```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hafezi.games.spaceshooter2d">

    <supports-screens
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="false"
        android:xlargeScreens="false" />

    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:screenOrientation="sensorLandscape"
            android:theme="@style/Theme.AppCompat.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
        <activity
            android:name=".GameActivity"
            android:screenOrientation="sensorLandscape"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <activity
            android:name=".OptionsActivity"
            android:screenOrientation="sensorLandscape"
            android:theme="@style/Theme.AppCompat.NoActionBar" />
        <activity
            android:name=".BluetoothActivity"
            android:screenOrientation="sensorLandscape"
            android:theme="@style/Theme.AppCompat.NoActionBar" />
        <activity
            android:name=".HighScoreActivity"
            android:screenOrientation="sensorLandscape"
            android:theme="@style/Theme.AppCompat.NoActionBar" ></activity>
    </application>

</manifest>
```

```java
package com.hafezi.games.spaceshooter2d.Database;

/**
 * Created by Mojtaba Hafezi on 28.02.2018.
 */

//constants for the database queries
public class Constants {
    public static final String DATABASE_NAME="datastorage.db";
    public static final int DATABASE_VERSION=1;
    public static final String TABLE_NAME="scores";
    public static final String SCORE="score";
    public static final String SHIPS="ships";
    public static final String KEY_ID = "_id";
}
```

```java
package com.hafezi.games.spaceshooter2d.Database;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.provider.SyncStateContract;
import android.util.Log;

/**
 * Created by Mojtaba Hafezi on 28.02.2018.
 */

public class GameDataBase {
    private SQLiteDatabase db;
    private final Context context;
    private final MyDBhelper dbhelper;

    public GameDataBase(Context context) {
        this.context = context;
        dbhelper = new MyDBhelper(context, Constants.DATABASE_NAME, null,
Constants.DATABASE_VERSION);
        openWritable();
    }

    public void close() {
        db.close();
    }

    public void openWritable() throws SQLiteException {
        try {
            db = dbhelper.getWritableDatabase();
        } catch (SQLiteException ex) {
            Log.e("DB", ex.getMessage());
            db = dbhelper.getReadableDatabase();
        }
    }
    //get the readable database in case no permission is granted for the
writable
    public void openReadable() {
        db = dbhelper.getReadableDatabase();
    }
```

```
    //insers the given values into the corresponding table
    public long insertScore(int time, int ships) {
        try {
            ContentValues newTaskValue = new ContentValues();
            newTaskValue.put(Constants.SCORE, time);
            newTaskValue.put(Constants.SHIPS,  ships);

            return db.insert(Constants.TABLE_NAME, null, newTaskValue);
        } catch (SQLiteException ex) {
            Log.e("DBInsert",
                    ex.getMessage() + "Insert into database exception
caught");
            return -1;
        }
    }

    //returns a cursor with the sorted query
    public Cursor getScores() {
        Cursor c = db.query(Constants.TABLE_NAME, null, null,
                null, null, null, Constants.SCORE + " DESC");
        return c;
    }
}
```

```
package com.hafezi.games.spaceshooter2d.Database;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

/**
 * Created by Mojtaba Hafezi on 28.02.2018.
 */

public class MyDBhelper extends SQLiteOpenHelper {

    //used when database is created or updated
    private static final String CREATE_TABLE = "create table " +
            Constants.TABLE_NAME + " (" +
            Constants.KEY_ID + " integer primary key autoincrement, " +
            Constants.SCORE + " int not null, " +
            Constants.SHIPS + " int not null);";

    public MyDBhelper(Context context, String name,
SQLiteDatabase.CursorFactory factory,
                      int version) {
        super(context, name, factory, version);
    }

    //create table
    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(CREATE_TABLE);
        } catch (SQLiteException ex) {
            Log.e("Create table exception", ex.getMessage());
        }
    }

    //on upgrade -> delete old data and create new table
```

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
                          int newVersion) {
        db.execSQL("drop table if exists " + Constants.TABLE_NAME);
        onCreate(db);
    }
}
```

```
package com.hafezi.games.spaceshooter2d.GameObjects;

import java.util.Random;

/**
 * Created by Mojtaba Hafezi on 21.02.2018.
 */

public class Dust extends GameObject {
    private Random random;


    public Dust(int screenX, int screenY)
    {
        random = new Random();
        setScreenX(screenX);
        setScreenY(screenY);
        setMinX(0);
        setMinY(0);
        setMaxX(getScreenX());
        setMaxY(getScreenY());
        //position the dots randomly
        setRandomAttributes();



    }
    @Override
    public void update() {
        setX(getX() - getSpeed());

        if(getX() <= getMinX())
        {
           setRandomAttributes();
        }
    }

    private void setRandomAttributes()
    {
        setX(random.nextInt(getMaxX()));
        setY(random.nextInt(getMaxY()));
        setSpeed(3+ random.nextInt(9));
    }
}
```

```
package com.hafezi.games.spaceshooter2d.GameObjects;

import android.content.Context;
import android.graphics.Rect;

import java.util.Random;

/**
```

```
 * Created by Mojtaba Hafezi on 21.02.2018.
 */

public class Enemy extends GameObject {

    private Random random;
    private int shield;

    public Enemy(Context context, int screenX, int screenY) {
        random = new Random();
        setContext(context);
        setScreenX(screenX);
        setScreenY(screenY);
        setMinX(0);
        setMinY(0);
        //choose a random enemy sprite
        String bitmapName = "enemy" + (random.nextInt(5) + 1);
        prepareBitmap(bitmapName);
        setMaxX(getScreenX());
        setMaxY(getScreenY());
        setWidth(getBitmap().getWidth());
        setHeight(getBitmap().getHeight());
        setHitbox(new Rect(getX(), getY(), getWidth(), getHeight()));
        setRandomAttributes();
    }

    public void setRandomAttributes() {
        int randomPosition = random.nextInt(getScreenY() -
getBitmap().getHeight());
        int randomSpeed = 5 + random.nextInt(5) ;
        int randomShields = 1 + random.nextInt(3);
        setX(getScreenX() + getBitmap().getWidth());
        setY(randomPosition);
        setSpeed(randomSpeed);
        setShield(randomShields);
    }

    @Override
    public void update() {

        setX(getX() - getSpeed());
        if ((getX() <= (getMinX() - getBitmap().getWidth())) || getShield()
<= 0) {
            setX(getMaxX());
            setRandomAttributes();
        }

        //update location of the rectangle collision hitbox
        getHitbox().left = getX();
        getHitbox().right = getX() + getBitmap().getWidth() -
getBitmap().getWidth() / 6;
        getHitbox().top = getY();
        getHitbox().bottom = getY() + getBitmap().getHeight() -
getBitmap().getHeight() / 6;

    }

    public int getShield() {
        return shield;
    }

    public void setShield(int shield) {
        this.shield = shield;
```

```
    }
}
package com.hafezi.games.spaceshooter2d.GameObjects;

import android.content.Context;

/**
 * Created by Mojtaba Hafezi on 21.02.2018.
 */

public class Explosion extends GameObject {

    public Explosion(Context context, int screenX, int screenY, String
name, int x, int y)
    {
        setContext(context);
        setScreenX(screenX);
        setScreenY(screenY);
        setX(x);
        setY(y);
        prepareBitmap(name);
        setWidth(getBitmap().getWidth());
        setHeight(getBitmap().getHeight());
    }

    public void setPosition(int x, int y)
    {
        setX(x);
        setY(y);
    }
    @Override
    public void update() {


    }

}
```

Explosion

```
package com.hafezi.games.spaceshooter2d.GameObjects;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Rect;
import android.os.Debug;
import android.util.Log;

import java.util.Random;

/**
 * Created by Mojtaba Hafezi on 19.02.2018.
 */
//Basic class for all game objects
public abstract class GameObject {

    //the sprite's width and height
    private int width;
    private int height;
    // the sprite
    private Bitmap bitmap;
    private Context context;
```

```
    //current position
    private int x, y;
    //limits on screen
    private int minY, maxY;
    private int minX, maxX;
    private int screenX, screenY;

    Random random;
    //collision box
    private Rect hitbox;
    private int speed;
    //all objects should have this method
    public abstract void update();

    //GETTERS AND SETTERS

    public int getWidth() {
        return width;
    }

    public void setWidth(int width) {
        this.width = width;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public Bitmap getBitmap() {
        return bitmap;
    }

    public Context getContext() {
        return context;
    }

    public void setContext(Context context) {
        this.context = context;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getMinY() {
        return minY;
    }
```

```
    public void setMinY(int minY) {
        this.minY = minY;
    }

    public int getMaxY() {
        return maxY;
    }

    public void setMaxY(int maxY) {
        this.maxY = maxY;
    }

    public int getMinX() {
        return minX;
    }

    public void setMinX(int minX) {
        this.minX = minX;
    }

    public int getMaxX() {
        return maxX;
    }

    public void setMaxX(int maxX) {
        this.maxX = maxX;
    }

    public Random getRandom() {
        return random;
    }

    public void setRandom(Random random) {
        this.random = random;
    }

    public Rect getHitbox() {
        return hitbox;
    }

    public void setHitbox(Rect hitbox) {
        this.hitbox = hitbox;
    }

    public int getSpeed() {
        return speed;
    }

    public void setSpeed(int speed) {
        this.speed = speed;
    }

    //prepares the bitmap by getting the identifier from the resources
    public void prepareBitmap(String bitmapName) {
        int resourceId =
getContext().getResources().getIdentifier(bitmapName, "drawable",
getContext().getPackageName());
        Bitmap bitmap =
BitmapFactory.decodeResource(getContext().getResources(), resourceId);
        this.bitmap = bitmap;
        //scaleBitmap();
    }
```

```
    public int getScreenY() {
        return screenY;
    }

    public void setScreenY(int screenY) {
        this.screenY = screenY;
    }

    public int getScreenX() {
        return screenX;
    }

    public void setScreenX(int screenX) {
        this.screenX = screenX;
    }

    //this method could have helped balancing the game play on different
screen resolutions
    //content scaling depending on different resolutions. Pixelated outcome
is bad but acceptable
    private void scaleBitmap() {
        int divider = 100;

        //picture has aspect ratio 4:3
        int standardWidth = 1600;
        int standardHeight = 1200;
        int optimalWidth = standardWidth / divider;
        int optimalHeight = standardHeight / divider;

        int currentWidth = getScreenX() / divider;
        int currentHeight = getScreenY() / divider;

        float widthMultiplier =  ( (float) currentWidth / (float)
optimalWidth);
        float heightMultiplier =  ( (float) currentHeight / (float)
optimalHeight);

        int desiredWidth = (int) (getBitmap().getWidth() *
widthMultiplier);
        int desiredHeight = (int) (getBitmap().getHeight() *
heightMultiplier);

        Bitmap scaledBitmap = getBitmap();
        scaledBitmap = Bitmap.createScaledBitmap(scaledBitmap,
                desiredWidth, desiredHeight, false);
        this.bitmap = scaledBitmap;
    }

}
```

GameObject

```
package com.hafezi.games.spaceshooter2d.GameObjects;

import android.content.Context;
import android.graphics.Rect;

import java.util.Random;

/**
 * Created by Mojtaba Hafezi on 27.02.2018.
 */
```

```
public class Laser extends GameObject {
    private Random random;
    private boolean isAvailable;

    public Laser(Context context,  int screenX, int screenY, int startX,
int startY) {
        setContext(context);
        setScreenX(screenX);
        setScreenY(screenY);
        setAvailable(true);
        random = new Random();
        int randomSpeed = 40 + random.nextInt(5);
        setSpeed(randomSpeed);
        prepareBitmap("laser");
        setMinY(0);
        setMinX(0);
        setMaxX(screenX - getBitmap().getWidth());
        setMaxY(screenY - getBitmap().getHeight());
        setWidth(getBitmap().getWidth());
        setHeight(getBitmap().getHeight());
        setHitbox(new Rect(getX(), getY(), getWidth(), getHeight()));
        setPosition(startX,startY);
    }

    public void setPosition(int x, int y) {
        if (x <= getMaxX() && x >= getMinX())
            setX(x);
        if (y <= getMaxY() && y >= getMinY())
            setY(y);
    }

    @Override
    public void update() {
        if(!isAvailable)
        {
            setX(getX() + getSpeed());
            if (getX() >= getMaxX()) {
                setX(getMaxX());
                setAvailable(true);
            } else
            {
                setAvailable(false);
            }

            //update location of the rectangle collision hitbox with some
margin
            getHitbox().left = getX();
            getHitbox().right = getX() + getBitmap().getWidth() -
getBitmap().getWidth() / 5;
            getHitbox().top = getY();
            getHitbox().bottom = getY() + getBitmap().getHeight() -
getBitmap().getHeight() / 5;
        }
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void setAvailable(boolean available) {
        isAvailable = available;
    }
}
```

```
package com.hafezi.games.spaceshooter2d.GameObjects;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Rect;
import android.util.Log;

import java.util.ArrayList;

/**
 * Created by Mojtaba Hafezi on 19.02.2018.
 */

public class Player extends GameObject {

    private int shields;
    private boolean moveUp;
    private boolean moveDown;
    private Laser laser;

    public Player(Context context, int startX, int startY, int speed, int
screenX, int screenY) {
        setContext(context);
        setScreenX(screenX);
        setScreenY(screenY);
        setY(startY);
        setSpeed(speed);
        setShields(2);
        prepareBitmap("player");
        setX(getBitmap().getWidth() / 3);
        setMinY(0);
        setMinX(0);
        setMaxX(screenX - getBitmap().getWidth());
        setMaxY(screenY - getBitmap().getHeight());
        setWidth(getBitmap().getWidth());
        setHeight(getBitmap().getHeight());
        setHitbox(new Rect(getX(), getY(), getWidth(), getHeight()));
        //laser is not visible until shot
        laser = new Laser(getContext(), getScreenX(), getScreenY(), -1000,
-1000);
    }

    @Override
    public void update() {
        laser.update();

        if (isMoveDown()) {
            setY(getY() + getSpeed());
        }
        if (isMoveUp()) {
            setY(getY() - getSpeed());
        }

        if (getY() <= getMinY()) {
            setY(getMinY());
        }
        if (getY() >= getMaxY()) {
            setY(getMaxY());
        }

        //update location of the rectangle collision hitbox with some
```

```
margin
        getHitbox().left = getX();
        getHitbox().right = getX() + getBitmap().getWidth() -
getBitmap().getWidth() / 5;
        getHitbox().top = getY();
        getHitbox().bottom = getY() + getBitmap().getHeight() -
getBitmap().getHeight() / 5;


    }

    public void fireLaser() {
        if (laser.isAvailable()) {
            laser.setAvailable(false);
            laser.setPosition(getX() + getWidth(), getY() + getHeight() /
2);
        }
    }

    public int getShields() {
        return shields;
    }

    public void setShields(int shields) {
        this.shields = shields;
    }


    public boolean isMoveUp() {
        return moveUp;
    }

    public void setMoveUp(boolean moveUp) {
        this.moveUp = moveUp;
    }

    public boolean isMoveDown() {
        return moveDown;
    }

    public void setMoveDown(boolean moveDown) {
        this.moveDown = moveDown;
    }

    public Laser getLaser() {
        return this.laser;
    }
}
```

```
package com.hafezi.games.spaceshooter2d.Utility;

import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.graphics.Color;
import android.support.annotation.NonNull;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import com.hafezi.games.spaceshooter2d.R;
```

```
import java.util.ArrayList;

/**
 * Created by Mojtaba Hafezi on 25.02.2018.
 */

//This class is required so that BluetoothDevices can be used in the
ListView
public class DeviceAdapter extends ArrayAdapter<BluetoothDevice> {

    private ArrayList<BluetoothDevice> devices;

    public DeviceAdapter(@NonNull Context context,
ArrayList<BluetoothDevice> devices) {

        super(context, 0, devices);
        this.devices = devices;
    }

    public View getView(int position, View convertView, ViewGroup parent)
    {
        BluetoothDevice device = devices.get(position);
        if (convertView == null) {
            convertView =
LayoutInflater.from(getContext()).inflate(R.layout.device, parent, false);
        }

        if (device != null) {
            TextView deviceName = (TextView)
convertView.findViewById(R.id.tvName);

            if (deviceName != null) {
                deviceName.setText(device.getName());
                deviceName.setTextColor(Color.CYAN);
            }
        }
        return convertView;
    }

}
```

```
package com.hafezi.games.spaceshooter2d.Utility;

import android.app.Activity;
import android.content.Context;
import android.graphics.Rect;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import android.os.Debug;
import android.text.InputType;
import android.util.Log;
import android.view.InputDevice;
import android.view.KeyEvent;
import android.view.MotionEvent;

import com.hafezi.games.spaceshooter2d.GameObjects.Player;
import com.hafezi.games.spaceshooter2d.GameView;
import com.hafezi.games.spaceshooter2d.SoundManager;

import java.util.ArrayList;
```

```java
/**
 * Created by Mojtaba Hafezi on 21.02.2018.
 */

public class InputController {
    //The screen is split up into following rectangle objects
    private Rect up;
    private Rect down;
    private Rect shoot;
    private GameView gameView;

    //required for accelerometer
    private float[] gravity = new float[]{0, 0, 0};
    private float[] linearAcceleration = new float[]{0, 0, 0};
    final float alpha = 0.915f;

    public InputController(GameView gameView, int screenWidth, int
screenHeight) {
        //divide the android screen into up and down area
        up = new Rect(0, 0, screenWidth / 2, screenHeight / 2);
        down = new Rect(0, (screenHeight / 2 - 1), screenWidth / 2,
screenHeight);
        //right half of the screen activates shooting
        shoot = new Rect(screenWidth / 2, 0, screenWidth, screenHeight);
        this.gameView = gameView;
    }

    //ACCELEROMETER
    public void handleSensorInput(SensorEvent sensorEvent, Player player) {
        if (sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            // Isolate the force of gravity with the low-pass filter.
            gravity[0] = alpha * gravity[0] + (1 - alpha) *
sensorEvent.values[0];
            gravity[1] = alpha * gravity[1] + (1 - alpha) *
sensorEvent.values[1];
            gravity[2] = alpha * gravity[2] + (1 - alpha) *
sensorEvent.values[2];

            // Remove the gravity contribution with the high-pass filter.
            linearAcceleration[0] = sensorEvent.values[0] - gravity[0];
            linearAcceleration[1] = sensorEvent.values[1] - gravity[1];
            linearAcceleration[2] = sensorEvent.values[2] - gravity[2];

            //Only the x value is in use
            float x = linearAcceleration[0];
            float y = linearAcceleration[1];
            float z = linearAcceleration[2];

            if (x >= 1) {
                player.setMoveUp(false);
                player.setMoveDown(true);
            }

            if (x <= -1) {
                player.setMoveDown(false);
                player.setMoveUp(true);
            }

            if (x > -1 && x < 1) {
                player.setMoveDown(false);
                player.setMoveUp(false);
            }
```

```
        }
    }

    //TOUCH INPUT
    public void handleTouchInput(MotionEvent motionEvent, Player player) {
        int horizontal = (int) motionEvent.getX();
        int vertical = (int) motionEvent.getY();

        switch (motionEvent.getAction() & motionEvent.ACTION_MASK) {
            //finger touches screen
            case MotionEvent.ACTION_DOWN:
                if (!gameView.isGameOver()) {
                    //if the right half of the screen is tapped
                    if (shoot.contains(horizontal, vertical)) {
                        if (player.getLaser().isAvailable())

gameView.getSoundManager().playSound(SoundManager.Sounds.LASER);
                            player.fireLaser();
                    }
                    //check if the user presses on the upper half or lower
half of the screen
                    if (up.contains(horizontal, vertical)) {
                        player.setMoveUp(true);
                        player.setMoveDown(false);
                    } else if (down.contains(horizontal, vertical)) {
                        player.setMoveDown(true);
                        player.setMoveUp(false);
                    }
                }
                // if the game is already over the high score activity
needs to be called
                else {
                    gameView.startNewActivity();
                }
                break;
            //finger is removed
            case MotionEvent.ACTION_UP:
                if (up.contains(horizontal, vertical)) {
                    player.setMoveUp(false);
                } else if (down.contains(horizontal, vertical)) {
                    player.setMoveDown(false);
                }
                break;
        }
    }

    // Handle gamepad and D-pad button presses to
    // navigate the ship and fire
    public void handleControllerKeysInput(KeyEvent event, Player player) {
        int keyCode = event.getKeyCode();
        int eventAction = event.getAction();
        boolean isGamePad = ((event.getSource() &
InputDevice.SOURCE_GAMEPAD) == InputDevice.SOURCE_GAMEPAD);
        boolean isJoystick = ((event.getSource() &
InputDevice.SOURCE_JOYSTICK) == InputDevice.SOURCE_JOYSTICK);
        boolean isDpad = ((event.getSource() & InputDevice.SOURCE_DPAD) ==
InputDevice.SOURCE_DPAD);

        if (isGamePad || isDpad || isJoystick) {
            switch (keyCode) {
                //pause game
                case KeyEvent.KEYCODE_BUTTON_START:
                    if (eventAction == KeyEvent.ACTION_DOWN) {
```

```
                    if (!gameView.isGameOver()) {
                        if (gameView.isPlaying())
                            gameView.pause();
                        else
                            gameView.resume();
                    } else
                        gameView.startNewActivity();
                }

                break;
            case KeyEvent.KEYCODE_DPAD_UP:
                if (eventAction == KeyEvent.ACTION_DOWN) {
                    player.setMoveUp(true);
                    player.setMoveDown(false);
                } else if (eventAction == KeyEvent.ACTION_UP) {
                    player.setMoveUp(false);
                    player.setMoveDown(false);
                }
                break;
            case KeyEvent.KEYCODE_DPAD_DOWN:
                if (eventAction == KeyEvent.ACTION_DOWN) {
                    player.setMoveUp(false);
                    player.setMoveDown(true);
                } else if (eventAction == KeyEvent.ACTION_UP) {
                    player.setMoveUp(false);
                    player.setMoveDown(false);
                }
                break;
            case KeyEvent.KEYCODE_DPAD_CENTER:
            case KeyEvent.KEYCODE_BUTTON_A:
            case KeyEvent.KEYCODE_BUTTON_X:
                if (!gameView.isGameOver()) {
                    if (player.getLaser().isAvailable())

gameView.getSoundManager().playSound(SoundManager.Sounds.LASER);
                    player.fireLaser();
                } else
                    gameView.startNewActivity();
                break;

            }
        }
    }

    //Handle joysticks
    public void handleControllerMotionInput(MotionEvent event, Player
player) {
        boolean isGamePad = ((event.getSource() &
InputDevice.SOURCE_GAMEPAD) == InputDevice.SOURCE_GAMEPAD);
        boolean isJoystick = ((event.getSource() &
InputDevice.SOURCE_JOYSTICK) == InputDevice.SOURCE_JOYSTICK);
        boolean isDpad = ((event.getSource() & InputDevice.SOURCE_DPAD) ==
InputDevice.SOURCE_DPAD);

        if (event.getAction() == MotionEvent.ACTION_MOVE && (isGamePad ||
isDpad || isJoystick)) {
            float vertical = event.getAxisValue(MotionEvent.AXIS_Y);
            if (vertical > 0.1) {
                player.setMoveDown(true);
                player.setMoveUp(false);

            } else if (vertical < -0.1) {
                player.setMoveUp(true);
```

```
                player.setMoveDown(false);
            } else {
                player.setMoveDown(false);
                player.setMoveUp(false);
            }
        }
    }
}
```

```
package com.hafezi.games.spaceshooter2d.Utility;

/**
 * Created by Mojtaba Hafezi on 23.02.2018.
 */

//When using the shared preferences the different strings could become
error prone
public enum Pref {
    AUDIO("AUDIO"),
    GAME("GAME"),
    TIME("TIME"),
    SENSOR("SENSOR"),
    SCORE("SCORE");

    // required to give a string back for the enums
    private final String text;

    Pref(final String text) {
        this.text = text;
    }

    @Override
    public String toString() {
        return text;
    }
}
```

```
package com.hafezi.games.spaceshooter2d;

import android.Manifest;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Build;
import android.os.Bundle;
import android.app.Activity;
import android.support.annotation.RequiresApi;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.Toast;
```

```
import com.hafezi.games.spaceshooter2d.Utility.DeviceAdapter;

import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Set;

public class BluetoothActivity extends AppCompatActivity implements
AdapterView.OnItemClickListener {

    Button exitButton;
    Button activateButton;
    Button discoverButton;

    private SoundManager soundManager;
    //bluetooth utilities
    private BluetoothAdapter bluetoothAdapter;
    private Set<BluetoothDevice> pairedDevices;
    public ArrayList<BluetoothDevice> bluetoothDevices = new ArrayList<>();
    private ListView newDevices;
    //required to convert array list of BT devices into ListView
    public DeviceAdapter deviceAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_bluetooth);
        soundManager = SoundManager.getInstance(this);
        soundManager.playMusic();
        exitButton = (Button) findViewById(R.id.backButton);
        activateButton = (Button) findViewById(R.id.activateButton);
        discoverButton = (Button) findViewById(R.id.discoverButton);
        setButtonListeners();

        //BLUETOOTH settings
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        bluetoothDevices = new ArrayList<>();
        newDevices = (ListView) findViewById(R.id.listView);
        newDevices.setOnItemClickListener(BluetoothActivity.this);

        //Broadcast when pairing status changes
        IntentFilter intentFilter = new
IntentFilter(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
        registerReceiver(broadCastPairing, intentFilter);

        //Broadcast when discovering new devices
        IntentFilter discoverDevicesIntent = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
        registerReceiver(broadCastDiscovery, discoverDevicesIntent);

    }

    private void activateBluetooth() {
        //if device supports bluetooth -> activate if not already on
        if (!(bluetoothAdapter == null)) {
            if (!bluetoothAdapter.isEnabled()) {
                // start intent and register the broadcast for activation
                Intent turnOn = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(turnOn, 0);
                IntentFilter intentFilter = new
IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
                registerReceiver(broadCastActivation, intentFilter);
```

```
            } else {
                bluetoothAdapter.disable();

                IntentFilter intentFilter = new
IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
                registerReceiver(broadCastActivation, intentFilter);
            }
        }
    }


    private void discoverDevices() {
        if (!bluetoothAdapter.isEnabled())
            activateBluetooth();

        if (bluetoothAdapter.isDiscovering()) {
            //if it is already discovering - cancel and restart
            bluetoothAdapter.cancelDiscovery();
            bluetoothAdapter.startDiscovery();
            //Broadcast when discovering new devices
            IntentFilter discoverDevicesIntent = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
            registerReceiver(broadCastDiscovery, discoverDevicesIntent);
            showShortToast(getBaseContext(), "Searching for devices...");

        } else {
            bluetoothAdapter.startDiscovery();
            //Broadcast when discovering new devices
            IntentFilter discoverDevicesIntent = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
            registerReceiver(broadCastDiscovery, discoverDevicesIntent);
            showShortToast(getBaseContext(), "Searching for devices...");

        }
    }

    private void setButtonListeners() {

        exitButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);
                unregisterReceiver();
                //soundManager.releasePlayer();
                finish();
            }
        });

        exitButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

exitButton.setBackgroundResource(R.drawable.red_button);
                else

exitButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

        activateButton.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);
                activateBluetooth();
            }
        });

        activateButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

activateButton.setBackgroundResource(R.drawable.red_button);
                else

activateButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

        discoverButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);
                discoverDevices();
            }
        });

        discoverButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

discoverButton.setBackgroundResource(R.drawable.red_button);
                else

discoverButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

    }

    // If the player hits the back button, quit the app
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            unregisterReceiver();
            finish();
            return true;
        }
        return false;
    }


    @Override
    protected void onStart() {
        super.onStart();
    }

    @Override
    protected void onPause() {
        super.onPause();
        soundManager.stopMusic();
    }
```

```
    @Override
    protected void onResume() {
        super.onResume();
        soundManager.playMusic();
    }

    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i,
long l) {
        //Cancel discovery to save energy
        bluetoothAdapter.cancelDiscovery();

        showShortToast(getBaseContext(), "Bluetooth enabled.");
        String deviceName = bluetoothDevices.get(i).getName();
        //After creating the bond the connection needs to be created -
TODO: Get it working
        bluetoothDevices.get(i).createBond();
    }

    //Shows a short toast with given text
    private void showShortToast(Context context, String text) {
        Toast toast = Toast.makeText(context, text, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
        toast.show();
    }

    //updates the list by adding all items from the bluetoothDevices
container
    private void updateList(Context context) {
        //Convert from ArrayList to ListView
        deviceAdapter = new DeviceAdapter(context, bluetoothDevices);
        newDevices.setAdapter(deviceAdapter);
    }

    private void unregisterReceiver() {
        try {
            unregisterReceiver(broadCastPairing);
            unregisterReceiver(broadCastActivation);
            unregisterReceiver(broadCastDiscovery);
        } catch (Exception e) {
            Log.e("BT", "Trying to unregister not registered receiver");
        }
        finish();
    }


    //BROADCASTS
    //Broadcast receiver for discovering
    private final BroadcastReceiver broadCastDiscovery = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (action.equals(BluetoothDevice.ACTION_FOUND)) {
                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                if (!bluetoothDevices.contains(device))
                    bluetoothDevices.add(device);
                updateList(context);
            }
        }
    };
```

```
    //Broadcast receiver for pairing
    private final BroadcastReceiver broadCastPairing = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)) {
                BluetoothDevice bluetoothDevice =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                //case: created a bond
                if (bluetoothDevice.getBondState() ==
BluetoothDevice.BOND_BONDED) {
                    showShortToast(context, "Bluetooth pairing finished.");
                }
                //case: creating a bond
                if (bluetoothDevice.getBondState() ==
BluetoothDevice.BOND_BONDING) {
                    showShortToast(context, "Pairing...");
                }
            }
        }
    };

    //Broadcast receiver for enabling/disabling BT
    private final BroadcastReceiver broadCastActivation = new
BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            // When discovery finds a device
            if (action.equals(bluetoothAdapter.ACTION_STATE_CHANGED)) {
                final int state =
intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, BluetoothAdapter.ERROR);
                switch (state) {
                    case BluetoothAdapter.STATE_OFF:
                        //disabled bluetooth -> no devices
                        showShortToast(context, "Bluetooth disabled.");
                        //get paired device
                        bluetoothDevices.clear();
                        updateList(getBaseContext());
                        break;
                    case BluetoothAdapter.STATE_ON:
                        //enabled bluetooth -> show devices
                        showShortToast(context, "Bluetooth enabled.");
                        //get paired device
                        bluetoothDevices.clear();
                        pairedDevices =
bluetoothAdapter.getBondedDevices();
                        for (BluetoothDevice bluetoothDevice :
pairedDevices) {
                            bluetoothDevices.add(bluetoothDevice);
                        }
                        updateList(getBaseContext());
                        break;
                }
            }
        }
    };


}
```

```java
package com.hafezi.games.spaceshooter2d;

import android.app.Activity;
import android.content.Context;
import android.graphics.Point;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.Log;
import android.view.Display;
import android.view.KeyEvent;
import android.view.MotionEvent;

/**
 * Created by Mojtaba Hafezi on 18.02.2018.
 */

public class GameActivity extends Activity {

    private GameView gameView;
    private SoundManager soundManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Detect the screen resolution and pass it on as point
        Display display = getWindowManager().getDefaultDisplay();
        Point point = new Point();
        display.getSize(point);
        gameView = new GameView(GameActivity.this, point.x, point.y);
        setContentView(gameView);
        soundManager = SoundManager.getInstance(this);
        soundManager.playMusic();
    }

    @Override
    protected void onStart() {
        super.onStart();
    }

    @Override
    protected void onPause() {
        super.onPause();
        gameView.pause();
        soundManager.stopMusic();
    }

    @Override
    protected void onResume() {
        super.onResume();
        gameView.resume();
        soundManager.playMusic();
    }

    //The events for the game controllers need to be registered in the
activity
    //These are passed down to the gameView
    @Override
    public boolean dispatchGenericMotionEvent(MotionEvent event) {
        gameView.handleControllerMotion(event);
        return super.dispatchGenericMotionEvent(event);
    }
```

```
    @Override
    public boolean dispatchKeyEvent(KeyEvent event) {
        gameView.handleControllerKeys(event);
        return super.dispatchKeyEvent(event);
    }
}
```

```
package com.hafezi.games.spaceshooter2d;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.SystemClock;
import android.os.Vibrator;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

import com.hafezi.games.spaceshooter2d.GameObjects.Dust;
import com.hafezi.games.spaceshooter2d.GameObjects.Enemy;
import com.hafezi.games.spaceshooter2d.GameObjects.Explosion;
import com.hafezi.games.spaceshooter2d.GameObjects.Laser;
import com.hafezi.games.spaceshooter2d.GameObjects.Player;
import com.hafezi.games.spaceshooter2d.Utility.InputController;
import com.hafezi.games.spaceshooter2d.Utility.Pref;

import java.util.ArrayList;

import static android.content.Context.VIBRATOR_SERVICE;

/**
 * Created by Mojtaba Hafezi on 18.02.2018.
 */
//View for the main game since everything needs to be drawn on screen
//Extends SurfaceView for drawing on its own thread
public class GameView extends SurfaceView implements Runnable,
SensorEventListener {

    //Thread related attributes
    volatile boolean playing;
    Thread gameThread = null;


    //Game objects
    private Player player;
    private Explosion[] explosions;
    private Explosion quickExplosion;
    private boolean isExplosionTriggered;
    private Enemy[] enemies;
    private ArrayList<Dust> whiteDusts;
    private ArrayList<Dust> yellowDusts;
    private ArrayList<Dust> redDusts;
```

```
    //number of dusts visible on the screen
    private final int WHITEDUST = 75;
    private final int YELLOWDUST = 45;
    private final int REDDUST = 30;
    private long enemiesDestroyed;
    private long score;

    //Attributes req. for drawing
    private Canvas canvas;
    private Paint paint;
    private SurfaceHolder surfaceHolder;
    private Context context;
    private int screenX;
    private int screenY;

    //Game loop relevant attributes
    private boolean gameOver;
    long startFrameTime;
    long timeThisFrame;
    long lastHit;
    long timeForExplosion;
    //measures time since game loop is running + tracks record
    private long longestTime = 0;
    private long timeTaken;
    private long timeStarted;

    //utility
    private SoundManager soundManager;
    private InputController inputController;
    private Vibrator vibrator;
    private long[] vibratorPattern = {300, 100, 300, 100, 600, 100, 1000,
100, 100};
    private boolean useSensor;
    private SensorManager sensorManager;
    private Sensor sensor;

    //persistence
    private SharedPreferences sharedPreferences;
    private SharedPreferences.Editor editor;

    public GameView(Context context) {
        super(context);
    }
    //constructor for the game view
    public GameView(Context context, int screenX, int screenY) {
        super(context);
        setContext(context);
        setScreenX(screenX);
        setScreenY(screenY);
        paint = new Paint();
        surfaceHolder = getHolder();
        sharedPreferences =
getContext().getSharedPreferences(Pref.GAME.toString(),
context.MODE_PRIVATE);
        editor = sharedPreferences.edit();
        soundManager = SoundManager.getInstance(context);
        vibrator = (Vibrator)
getContext().getSystemService(VIBRATOR_SERVICE);
        sensorManager = (SensorManager)
getContext().getSystemService(Context.SENSOR_SERVICE);
        sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        inputController = new InputController(this, screenX, screenY);
        initialiseGame();
```

```
        resume();
    }

    //initialises the game to a playable state
    public void initialiseGame() {
        setGameOver(false);
        setPlaying(true);
        lastHit = 0;
        timeTaken = 0;
        timeStarted = SystemClock.elapsedRealtime();
        //Load score and options for sensor
        longestTime = sharedPreferences.getLong(Pref.TIME.toString(), 0);
        score = sharedPreferences.getLong(Pref.SCORE.toString(), 0);
        enemiesDestroyed = 0;
        boolean usingSensor =
sharedPreferences.getBoolean(Pref.SENSOR.toString(), false);
        setUseSensor(usingSensor);
        //Initialisation of all game objects
        player = new Player(getContext(), 10, 0, 10, getScreenX(),
getScreenY());
        explosions = new Explosion[5];
        for (int i = 0; i < explosions.length; i++) {
            explosions[i] = new Explosion(getContext(), screenX, screenY,
"explosion" + (1 + i), 0, 0);
        }
        quickExplosion = new Explosion(getContext(), screenX, screenY,
"quickexplosion", -1000, -1000);
        isExplosionTriggered = false;
        timeForExplosion = 0;
        enemies = new Enemy[6];
        for (int i = 0; i < enemies.length; i++) {
            enemies[i] = new Enemy(getContext(), getScreenX(),
getScreenY());
        }
        whiteDusts = new ArrayList<>();
        yellowDusts = new ArrayList<>();
        redDusts = new ArrayList<>();
        for (int i = 0; i < WHITEDUST; i++) {
            Dust whiteDust = new Dust(getScreenX(), getScreenY());
            whiteDusts.add(whiteDust);
        }
        for (int i = 0; i < YELLOWDUST; i++) {
            Dust yellowDust = new Dust(getScreenX(), getScreenY());
            yellowDusts.add(yellowDust);
        }
        for (int i = 0; i < REDDUST; i++) {
            Dust redDust = new Dust(getScreenX(), getScreenY());
            redDusts.add(redDust);
        }

    }

    //game loop
    @Override
    public void run() {
        while (isPlaying()) {
            //get the time the execution of this code started
            startFrameTime = SystemClock.elapsedRealtime();
            //updates all the game objects
            update();
            //draws all the objects and graphical user interface
            draw();
            //get the time difference and control the frames per seconds
```

```
            //control the frames per seconds -> if drawing took too long
skip sleep call for thread
            timeThisFrame = SystemClock.elapsedRealtime() - startFrameTime;
            control();
        }
    }

    private void update() {
        //while the game is not over
        if (!isGameOver()) {
            //if game was paused -> time handled correctly through this
method
            if (timeStarted != 0)
                timeTaken += (SystemClock.elapsedRealtime() - timeStarted);
            timeStarted = SystemClock.elapsedRealtime();

            //update game objects
            player.update();
            for (Enemy enemy : enemies) {
                enemy.update();
            }
            for (Dust whiteDust : whiteDusts) {
                whiteDust.update();
            }
            for (Dust yellowDust : yellowDusts) {
                yellowDust.update();
            }
            for (Dust redDust : redDusts) {
                redDust.update();
            }

            //check for collisions between player and enemies
            boolean collisionDetected;
            for (Enemy enemy : enemies) {
                collisionDetected = collisionDetection(player, enemy);
                if (collisionDetected) {
                    enemiesDestroyed++;
                    isExplosionTriggered = true;
                    if (player.getShields() >= 1) {
                        soundManager.playSound(SoundManager.Sounds.HIT);
                        vibrator.vibrate(200);
                        //player is immune for 2 sec after a collision but
only once
                        if (lastHit == 0) {
                            lastHit = SystemClock.elapsedRealtime();
                            player.setShields(player.getShields() - 1);
                        }
                        if (startFrameTime - lastHit > 2000)
                            player.setShields(player.getShields() - 1);

                    }
                } else {
                    isExplosionTriggered = false;
                }
                //if laser hits enemy
                if (!player.getLaser().isAvailable()) {
                    collisionDetected = collisionWithLaser(player, enemy);
                    if (collisionDetected) {
                        player.getLaser().setAvailable(true);
                        enemy.setShield(enemy.getShield() - 1);
                        if (enemy.getShield() <= 0) {
                            enemiesDestroyed++;
                            quickExplosion.setPosition(enemy.getX() - 5,
```

```
enemy.getY() + enemy.getHeight() / 2);
                            enemy.setRandomAttributes();
                        }
                        isExplosionTriggered = true;
                        soundManager.playSound(SoundManager.Sounds.HIT);
                    } else {
                        isExplosionTriggered = false;
                    }
                }
            }

            //check for game status
            if (player.getShields() <= 0) {
                //play destroyed sound
                soundManager.playSound(SoundManager.Sounds.EXPLOSION);
                vibrator.vibrate(vibratorPattern, -1);
                setGameOver(true); //gameover
                if (timeTaken > longestTime) {
                    longestTime = timeTaken; //new hi-score
                    editor.putLong(Pref.TIME.toString(), longestTime);

                }
                if (enemiesDestroyed > score) {
                    editor.putLong(Pref.SCORE.toString(),
enemiesDestroyed);
                }
                editor.commit();
            }
        }
        //If game is over -> set the timing for the explosion animation
        else {
            if (timeForExplosion == 0)
                timeForExplosion = SystemClock.elapsedRealtime();
            //if player taps on screen again -> event triggers call to new
activity
        }

    }

    private void draw() {
        //only draw if valid
        if (surfaceHolder.getSurface().isValid()) {
            //if the game is not over
            if (!isGameOver()) {
                //Lock & repaint canvas
                canvas = surfaceHolder.lockCanvas();
                canvas.drawColor(Color.BLACK);

                //Draw game objects with corresponding paint color
                //Space dust is drawn as points
                paint.setColor(Color.YELLOW);
                for (Dust yellowDust : yellowDusts) {
                    canvas.drawPoint(yellowDust.getX(), yellowDust.getY(),
paint);
                }
                paint.setColor(Color.RED);
                for (Dust redDust : redDusts) {
                    canvas.drawPoint(redDust.getX(), redDust.getY(),
paint);
                }
                paint.setColor(Color.WHITE);
                for (Dust whiteDust : whiteDusts) {
                    canvas.drawPoint(whiteDust.getX(), whiteDust.getY(),
```

```
paint);
                }
                // draw player ship
                canvas.drawBitmap(player.getBitmap(), player.getX(),
player.getY(), paint);
                if (!player.getLaser().isAvailable()) {
                    Laser laser = player.getLaser();
                    canvas.drawBitmap(laser.getBitmap(), laser.getX(),
laser.getY(), paint);
                }
                //draw enemy objects
                for (Enemy enemy : enemies) {
                    canvas.drawBitmap(enemy.getBitmap(), enemy.getX(),
enemy.getY(), paint);
                }
                // draw explosion
                if (isExplosionTriggered)
                    canvas.drawBitmap(quickExplosion.getBitmap(),
quickExplosion.getX(), quickExplosion.getY(), paint);

                //USER INTERFACE - HUD
                paint.setTextAlign(Paint.Align.LEFT);
                paint.setColor(Color.CYAN);
                paint.setTextSize(30);
                canvas.drawText("Longest: " + (int) longestTime / 1000 + "
s", 10, 20, paint);
                canvas.drawText("Time: " + (int) timeTaken / 1000 + " s",
getScreenX() / 2, 20, paint);
                canvas.drawText("Shields: " + player.getShields(), 10,
getScreenY() - 20, paint);
                canvas.drawText("Destroyed: " + enemiesDestroyed,
getScreenX() / 2, getScreenY() - 20, paint);


                //unlock and post at the end
                surfaceHolder.unlockCanvasAndPost(canvas);
            }
            //Draw game over text, score and show ship explosion
            else {
                //Lock & repaint canvas
                canvas = surfaceHolder.lockCanvas();
                canvas.drawColor(Color.BLACK);

                //Draw game objects with corresponding paint color
                paint.setColor(Color.YELLOW);
                for (Dust yellowDust : yellowDusts) {
                    canvas.drawPoint(yellowDust.getX(), yellowDust.getY(),
paint);
                }
                paint.setColor(Color.RED);
                for (Dust redDust : redDusts) {
                    canvas.drawPoint(redDust.getX(), redDust.getY(),
paint);
                }
                paint.setColor(Color.WHITE);
                for (Dust whiteDust : whiteDusts) {
                    canvas.drawPoint(whiteDust.getX(), whiteDust.getY(),
paint);
                }

                //Explosion Animation
                //Draw explosion where player was before
                long animExplosion = startFrameTime - timeForExplosion;
```

```
                int result = -1;
                if (animExplosion <= 300)
                    result = 0;
                else if (animExplosion <= 600)
                    result = 1;
                else if (animExplosion <= 900)
                    result = 2;
                else if (animExplosion <= 1200)
                    result = 3;
                else if (animExplosion <= 1500)
                    result = 4;
                if (result > 0 && result <= 4)
                    canvas.drawBitmap(explosions[result].getBitmap(),
player.getX(), player.getY(), paint);

                //enemy objects
                for (Enemy enemy : enemies) {
                    canvas.drawBitmap(enemy.getBitmap(), enemy.getX(),
enemy.getY(), paint);
                }

                //GAMEOVER SCREEN
                paint.setTextSize(80);
                paint.setTextAlign(Paint.Align.CENTER);
                paint.setColor(Color.CYAN);
                canvas.drawText("GAME OVER", getScreenX() / 2, 100, paint);
                paint.setTextSize(25);
                canvas.drawText("Longest: " + (int) longestTime / 1000 + "
s", getScreenX() / 2, 160, paint);
                canvas.drawText("Time: " + (int) timeTaken / 1000 + " s",
getScreenX() / 2, 200, paint);
                canvas.drawText("Ships destroyed: " + enemiesDestroyed,
getScreenX() / 2, 240, paint);
                paint.setTextSize(80);
                canvas.drawText("Tap to continue!", getScreenX() / 2,
getScreenY() / 2, paint);


                //unlock and post at the end
                surfaceHolder.unlockCanvasAndPost(canvas);
            }
        }
    }

    //for constant frames per seconds
    private void control() {
        try {
            //took too long for the operations
            if (timeThisFrame >= 17) {
                return;
            } else
                //optionally 60 frames are shown per second
                //control frame rate (1000/60 = ca. 17) - subtract the time
taken for update/draw
                gameThread.sleep(17 - timeThisFrame);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    //checks for intersection between the hitboxes - used in the update
method
    private boolean collisionDetection(Player player, Enemy enemy) {
```

```
        if (Rect.intersects(player.getHitbox(), enemy.getHitbox())) {
            quickExplosion.setPosition(player.getX() + 10, player.getY() +
player.getHeight() / 2);
            enemy.setRandomAttributes();
            return true;
        }
        return false;
    }

    //checks if the laser has collided with an enemy
    private boolean collisionWithLaser(Player player, Enemy enemy) {
        if (Rect.intersects(player.getLaser().getHitbox(),
enemy.getHitbox())) {
            return true;
        }
        return false;
    }

    //pauses the game and thread
    public void pause() {
        setPlaying(false);
        sensorManager.unregisterListener(this);
        try {
            gameThread.join();
        } catch (InterruptedException e) {
            //Todo: error handling
            e.printStackTrace();
        }
    }

    //on start or on resume this method makes sure the game continues
correctly
    //creates a new thread and starts it
    public void resume() {
        if (sensorManager != null)
            sensorManager.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_NORMAL);
        timeStarted = SystemClock.elapsedRealtime();
        setPlaying(true);
        gameThread = new Thread(this);
        gameThread.start();
    }

    //InputController manages touch events
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        if (player != null) {
            inputController.handleTouchInput(event, player);
        }
        return true;
    }

    //transition to high-score activity and passes the parameters for time
and score
    public void startNewActivity() {
        Activity activity = (Activity) getContext();
        Intent i = new Intent(getContext(), HighScoreActivity.class);
        i.putExtra(Pref.TIME.toString(), (int) (timeTaken / 1000));
        i.putExtra(Pref.SCORE.toString(), (int) enemiesDestroyed);
        activity.finish();
        activity.startActivity(i);

    }
```

```
    //handle the accelerometer
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        if (isUseSensor())
            inputController.handleSensorInput(sensorEvent, player);
    }

    //Game controller joystick handling
    public void handleControllerMotion(MotionEvent event) {
        inputController.handleControllerMotionInput(event, player);
    }
    //Game controller key handling
    public void handleControllerKeys(KeyEvent event) {
        inputController.handleControllerKeysInput(event, player);
    }


    //Empty - required for Accelerometer
    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {

    }

    //GETTER AND SETTERS
    public boolean isPlaying() {
        return playing;
    }

    public void setPlaying(boolean playing) {
        this.playing = playing;
    }

    public int getScreenX() {
        return screenX;
    }

    public void setScreenX(int screenX) {
        this.screenX = screenX;
    }

    public int getScreenY() {
        return screenY;
    }

    public void setScreenY(int screenY) {
        this.screenY = screenY;
    }

    public void setContext(Context context) {
        this.context = context;
    }

    public boolean isGameOver() {
        return gameOver;
    }

    public void setGameOver(boolean gameOver) {
        this.gameOver = gameOver;
    }


    public boolean isUseSensor() {
```

```
        return useSensor;
    }

    public void setUseSensor(boolean useSensor) {
        this.useSensor = useSensor;
    }

    public SoundManager getSoundManager() {
        return soundManager;
    }
}
```

```
package com.hafezi.games.spaceshooter2d;

import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.graphics.Color;
import android.graphics.Typeface;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.hafezi.games.spaceshooter2d.Database.Constants;
import com.hafezi.games.spaceshooter2d.Database.GameDataBase;
import com.hafezi.games.spaceshooter2d.Database.MyDBhelper;
import com.hafezi.games.spaceshooter2d.Utility.Pref;

import org.w3c.dom.Text;

import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.List;

public class HighScoreActivity extends AppCompatActivity {

    private Button exitButton;
    private Button playButton;
    //High score relevant items
    //the text views are created on the run and passed to the layouts
    private LinearLayout scoreColumn;
    private LinearLayout shipsColumn;
    private ArrayList<TextView> scoreList;
    private ArrayList<TextView> shipList;
    //utility
    private SoundManager soundManager;

    //Database
    private GameDataBase gameDataBase;
    private long currentId;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_high_score);
```

```
        gameDataBase = new GameDataBase(this);
        scoreList = new ArrayList<>();
        shipList = new ArrayList<>();

        //If the player comes from the game activity: time and score will
be received as params
        Bundle b = getIntent().getExtras();
        if (b != null) {
            int timeTaken = b.getInt(Pref.TIME.toString());
            int enemiesDestroyed = b.getInt(Pref.SCORE.toString());
            if (timeTaken >= 0 && enemiesDestroyed >= 0) {
                currentId = saveToDB(timeTaken, enemiesDestroyed);
            }
        }

        soundManager = SoundManager.getInstance(this);
        soundManager.playMusic();
        scoreColumn = (LinearLayout) findViewById(R.id.scoreColumn);
        shipsColumn = (LinearLayout) findViewById(R.id.shipsColumn);
        exitButton = (Button) findViewById(R.id.hsBackButton);
        playButton = (Button) findViewById(R.id.hsPlayButton);
        //this method loads the data from the db and sets the linear
layouts of the view
        setColumns();
        setButtonListener();
    }

    private void setButtonListener() {
        exitButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                exitButton.setBackgroundResource(R.drawable.yellow_button);
                soundManager.playSound(SoundManager.Sounds.MENU);
               // Intent i = new Intent(HighScoreActivity.this,
MainActivity.class);
                finish();
            }
        });

        exitButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

exitButton.setBackgroundResource(R.drawable.red_button);
                else

exitButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

        playButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                playButton.setBackgroundResource(R.drawable.yellow_button);
                soundManager.playSound(SoundManager.Sounds.MENU);
                Intent i = new Intent(HighScoreActivity.this,
GameActivity.class);
                finish();
                startActivity(i);
            }
        });
```

```
        playButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

playButton.setBackgroundResource(R.drawable.red_button);
                else

playButton.setBackgroundResource(R.drawable.blue_button);
            }
        });
    }

    //returns the id - if it worked, else -1
    private long saveToDB(int score, int ships) {
        long id = -1;
        gameDataBase.openWritable();
        id = gameDataBase.insertScore(score, ships);
        gameDataBase.close();
        return id;
    }

    //Loads the data and also sets the text views with specific attributes
    private void setColumns() {
        loadDataFromDB();
        Typeface type = Typeface.createFromAsset(getAssets(),"space.ttf");
        for (TextView tv : scoreList) {
            tv.setTextColor(Color.BLACK);
            tv.setTextSize(25);
            tv.setTypeface(type);
            scoreColumn.addView(tv);
        }

        for (TextView tv : shipList) {
            tv.setTextColor(Color.BLACK);
            tv.setTextSize(25);
            tv.setTypeface(type);
            shipsColumn.addView(tv);
        }
    }

    //Requires readable access to db and loads the data as text views into
the array lists
    public void loadDataFromDB() {
        gameDataBase.openReadable();
        Cursor c = gameDataBase.getScores();
        //counter keeps track of which position the entry is since the
getScore() method is sorted
        int counter = 0;
        if (c.moveToFirst()) {
            do {
                counter++;
                Long id = c.getLong(c.getColumnIndex(Constants.KEY_ID));
                if(id == currentId)
                    showLongToast(this, "Your rank: " + counter);
                String score = counter + ". Time: " +
c.getInt(c.getColumnIndex(Constants.SCORE));
                String ships = "Destroyed: " +
c.getInt(c.getColumnIndex(Constants.SHIPS));
                TextView tv = new TextView(this);
                tv.setText(  score);
```

```
                TextView tv2 = new TextView(this);
                tv2.setText(ships);
                scoreList.add(tv);
                shipList.add(tv2);
            } while (c.moveToNext());
        }
        gameDataBase.close();
    }

    @Override
    protected void onResume() {
        super.onResume();
        soundManager.playMusic();
    }

    @Override
    protected void onPause() {
        super.onPause();
        soundManager.stopMusic();
    }

    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            finish();
            return true;
        }
        return false;
    }

    //Shows a short toast with given text
    private void showLongToast(Context context, String text) {
        Toast toast = Toast.makeText(context, text, Toast.LENGTH_LONG);
        toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
        toast.show();
    }
}
```

```
package com.hafezi.games.spaceshooter2d;

import android.Manifest;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothClass;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothHeadset;
import android.bluetooth.BluetoothManager;
import android.bluetooth.BluetoothProfile;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Build;
import android.os.Bundle;
import android.os.Debug;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.InputDevice;
import android.view.KeyEvent;
import android.view.View;
import android.widget.Button;

import java.util.ArrayList;
import java.util.Set;
```

```
public class MainActivity extends AppCompatActivity {

    Button playButton;
    Button optionButton;
    Button highscoreButton;
    Button exitButton;
    private SoundManager soundManager;
    private BluetoothAdapter bluetoothAdapter;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //get the bluetooth adapter -> on quit disable bluetooth
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        //set the instance of the soundManager
        soundManager = SoundManager.getInstance(this);
        soundManager.playMusic();

        //get the buttons
        playButton = (Button) findViewById(R.id.playButton);
        optionButton = (Button) findViewById(R.id.optionButton);
        highscoreButton = (Button) findViewById(R.id.scoreButton);
        exitButton = (Button) findViewById(R.id.exitButton);

        //Set the Listeners for the buttons
        setButtonListeners();

    }

    //play the music if the application continues
    @Override
    protected void onResume() {
        super.onResume();
        soundManager.playMusic();
        //reset the background images of the buttons
        resetButtons();
    }

    // pause the music as well
    @Override
    protected void onPause() {
        super.onPause();
        soundManager.stopMusic();
    }

    // If the player hits the back button, quit the app
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            soundManager.releasePlayer();
            finish();
            return true;
        }
        return false;
    }

    /*the listeners for change of focus and onClick are set here
accordingly
    // the soundManager is used to play the corresponding sound effects
    // the exit button additionally disables the bluetooth and released the
media player
```

```
    */
    private void setButtonListeners() {
        playButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);
                playButton.setBackgroundResource(R.drawable.yellow_button);
                Intent i = new Intent(MainActivity.this,
GameActivity.class);
                startActivity(i);
            }
        });

        playButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

playButton.setBackgroundResource(R.drawable.red_button);
                else

playButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

        exitButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);
                exitButton.setBackgroundResource(R.drawable.yellow_button);
                soundManager.releasePlayer();
                bluetoothAdapter.disable();
                finish();
            }
        });

        exitButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

exitButton.setBackgroundResource(R.drawable.red_button);
                else

exitButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

        highscoreButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);

highscoreButton.setBackgroundResource(R.drawable.yellow_button);
                Intent i = new Intent(MainActivity.this,
HighScoreActivity.class);
                startActivity(i);
            }
        });

        highscoreButton.setOnFocusChangeListener(new
```

```
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

highscoreButton.setBackgroundResource(R.drawable.red_button);
                else

highscoreButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

        optionButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);

optionButton.setBackgroundResource(R.drawable.yellow_button);
                Intent i = new Intent(MainActivity.this,
OptionsActivity.class);
                startActivity(i);
            }
        });

        optionButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

optionButton.setBackgroundResource(R.drawable.red_button);
                else

optionButton.setBackgroundResource(R.drawable.blue_button);
            }
        });
    }


    //simple method to reset all buttons to their initial background
    private void resetButtons() {
        playButton.setBackgroundResource(R.drawable.blue_button);
        highscoreButton.setBackgroundResource(R.drawable.blue_button);
        optionButton.setBackgroundResource(R.drawable.blue_button);
        exitButton.setBackgroundResource(R.drawable.blue_button);
    }


}
```

```
package com.hafezi.games.spaceshooter2d;

import android.bluetooth.BluetoothAdapter;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.PixelFormat;
import android.media.MediaPlayer;
import android.widget.MediaController;
import android.media.session.MediaSession;
import android.net.Uri;
import android.os.Build;
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.widget.Button;
import android.widget.VideoView;

import com.hafezi.games.spaceshooter2d.Utility.Pref;

public class OptionsActivity extends AppCompatActivity {

    Button audioEnableButton;
    Button audioDisableButton;
    Button accelEnableButton;
    Button accelDisableButton;
    Button tutorialButton;
    Button bluetoothButton;
    Button saveButton;

    //Utility
    private SoundManager soundManager;
    private MediaController mediaController;
    private VideoView videoHolder;

    //persistence
    private SharedPreferences sharedPreferences;
    private SharedPreferences.Editor editor;


    private boolean usingSensor;
    private boolean isMute;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_options);
        //get utility instances
        mediaController = new MediaController(this);
        videoHolder = new VideoView(OptionsActivity.this);
        soundManager = SoundManager.getInstance(this);
        soundManager.playMusic();
        //get the values for the options
        sharedPreferences = getSharedPreferences(Pref.GAME.toString(),
MODE_PRIVATE);
        editor = sharedPreferences.edit();
        //find buttons
        audioEnableButton = (Button) findViewById(R.id.audioEnableButton);
        audioDisableButton = (Button)
findViewById(R.id.audioDisableButton);
        accelEnableButton = (Button) findViewById(R.id.accelEnableButton);
        accelDisableButton = (Button)
findViewById(R.id.accelDisableButton);
        tutorialButton = (Button) findViewById(R.id.tutorialButton);
        bluetoothButton = (Button) findViewById(R.id.bluetoothButton);
        saveButton = (Button) findViewById(R.id.saveButton);
        //load the data and set the button listeners and their states
        loadData();
        setButtonListeners();
        setButtonStates();
    }

    //is used to bring the default state of the option screen back after
the video is played
    private void initialiseView() {
```

```
        soundManager.playMusic();
        audioEnableButton = (Button) findViewById(R.id.audioEnableButton);
        audioDisableButton = (Button)
findViewById(R.id.audioDisableButton);
        accelEnableButton = (Button) findViewById(R.id.accelEnableButton);
        accelDisableButton = (Button)
findViewById(R.id.accelDisableButton);
        tutorialButton = (Button) findViewById(R.id.tutorialButton);
        bluetoothButton = (Button) findViewById(R.id.bluetoothButton);
        saveButton = (Button) findViewById(R.id.saveButton);
        setButtonListeners();
        setButtonStates();
    }

    //loads the boolean values from the shared preferences
    private void loadData() {
        boolean isMute =
sharedPreferences.getBoolean(Pref.AUDIO.toString(), false);
        setUsingSensor(sharedPreferences.getBoolean(Pref.SENSOR.toString(),
false));
        setMute(isMute);
        soundManager.setMute(isMute());
    }

    // private method to save data using shared preferences
    private void saveOptions() {
        editor.putBoolean(Pref.SENSOR.toString(), isUsingSensor());
        editor.putBoolean(Pref.AUDIO.toString(), isMute());
        editor.commit();
    }


    // If the player hits the back button while video is playing leads to
closing the video player
    // Should it already been closed then the changed data (boolean values)
are discarded
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            if (videoHolder.isPlaying()) {
                videoHolder.stopPlayback();
                videoHolder.clearFocus();

OptionsActivity.this.setContentView(R.layout.activity_options);
                initialiseView();
            } else {
                //if quit without saving -> load old data
                loadData();
                finish();
            }


            return true;
        }
        return false;
    }


    /*
    *   Setting the focus to the video holder and reinitialising the
content view to default
    *   is implemented into the tutorial button.
    *   Otherwise the following lines of code are similar to main activity
    */
```

```
    private void setButtonListeners() {

        tutorialButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);
                //with controls if the apk allows it
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
{
                    videoHolder.setMediaController(mediaController);
                }
                //get the tutorial video inside the raw folder
                Uri video = Uri.parse("android.resource://" +
getPackageName() + "/" + R.raw.tutorial);
                videoHolder.setVideoURI(video);
                //change content of the activity
                setContentView(videoHolder);
                soundManager.stopMusic();
                videoHolder.requestFocus();
                videoHolder.start();

                //when video finishes the content is set back to the right
layout and set to default state
                videoHolder.setOnCompletionListener(new
MediaPlayer.OnCompletionListener() {
                    @Override
                    public void onCompletion(MediaPlayer mediaPlayer) {
                        videoHolder.stopPlayback();
                        videoHolder.clearFocus();

OptionsActivity.this.setContentView(R.layout.activity_options);
                        initialiseView();
                    }
                });
            }
        });

        tutorialButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

tutorialButton.setBackgroundResource(R.drawable.red_button);
                else

tutorialButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

        bluetoothButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);
                //save and start new activity
                saveOptions();
                Intent i = new Intent(OptionsActivity.this,
BluetoothActivity.class);
                finish();
                startActivity(i);
            }
        });
```

```
        bluetoothButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

bluetoothButton.setBackgroundResource(R.drawable.red_button);
                else

bluetoothButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

        saveButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                soundManager.playSound(SoundManager.Sounds.MENU);
                saveOptions();
                //soundManager.releasePlayer();
                finish();
            }
        });

        saveButton.setOnFocusChangeListener(new
View.OnFocusChangeListener() {
            @Override
            public void onFocusChange(View view, boolean b) {
                if (b)

saveButton.setBackgroundResource(R.drawable.red_button);
                else

saveButton.setBackgroundResource(R.drawable.blue_button);
            }
        });

        audioEnableButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                setMute(false);
                soundManager.setMute(false);
                setButtonStates();
                soundManager.playSound(SoundManager.Sounds.MENU);
                soundManager.playMusic();
            }
        });

        audioDisableButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                setMute(true);
                soundManager.setMute(true);
                setButtonStates();
                soundManager.stopMusic();
            }
        });


        accelEnableButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                setUsingSensor(true);
                setButtonStates();
```

```
                soundManager.playSound(SoundManager.Sounds.MENU);
            }
        });

        accelDisableButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                setUsingSensor(false);
                setButtonStates();
                soundManager.playSound(SoundManager.Sounds.MENU);
            }
        });

    }

    //Set the buttons states depending on the boolean values - changes
alpha value for transparency
    private void setButtonStates() {
        if (isMute()) {

audioDisableButton.setBackgroundResource(R.drawable.blue_button);
            audioDisableButton.setAlpha(1f);
            audioEnableButton.setBackgroundResource(R.drawable.red_button);
            audioEnableButton.setAlpha(.5f);
        } else {

audioEnableButton.setBackgroundResource(R.drawable.blue_button);
            audioEnableButton.setAlpha(1f);

audioDisableButton.setBackgroundResource(R.drawable.red_button);
            audioDisableButton.setAlpha(.5f);
        }

        if (isUsingSensor()) {

accelEnableButton.setBackgroundResource(R.drawable.blue_button);
            accelEnableButton.setAlpha(1f);

accelDisableButton.setBackgroundResource(R.drawable.red_button);
            accelDisableButton.setAlpha(.5f);
        } else {

accelDisableButton.setBackgroundResource(R.drawable.blue_button);
            accelDisableButton.setAlpha(1f);
            accelEnableButton.setBackgroundResource(R.drawable.red_button);
            accelEnableButton.setAlpha(.5f);
        }
    }


    @Override
    protected void onPause() {
        super.onPause();
        soundManager.stopMusic();
    }

    @Override
    protected void onResume() {
        super.onResume();
        soundManager.playMusic();
    }
```

```
    //GETTERS AND SETTERS
    public boolean isUsingSensor() {
        return usingSensor;
    }

    public void setUsingSensor(boolean usingSensor) {
        this.usingSensor = usingSensor;
    }

    public boolean isMute() {
        return isMute;
    }

    public void setMute(boolean mute) {
        this.isMute = mute;
    }
}
```

```
package com.hafezi.games.spaceshooter2d;

import android.content.Context;
import android.content.SharedPreferences;
import android.content.res.AssetFileDescriptor;
import android.content.res.AssetManager;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.media.SoundPool;
import android.util.Log;

import com.hafezi.games.spaceshooter2d.Utility.Pref;

import java.io.IOException;

/**
 * Created by Mojtaba Hafezi on 18.02.2018.
 */
// The SoundManager needs to be accessible from all other activities ->
singleton design pattern
public class SoundManager {
    //the instance every other class will access
    private static SoundManager instance;
    private Context context;
    //classes required for sound and music
    private SoundPool soundPool;
    private MediaPlayer mediaPlayer;
    //keep track where mediaplayer stopped to continue whenever the game is
paused
    private int length;
    private boolean mute;

    // ids for the sound effects - will be loaded upon instantiation of the
class
    int menu = -1;
    int explosion = -1;
    int hit = -1;
    int laser = -1;

    //persistence - gets the mute value
    private SharedPreferences sharedPreferences;

    //enum for the sound effects
    public enum Sounds {
```

```
        MENU, EXPLOSION, HIT, LASER
    }

    private SoundManager(Context context) {
        this.context = context;
        // If id doesn't exist one is created
        sharedPreferences =
context.getSharedPreferences(Pref.GAME.toString(), context.MODE_PRIVATE);
        boolean toMute =
sharedPreferences.getBoolean(Pref.AUDIO.toString(), false);
        setMute(toMute);
        //loads all sound effects so it can play them whenever required
        loadSound(context);
    }

    //returns the actual instance - only one instance of this class will be
available
    public static SoundManager getInstance(Context context) {
        if (instance == null) {
            instance = new SoundManager(context);
        }
        return instance;
    }

    private void loadSound(Context context) {
        //Sound
        soundPool = new SoundPool(10, AudioManager.STREAM_MUSIC, 0);
        try {
            //Create objects of the 2 required classes
            AssetManager assetManager = context.getAssets();
            AssetFileDescriptor descriptor;
            //create the sounds
            descriptor = assetManager.openFd("explosion.ogg");
            explosion = soundPool.load(descriptor, 0);
            descriptor = assetManager.openFd("hit.ogg");
            hit = soundPool.load(descriptor, 0);
            descriptor = assetManager.openFd("menu.ogg");
            menu = soundPool.load(descriptor, 0);
            descriptor = assetManager.openFd("laser.ogg");
            laser = soundPool.load(descriptor, 0);
        } catch (IOException e) {
            Log.e("error", "failed to load sound files");
        }
        //Media
        mediaPlayer = MediaPlayer.create(context, R.raw.ambient);
    }

    //if sound is enabled then the sound effect is played once
    public void playSound(Sounds sound) {
        if (isMute())
            return;
        switch (sound) {
            case HIT:
                soundPool.play(hit, 1, 1, 0, 0, 1);
                break;
            case EXPLOSION:
                soundPool.play(explosion, 1, 1, 0, 0, 1);
                break;
            case MENU:
                soundPool.play(menu, 1, 1, 0, 0, 1);
                break;
            case LASER:
                soundPool.play(laser, 1, 1, 0, 0, 1);
```

```
                break;
        }

    }

    //if sound is enabled the music will be played in a loop
    public void playMusic() {
        if (isMute())
            return;
        //use of media player is recommended by Google instead of sound
pool for ambient music
        if (mediaPlayer == null || !mediaPlayer.isPlaying()) {
            mediaPlayer = MediaPlayer.create(this.context, R.raw.ambient);
            mediaPlayer.setLooping(true);
            //if the music was stopped before - continue
            if (length > 0) {
                mediaPlayer.seekTo(length);
            }
            mediaPlayer.start();
        }
    }
    //should the music be paused then the current position will be stored
    public void stopMusic() {
        if (mediaPlayer != null) {
            mediaPlayer.stop();
            length = mediaPlayer.getCurrentPosition();
        } else
            length = 0;
    }
    //the media player needs to be released once the application is about
to exit
    public void releasePlayer() {
        if (mediaPlayer != null) {
            mediaPlayer.release();
            mediaPlayer = null;
            length = 0;
        }

    }

    public boolean isMute() {
        return mute;
    }

    public void setMute(boolean mute) {
        this.mute = mute;
    }
}
```