch.

* On touch event

**private** **void** shootProjectile(**final** **float** pX, **final** **float** pY) {

**int** offX = (**int**) (player.getX() - pX);

**int** offY = (**int**) (pY - player.getY());

**if** (offX <= 0)

**return**;

**int** realX = (**int**) (mCamera.getWidth() + projectile.getWidth() / 2.0f);

**float** ratio = (**float**) offY / (**float**) offX;

**int** realY = (**int**) ((realX \* ratio) + projectile.getY());

**int** offRealX = (**int**) (realX + projectile.getX());

**int** offRealY = (**int**) (realY - projectile.getY());

Vector2 shoot = **new** Vector2(-offRealX,offRealY\*2);

body.setLinearVelocity(shoot);

**Appendix B:**

This part of code used in shoot game, it is calculate the angle (direction) and the force that the projectile will release.

* On action move

**this**.setPosition(pSceneTouchEvent.getX() - **this**.getWidth() / 2, pSceneTouchEvent.getY() - **this**.getHeight() / 2);

init\_x=pSceneTouchEvent.getX();

init\_y=pSceneTouchEvent.getY();

DistanceX = init\_x - 80;

DistanceY = init\_y - 170;

* On action up

Float Distance=(**float**) Math.*sqrt*(DistanceX\*DistanceX+DistanceY\*DistanceY);

**float** angleRad =(**float**)Math.*atan2*(DistanceY, DistanceX);

**float** yy = (**float**) (-Distance\*Math.*sin*(angleRad)/4);

**float** xx = (**float**) (-Distance\*Math.*cos*(angleRad)/4);

shoot++;

*shootMuzic*.play();

body.setLinearVelocity(**new** Vector2(xx,yy));

mPhysicsWorld.setGravity(**new** Vector2(0,SensorManager.*GRAVITY\_EARTH*));

**Appendix C:**

This code used in Billiard game to determine the force and direction that will apply to the white ball.

* On load scene

revoluteJointDef = **new** RevoluteJointDef();

revoluteJointDef.initialize(body, hid\_body, body.getWorldCenter());

revoluteJointDef.enableMotor = **true**;

revoluteJointDef.maxMotorTorque = 200;

**this**.mPhysicsWorld.createJoint(revoluteJointDef);

* On action move:

init\_x=pSceneTouchEvent.getX();

init\_y=pSceneTouchEvent.getY();

DistanceX = init\_x - projectile.getX();

DistanceY = init\_y - centery;

* On action up:

connectionLine.setPosition(150 + spriteWidth / 2, centery + spriteHeight / 2, pSceneTouchEvent.getX(), pSceneTouchEvent.getY());

Float Distance=(**float**) Math.*sqrt*(DistanceX\*DistanceX+DistanceY\*DistanceY);

**float** angleRad =(**float**)Math.*atan2*(DistanceY, DistanceX);

**float** yy = (**float**) (-Distance\*Math.*sin*(angleRad)/4);

**float** xx = (**float**) (-Distance\*Math.*cos*(angleRad)/4);

body.setLinearVelocity(**new** Vector2(xx,yy));

connectionLine.setVisible(**false**);

mPhysicsWorld.destroyBody(hid\_body);

mMainScene.detachChild(hid);

hid=**null**;

