

“Run Steve, Run!” An Android Game written in Java

By John Oldfield

BSc (Hons) Software Engineering

AE2

Contents

[Background 4](#_Toc503609627)

[Integrated Technologies 4](#_Toc503609628)

[DisplayMetrics 4](#_Toc503609629)

[Graphics 4](#_Toc503609630)

[Bitmap & BitmapFactory 4](#_Toc503609631)

[Canvas 5](#_Toc503609632)

[Rect 5](#_Toc503609633)

[Point 5](#_Toc503609634)

[MediaPlayer 5](#_Toc503609635)

[MotionEvent 5](#_Toc503609636)

[WindowManager 5](#_Toc503609637)

[Aims & Objectives 6](#_Toc503609638)

[Aims 6](#_Toc503609639)

[Objectives 6](#_Toc503609640)

[Implementation 6](#_Toc503609641)

[App Startup 6](#_Toc503609642)

[Handling the SurfaceView 7](#_Toc503609643)

[Thread to Limit FPS 8](#_Toc503609644)

[Handling Touch Events 8](#_Toc503609645)

[Displaying the Game 9](#_Toc503609646)

[Handling Player Movement 9](#_Toc503609647)

[Handling Animation 10](#_Toc503609648)

[Handling Obstacles (Walls) 11](#_Toc503609649)

[Handling the Wall Gap 11](#_Toc503609650)

[Infinite Walls 12](#_Toc503609651)

[Handling Collision 13](#_Toc503609652)

[Evaluation 13](#_Toc503609653)

[Positives 13](#_Toc503609654)

[Negatives 14](#_Toc503609655)

[Problems Encountered 14](#_Toc503609656)

[References 16](#_Toc503609657)

[Appendix A – Source Code 17](#_Toc503609658)

[A.1 - Animation.java 17](#_Toc503609659)

[A.2 - AnimationManager.java 18](#_Toc503609660)

[A.3 - Constants.java 19](#_Toc503609661)

[A.4 - GameObject.java 19](#_Toc503609662)

[A.5 - GameScene.java 20](#_Toc503609663)

[A.6 - GameScreen.java 22](#_Toc503609664)

[A.7 - MainActivity.java 24](#_Toc503609665)

[A.8 - MainThread.java 25](#_Toc503609666)

[A.9 - Player.java 26](#_Toc503609667)

[A.10 - Scene.java 28](#_Toc503609668)

[A.11 - SceneManager.java 28](#_Toc503609669)

[A.12 - Wall.java 29](#_Toc503609670)

[A.13 - WallManager.java 30](#_Toc503609671)

[A.14 – Manifests.xml 32](#_Toc503609672)

# Background

This project intended on developing a simple mobile game for Android devices where the player must navigate a character through random openings for as long as they can. Gaming is an essential pastime for all people of varying ages, with there being 2.2 billion gamers across the globe (McDonald, 2017a). Mobile gaming makes up a huge amount of the gaming market, with it being estimated to have generated $46.1 billion in 2017, or 42% of all global game revenues (McDonald, 2017b). With such a large market it is obvious that almost all smartphones have the capacity to play games in some form, whether it be a simple 2D action game or a complex 3D story driven game rivalling those of consoles. It is for this reason that a mobile game was chosen as a project.

By combining Games Development and Android Development a wide range of knowledge can be gained, as the playability/usability side of the gaming will require much investigation into the functionalities of Android. In particular, finding out about how essential topics in Game Development, such as achieving a good FPS (Frames Per Second) can be applied using Android. The project investigates how this can be achieved whilst generating all the features of the game such as characters and obstacles. To ensure there is a good replay value for this game a strong focus on randomization will be used so that the player receives a different challenge every time.

# Integrated Technologies

## DisplayMetrics

The application uses the DisplayMetrics import to retrieve the screen dimensions of the device. These dimensions are stored and later used to calculate different variables used in different aspects of the game. Using the dimensions, the game could also be scaled so that no matter what device the game was being played on it had a similar look, feel and difficulty. The screen height was important as this was used to produce walls “above” the screen, generate new walls once they had gone “below” the screen and calculate a quickening speed of which the walls would fall. The screen width was vital as it was used to ensure there was always a space between the two walls for the player to go through.

## Graphics

### Bitmap & BitmapFactory

Whilst developing the application, a rectangle was used to the game mechanics however to make the app more aesthetically pleasing an actual character for the player to navigate was implemented. To display the character, who is a PNG image, on the screen the Bitmap import was required. To further improve the aesthetics of the game, animation was used, and this required having multiple bitmaps that when showed one after another would simulate the character walking. The Bitmap factory was used to edit the Bitmaps further, flipping them horizontally so that the character could face different directions.

### Canvas

The Canvas class was one of the most important to this application as this is how the game was drawn and displayed to the player. When being played the application is constantly updating the Canvas, updating the walls, the score and most importantly the player position.

### Rect

The Rect class was essentially the foundation on which the application was built on, with the walls being rectangles and the character and its hitbox fundamentally being a rectangle. Using rectangles this way created an effective and simple way to draw shapes, and also check for collisions by checking if the player rectangle had collided with a wall rectangle. It could also be used for clearly indicating the direction the user was dragging the rectangle by comparing changes in the position of the coordinates.

### Point

Point was an essential class used as it meant that the character could be initially set to a position relative to the screen height and width, further allowing the use of the app on different types of devices. It also meant that the character could be centred on the position where the user presses, so they could accurately tell where they were moving the character to.

## MediaPlayer

The MediaPlayer class was used by the application to provide background music whilst playing the game. As this was simply just to improve the user experience whilst using the app its usage in the app was simple, as it just played while the game was open and looped once it was finished. Using the Activity Lifecycle methods in conjunction with the pausing and starting methods of MediaPlayer, the music for the game can be paused and started with the activity which means that the music will start and stop when a user presses the home button for example.

## MotionEvent

The MotionEvent class was used to detect touches from the player and act accordingly. For this game pressing on the character begins the game and when they drag their finger on the screen the character follows the same movements. If they release their finger the character stops moving.

## WindowManager

The WindowManager class was used by the application to hide standard aspects of applications. For this application the Status bar and activity title were hidden so that the game and player could benefit from seeing the whole screen. This made the game more enjoyable and more aesthetically pleasing.

# Aims & Objectives

## Aims

The main aim for this project was to utilize topics from the Engineering Mobile Applications unit as well as additional Android APIs to produce a robust and competitive mobile software application. For this application 2D Graphics and Touch Events were used at the core of the game but Multimedia and design principles were also incorporated to cement the application as a game.

## Objectives

* Develop a competitive mobile game for Android Devices using Android Java

# Implementation

## App Startup

On starting the game, the default layout is first changed programmatically (See Figure 1) in the onCreate of the MainActivity (Appendix A.7) so that the status bar and activity title are hidden.

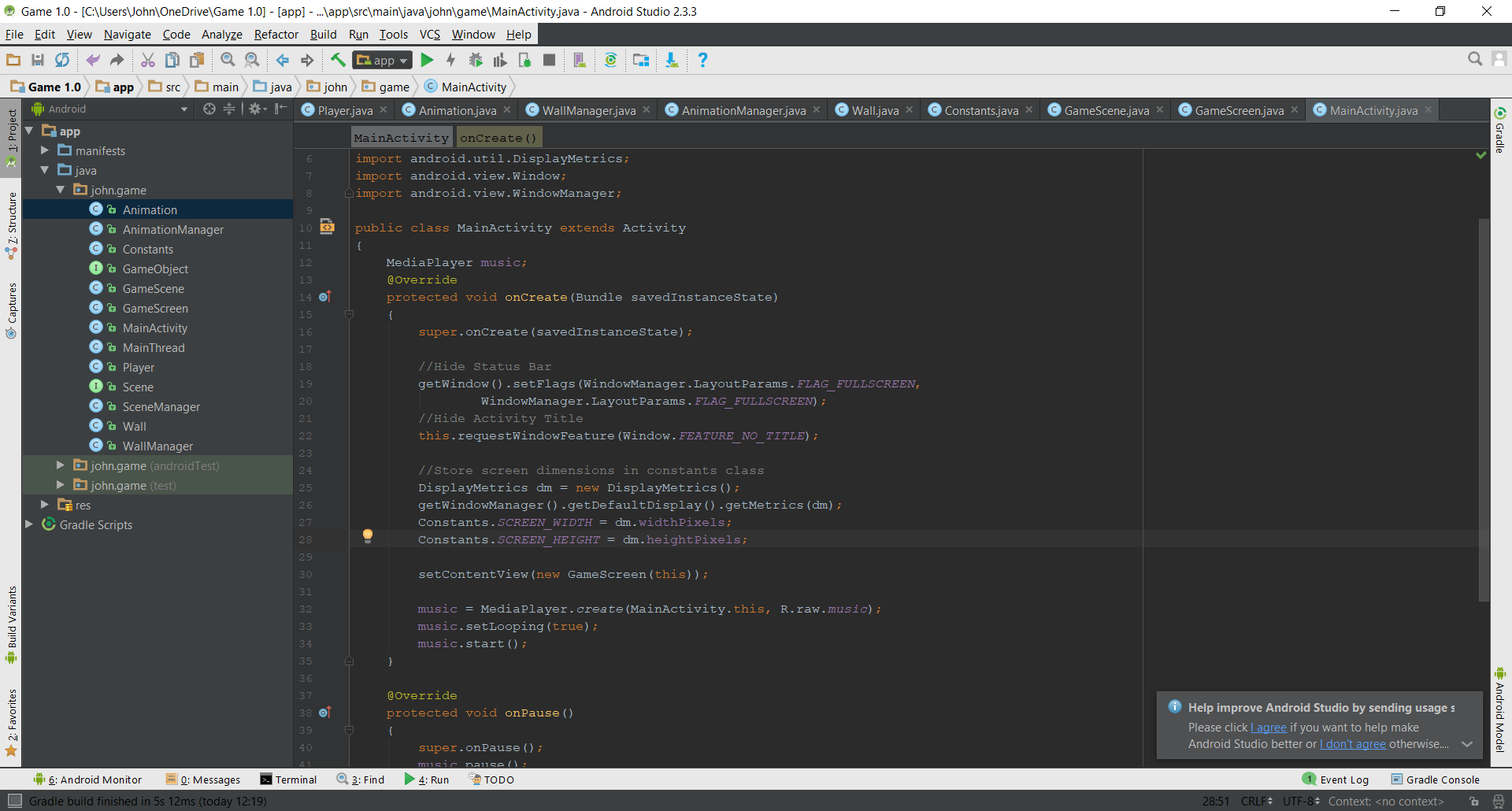


Figure 1 – Hiding status bar and activity title

A DisplayMetrics object is then created and used to get to the dimensions of the device screen. These dimensions are stored in public variables in the Constants class (Appendix A.3) for later use.

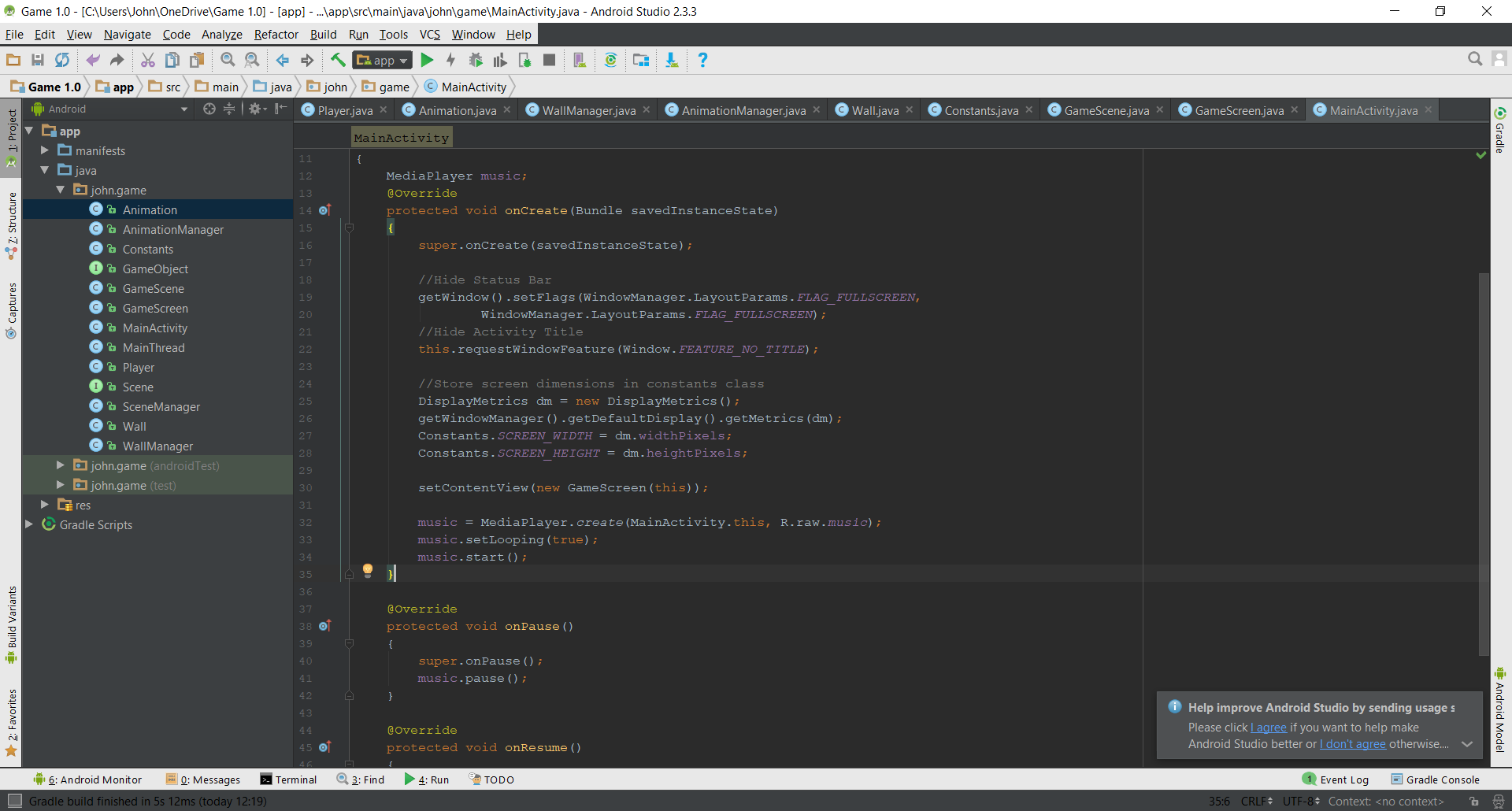


Figure 2 – Retrieving and storing screen dimensions

The MediaPlayer object is then created using the background music stored in the raw folder in the resources. This music is set to loop and then started.

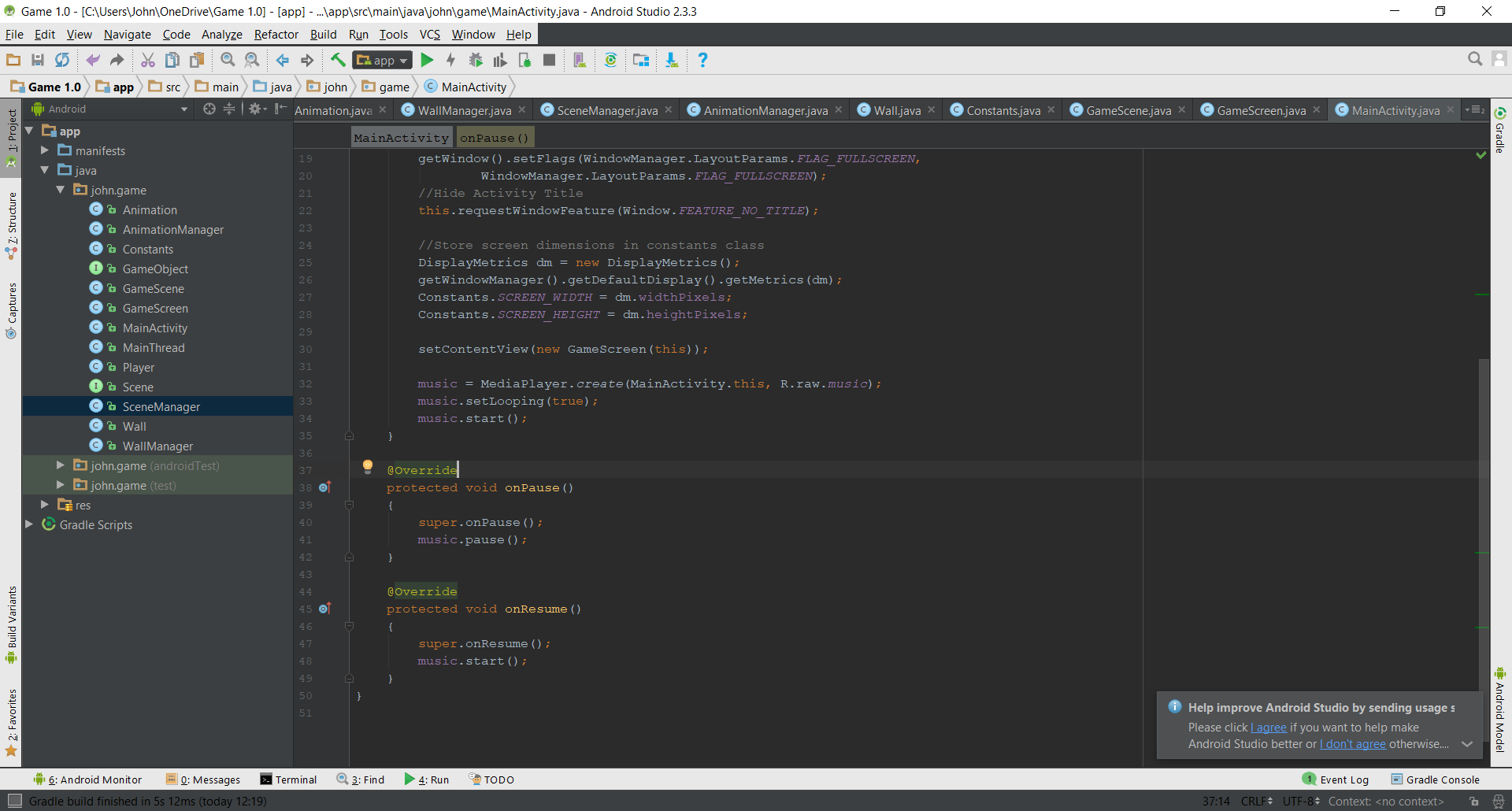


Figure 3 – Setting a music object to loop and start

Finally, the UI is loaded using the GameScreen class (Appendix A.6) passing in the **this** context as a constructor to the setContentView method.

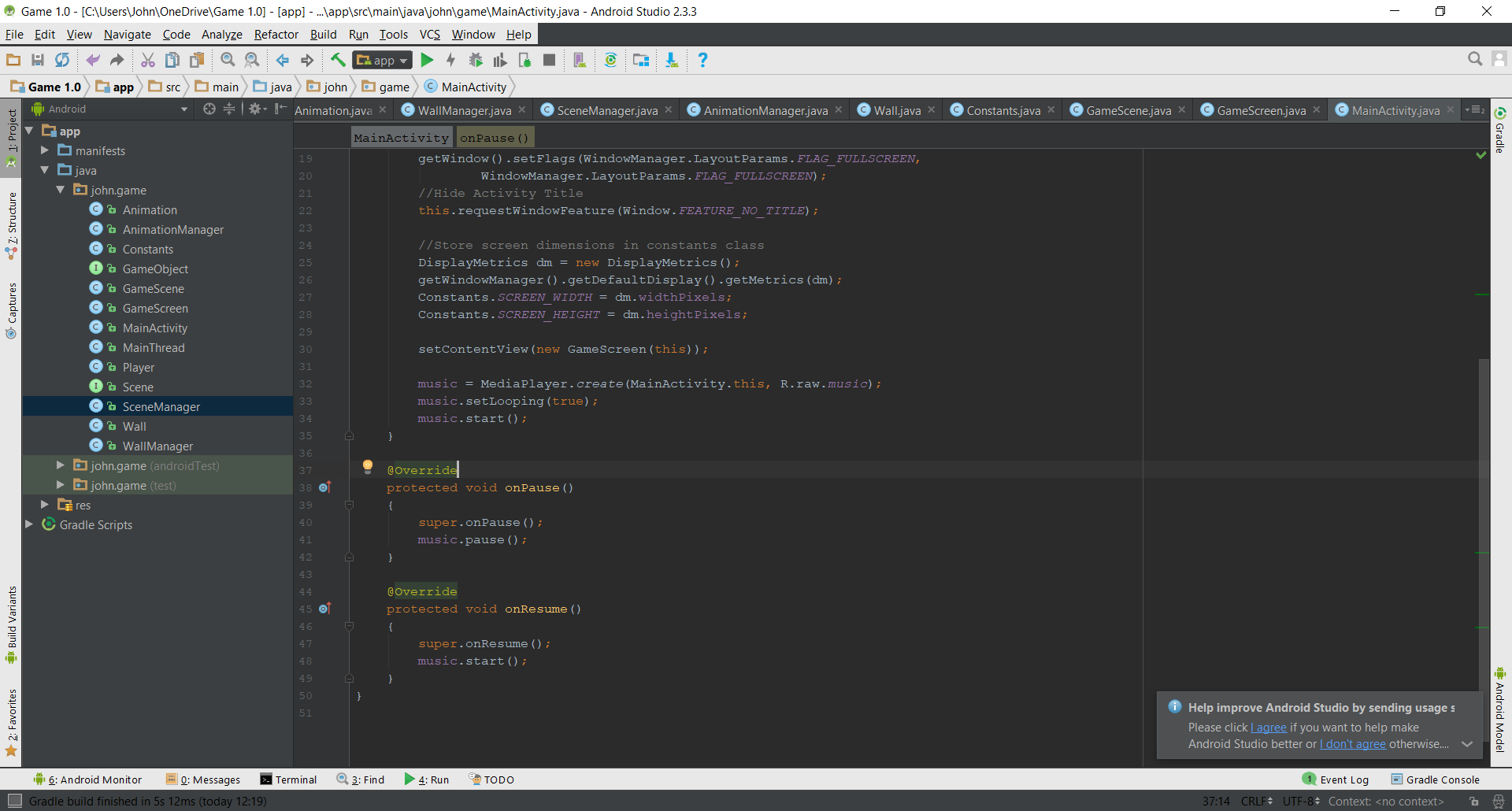


Figure 4 – Setting the Content View

The GameScreen class handles the following functionality:

* SurfaceView and its required methods.
* Starting the thread, which handles the games FPS.
* Handling touch events
* Updating the game frame by frame
* Displaying the game.

## Handling the SurfaceView

By extending the SurfaceView class, its required methods, which are surfaceCreated(), surfaceChanged() and surfaceDestroyed() need to be implemented. In the surfaceCreated() method the MainThread class(Appendix A.8) is initialized and is started. The surfaceChanged() method is left blank as this is usually used to handle orientation changes however as my game has been forced to only be in portrait in the manifest (Appendix A.14) nothing is required. In the surfaceDestroyed() method the current thread is stopped, thus stopping the game loop and the game from continuing in the background.

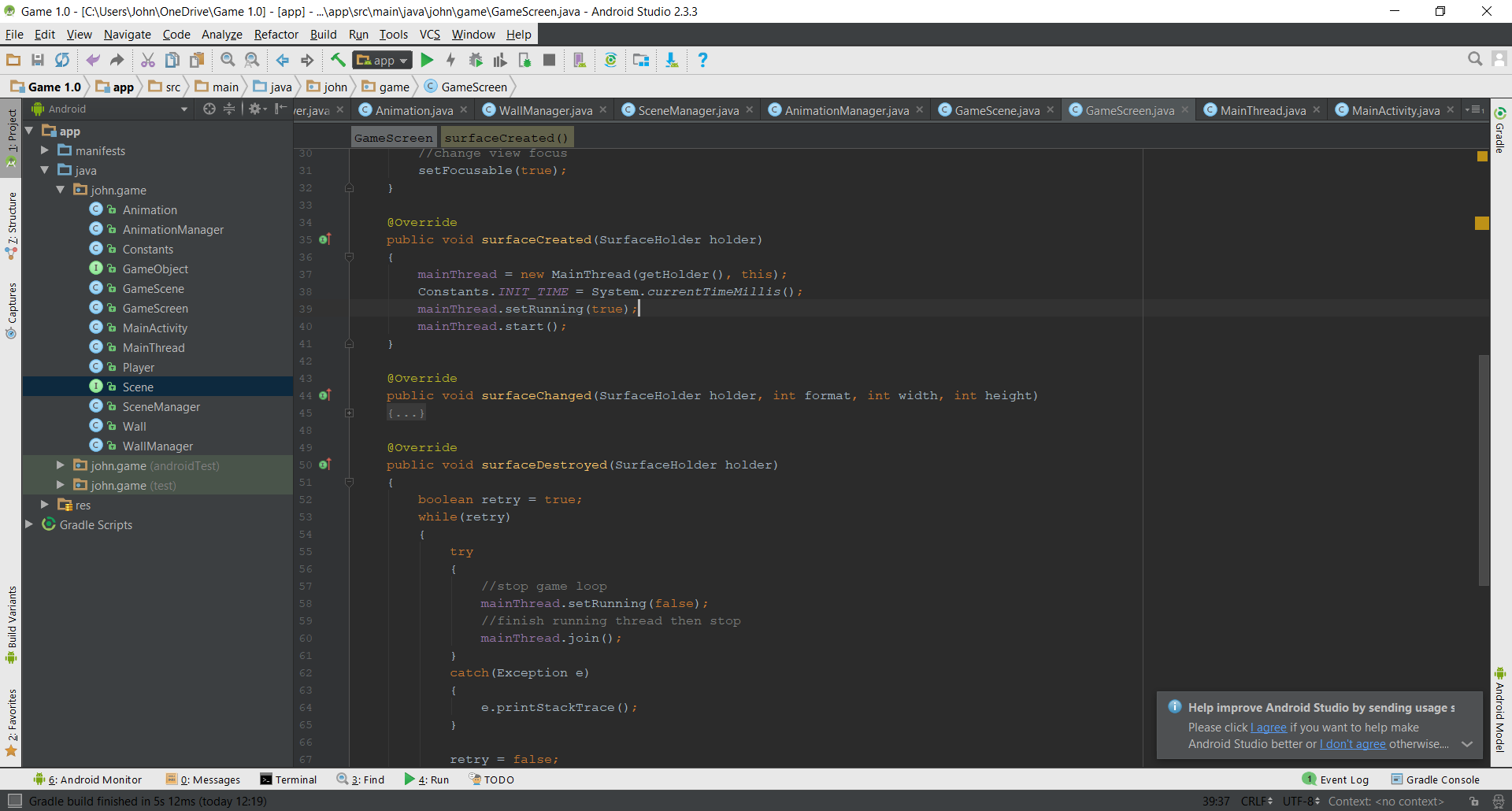


Figure 5.1 – Starting the Thread

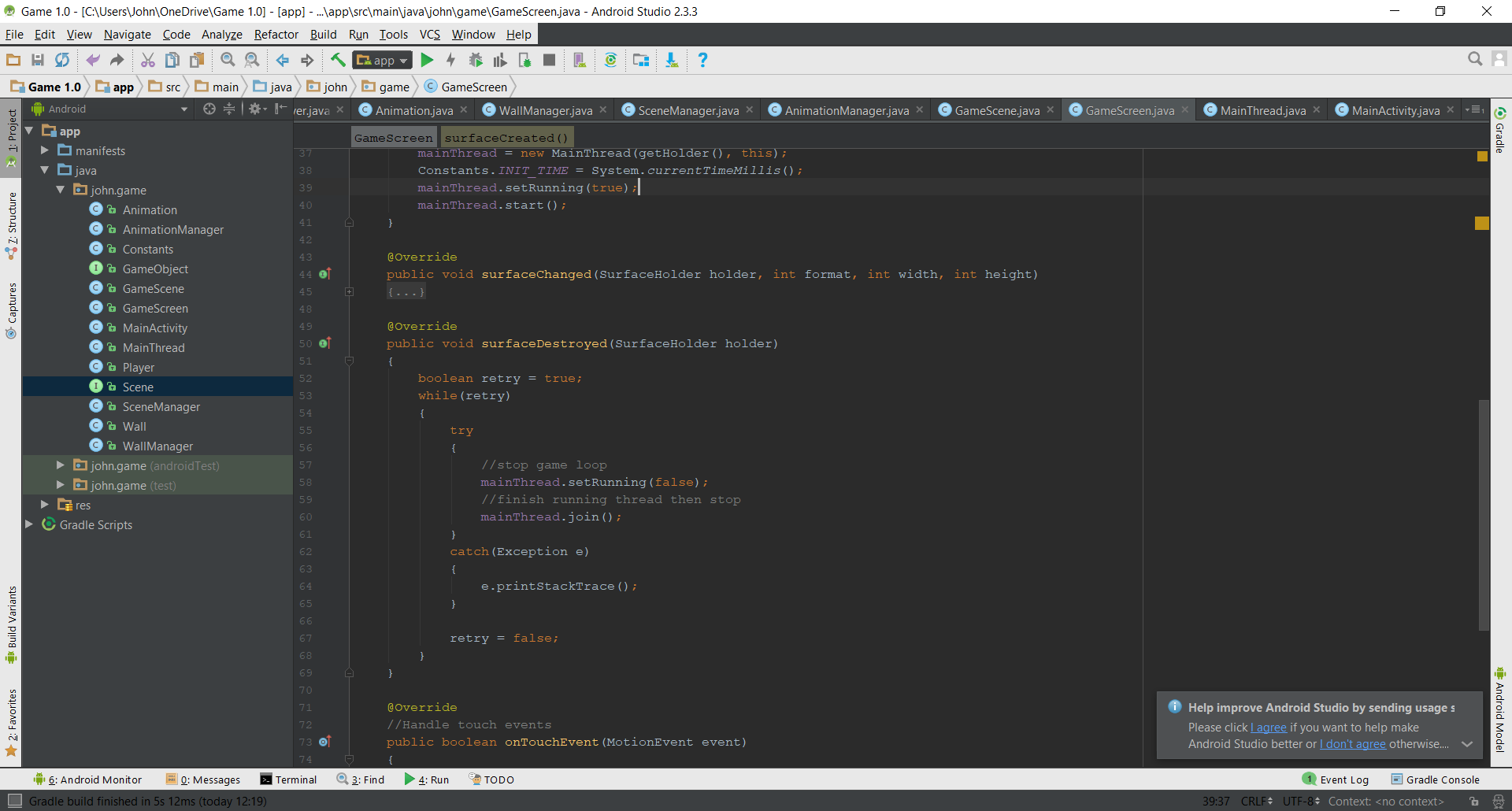


Figure 5.2 – Stopping the Thread

## Thread to Limit FPS

With all types of gaming, FPS is vital as too low an FPS can make the game appear as if it is lagging which will negatively affect the user’s thoughts on the game. Most smart phones today can easily run a simple 2D game such as this one at 30 FPS, which is generally considered to be good however I have chosen to limit it from being over as it will use up the phones resources unnecessarily. The thread is constantly updating the canvas making lots of frames and this is capped by calculating the average FPS and making sure that once it equals 30 that the process is restarted by resetting a timer. This task required me to investigate how to calculate the FPS, which I did by adapting code found on 41post.com (41Post, 2011).

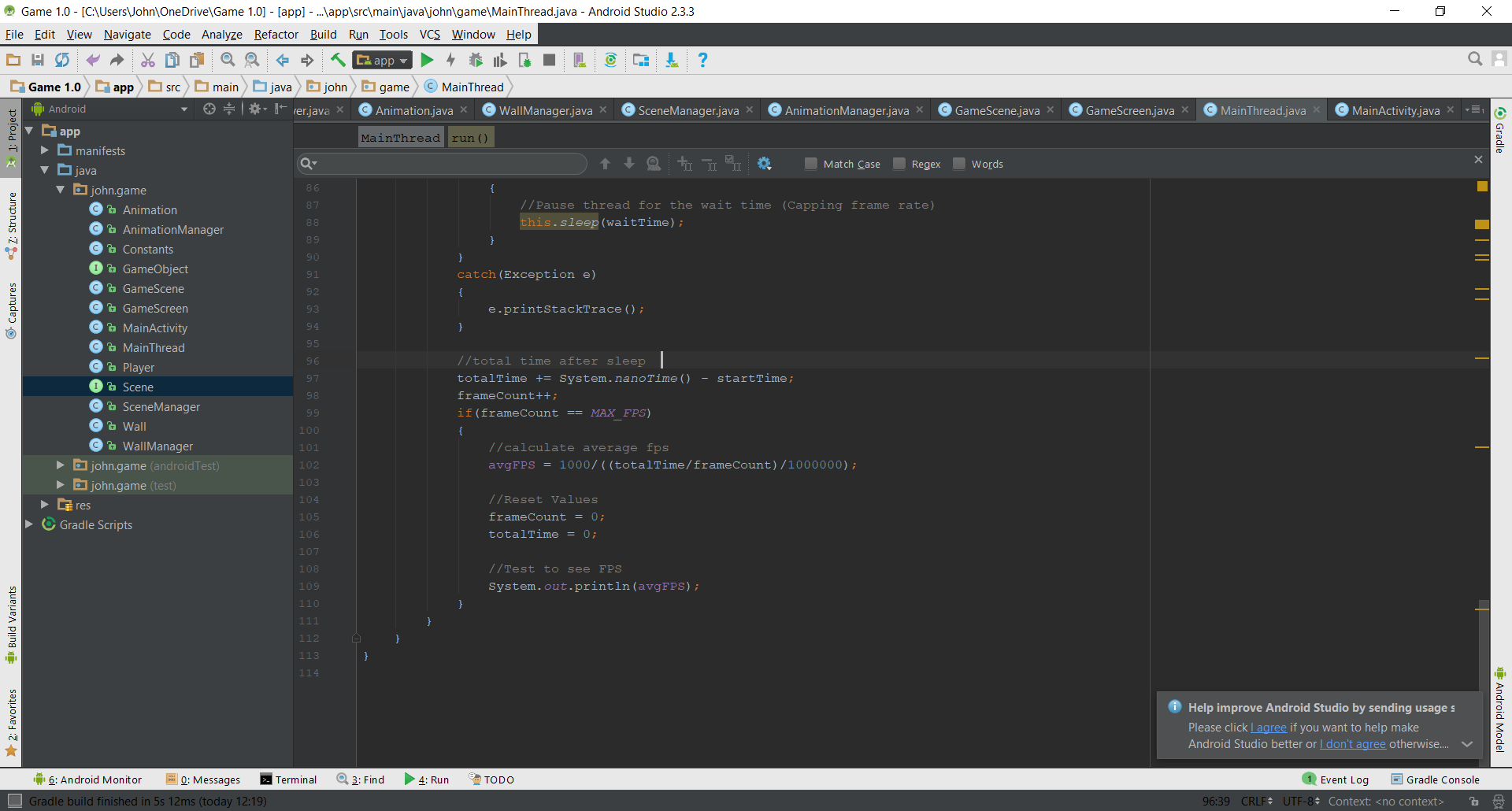


Figure 6 – Limiting the FPS

## Handling Touch Events

A SceneManager class (See Appendix A.11) was first implemented to handle multiple scenes however due to time constraints, the only scene that was developed was the GameScene (Appendix A.5). In the GameScene class a receiveTouchMethod uses the MotionEvent class to get the users action. Based on the action the appropriate action can then be carried out.

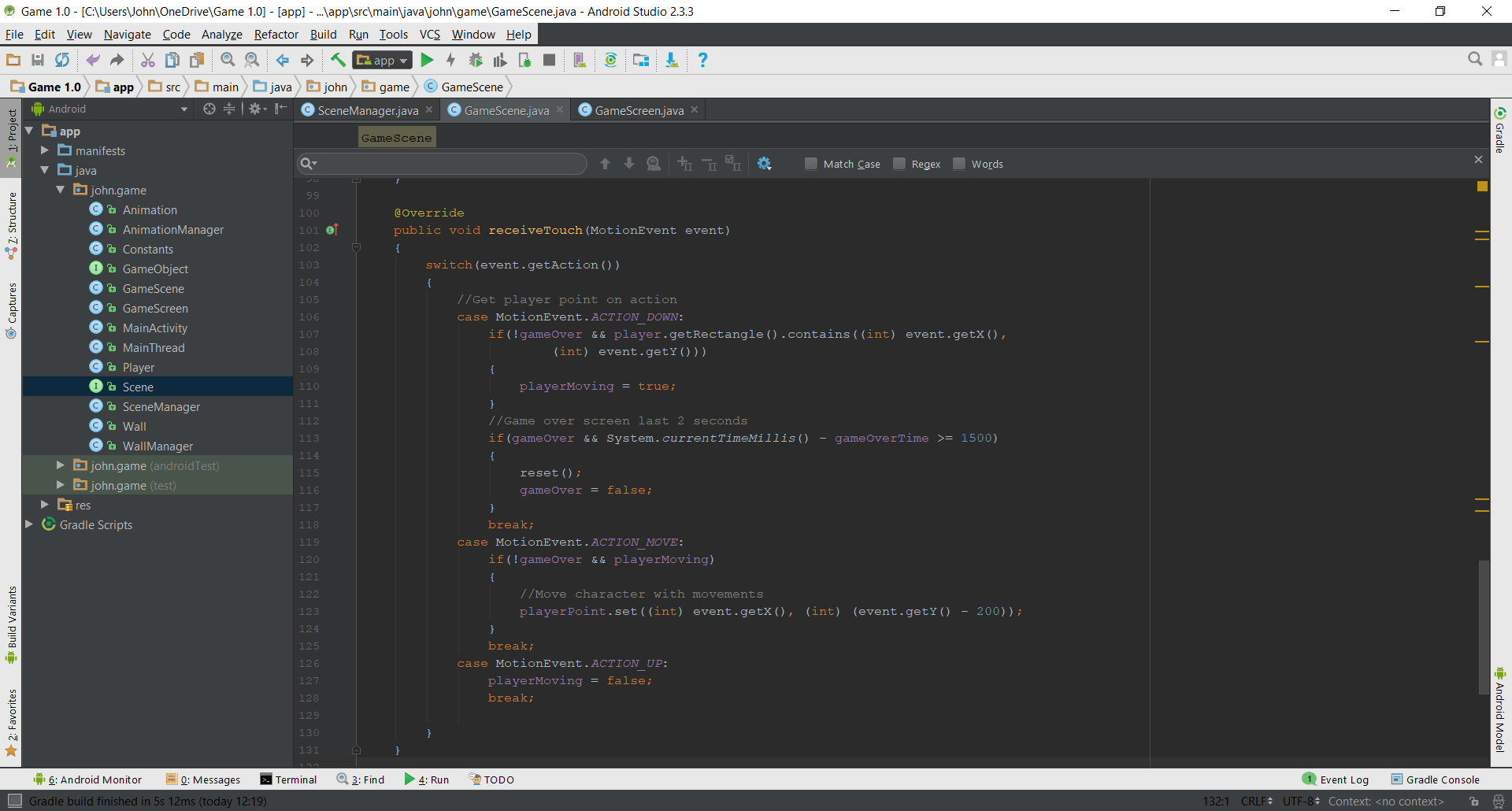
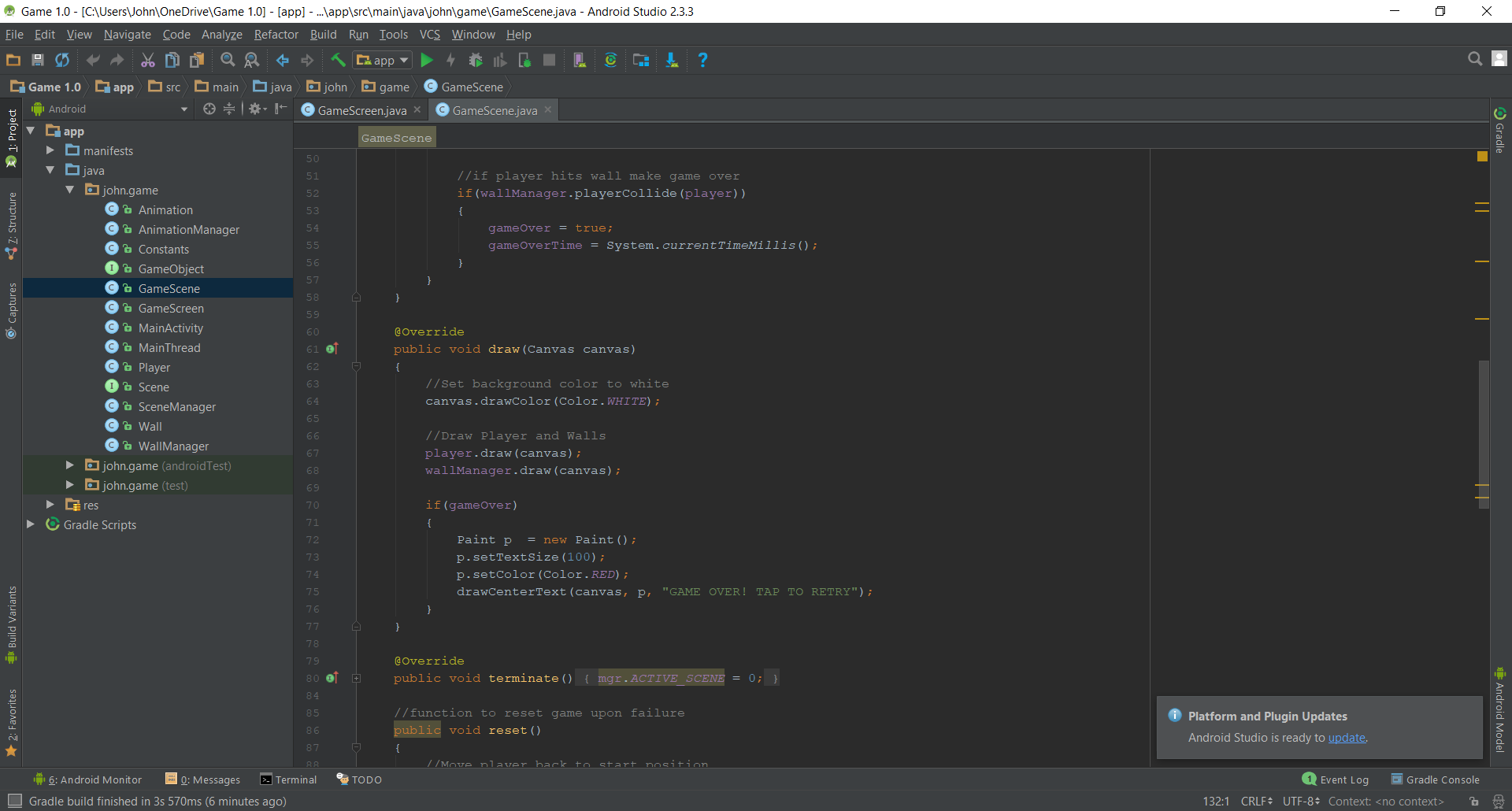


Figure 7 – Using Switch statement to read and react to user interaction.

## Displaying the Game

The displaying of the game is done through a draw() method in the GameScene class (Appendix A.5). The draw method takes in the canvas, or the background in terms of the game and initially sets it to white. The draw method of a player and wallManager instance variables are continuously called so walls and the player movement can be seen on the screen. An IF statement is used to check if the game has ended in which case it simply displays a Game Over screen to the user.

  
Figure 8 – Initial drawing of Game.

## Handling Player Movement

As seen in Figure 9 an update method in the GameScene class (Appendix A.5) is continuously being run whilst the game is not over. The point where the user is currently pressing is being passed to an instance of the Player class.

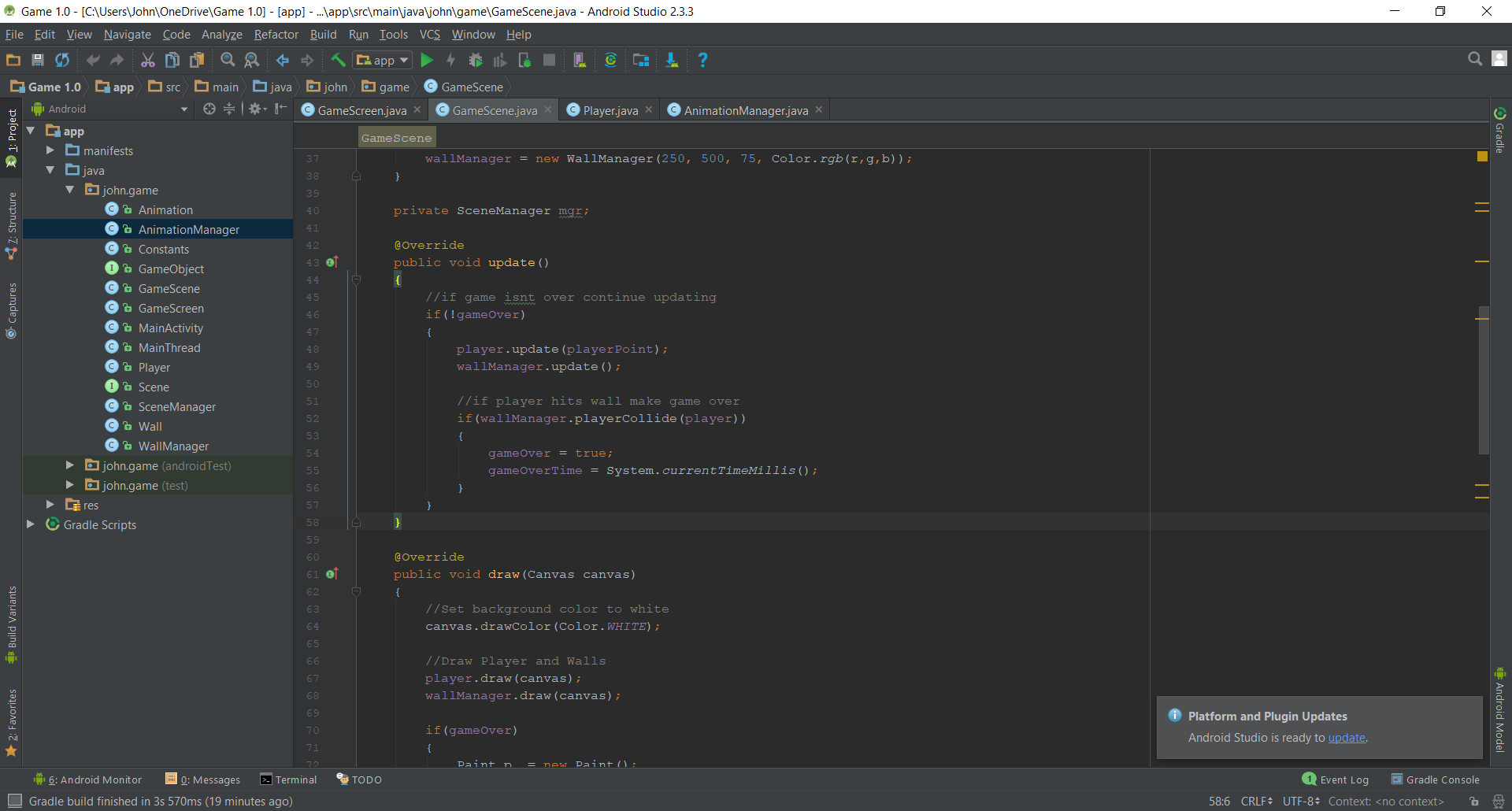


Figure 9 – GameScene updating player and walls.

Fundamentally, the game character is contained within a rectangle which grants access to use methods to get the rectangles dimensions. This is utilized when detecting player movement. In the Player class this the value of the position of the Point is used to centre the rectangle/character around this point. This is done by setting the point to the same position as the centre of the rectangle, which as shown in Figure 10 is half of the width and height of the rectangle. By doing this the user has a clear indication of the character movement and can easily dictate and anticipate the movement of it.

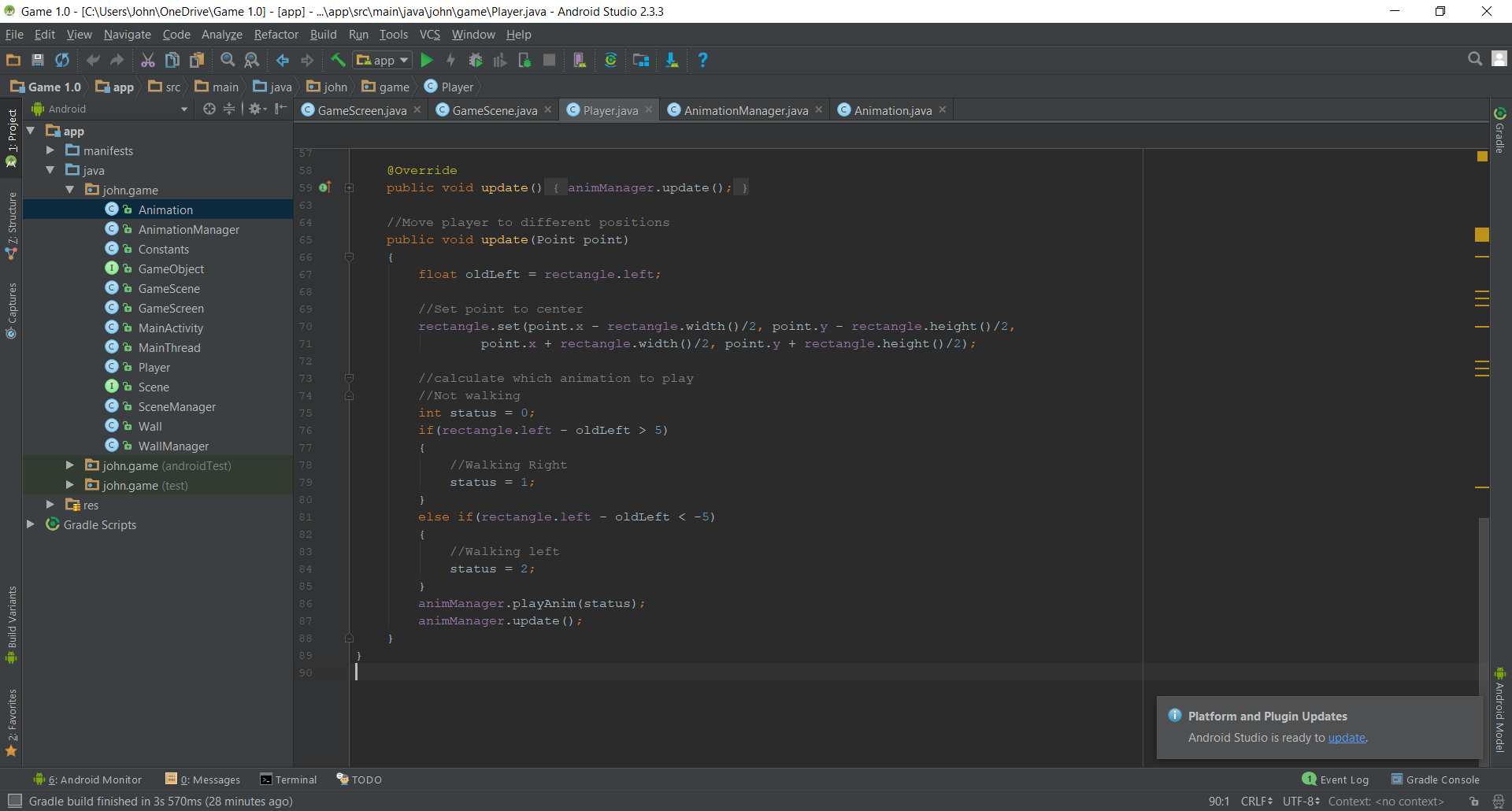


Figure 10 – Centring a rectangle around a point

### Handling Animation

Another useful feature available from containing the character within a rectangle is being able to call methods to get the position of the corners. This was utilized to provide animation to the character, with the character appearing as if it is walking in the direction the user is dragging. This was done by setting a variable for the X coordinate of the rectangle before it is centred around the Point and then comparing it to the new X coordinate of the rectangle. If it has moved five pixels either left or right, then a certain status is set as part of handling the Animation of the character (See Figure 11).

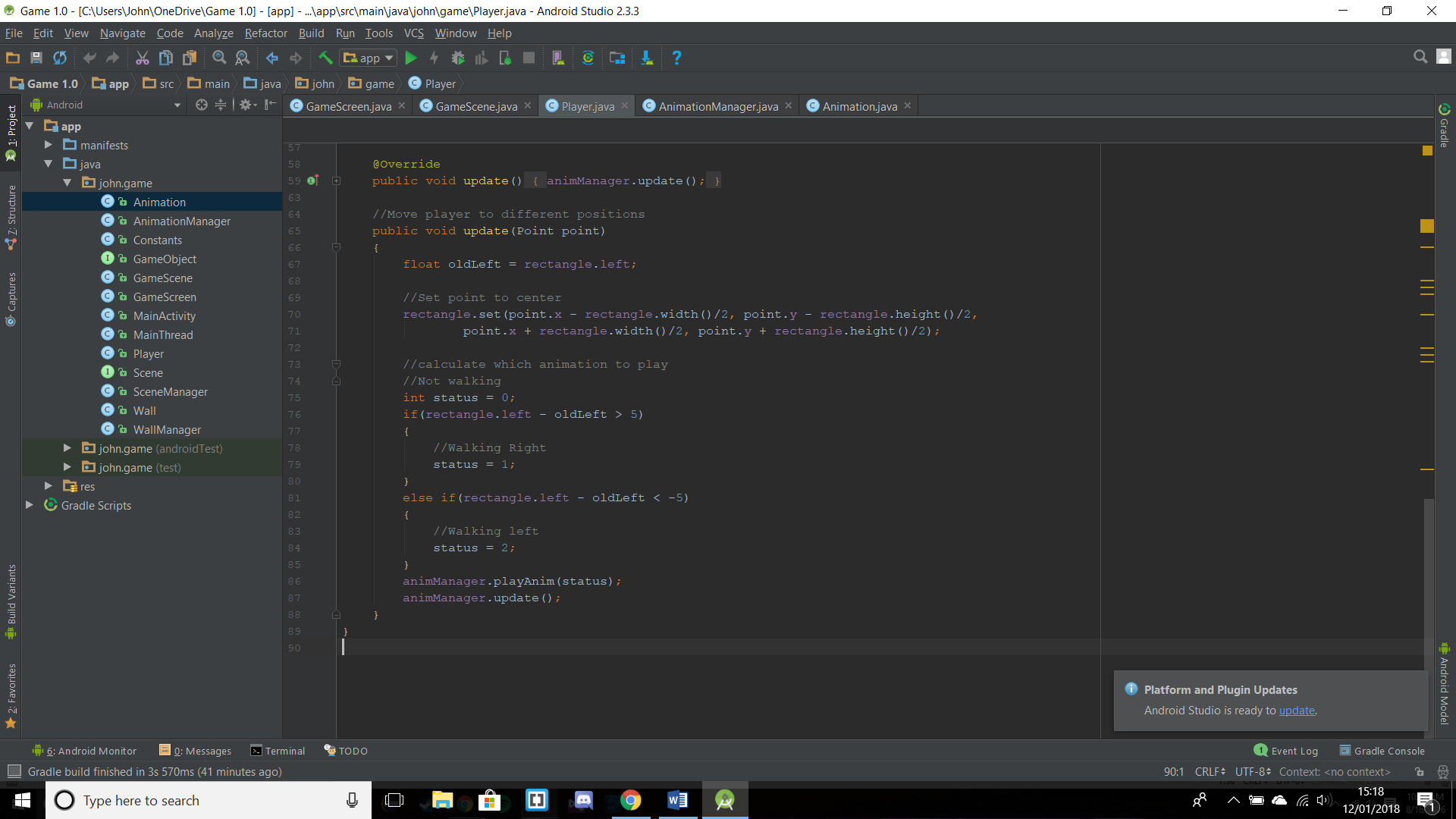


Figure 11 – Checking for movement by comparing coordinates.

In the Player class (Appendix A.9) Bitmap arrays are created storing the images which form the different steps of the Animation cycle and these are used to create Animation objects. Also, in this class an array of animations is created passing the different animations which have been created (See Figure 12). When a player has moved, and the status of direction is set, as mentioned previously the status is passed to the AnimationManager (Appendix A.2) which checks an array of animations for the corresponding animation which is subsequently played using the drawBitmap method of the canvas class.

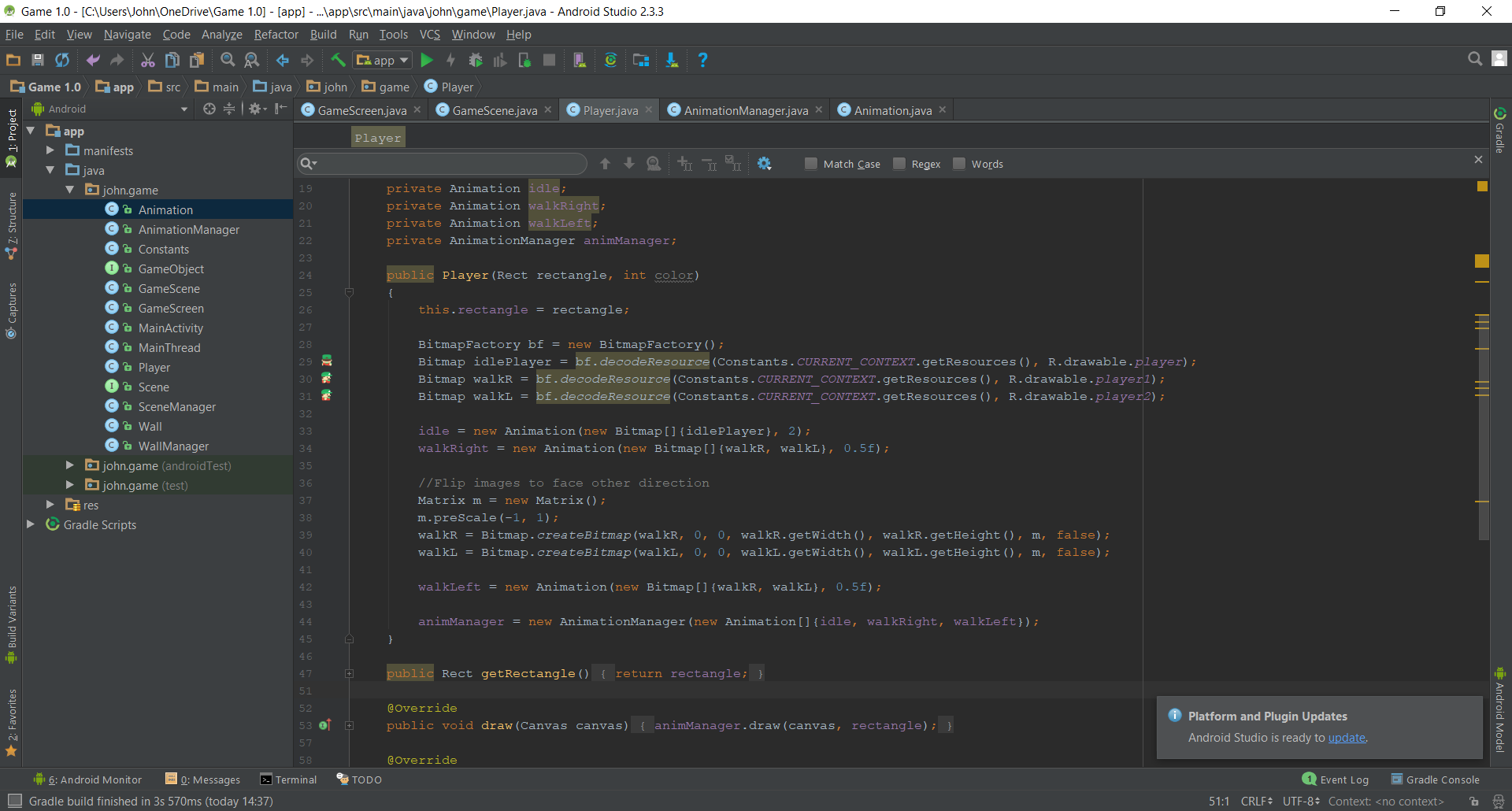


Figure 12 – The process of creating bitmaps and relative animations.

## Handling Obstacles (Walls)

### Handling the Wall Gap

Each wall that comes down a screen is a new Wall object (See Figure 13). The Wall object takes in various important parameters which make the game functional. The most important of these is the wall gap which determines the space between the 2 rectangles, allowing the character a space to go through.

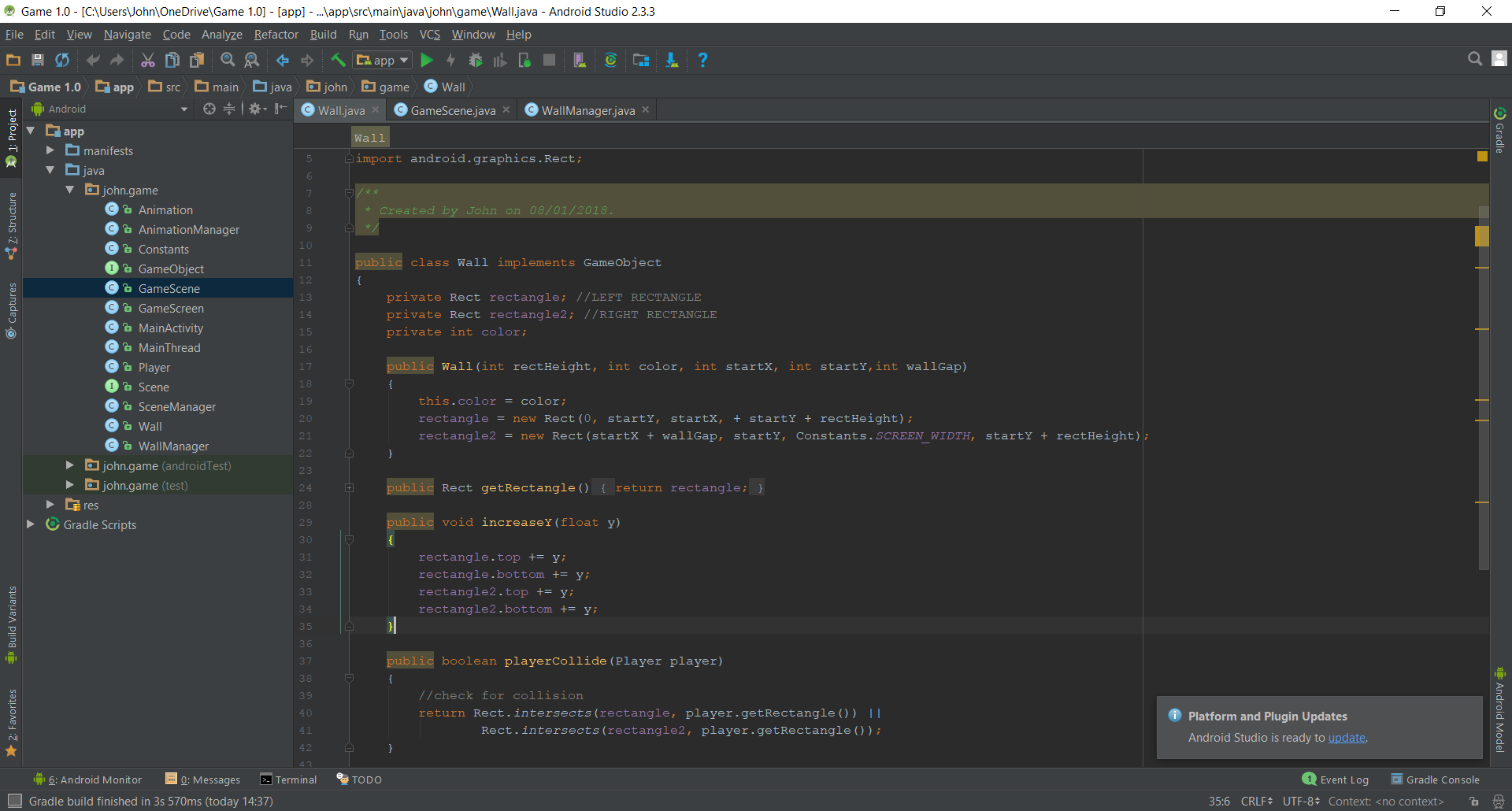


Figure 13 – Wall object parameters being used to position the 2 rectangles that form a wall.

Using this number, the 2 rectangles can be drawn a certain distance apart from each other. To make the gap appear anywhere along the width of the screen a populateWalls method in the WallManager class (Appendix A.13) calculates an X coordinate for the second rectangle to be placed by subtracting the wall gap from the width of the screen and multiplying this by a random number. Then when the wall is being created the wall gap is added to the start X coordinate of the second rectangle leaving a space between the two.

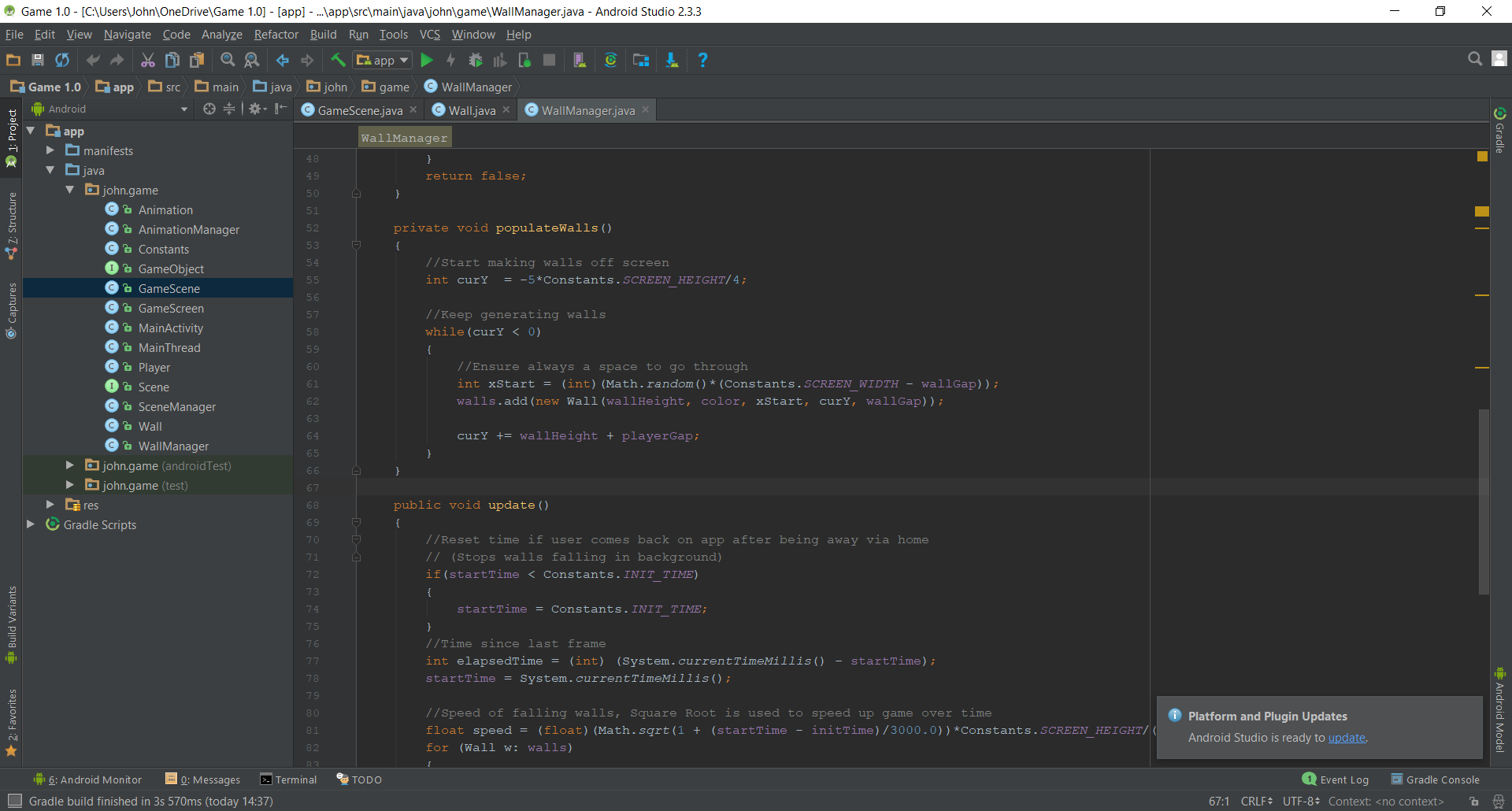


Figure 14 – How the X coordinate for the 2nd Rectangle is calculated.

### Infinite Walls

To create the never-ending arcade feel to this game the walls are infinitely produced, so as long as the player can navigate the game without colliding with one. To make this game function on different devices the walls are generated based on the screen height of the device. These walls need to be generated above the screen so that the user always has walls to navigate whilst playing. This is done by setting an initial Y value which is located above the screen. A while loop will check that is value is greater than zero which it will always be, meaning walls will keep being created. The wall gap is added as mentioned previously and the Y value for which the rectangles will be drawn is set to the same Y value which is off screen. Each set of walls has a fixed gap between them vertically so that the character has space to navigate and this is added to the initial Y position along with the wall height so that the next set of walls is separated from the previous.

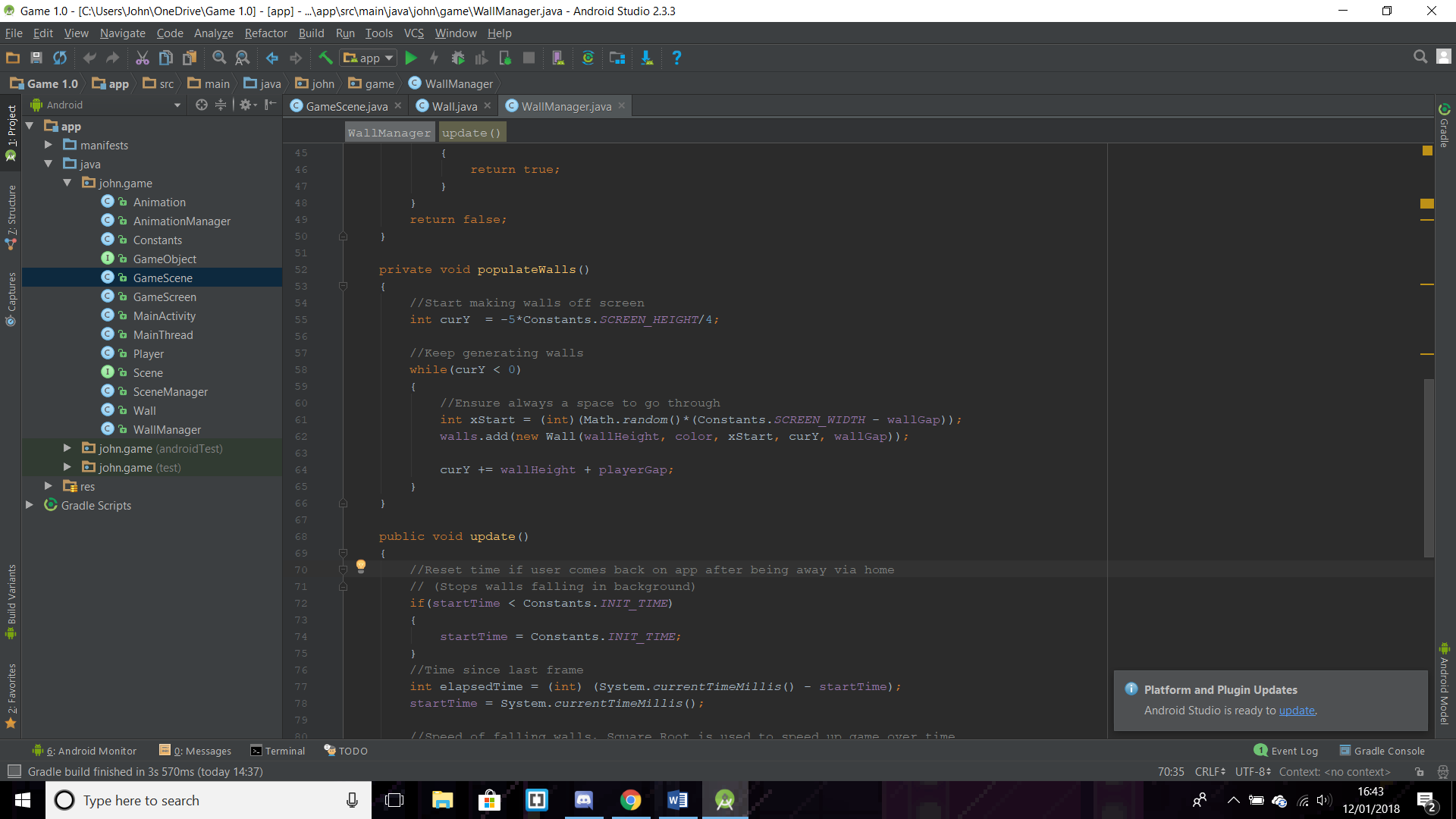


Figure 15 – The process of creating walls.

All the walls that are stored in the array are moved downwards with each update of frame. This done by calling a method which adds to the Y values of the rectangles in the array. To calculate if a wall has gone off the screen then an IF statement is used to see if the first rectangle in the array, which will be lowest on screen, has passed the height of the screen. If this is true then a new wall is created and the wall below the screen is removed from the array.

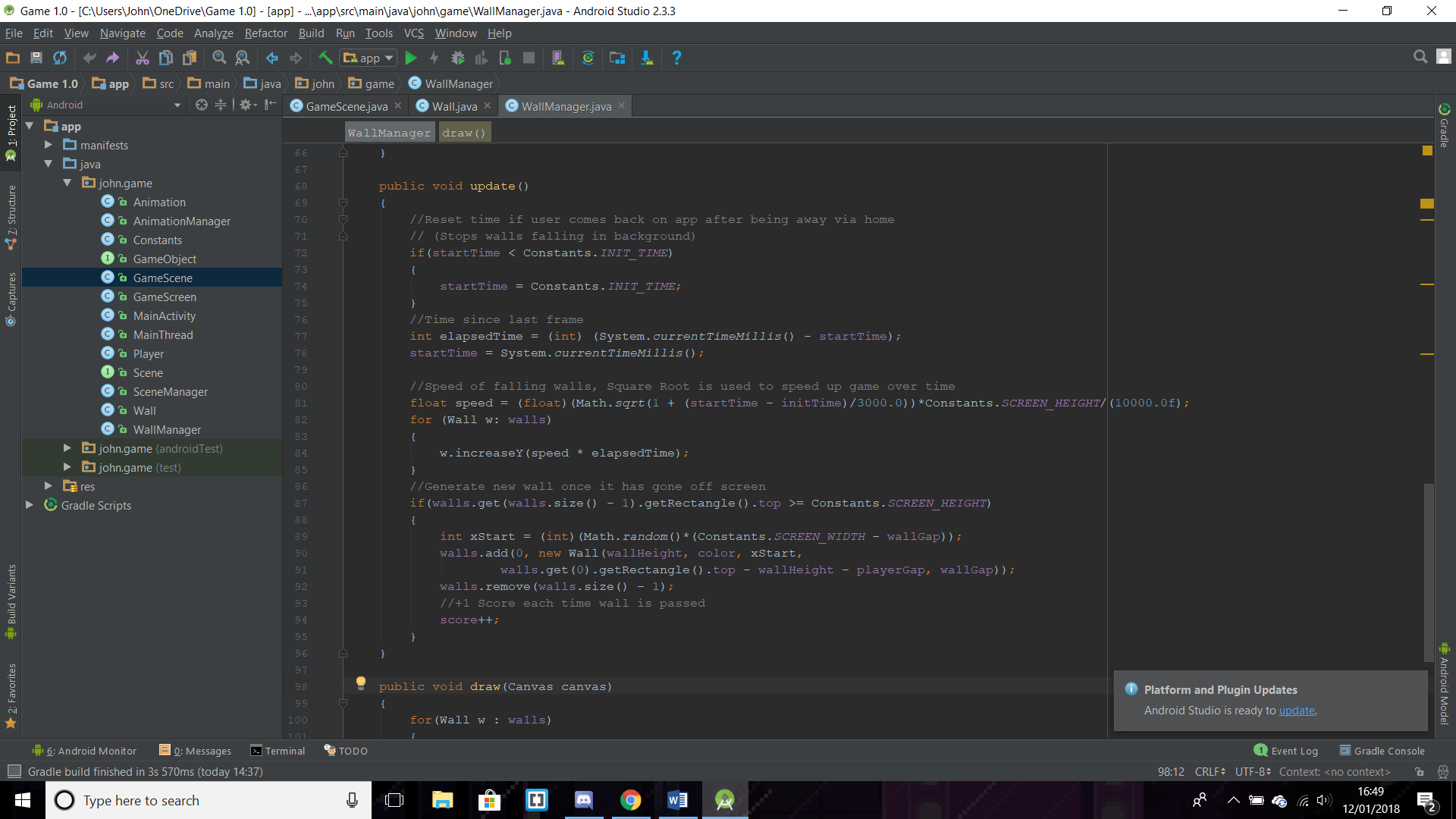


Figure 16 – How walls are generated once they have gone off the screen

### Handling Collision

To check if the character has collided with any of the walls a playerCollide method which returns a Boolean, checks to see if the characters rectangle is intersecting any of the walls. If this is the case then the method returns true and a gameOver Boolean is made true. Upon this happening the game will stop and a Game Over screen will be shown to the player.

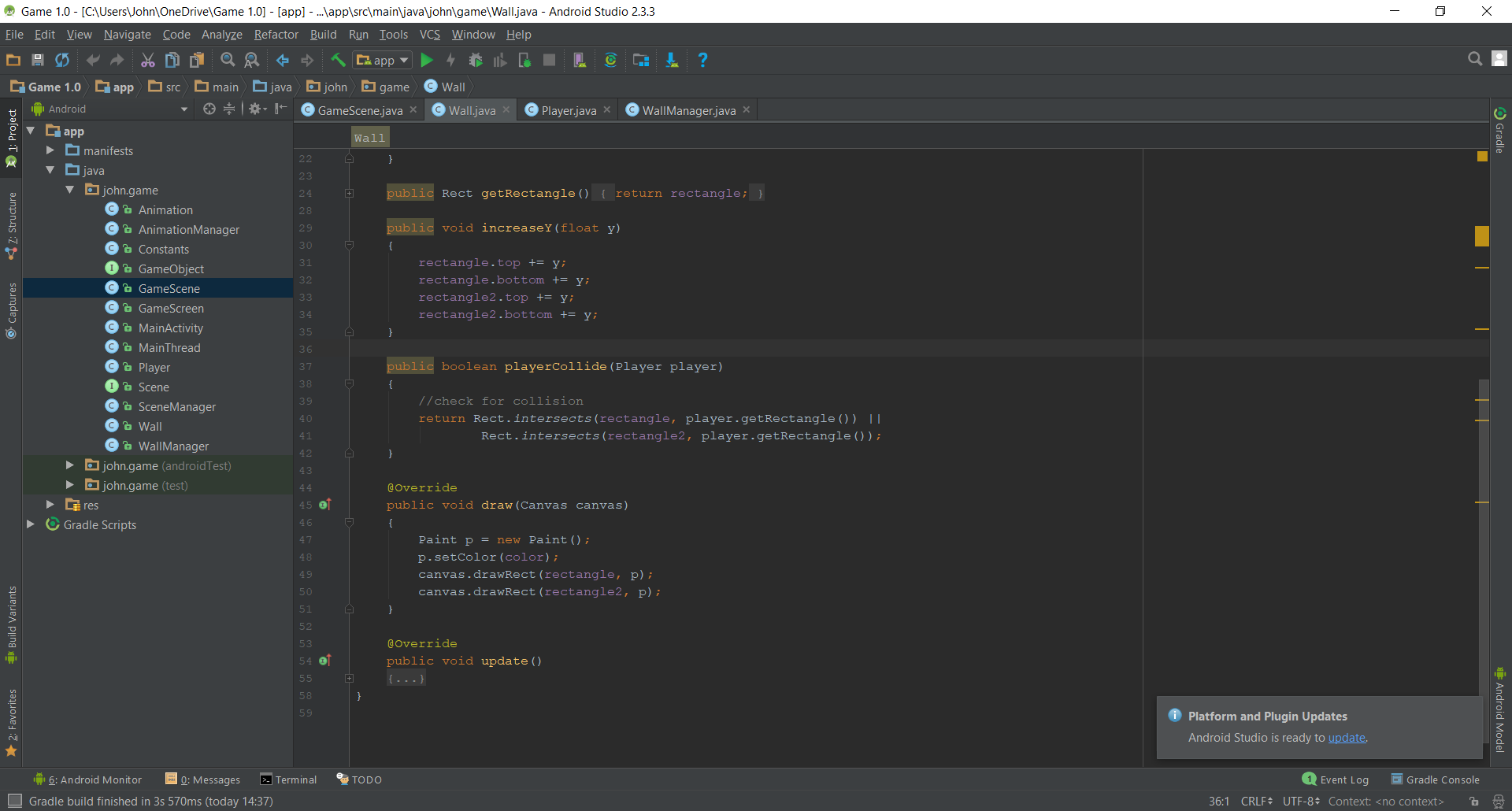


Figure 17 – Method to check for character colliding with a wall

# Evaluation

## Positives

As a game Run Steve, Run! (RSR!) has the same never ending, addictive feel seen by similar games such as Flappy Bird, a game which at one point was so downloaded it was the “number-one free game in 53 countries in the App Store” (Warren, 2014). In Flappy Bird the user must navigate a character through vertical walls, a concept very similar to the one seen in this game. Also having a video game with good music is always useful as a tool for keeping a user engaged. The music the game has included is a very catchy, electronic theme, a style adopted by many other similar arcade games.

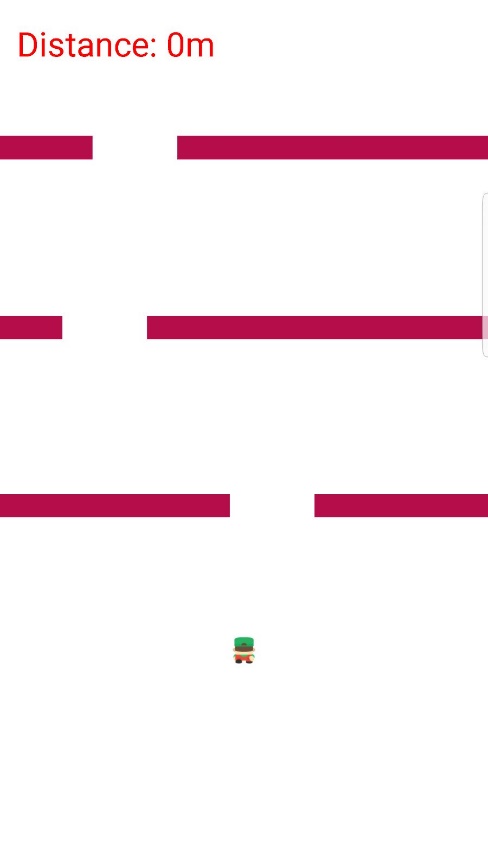


Figure 18 – Screenshot from Run Steve, Run!

RSR! also has a scoring system, which is seen in almost every video game, giving it familiarity to any gamers who would play this. The scoring system coupled with the never-ending walls would make this game very replayable as it would entice users to forever better their score and compare with their friends and family.

## Negatives

Due to the time scale the project had to be scoped to have a working application prior to the deadline. This meant that some features which would have greatly improved the game had to be left out. The first of these is the lack of an indication of a high score. This was intended to be developed using the sharedPreferences class to store a high score for the game. If the score the user achieved was greater than the high score in the preferences, then it would be updated. This would have cemented the application as a true mobile game as a high score is a concept featured in many games.

During development of the game, it was intended that there would be multiple scene classes rather than just the one GameScene hence why there is a Scene interface. An example of another Scene which could have been made is a menu prior to when the game starts which is a feature most apps have even non-gaming ones. This would have been done by creating a new scene class and having that appear on start-up instead. This could have been extended further, allowing a settings menu for the user to change certain characteristics about the game such as the character model.

## Problems Encountered

During development a bug was found where pressing the home button and leaving the app running in the background would cause the screen to go black and the game to crash upon returning. This was due to a logic error in the surfaceDestroyed code seen in Figure 19. Previously it stated while(true) which meant that the variable retry was never being accessed and causing the issue so changing this to while(retry) fixed the problem. This is issue also occurred when changing the orientation to the phone so I forced the application to only be in portrait by adding **android:screenOrientation="portrait"** to the manifest file.

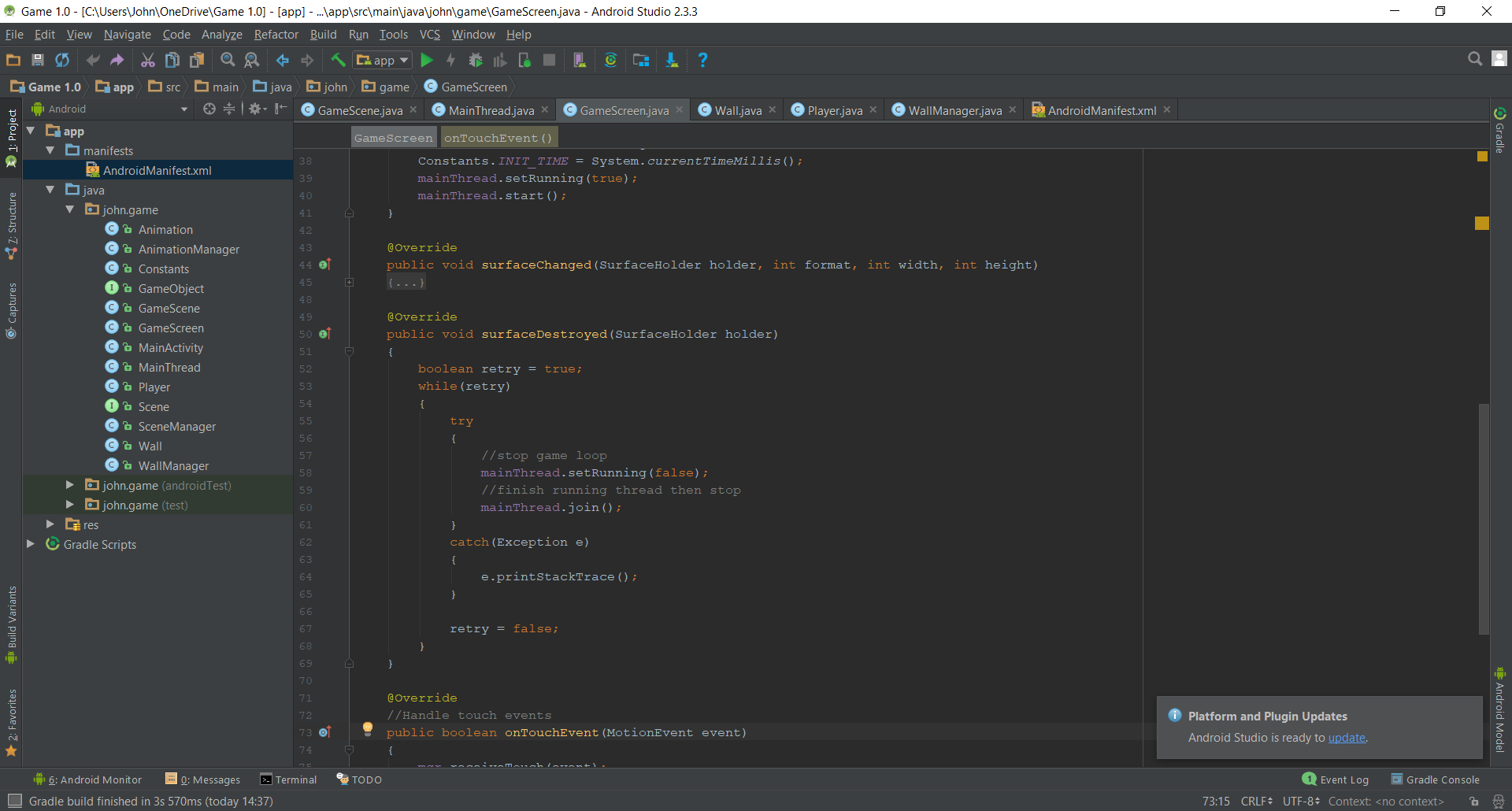


Figure 19 – Corrected surfaceDestroyed method

There was also an issue where the walls would continue falling if the user went onto another app, meaning that when they reopened it, they would have definitely collided, and the game would be over. This is where the INIT\_TIME variable in the Constants class was introduced. This variable is set in the surfaceCreated method to the time the surface is created using System.currentTimeMillis(). Then an IF statement in the WallManager class checks to see if the time the class was initialized is less than the INIT\_TIME. If the user has gone away from the game via the home button, then this IF statement will trigger and reset the startTime to the INIT\_TIME so that a time before the app was put in the background is still not being referenced.

# References

41post.com. (2011). *Game Programming Basics: Creating a FPS counter - a tutorial | 41 Post*. [online] Available at: http://www.41post.com/3099/programming/game-programming-basics-creating-a-fps-counter [Accessed 13 Jan. 2018].

McDonald, E. (2017). *THE GLOBAL GAMES MARKET WILL REACH $108.9 BILLION IN 2017 WITH MOBILE TAKING 42%*. [online] Newzoo. Available at: https://newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42/ [Accessed 11 Jan. 2018].

Warren, C. (2014). *28 Days of Fame: The Strange, True Story of 'Flappy Bird'*. [online] Mashable. Available at: http://mashable.com/2014/02/10/flappy-bird-story/#2kPboWHSEgqz [Accessed 12 Jan. 2018].

# Appendix A – Source Code

## A.1 - Animation.java

package john.game;  
  
import android.graphics.Bitmap;  
import android.graphics.Canvas;  
import android.graphics.Paint;  
import android.graphics.Rect;  
  
public class Animation  
{  
 private Bitmap[] frames;  
 private int frameIndex;  
  
 private boolean isPlaying = false;  
  
 public boolean isPlaying()  
 {  
 return isPlaying;  
 }  
  
 public void play()  
 {  
 isPlaying = true;  
 frameIndex = 0;  
 lastFrame = System.*currentTimeMillis*();  
 }  
  
 public void stop()  
 {  
 isPlaying = false;  
 }  
  
 //time in between frames  
 private float frameTime;  
  
 private long lastFrame;  
  
 public Animation(Bitmap[] frames, float animTime)  
 {  
 this.frames = frames;  
 frameIndex = 0;  
  
 frameTime = animTime/frames.length;  
 lastFrame = System.*currentTimeMillis*();  
 }  
  
 public void draw(Canvas canvas, Rect dest)  
 {  
 if(!isPlaying)  
 {  
 return;  
 }  
 scaleRect(dest);  
  
 canvas.drawBitmap(frames[frameIndex], null, dest, new Paint());  
 }  
  
 private void scaleRect(Rect rect)  
 {  
 float ratio = (float) (frames[frameIndex].getWidth())/frames[frameIndex].getHeight();  
 if(rect.width() > rect.height())  
 {  
 rect.left = rect.bottom - (int)(rect.height() \* ratio);  
 }  
 else  
 {  
 rect.top = rect.bottom - (int) (rect.width() \* (1/ratio));  
 }  
 }  
  
  
 public void update()  
 {  
 if(!isPlaying)  
 {  
 return;  
 }  
  
 if(System.*currentTimeMillis*() - lastFrame > frameTime\*1000)  
 {  
 //go to next frame  
 frameIndex++;  
 frameIndex = frameIndex >= frames.length ? 0 : frameIndex;  
 lastFrame = System.*currentTimeMillis*();  
 }  
 }  
}

## A.2 - AnimationManager.java

package john.game;  
  
import android.graphics.Canvas;  
import android.graphics.Rect;  
  
public class AnimationManager  
{  
 private Animation[] anims;  
 private int animIndex = 0;  
  
 public AnimationManager(Animation[] anims)  
 {  
 this.anims = anims;  
 }  
  
 public void playAnim(int index)  
 {  
 for(int i = 0; i < anims.length; i++)  
 {  
 if(i == index)  
 {  
 //if not already playing  
 if(!anims[index].isPlaying())  
 {  
 anims[i].play();  
 }  
 }  
 else  
 {  
 anims[i].stop();  
 }  
 }  
 animIndex = index;  
 }  
  
 public void draw(Canvas canvas, Rect rect)  
 {  
 if(anims[animIndex].isPlaying())  
 {  
 anims[animIndex].draw(canvas, rect);  
 }  
 }  
  
 public void update()  
 {  
 if(anims[animIndex].isPlaying())  
 {  
 anims[animIndex].update();  
 }  
 }  
}

## A.3 - Constants.java

package john.game;  
  
import android.content.Context;  
  
public class Constants  
{  
 //Screen dimensions  
 public static int *SCREEN\_WIDTH*;  
 public static int *SCREEN\_HEIGHT*;  
  
 public static Context *CURRENT\_CONTEXT*;  
  
 //Time app initialized (for backing out of game via home etc)  
 public static long *INIT\_TIME*;  
}

## A.4 - GameObject.java

package john.game;  
  
import android.graphics.Canvas;  
  
public interface GameObject  
{  
 public void draw(Canvas canvas);  
 public void update();  
}

## A.5 - GameScene.java

package john.game;  
  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.graphics.Point;  
import android.graphics.Rect;  
import android.view.MotionEvent;  
  
public class GameScene implements Scene  
{  
 private Player player;  
 private Rect r = new Rect();  
  
 private Point playerPoint;  
 private WallManager wallManager;  
  
 private boolean playerMoving = false;  
 private boolean gameOver = false;  
 private long gameOverTime;  
  
 public GameScene()  
 {  
 player = new Player(new Rect(100, 100, 200, 200), Color.*rgb*(255, 0, 0));  
 //Set player starting point to middle and 3/4 down screen  
 playerPoint = new Point(Constants.*SCREEN\_WIDTH*/2, 3\*Constants.*SCREEN\_HEIGHT*/4);  
 player.update(playerPoint);  
  
 int r = (int)(Math.*random*()\*256);  
 int g = (int)(Math.*random*()\*256);  
 int b = (int)(Math.*random*()\*256);  
  
 wallManager = new WallManager(250, 500, 75, Color.*rgb*(r,g,b));  
 }  
  
 private SceneManager mgr;  
  
 @Override  
 public void update()  
 {  
 //if game isnt over continue updating  
 if(!gameOver)  
 {  
 player.update(playerPoint);  
 wallManager.update();  
  
 //if player hits wall make game over  
 if(wallManager.playerCollide(player))  
 {  
 gameOver = true;  
 gameOverTime = System.*currentTimeMillis*();  
 }  
 }  
 }  
  
 @Override  
 public void draw(Canvas canvas)  
 {  
 //Set background color to white  
 canvas.drawColor(Color.*WHITE*);  
  
 //Draw Player and Walls  
 player.draw(canvas);  
 wallManager.draw(canvas);  
  
 if(gameOver)  
 {  
 Paint p = new Paint();  
 p.setTextSize(100);  
 p.setColor(Color.*RED*);  
 drawCenterText(canvas, p, "GAME OVER! TAP TO RETRY");  
 }  
 }  
  
 @Override  
 public void terminate()  
 {  
 mgr.*ACTIVE\_SCENE* = 0;  
 }  
  
 //function to reset game upon failure  
 public void reset()  
 {  
 //Move player back to start position  
 playerPoint = new Point(Constants.*SCREEN\_WIDTH*/2, 3\*Constants.*SCREEN\_HEIGHT*/4);  
 player.update(playerPoint);  
 //Reset walls  
 int r = (int)(Math.*random*()\*256);  
 int g = (int)(Math.*random*()\*256);  
 int b = (int)(Math.*random*()\*256);  
  
 wallManager = new WallManager(250, 500, 75, Color.*rgb*(r,g,b));  
 playerMoving = false;  
 }  
  
 @Override  
 public void receiveTouch(MotionEvent event)  
 {  
 switch(event.getAction())  
 {  
 //Get player point on action  
 case MotionEvent.*ACTION\_DOWN*:  
 if(!gameOver && player.getRectangle().contains((int) event.getX(),  
 (int) event.getY()))  
 {  
 playerMoving = true;  
 }  
 //Game over screen last 2 seconds  
 if(gameOver && System.*currentTimeMillis*() - gameOverTime >= 1500)  
 {  
 reset();  
 gameOver = false;  
 }  
 break;  
 case MotionEvent.*ACTION\_MOVE*:  
 if(!gameOver && playerMoving)  
 {  
 //Move character with movements  
 playerPoint.set((int) event.getX(), (int) (event.getY() - 200));  
 }  
 break;  
 case MotionEvent.*ACTION\_UP*:  
 playerMoving = false;  
 break;  
  
 }  
 }  
  
 //Method to draw text in center of screen  
 private void drawCenterText(Canvas canvas, Paint paint, String text) {  
 paint.setTextAlign(Paint.Align.*LEFT*);  
 canvas.getClipBounds(r);  
 int cHeight = r.height();  
 int cWidth = r.width();  
 paint.getTextBounds(text, 0, text.length(), r);  
 float x = cWidth / 2f - r.width() / 2f - r.left;  
 float y = cHeight / 2f + r.height() / 2f - r.bottom;  
 canvas.drawText(text, x, y, paint);  
 }  
}

## A.6 - GameScreen.java

package john.game;  
  
import android.content.Context;  
import android.graphics.Canvas;  
import android.view.MotionEvent;  
import android.view.SurfaceHolder;  
import android.view.SurfaceView;  
  
public class GameScreen extends SurfaceView implements SurfaceHolder.Callback  
{  
 private MainThread mainThread;  
 private SceneManager mgr;  
  
 public GameScreen(Context context)  
 {  
 super(context);  
  
 getHolder().addCallback(this);  
  
 //Make current context equal to this context  
 Constants.*CURRENT\_CONTEXT* = context;  
  
 mainThread = new MainThread(getHolder(), this);  
 mgr = new SceneManager();  
  
 //change view focus  
 setFocusable(true);  
 }  
  
 @Override  
 public void surfaceCreated(SurfaceHolder holder)  
 {  
 mainThread = new MainThread(getHolder(), this);  
 Constants.*INIT\_TIME* = System.*currentTimeMillis*();  
 mainThread.setRunning(true);  
 mainThread.start();  
 }  
  
 @Override  
 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height)  
 {  
  
 }  
  
 @Override  
 public void surfaceDestroyed(SurfaceHolder holder)  
 {  
 boolean retry = true;  
 while(retry)  
 {  
 try  
 {  
 //stop game loop  
 mainThread.setRunning(false);  
 //finish running thread then stop  
 mainThread.join();  
 }  
 catch(Exception e)  
 {  
 e.printStackTrace();  
 }  
  
 retry = false;  
 }  
 }  
  
 @Override  
 //Handle touch events  
 public boolean onTouchEvent(MotionEvent event)  
 {  
 mgr.receiveTouch(event);  
 return true;  
 }  
  
 //Update game frame by frame  
 public void update()  
 {  
 mgr.update();  
 }  
  
 //Display Game  
 @Override  
 public void draw(Canvas canvas)  
 {  
 super.draw(canvas);  
 mgr.draw(canvas);  
 }  
}

## A.7 - MainActivity.java

package john.game;  
  
import android.app.Activity;  
import android.media.MediaPlayer;  
import android.os.Bundle;  
import android.util.DisplayMetrics;  
import android.view.Window;  
import android.view.WindowManager;  
  
public class MainActivity extends Activity  
{  
 MediaPlayer music;  
 @Override  
 protected void onCreate(Bundle savedInstanceState)  
 {  
 super.onCreate(savedInstanceState);  
  
 //Hide Status Bar  
 getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*,  
 WindowManager.LayoutParams.*FLAG\_FULLSCREEN*);  
 //Hide Activity Title  
 this.requestWindowFeature(Window.*FEATURE\_NO\_TITLE*);  
  
 //Store screen dimensions in constants class  
 DisplayMetrics dm = new DisplayMetrics();  
 getWindowManager().getDefaultDisplay().getMetrics(dm);  
 Constants.*SCREEN\_WIDTH* = dm.widthPixels;  
 Constants.*SCREEN\_HEIGHT* = dm.heightPixels;  
  
 setContentView(new GameScreen(this));  
  
 music = MediaPlayer.*create*(MainActivity.this, R.raw.*music*);  
 music.setLooping(true);  
 music.start();  
 }  
  
 @Override  
 protected void onPause()  
 {  
 super.onPause();  
 music.pause();  
 }  
  
 @Override  
 protected void onResume()  
 {  
 super.onResume();  
 music.start();  
 }  
}

## A.8 - MainThread.java

package john.game;  
  
import android.graphics.Canvas;  
import android.view.SurfaceHolder;  
  
public class MainThread extends Thread  
{  
 //Cap fps for game to avoid unecessary calls to game loop  
 public static final int *MAX\_FPS* = 30;  
 private double avgFPS;  
 private SurfaceHolder surfaceHolder;  
 private GameScreen gameScreen;  
 private boolean running;  
 public static Canvas *canvas*;  
  
 public void setRunning(boolean running)  
 {  
 this.running = running;  
 }  
  
 public MainThread(SurfaceHolder surfaceHolder, GameScreen gameScreen)  
 {  
 super();  
 this.surfaceHolder = surfaceHolder;  
 this.gameScreen = gameScreen;  
 }  
  
 @Override  
 public void run()  
 {  
 long startTime;  
 //1000 Ms = 1 Second  
 long timeMillis = 1000/*MAX\_FPS*;  
 long waitTime;  
 int frameCount = 0;  
 int totalTime = 0;  
 long targetTime = 1000/*MAX\_FPS*;  
  
 while(running)  
 {  
 //Get System Time  
 startTime = System.*nanoTime*();  
 *canvas* = null;  
  
 try  
 {  
 //Create surface area to write to and prevent simultaneous accessing of code  
 *canvas* = this.surfaceHolder.lockCanvas();  
  
 // prevents concurrent access to a block of code  
 synchronized (surfaceHolder)  
 {  
 this.gameScreen.update(); //Handle Changes in Game  
 this.gameScreen.draw(*canvas*); //Draw changes  
 }  
 }  
 catch(Exception e)  
 {  
 e.printStackTrace();  
 }  
 finally  
 {  
 if(*canvas* != null)  
 {  
 try  
 {  
 surfaceHolder.unlockCanvasAndPost(*canvas*);  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
 }  
 }  
 //Divide by 1M to get time in millis  
 timeMillis = (System.*nanoTime*() - startTime)/1000000;  
 waitTime = targetTime - timeMillis;  
 try  
 {  
 //If frame finishes before target time  
 if(waitTime > 0)  
 {  
 //Pause thread for the wait time (Capping frame rate)  
 this.*sleep*(waitTime);  
 }  
 }  
 catch(Exception e)  
 {  
 e.printStackTrace();  
 }  
  
 //total time after sleep  
 totalTime += System.*nanoTime*() - startTime;  
 frameCount++;  
 if(frameCount == *MAX\_FPS*)  
 {  
 //calculate average fps  
 avgFPS = 1000/((totalTime/frameCount)/1000000);  
  
 //Reset Values  
 frameCount = 0;  
 totalTime = 0;  
  
 //Test to see FPS  
 System.*out*.println(avgFPS);  
 }  
 }  
 }  
}

## A.9 - Player.java

package john.game;  
  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Canvas;  
import android.graphics.Matrix;  
import android.graphics.Point;  
import android.graphics.Rect;  
  
public class Player implements GameObject  
{  
 private Rect rectangle;  
  
 //Animation Variables  
 private Animation idle;  
 private Animation walkRight;  
 private Animation walkLeft;  
 private AnimationManager animManager;  
  
 public Player(Rect rectangle, int color)  
 {  
 this.rectangle = rectangle;  
  
 BitmapFactory bf = new BitmapFactory();  
 Bitmap idlePlayer = bf.*decodeResource*(Constants.*CURRENT\_CONTEXT*.getResources(), R.drawable.*player*);  
 Bitmap walkR = bf.*decodeResource*(Constants.*CURRENT\_CONTEXT*.getResources(), R.drawable.*player1*);  
 Bitmap walkL = bf.*decodeResource*(Constants.*CURRENT\_CONTEXT*.getResources(), R.drawable.*player2*);  
  
 idle = new Animation(new Bitmap[]{idlePlayer}, 2);  
 walkRight = new Animation(new Bitmap[]{walkR, walkL}, 0.5f);  
  
 //Flip images to face other direction  
 Matrix m = new Matrix();  
 m.preScale(-1, 1);  
 walkR = Bitmap.*createBitmap*(walkR, 0, 0, walkR.getWidth(), walkR.getHeight(), m, false);  
 walkL = Bitmap.*createBitmap*(walkL, 0, 0, walkL.getWidth(), walkL.getHeight(), m, false);  
  
 walkLeft = new Animation(new Bitmap[]{walkR, walkL}, 0.5f);  
  
 animManager = new AnimationManager(new Animation[]{idle, walkRight, walkLeft});  
 }  
  
 public Rect getRectangle()  
 {  
 return rectangle;  
 }  
  
 @Override  
 public void draw(Canvas canvas)  
 {  
 animManager.draw(canvas, rectangle);  
 }  
  
 @Override  
 public void update()  
 {  
 animManager.update();  
 }  
  
 //Move player to different positions  
 public void update(Point point)  
 {  
 float oldLeft = rectangle.left;  
  
 //Set point to center  
 rectangle.set(point.x - rectangle.width()/2, point.y - rectangle.height()/2,  
 point.x + rectangle.width()/2, point.y + rectangle.height()/2);  
  
 //calculate which animation to play  
 //Not walking  
 int status = 0;  
 if(rectangle.left - oldLeft > 5)  
 {  
 //Walking Right  
 status = 1;  
 }  
 else if(rectangle.left - oldLeft < -5)  
 {  
 //Walking left  
 status = 2;  
 }  
 animManager.playAnim(status);  
 animManager.update();  
 }  
}

## A.10 - Scene.java

package john.game;  
  
import android.graphics.Canvas;  
import android.view.MotionEvent;  
  
public interface Scene  
{  
 public void update();  
 public void draw(Canvas canvas);  
 //Tells Scene manager to switch active screen  
 public void terminate();  
 public void receiveTouch(MotionEvent event);  
}

## A.11 - SceneManager.java

package john.game;  
  
import android.graphics.Canvas;  
import android.view.MotionEvent;  
import java.util.ArrayList;  
  
public class SceneManager  
{  
 private ArrayList<Scene> scenes = new ArrayList<>();  
 public static int *ACTIVE\_SCENE*;  
  
 public SceneManager()  
 {  
 *ACTIVE\_SCENE* = 0;  
 scenes.add(new GameScene());  
 }  
  
 //Send touch event to scene  
 public void receiveTouch(MotionEvent event)  
 {  
 scenes.get(*ACTIVE\_SCENE*).receiveTouch(event);  
 }  
  
 public void update()  
 {  
 scenes.get(*ACTIVE\_SCENE*).update();  
 }  
  
 public void draw(Canvas canvas)  
 {  
 scenes.get(*ACTIVE\_SCENE*).draw(canvas);  
 }  
}

## A.12 - Wall.java

package john.game;  
  
import android.graphics.Canvas;  
import android.graphics.Paint;  
import android.graphics.Rect;  
  
public class Wall implements GameObject  
{  
 private Rect rectangle; //LEFT RECTANGLE  
 private Rect rectangle2; //RIGHT RECTANGLE  
 private int color;  
  
 public Wall(int rectHeight, int color, int startX, int startY,int wallGap)  
 {  
 this.color = color;  
 rectangle = new Rect(0, startY, startX, + startY + rectHeight);  
 rectangle2 = new Rect(startX + wallGap, startY, Constants.*SCREEN\_WIDTH*, startY + rectHeight);  
 }  
  
 public Rect getRectangle()  
 {  
 return rectangle;  
 }  
  
 public void increaseY(float y)  
 {  
 rectangle.top += y;  
 rectangle.bottom += y;  
 rectangle2.top += y;  
 rectangle2.bottom += y;  
 }  
  
 public boolean playerCollide(Player player)  
 {  
 //check for collision  
 return Rect.*intersects*(rectangle, player.getRectangle()) ||  
 Rect.*intersects*(rectangle2, player.getRectangle());  
 }  
  
 @Override  
 public void draw(Canvas canvas)  
 {  
 Paint p = new Paint();  
 p.setColor(color);  
 canvas.drawRect(rectangle, p);  
 canvas.drawRect(rectangle2, p);  
 }  
  
 @Override  
 public void update()  
 {  
  
 }  
}

## A.13 - WallManager.java

package john.game;  
  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import java.util.ArrayList;  
  
  
public class WallManager  
{  
 private ArrayList<Wall> walls; // Lowest Index is the highest wall on screen  
 private int wallGap; // GAP TO GO THROUGH  
 private int playerGap; //GAP BETWEEN WALLS  
 private int wallHeight;  
 private int color;  
  
 private long startTime;  
 private long initTime; // Time class was initialized  
  
 private int score = 0;  
  
 public WallManager(int wallGap, int playerGap, int wallHeight, int color)  
 {  
 this.wallGap = wallGap;  
 this.playerGap = playerGap;  
 this.wallHeight = wallHeight;  
 this.color = color;  
  
 startTime = initTime = System.*currentTimeMillis*();  
  
 walls = new ArrayList<>();  
 populateWalls();  
 }  
  
 //Detect if player is hitting a wall  
 public boolean playerCollide(Player player)  
 {  
 for(Wall w : walls)  
 {  
 if(w.playerCollide(player))  
 {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private void populateWalls()  
 {  
 //Start making walls off screen  
 int curY = -5\*Constants.*SCREEN\_HEIGHT*/4;  
  
 //Keep generating walls  
 while(curY < 0)  
 {  
 //Ensure always a space to go through  
 int xStart = (int)(Math.*random*()\*(Constants.*SCREEN\_WIDTH* - wallGap));  
 walls.add(new Wall(wallHeight, color, xStart, curY, wallGap));  
  
 curY += wallHeight + playerGap;  
 }  
 }  
  
 public void update()  
 {  
 //Reset time if user comes back on app after being away via home  
 // (Stops walls falling in background)  
 if(startTime < Constants.*INIT\_TIME*)  
 {  
 startTime = Constants.*INIT\_TIME*;  
 }  
 //Time since last frame  
 int elapsedTime = (int) (System.*currentTimeMillis*() - startTime);  
 startTime = System.*currentTimeMillis*();  
  
 //Speed of falling walls, Square Root is used to speed up game over time  
 float speed = (float)(Math.*sqrt*(1 + (startTime - initTime)/3000.0))\*Constants.*SCREEN\_HEIGHT*/(10000.0f);  
 for (Wall w: walls)  
 {  
 w.increaseY(speed \* elapsedTime);  
 }  
 //Generate new wall once it has gone off screen  
 if(walls.get(walls.size() - 1).getRectangle().top >= Constants.*SCREEN\_HEIGHT*)  
 {  
 int xStart = (int)(Math.*random*()\*(Constants.*SCREEN\_WIDTH* - wallGap));  
 walls.add(0, new Wall(wallHeight, color, xStart,  
 walls.get(0).getRectangle().top - wallHeight - playerGap, wallGap));  
 walls.remove(walls.size() - 1);  
 //+1 Score each time wall is passed  
 score++;  
 }  
 }  
  
 public void draw(Canvas canvas)  
 {  
 for(Wall w : walls)  
 {  
 w.draw(canvas);  
 }  
 Paint p = new Paint();  
 p.setTextSize(100);  
 p.setColor(Color.*RED*);  
 //p.descent - p.ascent distance between bottom and top of text  
 canvas.drawText("Distance: " + score + "m", 50, 50 + p.descent() - p.ascent(), p);  
 }  
}

## A.14 – Manifests.xml

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="john.game">  
  
 <application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme">  
 <activity android:name=".MainActivity"  
 android:screenOrientation="portrait">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
  
</manifest>