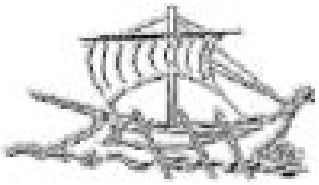


ΑΕΙ ΠΕΙΡΑΙΑ Τ.Τ.
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. "ΕΦΑΡΜΟΣΜΕΝΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εκπαιδευτικό υλικό σε Android Studio για τη μέση εκπαίδευση

Μαρία – Λεμονιά Καρδαρά

Εισηγητής: Δρ Ιωάννης Έλληνας

ΑΘΗΝΑ
ΙΑΝΟΥΑΡΙΟΣ 2018

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εκπαιδευτικό υλικό σε Android Studio για τη μέση εκπαίδευση

Μαρία – Λεμονιά Καρδαρά
A.M. ais0077

Εισηγητής:

Δρ Ιωάννης Έλληνας

Εξεταστική Επιτροπή:

Ημερομηνία εξέτασης:

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/Η κάτωθι υπογεγραμμένος/η Καρδαρά Μαρία – Λεμονιά, του Βασιλείου, με αριθμό μητρώου ais0077, φοιτητής/τρια του Τμήματος Μηχανικών Η/Υ Συστημάτων Τ.Ε. του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία ολοκληρώθηκε μετά από επίμονες προσπάθειες και δυσκολείες. Την προσπάθειά μου αυτή υποστήριξε ο επιβλέπων καθηγητής μου, τον οποίο θα ήθελα να ευχαριστήσω, όπως και για την κατανόηση που έδειξε.

Ένα ιδιαίτερο ευχαριστώ θα ήλεθα να δώσω στον κ. Νίκο Ξεφτεράκη, για την πολύτιμη βοήθεια και την επιστημονική γνώση που μου πρόσφερε. Τέλος, θα ήθελα να ευχαριστήσω τον σύζυγό μου Κώστα, για την απέραντη υπομονή και έμπρακτη υποστήριξη που μου παρείχε, καθώς και τις δύο νεογέννητες κόρες μου Ραφαέλα και Εριφύλη, για τον όποιον χρόνο μπορούσαν να παραμείνουν ήσυχες χωρίς να απαιτούν την προσοχή μου, ώστε να καταφέρω να ολοκληρώσω την εργασία μου.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία σχετίζεται με μία ενότητα του βιβλίου του μαθήματος «Ειδικά Θέματα στον Προγραμματισμό Υπολογιστών», το οποίο διδάσκεται στη Γ’ τάξη του τομέα Πληροφορικής του Επαγγελματικού Λυκείου, της Δευτεροβάθμιας Εκπαίδευσης. Συγκεκριμένα ασχολείται με την Ενότητα 2α: «Ανάπτυξη Εφαρμογών (APPS) για Android». Η εν λόγω ενότητα είναι αρκετά θεωρητική, καθώς αποτελείται κυρίως από παρουσίαση και εξήγηση προγραμματιστικών εννοιών, χωρίς να περιλαμβάνει παραδείγματα με κώδικα. Επειδή λοιπόν, υπάρχουν πολλοί καθηγητές που πρέπει να διδάξουν το μάθημα και δεν γνωρίζουν από ανάπτυξη εφαρμογών Android, αυτή η πτυχιακή εργασία αποτελεί ουσιαστικά έναν οδηγό με έτοιμα παραδείγματα εφαρμογών σε Android Studio και μια σύντομη εξήγησή τους.

ABSTRACT

The present thesis is related to a unit of the book "Special Topics in Computer Programming", which is taught in the 3rd grade, in the field of Informatics of the Vocational High School of Secondary Education. Specifically, it deals with the Unit 2a: "Android Applications Development (APPS)". This unit is quite theoretical, as it deals mainly with the presentation and the explanation of programming concepts, without including code examples. So, considering the many teachers who have to teach this lesson and are not familiar with Android applications development, this thesis is basically a guide with examples of Android Studio applications and a brief explanation of them.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Προγραμματισμός εφαρμογών για συσκευές Android
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Android Studio, Android, Java, applications, αλγόριθμος.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο αναλύεται το αντικείμενο της πτυχιακής εργασίας και περιγράφονται οι λόγοι και οι στόχοι συγγραφής της.

1.1 Περιγραφή του αντικειμένου της πτυχιακής εργασίας

Η παρούσα εργασία έχει γραφεί με σκοπό τη διευκόλυνση μαθητών – και κυρίως καθηγητών – στη διερεύνηση των εννοιών του μαθήματος «Ειδικά θέματα στον προγραμματισμό υπολογιστών» της Γ΄ τάξης του τομέα Πληροφορικής των Επαγγελματικών Λυκείων (ΕΠΑ.Λ.). Το μάθημα έχει σχεδιαστεί εξ' αρχής με αρθρωτή λογική και σε δύο μέρη. Το πρώτο μέρος ασχολείται με τις έννοιες του αντικειμενοστραφούς προγραμματισμού και τη γλώσσα JAVA. Το δεύτερο μέρος αποτελείται από δύο εναλλακτικά αρθρώματα, τα οποία αφορούν στην ανάπτυξη εφαρμογών για την πλατφόρμα Android που κατά κόρο συναντάται σε φορητές συσκευές, όπως tablets, έξυπνα τηλέφωνα, αλλά και σε έξυπνες πολυμεσικές συσκευές όπως τηλεοράσεις, media boxes κ.α.

Σημαντικό σημείο εδώ, είναι η κατανόηση ότι όταν γίνεται αναφορά στο Android, δεν συζητάμε για ένα λειτουργικό σύστημα, αλλά για μια ολοκληρωμένη πολυεπίπεδη πλατφόρμα όπου βασίζεται στο λειτουργικό σύστημα Linux. Πάνω σε αυτή είναι εγκατεστημένες μια σειρά από βιβλιοθήκες και μια μηχανή εκτέλεσης εφαρμογών σε γλώσσα java, έτσι ώστε να παρέχουν σε υψηλότερο επίπεδο ένα εκτεταμένο πλαίσιο από ρουτίνες. Η δομή αυτή, δίνει τη δυνατότητα στους προγραμματιστές να αναπτύξουν πολυσύνθετες εφαρμογές με πολύ πιο εύκολο τρόπο, προσφέροντας ανεξαρτησία από το υλικό.

Επιστρέφοντας στη δομή του μαθήματος, το πρώτο άρθρωμα του δεύτερου μέρους του χρησιμοποιεί για προγραμματισμό Android το περιβάλλον ανάπτυξης Eclipse, το οποίο συναντάται επίσης και στο πρώτο μέρος. Εναλλακτικά αυτού, το δεύτερο άρθρωμα του δεύτερου μέρους αφορά επίσης σε προγραμματισμό για Android, με χρήση όμως του περιβάλλοντος οπτικού προγραμματισμού AppInventor.

Στο πρώτο άρθρωμα του δεύτερου μέρους αφορά η προσπάθεια της παρούσης πτυχιακής εργασίας (Ενότητα 2α). Η επιλογή του Eclipse όταν γραφόταν το αναλυτικό πρόγραμμα και αργότερα το βιβλίο του μαθήματος, ήταν εύκολη: επρόκειτο για ελεύθερο λογισμικό υψηλών δυνατοτήτων, με έτοιμους προσομοιωτές για τις περισσότερες τηλεφωνικές συσκευές της αγοράς και μπορούσε να εξυπηρετήσει τόσο το πρώτο, όσο και το δεύτερο μέρος. Ήταν όμως εξαρχής γνωστό ότι το, βαρύτερο σχετικά, με αυξημένες απαιτήσεις από πλευρά υλικού, αλλά και συνεχώς υπό ανάπτυξη, Android Studio θα αντικαθιστούσε το Eclipse με την πάροδο του χρόνου.

Σήμερα, μόλις λίγα χρόνια από την εφαρμογή του νέου προγράμματος σπουδών, η Google δεν υποστηρίζει πλέον το Eclipse ενώ το Android Studio είναι λειτουργικό, έχει προσομοιωτές για τις περισσότερες συσκευές της αγοράς και κερδίζει διαρκώς μερίδιο στην αγορά ανάπτυξης λογισμικού για Android. Προτιμάται επειδή παρέχει μια σειρά από αυτοματοποιημένες λειτουργίες εισαγωγής κώδικα, καθώς και πολλαπλά εργαλεία για τον έλεγχο, την εκτέλεση και τη διόρθωση του κώδικα. Επιπλέον, ένα σημαντικό πλεονέκτημα είναι και η τεκμηρίωση του δικτυακού τόπου Android Developer που υποστηρίζει το περιβάλλον του Android Studio.

Ατυχώς ακόμα δεν μπορεί να εγκατασταθεί στα περισσότερα Εργαστήρια Πληροφορικής, τα οποία δεν έχουν ανανεωθεί πρόσφατα από πλευράς υλικού. Παρόλα αυτά, για τα εργαστήρια που μπορούν να το σηκώσουν, είναι σκόπιμη η επιλογή του από πλευράς επαγγελματικής ανάπτυξης των μαθητών της ειδικότητας. Αυτή την ανάγκη καλύπτει το υλικό της παρούσας πτυχιακής εργασίας, ενώ ταυτόχρονα αποτελεί σημαντικό βοήθημα για τον εκπαιδευτικό που διδάσκει το μάθημα.

Το υλικό αποτελείται από τέσσερις μικρές αλλά χαρακτηριστικές εφαρμογές (Android projects), που αναδεικνύουν συχνά χρησιμοποιούμενες μεθόδους, και μία εκτεταμένη που αναδεικνύει σημαντικά ζητήματα και περιπλοκές στην ανάπτυξη εφαρμογών μεγαλύτερης εμβέλειας.

Χωρίς να επαναλαμβάνονται στοιχεία θεωρίας, προσφέρονται για τον διδάσκοντα οδηγίες για την αξιοποίηση του υλικού, παρέχοντας έτσι μία αναβάθμιση του υπάρχοντος υλικού του βιβλίου και ταυτόχρονα ένα πολύτιμο βοήθημα για το άρθρωμα που διαπραγματεύεται.

Προσωπικά, γνωρίζω ορισμένους από την ομάδα εκπαιδευτικών συγγραφέων του βιβλίου και έχω παρακολουθήσει από κοντά τους προβληματισμούς τους. Συμμετείχα επίσης, στην ανάπτυξη κώδικα για το δεύτερο άρθρωμα του δεύτερου μέρους (Ενότητα 2β: Ανάπτυξη Εφαρμογών με AppInventor). Θεωρώ λοιπόν ότι υπήρχε ανάγκη για ανάπτυξη συμβατού υλικού για Android Studio και για την ενότητα 2α, όσο και του τρόπου για να γίνει αυτό εκπαιδευτικά αξιοποιήσιμο.

Σημείωση: Ολοκληρωμένος ο κώδικας όλων των εφαρμογών βρίσκεται στο Παράρτημα της εργασίας, καθώς επίσης είναι και αποθηκευμένα στο CD που συνοδεύει την εργασία. Επιπλέον, για τη μελέτη και την εκτέλεση των εφαρμογών μέσα από την πλατφόρμα Android Studio, πολύ βοηθητικό είναι το βίντεο του Karthik M., "*How to import existing Android Studio Project In to Android Studio with New Package Name"*, όπου δείχνει βήμα-βήμα πως επιτυγχάνεται η εισαγωγή ενός project στο Android Studio με καινούριο package name, και βρίσκεται στη διεύθυνση: https://www.youtube.com/watch?v=isr3zXK42po.

ΚΕΦΑΛΑΙΟ 2 – Η πρώτη εφαρμογή

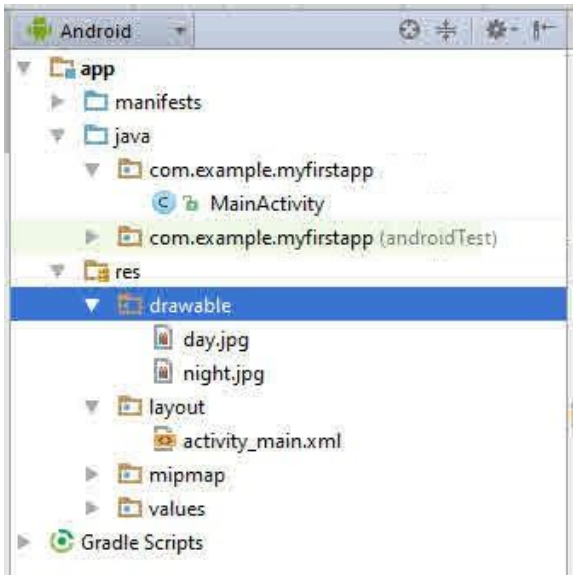
ΕΙΣΑΓΩΓΗ

Η πρώτη εφαρμογή που θα παρουσιαστεί είναι μια απλή εφαρμογή, όπου αναλόγως το κουμπί που θα πατηθεί (*Μέρα ή Νύχτα*) θα αλλάζει το σκηνικό στην οθόνη της συσκευής.

Εικόνες 2.1, 2.2: Στιγμιότυπα από την εφαρμογή MyFirstApp

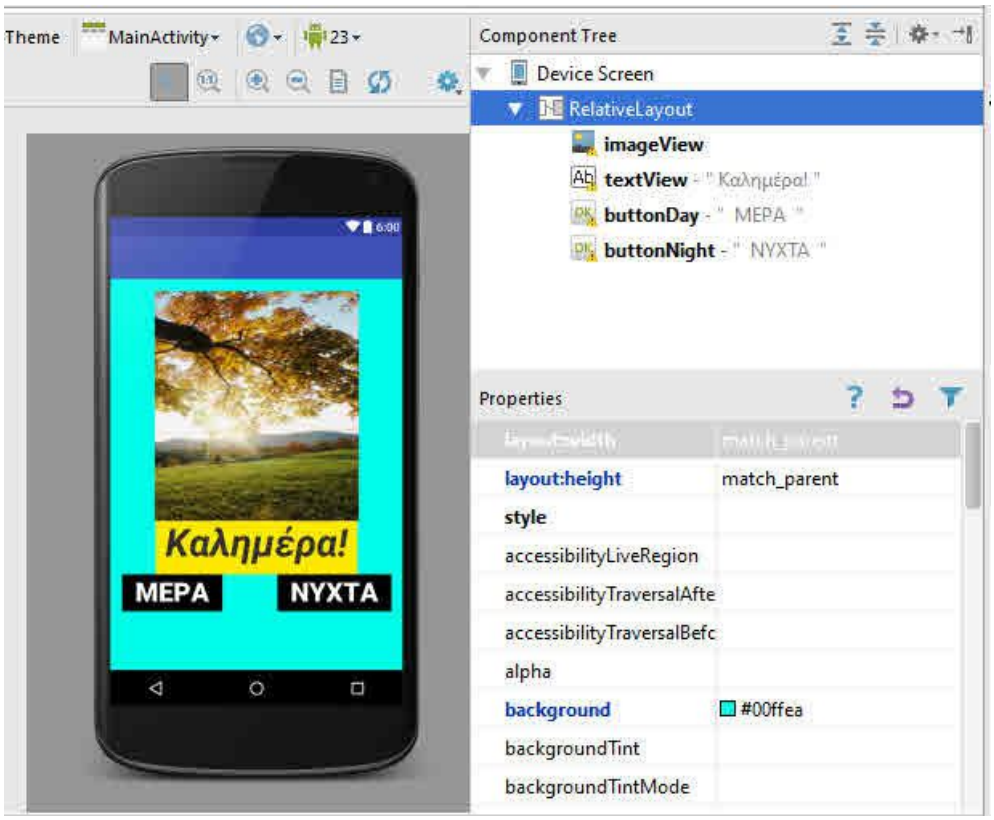
2.1 Layout – αρχείο xml της εφαρμογής MyFirstApp

Αρχικά, δημιουργείται ένα νέο project (*File → New → New Project)* με ονομασία *MyFirstApp*. Χρειάζονται επίσης και δύο φωτογραφίες που να δείχνουν τη μέρα και τη νύχτα, οι οποίες θα τοποθετούν μέσα στον φάκελο *drawable (*day.jpg και night.jpg*)*.



Εικόνα 2.3: Ο package explorer με τη δομή της εφαρμογής (κώδικες, resources, κλπ)

Η πρώτη δουλειά είναι να φτιαχτεί το layout της εφαρμογής. Ανοίγοντας το αρχείο *activity_main.xml* (στον package explorer, μέσα στο φάκελο *layout* του φακέλου *res*), επιλέγεται η καρτέλα *Design* και από τη παλέτα με τα αντικείμενα, με την μέθοδο drag-and-drop, εισάγονται στην εικόνα της συσκευής τα εξής αντικείμενα: από την ομάδα *widgets* ένα *ImageView*, ένα *Plain TextView* και δύο *Button*.



Εικόνα 2.4: Καρτέλα Design που δείχνει τη γραφική απεικόνιση του xml αρχείου

Μέσα από το χώρο των *Properties* υπάρχει η δυνατότητα να μεταβληθούν οι απαραίτητες ιδιότητες των αντικειμένων, όπως id (imageView για την εικόνα, textView για την ετικέτα, buttonDay για το κουμπί της Μέρας και buttonNight για το κουμπί της Νύχτας), background, text, κ.α. Επίσης υπάρχει και η δυνατότητα μέσω της καρτέλας *Text* να προγραμματιστούν οι τιμές των ιδιοτήτων σε γλώσσα xml, όπως φαίνεται στα παρακάτω κομμάτια του κώδικα:

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="300dp"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:background="@drawable/day" />
```

Εδώ, γίνεται εισαγωγή ενός αντικειμένου *ImageView*, στο οποίο θα εμφανίζεται η φωτογραφία για τη Μέρα και τη Νύχτα, αναλόγως ποιο κουμπί θα πατηθεί. Ορίζονται το id (imageView) μέσω του οποίου θα αναφέρεται, το ύψος και το πλάτος του (layout_height, layout_width), σε ποιο σημείο θα είναι τοποθετημένο και στοιχισμένο στην οθόνη της συσκευής (layout_alignParentTop, layout_centerHorizontal), καθώς και ποια φωτογραφία θα έχει αρχικά ως φόντο (background), δηλαδή η εικόνα της μέρας (day.jpg), που βρίσκεται μέσα στον φάκελο drawable.

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="70dp"
    android:layout_below="@+id/imageView"
    android:layout_centerHorizontal="true"
    android:text=" Καλημέρα! "
    android:background="#ffea00"
    android:textSize="50dp"
    android:textStyle="bold|italic"
    android:textColor="#313030" />
```

Στη συνέχεια, γίνεται εισαγωγή ενός αντικειμένου *TextView*, όπου αναλόγως θα εμφανίζεται το κείμενο “Καλημέρα” και “Καληνύχτα”. Ορίζονται το id του (textView), το ύψος και το πλάτος του, σε ποιο σημείο θα είναι τοποθετημένο σε

σχέση με το ImageView, καθώς και το κείμενο που θα εμφανίζεται αρχικά, με το επιθυμητό μέγεθος, στυλ και χρώμα γραμματοσειράς και φόντου.

```
<Button
    android:id="@+id/buttonDay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:text="  ΜΕΡΑ  "
    android:textStyle="bold"
    android:textSize="35dp"
    android:background="#000000"
    android:textColor="#ffffff"
    android:onClick="setDay"/>

<Button
    android:id="@+id/buttonNight"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:text="  ΝΥΧΤΑ  "
    android:textStyle="bold"
    android:textSize="35dp"
    android:background="#000000"
    android:textColor="#ffffff"
    android:onClick="setNight"/>
```

Τέλος, προσθέτονται δύο *Buttons*, όπου με το πάτημα στο καθένα θα εμφανίζονται τα στοιχεία της Μέρας και της Νύχτας αντίστοιχα. Ορίζονται το id τους (buttonDay και buttonNight), το ύψος, πλάτος, τη σχετική με τα άλλα αντικείμενα τοποθέτησή τους και το κείμενο που θα αναγράφεται πάνω τους. Μια πολύ σημαντική ιδιότητα που το όνομά της ορίζεται επίσης, είναι το *onClick* (setDay και setNight), το οποίο είναι μια ενέργεια (event) που θα εκτελεστεί όταν πατηθεί το κουμπί. Το ποιες εντολές θα εκτελεστούν ακριβώς θα παρουσιαστούν αμέσως παρακάτω στον κώδικα java της εφαρμογής.

2.2 Κώδικας java της εφαρμογής MyFirstApp

Ανοίγοντας το αρχείο *MainActivity.java*, θα βρεθούμε στο περιβάλλον όπου υλοποιείται ο προγραμματισμός του κυρίου κώδικα της εφαρμογής. Το Android

Studio δίνει έτοιμα κάποια κομμάτια κώδικα, όπως είναι η μέθοδος onCreate που αποτελεί ένα στάδιο του κύκλου ζωής ενός activity.

```
package com.example.myfirstapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Πάνω στα παραπάνω θα χτιστεί και ο υπόλοιπος κώδικας. Κομμάτια του κώδικα της εφαρμογής παραθέτονται παρακάτω:

```
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
```

Αρχικά, γίνεται προσθήκη των απαραίτητων βιβλιοθηκών που θα χρειαστούν τα αντικείμενα που θα χρησιμοποιηθούν. Το Android Studio έχει την δυνατότητα να ειδοποιεί με μήνυμα όταν χρειάζεται να εισαχθεί μια βιβλιοθήκη, οπότε μπορεί η προσθήκη να γίνει και κατά τη διάρκεια συγγραφής του κώδικα και όχι στην αρχή (πράγμα που είναι πιο δύσκολο, αφού μπορεί να μην γνωρίζουμε ποιες ακριβώς θα χρειαστούν).

```
public class MainActivity extends AppCompatActivity {
    private ImageView imgView;
    private TextView txtView;
    private Button btnDay, btnNight;
```

Στη συνέχεια, στο κύριο activity της εφαρμογής, το *MainActivity*, ορίζονται τα χαρακτηριστικά του: μια εικόνα *(ImageView)*, ένα κείμενο *(TextView)* και δυο κουμπιά *(Button)*.

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

    imgView=(ImageView) findViewById(R.id.imageView);
    txtView=(TextView) findViewById(R.id.textView);
    btnDay=(Button) findViewById(R.id.buttonDay);
    btnNight=(Button) findViewById(R.id.buttonNight);
}
```

Στην αρχή της μεθόδου *onCreate*, συνδέονται τα χαρακτηριστικά που μόλις πριν ορίστηκαν, με τα αντίστοιχα αντικείμενα του layout από το αρχείο activity_mail.xml.

```
// Μέθοδος όπου αλλάζει το σκηνικό σε Μέρα
public void setDay(View view) {
    imgView.setBackgroundResource(R.drawable.day);
    txtView.setText(" Καλημέρα! ");
    txtView.setBackgroundColor(Color.YELLOW);
    txtView.setTextColor(Color.DKGRAY);
}

// Μέθοδος όπου αλλάζει το σκηνικό σε Νύχτα
public void setNight(View view) {
    imgView.setBackgroundResource(R.drawable.night);
    txtView.setText(" Καληνύχτα! ");
    txtView.setBackgroundColor(Color.DKGRAY);
    txtView.setTextColor(Color.RED);
}
```

Τέλος, υλοποιείται η μέθοδος *setDay*, η οποία θα εκτελεστεί όταν ενεργοποιηθεί το event *onClick*, του button *"buttonDay"* στο layout (xml) της εφαρμογής, δηλαδή στην ουσία όταν πατηθεί το κουμπί "Μέρα". Οι ενέργειες που θα εκτελεστούν θα είναι να αλλάξει το background του imageView στην εικόνα της Μέρας (day.jpg), καθώς και να αλλάξει το κείμενο του textView σε "Καλημέρα!", με χρώμα φόντου κίτρινο και χρώμα γραμματοσειράς σκούρο γκρι.

Αντιστοίχως θα λειτουργήσει και η μέθοδος *setNight*, όπου θα εκτελεστεί όταν πατηθεί το συγκεκριμένο κουμπί *"buttonNight"* ("Νύχτα"). Θα γίνει αλλαγή στο background του imageView στην εικόνα της Νύχτας (night.jpg), και στο κείμενο του textView σε "Καληνύχτα!", με χρώμα φόντου σκούρο γκρι και χρώμα γραμματοσειράς κόκκινο.

Ολοκληρωμένος ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα, σελ. 87.

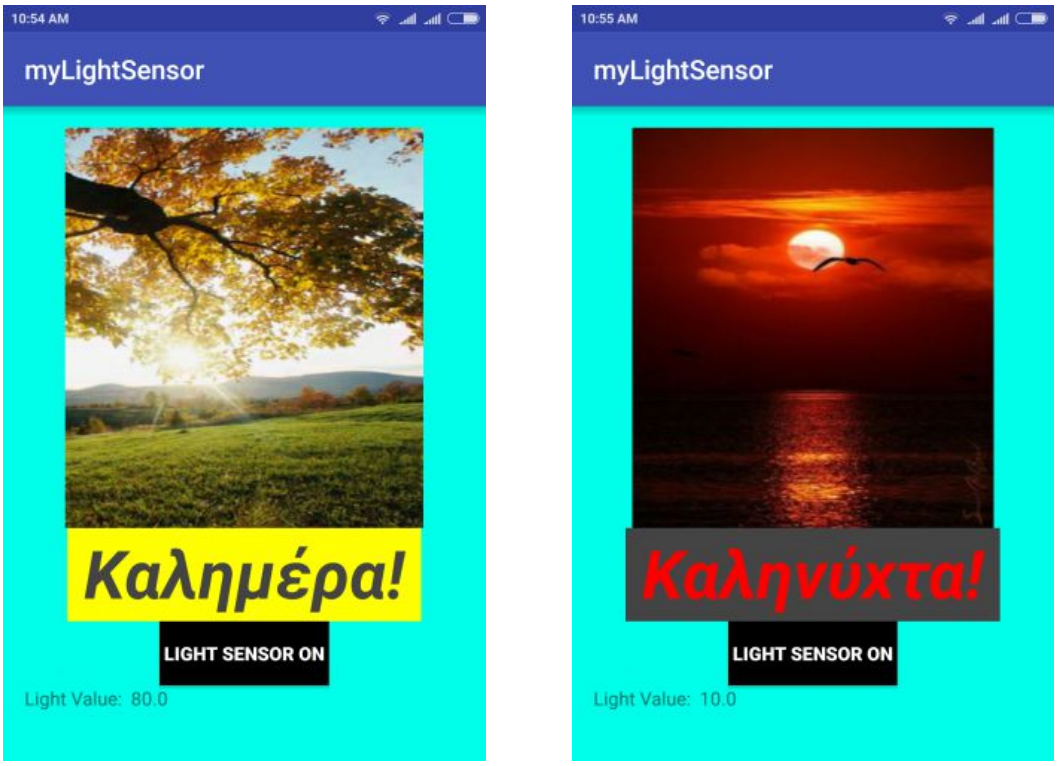
ΚΕΦΑΛΑΙΟ 3 – Επικοινωνία με τους αισθητήρες

Στο κεφάλαιο αυτό θα περιγραφούν δύο εφαρμογές που κεντρικό τους στοιχείο είναι οι αισθητήρες που υπάρχουν στις συσκευές.

3.1 Αισθητήρας φωτός

ΕΙΣΑΓΩΓΗ

Η δεύτερη εφαρμογή που θα υλοποιηθεί είναι ουσιαστικά μια παραλλαγή της πρώτης, με τη διαφορά ότι το σκηνικό στην οθόνη της συσκευής θα αλλάζει από Μέρα σε Νύχτα και αντίστροφα, σύμφωνα με την ένταση του φωτός που υπάρχει εκείνη τη στιγμή. Για να επιτευχθεί αυτό, χρειάζεται να εγκατασταθεί επικοινωνία μεταξύ της εφαρμογής και του αισθητήρα φωτός (light sensor) που παρέχει η συσκευή, ώστε να μετρήσει τις τιμές της έντασης του φωτός, να διαβαστούν μέσω του κώδικα και αναλόγως των τιμών αυτών να εμφανιστούν τα στοιχεία της Μέρας ή της Νύχτας.

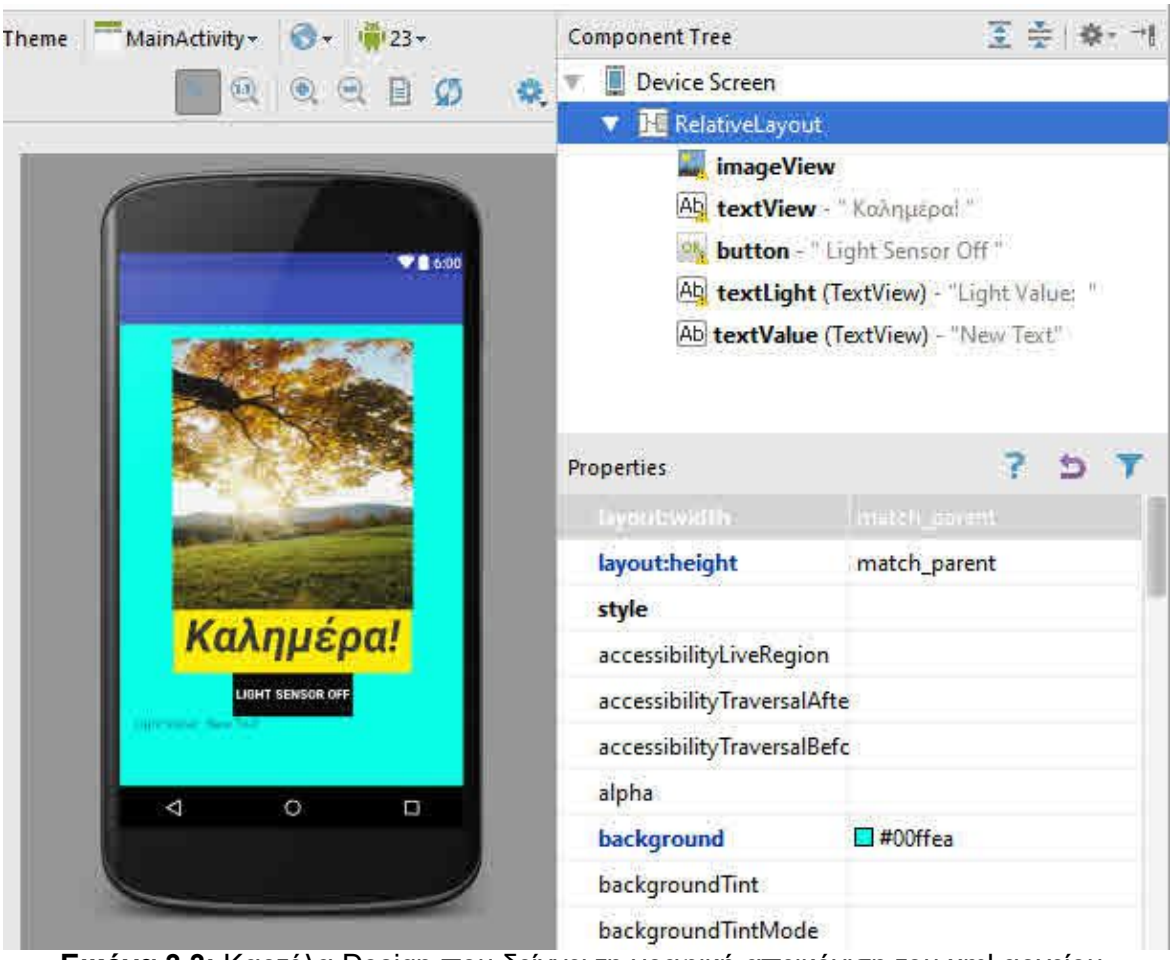


Εικόνες 3.1, 3.2: Στιγμιότυπα από την εφαρμογή MyLightSensor

3.1.1 Layout – αρχείο xml της εφαρμογής MyLightSensor

Δημιουργείται νέο project με ονομασία *MyLightSensor* και τοποθετούνται οι φωτογραφίες της Μέρας και της Νύχτας μέσα στον φάκελο *drawable*, όπως ακριβώς στην πρώτη εφαρμογή.

Το layout της εφαρμογής *(activity_main.xml)* θα περιέχει τα γνωστά αντικείμενα *ImageView* και *Plain TextView*, για την εμφάνιση της φωτογραφίας και του συναφούς κειμένου αντίστοιχα, και επιπρόσθετα ένα *Button* για την ενεργοποίηση / απενεργοποίηση του αισθητήρα φωτός, και δύο *Plain TextView* που θα δείχνουν την τιμή του φωτός που έχει μετρήσει ο αισθητήρας.



Εικόνα 3.3: Καρτέλα Design που δείχνει τη γραφική απεικόνιση του xml αρχείου

Μέσα από το χώρο των *Properties* (στην καρτέλα *Design*) ή / και μέσω κώδικα xml (στην καρτέλα *Text*) μπορούν να μεταβληθούν οι ιδιότητες των παραπάνω αντικειμένων. Κομμάτια του κώδικα xml φαίνονται παρακάτω:

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="300dp"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:background="@drawable/day" />
```

Στο αντικείμενο *ImageView*, όπου θα εμφανίζεται η φωτογραφία για τη Μέρα και τη Νύχτα, όπως και στην πρώτη εφαρμογή, ορίζονται η ιδιότητα id σε imageView, με την οποία θα αναφερόμαστε σ' αυτό, το ύψος και το πλάτος του (layout_height, layout_width), η τοποθέτηση και στοίχιση του στην οθόνη της συσκευής (layout_alignParentTop, layout_centerHorizontal), καθώς και ποια φωτογραφία θα έχει αρχικά ως φόντο (background), δηλαδή η εικόνα της μέρας (day.jpg).

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="70dp"
    android:layout_below="@+id/imageView"
    android:layout_centerHorizontal="true"
    android:text=" Καλημέρα! "
    android:background="#ffea00"
    android:textSize="50dp"
    android:textStyle="bold|italic"
    android:textColor="#313030" />
```

Επίσης, όπως και στην πρώτη εφαρμογή, εισάγεται ένα αντικείμενο *TextView*, στο οποίο θα εμφανίζεται το κείμενο "Καλημέρα" ή "Καληνύχτα". Μεταβάλλεται το id του σε textView, το ύψος και το πλάτος του, η τοποθέτησή του σε σχέση με το ImageView, και το κείμενο που θα εμφανίζεται αρχικά με το μέγεθος, το στυλ και το χρώμα της γραμματοσειράς και του φόντου.

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true"
    android:text=" Light Sensor Off "
    android:background="#000000"
    android:textColor="#ffffff"
    android:textStyle="bold"
    android:onClick="sensorOnOff" />
```

Στη συνέχεια, γίνεται εισαγωγή ενός *Button*, με το οποίο θα επικοινωνεί ή θα παύει να επικοινωνεί η εφαρμογή με τον αισθητήρα φωτός της Android συσκευής. Ορίζονται οι εξής ιδιότητες: το id του σε button, το ύψος και το πλάτος του, η στοίχιση του στην οθόνη (layout_below, layout_centerHorizontal), το κείμενο που θα αναγράφεται αρχικά μαζί με το στυλ και το χρώμα της γραμματοσειράς και του φόντου (text = ”Light Sensor Off” – με το ξεκίνημα της εφαρμογής θα είναι απενεργοποιημένη η επικοινωνία). Επιπλέον, ορίζεται και η ιδιότητα onClick (sensorOnOff), ένα event που θα εκτελέσει στον κώδικα java τις εντολές ενεργοποίησης / απενεργοποίησης της επικοινωνίας όταν πατηθεί το κουμπί.

```
<TextView
    android:id="@+id/textLight"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:text="Light Value:  " />
```

Το συγκεκριμένο *TextView* (με id textLight) είναι απλώς μια ετικέτα, ένας τίτλος που θα γράφει το κείμενο “Light Value:” και τοποθετείται ακριβώς μπροστά από το επόμενο TextView, το οποίο δείχνει την τιμή της έντασης του φωτός.

```
<TextView
    android:id="@+id/textValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/button"
    android:layout_toRightOf="@+id/textLight"
    android:layout_toEndOf="@+id/textLight"
    android:text=" ... " />
```

Το επόμενο *TextView* δείχνει την τιμή της υπάρχουσας έντασης του φωτός που έχει μετρήσει ο αισθητήρας φωτός. Θέτονται το id του σε textValue, το ύψος και το πλάτος του, η σχετική τοποθέτησή του (κάτω από το button και δίπλα, δεξιά της προηγούμενης ετικέτας), καθώς και το κείμενο που θα εμφανίζει όταν ξεκινάει η εφαρμογή, όπου δεν υπάρχει ακόμα επικοινωνία με τον αισθητήρα για να στείλει κάποια τιμή (“…”).

3.1.2 Κώδικας java της εφαρμογής MyLightSensor

Αφού ολοκληρωθεί ο σχεδιασμός του layout της εφαρμογής, το επόμενο βήμα είναι η υλοποίηση του κύριου κώδικα στο περιβάλλον προγραμματισμού της γλώσσας java, στο αρχείο *MainActivity.java*.

Κομμάτια του κώδικα της εφαρμογής παραθέτονται παρακάτω:

```
import android.content.Context;
import android.graphics.Color;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
```

Υλοποιείται η σύνδεση με τις απαραίτητες βιβλιοθήκες, είτε σε αυτό το σημείο για όσες ήδη γνωρίζουμε ότι θα χρειαστούν, είτε κατά τη διάρκεια της συγγραφής του κώδικα, κάθε φορά που θα εμφανίζεται μήνυμα για προσθήκη συγκεκριμένης βιβλιοθήκης.

```
public class MainActivity extends AppCompatActivity implements
SensorEventListener {
    private ImageView imgView;
    private TextView txtView, txtValue;
    private Button btn;
    private SensorManager sensorManager;
    private Sensor lightSensor;
    private boolean isStarted=false;
```

Εδώ, ορίζεται η κύρια class της εφαρμογής *(MainActivity)*, εφαρμόζοντας και έναν Listener. Συγκεκριμένα τον Listener που έχει την δυνατότητα να "ακούει", δηλαδή να επικοινωνεί με τους αισθητήρες της Android συσκευής. Ακολουθεί η προσθήκη των χαρακτηριστικών της κλάσης: μια εικόνα *(ImageView)*, δύο κείμενα *(TextView)*, ένα κουμπί *(Button)*, καθώς και έναν *SensorManager*, ο οποίος δίνει πρόσβαση στους αισθητήρες της συσκευής, έναν *LightSensor*, που αντιπροσωπεύει τον αισθητήρα φωτός συγκεκριμένα, και μια boolean μεταβλητή *isStarted* για να ελέγχεται εάν υπάρχει ή δεν υπάρχει επικοινωνία με τον αισθητήρα.

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

    imgView=(ImageView) findViewById(R.id.imageView);
    txtView=(TextView) findViewById(R.id.textView);
    btn=(Button)findViewById(R.id.button);
    txtValue=(TextView)findViewById(R.id.textValue);

    sensorManager=(SensorManager)getSystemService(Context.
SENSOR_SERVICE);
    lightSensor=sensorManager.getDefaultSensor(Sensor.
TYPE_LIGHT);
}
```

Στην αρχή της μεθόδου *onCreate*, συνδέονται τα χαρακτηριστικά που ορίστηκαν παραπάνω (imgView, txtView, btn, txtValue), με τα αντίστοιχα αντικείμενα του layout από το αρχείο activity_mail.xml. Επιπλέον, μέσω του sensorManager, ορίζεται και ο αισθητήρας φωτός (lightSensor) που θα χρησιμοποιηθεί *(TYPE_LIGHT)*.

```
public void sensorOnOff(View view) {
    if (isStarted) {
        isStarted=false;
        sensorManager.unregisterListener(this);
        btn.setText(" Light Sensor Off ");
    }
    else {
        isStarted=true;
        sensorManager.registerListener(this, lightSensor,
SensorManager.SENSOR_DELAY_NORMAL);
        btn.setText(" Light Sensor On ");
    }
}
```

Συνέχεια με τη μέθοδο *sensorOnOff*, η οποία θα εκτελεστεί όταν ενεργοποιηθεί το event *onClick*, του button στο layout (xml) της εφαρμογής, δηλαδή όταν πατηθεί το κουμπί. Μέσα σ' αυτή τη μέθοδο, ελέγχεται μέσω της boolean μεταβλητής σε μια εντολή if, εάν είναι ενεργοποιημένη ή όχι η επικοινωνία με τον light sensor. Εάν είναι ενεργοποιημένη τότε μέσω του *sensor manager* απενεργοποιείται *(unregisterListener)*, και αλλάζει το κείμενο του button σε "Light Sensor Off". Στην αντίθετη περίπτωση, ο *sensor manager* ανοίγει κανάλι επικοινωνίας με τους αισθητήρες της συσκευής, και συγκεκριμένα με τον αισθητήρα φωτός *(lightSensor)*. Ταυτόχρονα ορίζει το κάθε πότε θα καταγράφει ο αισθητήρας τις τιμές της έντασης του φωτός (*SENSOR_DELAY_NORMAL*, ταχύτητα normal), καθώς μεταβάλλει και το κείμενο του button σε "Light Sensor On".

```
@Override
public void onSensorChanged(SensorEvent event) {
    txtValue.setText(Float.toString(event.values[0]));

    if (event.values[0]>10) {
        imgView.setBackgroundResource(R.drawable.day);
        txtView.setText(" Καλημέρα! ");
        txtView.setBackgroundColor(Color.YELLOW);
        txtView.setTextColor(Color.DKGRAY);
    }
```

```
        else {
            imgView.setBackgroundResource(R.drawable.night);
            txtView.setText(" Καληνύχτα! ");
            txtView.setBackgroundColor(Color.DKGRAY);
            txtView.setTextColor(Color.RED);
        }
}
```

Τελειώνοντας με τις μεθόδους που χειρίζονται τις αλλαγές που συμβαίνουν στον light sensor, υπάρχει η *onSersorChanged*, η οποία είναι η μέθοδος που καλείται όταν συμβαίνουν αλλαγές στις τιμές που διαβάζει ο αισθητήρας. Εκτελεί τα ακόλουθα: Απεικονίζει στο αντικείμενο textValue στο layout της εφαρμογής, την τιμή της έντασης του φωτός που διάβασε και με μια εντολή if την ελέγχει. Εάν είναι μεγαλύτερη του 10 τότε αλλάζει το σκηνικό σε Μέρα, δηλαδή φορτώνει στο imageView την φωτογραφία της μέρας (day.jpg) από τα resources, θέτει το κείμενο του txtView σε “Καλημέρα!”, με χρώμα φόντου κίτρινο και χρώμα γραμματοσειράς σκούρο γκρι. Αλλιώς (*else*), αλλάζει το σκηνικό σε Νύχτα, φορτώνοντας στο imageView την εικόνα της νύχτας (night.jpg), με χρώμα φόντου σκούρο γκρι και χρώμα γραμματοσειράς κόκκινο.

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) { }
```

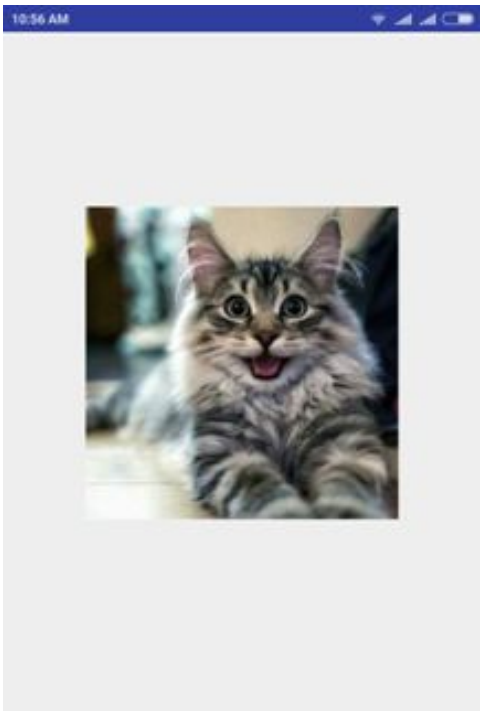
Υπάρχει και η μέθοδος *onAccuracyChanged* όπου καλείται όταν συμβαίνουν αλλαγές στην ακρίβεια του light sensor, αλλά στην εφαρμογή δεν χρειάζεται, οπότε αφήνεται κενή χωρίς να προγραμματιστεί καμία ενέργεια.

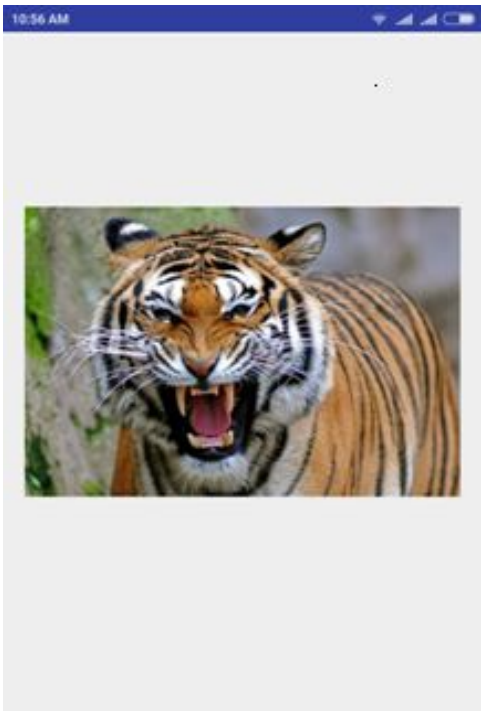
Ολοκληρωμένος ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα, σελ. 89.

3.2 Αισθητήρας απόστασης

ΕΙΣΑΓΩΓΗ

Σε αυτή την εφαρμογή θα πειραματιστούμε με τον αισθητήρα απόστασης (proximity sensor). Πιο συγκεκριμένα, θα υλοποιηθεί μια απλή εφαρμογή, η οποία θα μπορεί να ανιχνεύει όποτε κάποιο αντικείμενο πλησιάζει τη συσκευή. Όταν δεν υπάρχει κάποιο αντικείμενο κοντά της θα εμφανίζεται η φωτογραφία μιας γάτας και θα ακούγεται και ένα νιαούρισμα. Όταν όμως, κάποιο αντικείμενο (π.χ. ένα χέρι) πλησιάσει πολύ το τηλέφωνο ή το tablet, τότε θα εμφανίζεται η φωτογραφία μιας τίγρης και θα ακούγεται ο βρυχηθμός της.



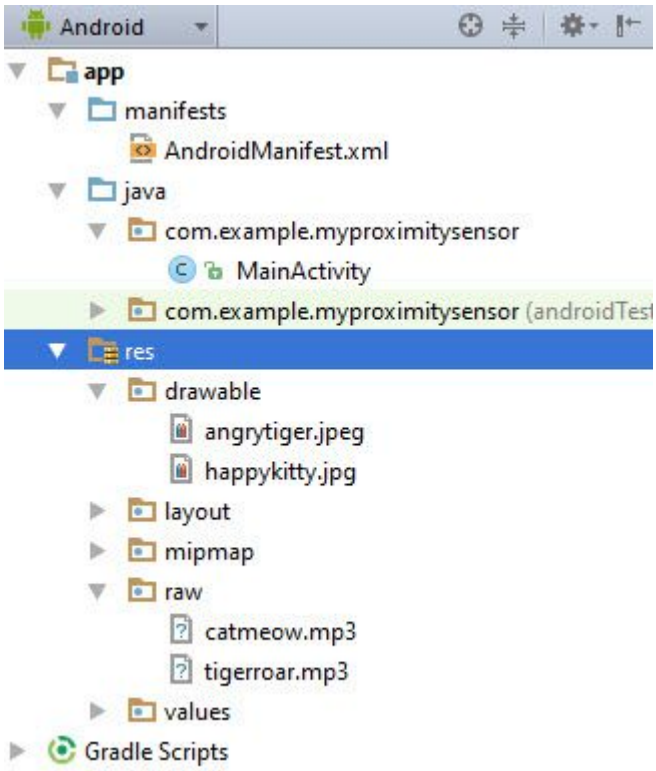


Εικόνες 3.4, 3.5: Στιγμιότυπα από την εφαρμογή MyProximitySensor

3.2.1 Layout – αρχείο xml της εφαρμογής MyProximitySensor

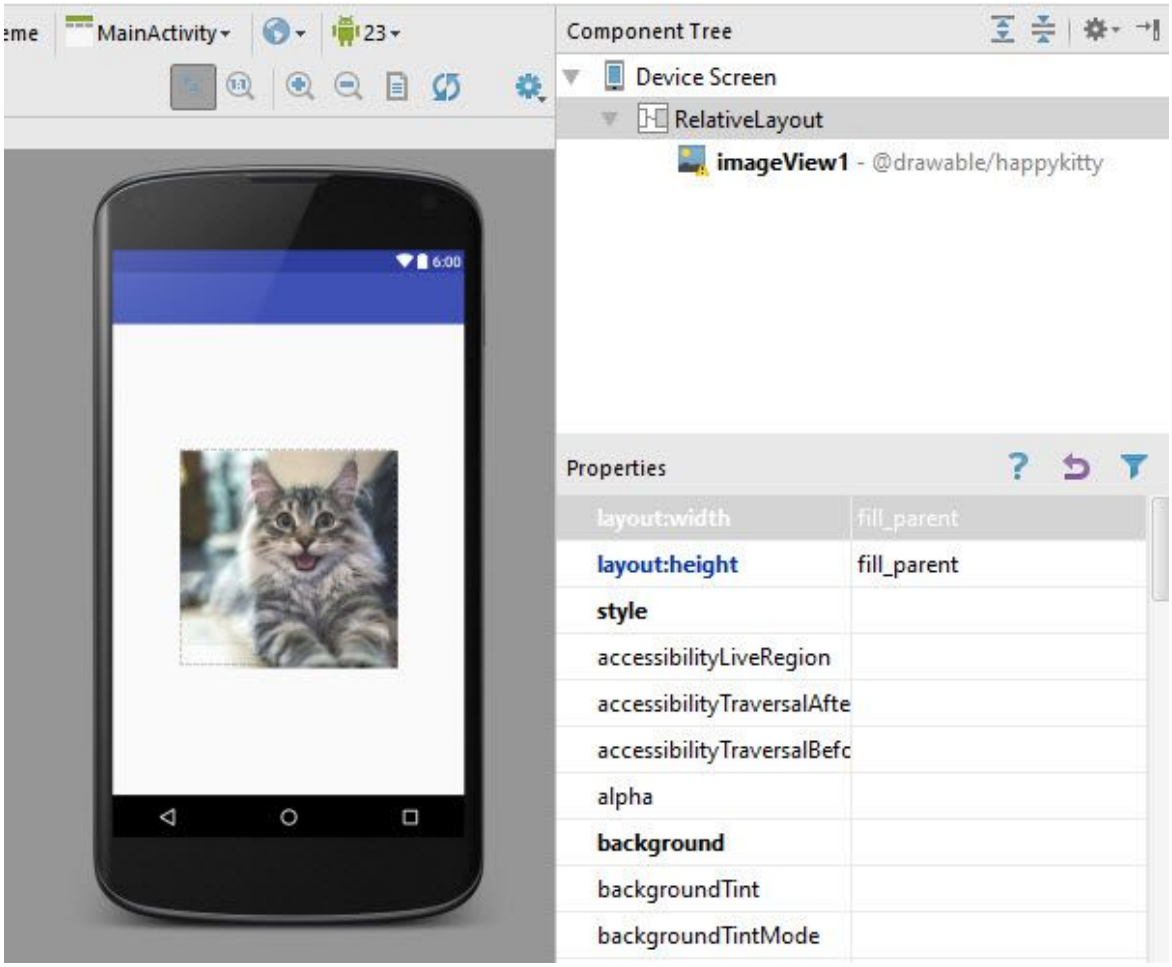
Αρχικά, θα δημιουργηθεί ένα νέο project με ονομασία *MyProximitySensor*. Χρειάζονται δύο φωτογραφίες, μιας γάτας και μιας τίγρης, που θα τοποθετηθούν μέσα στον φάκελο *drawable (happykitty.jpg* και *anrytiger.jpeg)*. Επιπλέον, χρειάζονται και δύο ήχοι, ένα νιαούρισμα και ένα βρυχηθμό *(catmeow.mp3* και *tigerroar.mp3)*. Οι ήχοι αυτοί, θα αποθηκευτούν στον φάκελο με ονομασία *raw* που θα έχουμε ήδη δημιουργήσει μέσα στον φάκελο res *(δεξί κλικ πάνω στο res*

→ *New* → *Directory)*. Η παρακάτω εικόνα δείχνει πώς θα φαίνονται τα αρχεία στον package explorer.



Εικόνα 3.6: Ο package explorer με τη δομή της εφαρμογής (κώδικες, resources, κλπ)

Στο αρχείο *activity_main.xml* σχεδιάζεται το layout της εφαρμογής. Το μόνο που χρειάζεται είναι ένα αντικείμενο στο οποίο θα φορτώνεται η εκάστοτε φωτογραφία. Από τη παλέτα με τα αντικείμενα, στη καρτέλα *Design*, εισάγεται στην εικόνα της συσκευής ένα *ImageView* από την ομάδα *widgets*, η οποία και τοποθετείται στο κέντρο της οθόνης.



Εικόνα 3.7: Καρτέλα Design που δείχνει τη γραφική απεικόνιση του xml αρχείου

Από το χώρο των *Properties* (στην καρτέλα *design*) ή / και μέσω κώδικα xml (στην καρτέλα *text*) πραγματοποιείται η αλλαγή στις ιδιότητες του *image view*:

```
<ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true"
    android:src="@drawable/happykitty" />
```

Προσδιορίζονται η ιδιότητα id σε imageView1, το ύψος και το πλάτος του (layout_height, layout_width), η τοποθέτηση στο κέντρο της οθόνης της συσκευής (layout_alignParentTop, layout_centerHorizontal), καθώς και η εικόνα που θα εμφανίζεται αρχικά στο background, δηλαδή η εικόνα της γάτας (happykitty.jpg).

3.2.2 Κώδικας java της εφαρμογής MyProximitySensor

Όταν ολοκληρωθεί ο σχεδιασμός του layout της εφαρμογής, ακολουθεί ο προγραμματισμός του κύριου κώδικα, στο αρχείο *MainActivity.java*.

Κομμάτια του κώδικα της εφαρμογής παραθέτονται παρακάτω:

```
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.widget.ImageView;
```

Εδώ πραγματοποιείται τη σύνδεση με τις απαραίτητες βιβλιοθήκες. Η εργασία αυτή, όπως έχει αναφερθεί και νωρίτερα, γίνεται σχεδόν αυτόματα, αφού το Android Studio με κατάλληλο μήνυμα ειδοποιεί τη στιγμή που χρειάζεται μια συγκεκριμένη βιβλιοθήκη για να την προσθέσει.

```
public class MainActivity extends Activity implements
SensorEventListener {
    private SensorManager mSensorManager;
    private Sensor proxSensor;
    private ImageView imgView;
    private MediaPlayer meowSound, roarSound;
```

Στο κύριο activity της εφαρμογής *(MainActivity)*, εφαρμόζεται και ο Listener που έχει την δυνατότητα να επικοινωνεί με τους αισθητήρες της Android συσκευής. Μέσα στη κύρια κλάση πραγματοποιείται η προσθήκη των χαρακτηριστικών της: ενός *SensorManager*, ο οποίος μας δίνει πρόσβαση στους αισθητήρες της συσκευής, ενός Proximity Sensor (*proxSensor)* για τον αισθητήρα απόστασης, μιας εικόνα *(ImageView)*, καθώς και δύο αντικειμένων τύπου MediaPlayer (*meowSound* και *roarSound*), μέσω τον οποίων θα παίζουν οι ήχοι της γάτας και της τίγρης.

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

    imgView = (ImageView) findViewById(R.id.imageView1);

    mSensorManager = (SensorManager) getSystemService
(SENSOR_SERVICE);
    proxSensor= mSensorManager.getDefaultSensor(Sensor.
TYPE_PROXIMITY);
}
```

Στη μέθοδο *onCreate*, συνδέεται το imgView με το αντίστοιχο αντικείμενο του layout από το αρχείο activity_mail.xml. Επιπλέον, μέσω του sensorManager, ορίζεται και ο αισθητήρας απόστασης (proxSensor) που θα χρησιμοποιηθεί *(TYPE_PROXIMITY)*.

```
@Override
protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, proxSensor,
SensorManager.SENSOR_DELAY_NORMAL);
}

protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}
```

Σε αυτήν την εφαρμογή η ενεργοποίηση / απενεργοποίηση της επικοινωνίας με τον αισθητήρα προσεγγίζεται με άλλον τρόπο σε σχέση με την προηγούμενη εφαρμογή “MyLightSensor”. Εδώ γίνεται χρήση των μεθόδων *onResume* και *onPause*, οι οποίες είναι δύο από τα στάδια του κύκλου ζωής του activity.

Στην *onResume*, κατά τη διάρκεια της οποίας η εφαρμογή είναι στο προσκήνιο και εκτελείται έναντι άλλων εφαρμογών, ο *sensor manager* ανοίγει κανάλι επικοινωνίας με τον αισθητήρα απόστασης *(proxSensor)*, ορίζοντας ταυτόχρονα τη ταχύτητα που θα καταγράφει τις τιμές της απόστασης, σε normal.

Κατά τη διάρκεια της *onPause* τώρα, όπου το σύστημα έχει βάλει την εφαρμογή σε παύση, διότι πρέπει να εκτελέσει κάποια άλλη ενέργεια ή activity, μέσω του *sensor manager* απενεργοποιείται η επικοινωνία μεταξύ της εφαρμογής και του αισθητήρα *(unregisterListener)*.

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.values[0] == 0) {
        imgView.setImageResource(R.drawable.angrytiger);
        roarSound = MediaPlayer.create(this,
R.raw.tigerroar);
        roarSound.start();
    }
    else
    {
        imgView.setImageResource(R.drawable.happykitty);
        meowSound = MediaPlayer.create(this,
R.raw.catmeow);
        meowSound.start();
    }
}
```

Η μέθοδος *onSersorChanged* καλείται κάθε φορά που συμβαίνουν αλλαγές στις τιμές που διαβάζει ο αισθητήρας. Συγκεκριμένα εδώ, ελέγχεται με μια εντολή if η τιμή αυτή, και εάν η τιμή είναι 0, δηλαδή εάν έχει πλησιάσει πολύ ένα αντικείμενο στη συσκευή, τότε μέσα από το φάκελο drawable στα resources, φορτώνει στο imageView την εικόνα της τίγρης (angrytiger.jpeg). Ταυτόχρονα μέσω του αντικειμένου roarSound που είναι τύπου MediaPlayer, παίζει ο βρυχηθμός της τίγρης (tigerroar.mp3), που έχει φορτωθεί από το φάκελο raw μέσα στα resources. Εάν η τιμή που διάβασε είναι μεγάλη, δηλαδή δεν υπάρχει αντικείμενο κοντά στη συσκευή, τότε φορτώνει στο imageView την εικόνα της γάτας (happykitty.jpg) και μέσω του MediaPlayer αντικειμένου meowSound παίζει το νιαούρισμα της γάτας (catmeow.mp3).

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) { }
```

Η μέθοδος *onAccuracyChanged*, όπου καλείται όταν συμβαίνουν αλλαγές στην ακρίβεια του αισθητήρα, δεν χρειάζεται οπότε δεν προγραμματίζεται κάποιος κώδικας εκεί.

```
@Override
protected void onDestroy() {
    super.onDestroy();

    roarSound.stop();
    roarSound.release();
    meowSound.stop();
    meowSound.release();
}
```

Τέλος, η μέθοδος *onDestroy*, που αποτελεί κι αυτή μέρος του κύκλου ζωής του activity, καλείται όταν έχει τελειώσει το activity και το σύστημα θέλει να καταστρέψει όλες τα threads και τις ενέργειες που εκτελούνται από αυτό, για να απελευθερώσει χώρο στη μνήμη. Όντως, βλέπουμε ότι εκτελούνται εντολές οι οποίες σταματούν τα δύο αντικείμενα MediaPlayer και πραγματοποιείται απελευθέρωση (release) της μνήμης.

Ολοκληρωμένος ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα, σελ. 91.

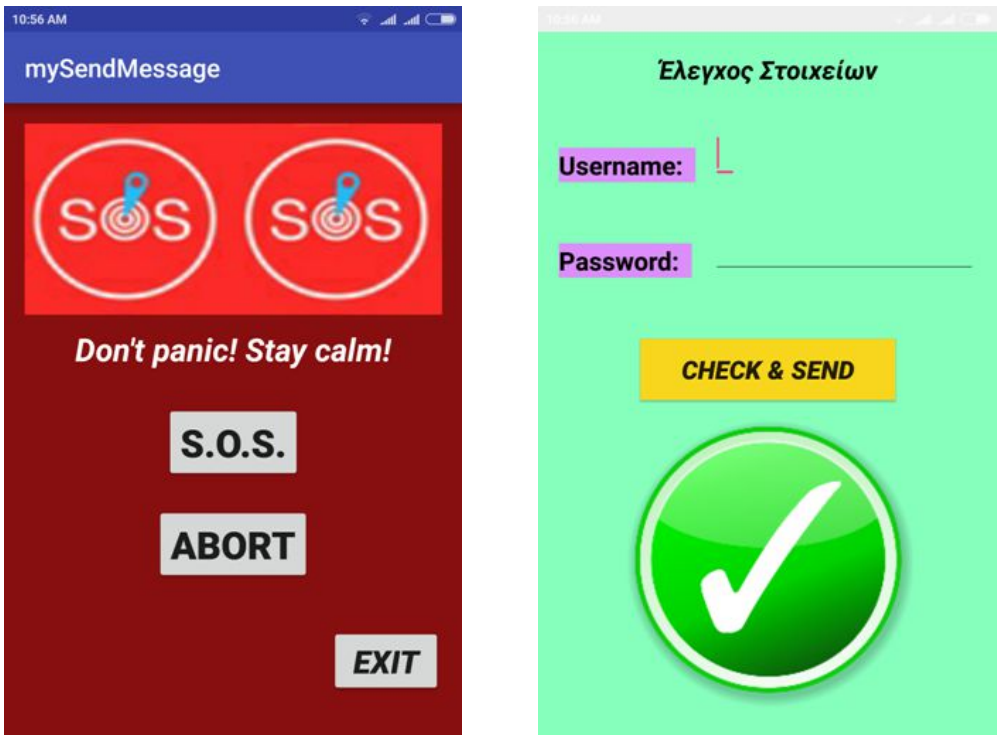
ΚΕΦΑΛΑΙΟ 4 – Χρήση GPS και αποστολή μηνύματος

ΕΙΣΑΓΩΓΗ

Στο παρόν παράδειγμα θα κατασκευαστεί μια πιο σύνθετη εφαρμογή, η οποία θα έχει τη δυνατότητα άμεσης ειδοποίησης σε περιστάσεις κινδύνου ή σοβαρής ανάγκης. Συγκεκριμένα, στην υποθετική περίσταση όπου ο χρήστης θα βρεθεί σε μια κατάσταση εκτάκτου ανάγκης, η εφαρμογή θα επικοινωνεί με το GPS της συσκευής ζητώντας τις συντεταγμένες του σημείου που βρίσκεται, και θα τις στέλνει μέσω γραπτού μηνύματος σε κάποιον γνωστό παραλήπτη, με το εξής περιεχόμενο: “Παρακαλώ βοήθεια! Είμαι στην τοποθεσία με γεωγραφικό μήκος: … και πλάτος: …”.

Στην περίπτωση τώρα, που όλα είναι καλά ή έχει περάσει ο κίνδυνος, ο χρήστης μέσω ενός δεύτερου activity, θα εισάγει όνομα χρήστη και κωδικό πρόσβασης, και εφόσον επαληθευτούν η εφαρμογή θα στέλνει ένα δεύτερο μήνυμα με περιεχόμενο: “Άκυρος ο συναγερμός. Είμαι εντάξει.”. Η επαλήθευση είναι αναγκαία για να υπάρχει βεβαιότητα ότι το μήνυμα το έστειλε ο ιδιοκτήτης του κινητού και όχι κάποιο τρίτο πρόσωπο.

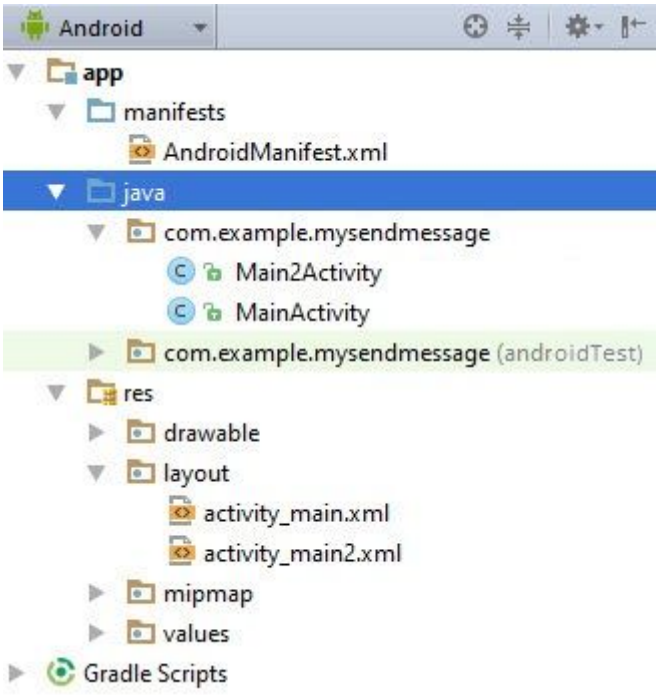
Σημειώνεται, ότι η αποστολή των μηνυμάτων δεν είναι δωρεάν, αλλά υπόκειται στις τιμές χρέωσης του παρόχου κινητής τηλεφωνίας που χρησιμοποιείται.



Εικόνες 4.1, 4.2: Στιγμιότυπα από την εφαρμογή MySendMessage

Δημιουργείται ένα νέο project με ονομασία *MySendMessage*. Προαιρετικά, εάν θέλουμε να υπάρχουν και εικόνες στην εφαρμογή, προσθέτουμε δύο εικόνες μέσα στον φάκελο *drawable (sos.jpg* και *check.png)*.

Επιπλέον, όπως αναφέρθηκε νωρίτερα, θα κατασκευαστούν δύο activity: το πρώτο που είναι το κύριο, και ένα δεύτερο όπου ο χρήστης θα εισάγει το username και το password του. Το δεύτερο activity λοιπόν, θα δημιουργηθεί με δεξί κλικ πάνω στο φάκελο *java → New → Activity → Empty Activity → Activity Name: Main2Activity*. Έτσι θα έχουμε το αποτέλεσμα που φαίνεται στην παρακάτω εικόνα:



Εικόνα 4.3: Ο package explorer με τη δομή της εφαρμογής (κώδικες, resources, κλπ)

4.1 Manifest της εφαρμογής MySendMessage

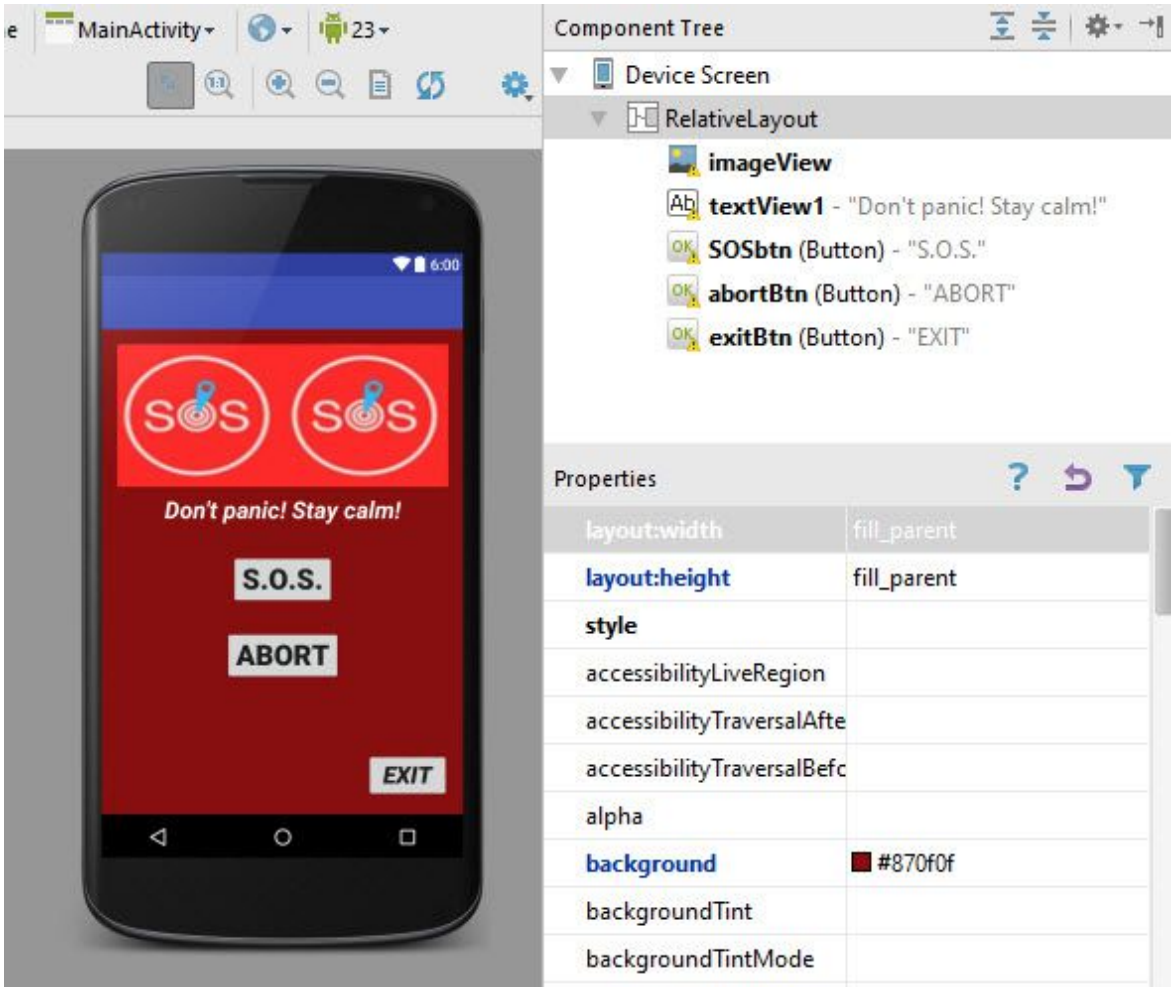
Σε αυτήν την εφαρμογή θα χρειαστεί επέμβαση και στον κώδικα του manifest (στον package explorer μέσα στον φάκελο *manifests*, το αρχείο *AndroidManifest.xml*). Συμπληρώνονται οι παρακάτω τρεις εντολές που βρίσκονται ανάμεσα στα σχόλια.

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

<!-- Ορισμός των απαραίτητων δικαιωμάτων πρόσβασης σε υπηρεσίες
του τηλεφώνου -->
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.SEND_SMS" />

<!-- Υπόλοιπος κώδικας -->
```

Η προσθήκη αυτών των εντολών θα δώσει στην εφαρμογή τα απαραίτητα δικαιώματα πρόσβασης προς την υπηρεσία GPS της συσκευής, καθώς και προς την υπηρεσία αποστολής μηνυμάτων. Χωρίς αυτές τις εντολές στο manifest, η συσκευή δεν θα επέτρεπε στην εφαρμογή να έχει πρόσβαση στο GPS, ούτε στα μηνύματα.

4.2 Layout της κεντρικής οθόνης (κύριο activity) – αρχείο xml της εφαρμογής MySendMessage

Ακολουθεί το layout της εφαρμογής. Αρχικά θα γραφτεί κώδικας για το αρχείο xml του κύριου activity *(activity_main.xml)*. Η κεντρική οθόνη θα αποτελείται από ένα *ImageView*, ένα *Plain TextView* και τρία *Button*, όλα τα οποία εισάγονται στην καρτέλα *Design*, από την ομάδα *widgets* στην παλέτα με τα αντικείμενα.



Εικόνα 4.4: Καρτέλα Design που δείχνει τη γραφική απεικόνιση του xml αρχείου

Μέσα από το χώρο των *Properties* (στην καρτέλα *Design*) ή / και μέσω κώδικα xml (στην καρτέλα *Text*) προγραμματίζονται οι αλλαγές στις ιδιότητες των παραπάνω αντικειμένων που εισήχθησαν. Κομμάτια του κώδικα xml φαίνονται παρακάτω:

```
<ImageView
    android:layout_width="350dp"
    android:layout_height="150dp"
    android:id="@+id/imageView"
    android:background="@drawable/sos"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance=
"?android:attr/textAppearanceLarge"
    android:text="Don&apos;t panic! Stay calm!"
    android:id="@+id/textView1"
    android:textSize="25dp"
    android:textStyle="bold|italic"
    android:textColor="#ffffff"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="160dp" />
```

Η τοποθέτηση των *ImageView* και *TextView* είναι προαιρετική, δεν έχουν κάποια συγκεκριμένη λειτουργία για την εφαρμογή, απλώς για ομορφιά. Στο ImageView φορτώνεται η εικόνα S.O.S. (sos.jpg) από τον φάκελο drawable και στο TextView γράφεται το μήνυμα “Don’t panic! Stay calm!”. Ορίζονται τα id τους, τα ύψη και τα πλάτη, η τοποθέτηση και στοίχισή τους στην οθόνη της συσκευής, καθώς και η μορφή της γραμματοσειράς.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="S.O.S."
    android:id="@+id/SOSbtn"
    android:textSize="30dp"
    android:textStyle="bold"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="220dp" />
```

Το πρώτο *Button* είναι για την αποστολή του μηνύματος που θα ζητάει ο χρήστης βοήθεια. Ορίζονται το id του, ένα κείμενο πάνω του που θα γράφει “S.O.S”, το ύψος και πλάτος του και η σχετική τοποθέτησή του. Σημαντικό είναι να σημειωθεί ότι δεν έχει οριστεί η ιδιότητα onClick, όπως στις προηγούμενες εφαρμογές. Θα παρουσιαστεί στον κώδικα java της εφαρμογής *(MainActivity.java)* ένας άλλος τρόπος που έχει δυνατότητα το Android Studio να αναγνωρίσει το event του πατήματος ενός κουμπιού.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="ABORT"
    android:id="@+id/abortBtn"
    android:textSize="30dp"
    android:textStyle="bold"
    android:onClick="checkAbort"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="300dp" />
```

Στη συνέχεια περιγράφεται το *Button* το οποίο θα μεταφέρει το χρήστη στο δεύτερο activity, ώστε να ελέγξει τα στοιχεία του και να στείλει το μήνυμα ότι όλα είναι καλά, άκυρος ο συναγερμός. Θέτονται το id του, το κείμενο “ABORT” που θα αναγράφεται πάνω του, το ύψος, το πλάτος και η τοποθέτησή του. Ορίζεται και η ιδιότητα onClick (checkAbort), ένα event που θα εκτελέσει τις εντολές μεταφοράς στο δεύτερο activity.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="EXIT"
    android:id="@+id/exitBtn"
    android:textSize="25dp"
    android:textStyle="bold|italic"
    android:onClick="exitApp"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
```

Τελειώνοντας το layout του κύριου activity, με το *Button* για την έξοδο από την εφαρμογή. Ορίζονται το id του, ένα κείμενο “EXIT”, το ύψος, το πλάτος, η τοποθέτησή του, καθώς και η ιδιότητα onClick (exitApp), το event για το κλείσιμο της εφαρμογής.

4.3 Κώδικας java του κύριου activity της εφαρμογής MySendMessage

Αφού ολοκληρωθεί ο σχεδιασμός του layout της κεντρικής οθόνης, ακολουθεί η υλοποίηση του κώδικα του κύριου activity, στο αρχείο *MainActivity.java*.

Κομμάτια του κώδικα παραθέτονται παρακάτω:

```
import android.Manifest;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import static java.lang.System.exit;
```

Αρχικά πραγματοποιείται η σύνδεση με τις απαραίτητες βιβλιοθήκες. Στην εφαρμογή αυτή, χρειάζονται συν τις άλλοις και οι βιβλιοθήκες για την εύρεση της τοποθεσίας που βρίσκεται η συσκευή κάθε στιγμή (android.location), καθώς και η βιβλιοθήκη για την υπηρεσία αποστολής μηνύματος (android.telephony. SmsManager).

```
public class MainActivity extends AppCompatActivity
implements LocationListener {

    Button sendBtn, abortBtn, exitBnt;
    String phoneNo = "69......";  // αριθμός κινητού
    String msgSOS;
    String toastMsg;

    static String longitude;
    static String latitude;
    LocationManager locManager;
```

Παραπάνω φαίνεται το πρώτο κύριο activity της εφαρμογής (*MainActivity)*, στο οποίο εφαρμόζονται και οι μέθοδοι της υπηρεσίας εντοπισμού θέσης *(implements LocationListener)*.

Ορίζονται τα χαρακτηριστικά της κλάσης: τρία κουμπιά *(Button)*, τρία *String*: ένα για τον αριθμό κινητού που θα σταλούν τα μηνύματα (phoneNo), ένα για το μήνυμα (msgSOS) και ένα για την αναφορά της αποστολής (toastMsg). Επίσης, ορίζονται και τα στοιχεία που θα χρειαστούν για την υπηρεσία GPS: δύο *String* (longitude, latitude) όπου θα αποθηκεύονται το γεωγραφικό μήκος και πλάτος, και

ένας LocationManager ο οποίος είναι υπεύθυνος για την επικοινωνία της εφαρμογής με το GPS της συσκευής.

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

     sendBtn = (Button) findViewById(R.id.SOSbtn);
    abortBtn = (Button) findViewById(R.id.abortBtn);
    exitBnt = (Button) findViewById(R.id.exitBtn);
    locManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
```

Στην αρχή της μεθόδου *onCreate*, συνδέονται τα χαρακτηριστικά με τα αντίστοιχα αντικείμενα του layout από το αρχείο activity_main.xml, καθώς και με την υπηρεσία εντοπισμού θέσης.

```
if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
!= PackageManager.PERMISSION_GRANTED
&& ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION)
!= PackageManager.PERMISSION_GRANTED) {

     locManager.requestLocationUpdates(LocationManager.
GPS_PROVIDER, 0, 0, this);
    return;
}
else locManager.requestLocationUpdates(LocationManager.
GPS_PROVIDER, 0, 0, this);
```

Συνεχίζοντας στην μέθοδο *onCreate*, γίνονται οι απαραίτητοι έλεγχοι για το κατά πόσο το GPS είναι ενεργοποιημένο και η εφαρμογή έχει τα σχετικά δικαιώματα πρόσβασης σε αυτό. Εφόσον όλα είναι έγκυρα, πραγματοποιείται η μεταφορά των δεδομένων, δηλαδή τις τιμές που έχει διαβάσει το GPS, στον Location Manager.

```
sendBtn.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {

        if (longitude != null) {
            msgSOS = "Παρακαλώ βοήθεια! Βρίσκομαι στην
τοποθεσία με γεωγραφικό μήκος: " + longitude + "και πλάτος: "
+ latitude;
            toastMsg = "Μήνυμα εστάλη!!";
        }
        else {
            msgSOS = "Παρακαλώ βοήθεια!";
            toastMsg = "Μήνυμα εστάλη χωρίς την τοποθεσία.";
        }

        sendMsgSOS(phoneNo, msgSOS);
    }
});
}
```

Σ' αυτό το κομμάτι της *onCreate*, υλοποιείται ο άλλος τρόπος που μπορεί το Android Studio να καταλάβει την ενέργεια του πατήματος ενός κουμπιού, χωρίς να χρειαστεί να οριστεί στην ιδιότητα onClick του κουμπιού (SOSbtn) στο layout αρχείο (activity_mail.xml). Χρησιμοποιείται ένας ClickListener, ο οποίος έχει τη δυνατότητα να "ακούει" – ανιχνεύει τα πατήματα στο κουμπί.

Ορίζεται λοιπόν, ένας ClickListener στο στοιχείο sendBtn, το οποίο παραπάνω στον κώδικα έχει συνδεθεί με το κουμπί SOSbtn, ώστε να εντοπίσει το event του click και να τρέξει την μέθοδο onClick.

Μέσα στην μέθοδο *onClick*, ελέγχεται (εντολή if) κατά πόσο η μεταβλητή longitude έχει κάποια τιμή, δηλαδή εάν το GPS λειτούργησε κανονικά, έδωσε στίγμα και έτσι αποθηκεύτηκαν οι τιμές (γεωγραφικό μήκος και πλάτος) που διάβασε στις μεταβλητές longitude και latitude. Εάν ισχύει αυτό, τότε δημιουργείται το μήνυμα msgSOS με τις συντεταγμένες που ελήφθησαν από το GPS, καθώς και το μήνυμα της αναφοράς για την αποστολή του μηνύματος (toastMsg).

Εάν δεν ισχύει, δηλαδή το GPS δεν έδωσε στίγμα, και άρα οι μεταβλητές longitude και latitude δεν έχουν πάρει τιμή, τότε δημιουργείται ένα απλοποιημένο μήνυμα msgSOS χωρίς συντεταγμένες, μιας και δεν ελήφθη κάτι από το GPS, και το μήνυμα αναφοράς (toastMsg) αναφέρει ότι εστάλη το μήνυμα, αλλά χωρίς το σημείο της τοποθεσίας.

Τέλος, καλείται η μέθοδος για την αποστολή μηνύματος *sendMsgSOS*, η οποία θα αναλυθεί αμέσως παρακάτω, με παραμέτρους το τηλέφωνο όπου θα σταλεί το μήνυμα για βοήθεια και το ίδιο το μήνυμα.
Εδώ, ολοκληρώνεται η μέθοδος onCreate.

```
private void sendMsgSOS(String phoneNo, String msgSOS)
{
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(phoneNo, null, msgSOS, null, null);
    Toast.makeText(getBaseContext(), toastMsg,
Toast.LENGTH_LONG).show();
    }
```

Η μέθοδος *sendMsgSOS* είναι υπεύθυνη για την αποστολή μηνυμάτων, εφόσον έχουν δοθεί τα απαραίτητα δικαιώματα πρόσβασης για την αποστολή. Ως παραμέτρους έχει το μήνυμα που θα στείλει (msgSOS) και τον αριθμό τηλεφώνου στο οποίο θα το στείλει (phoneNo). Τέλος, εμφανίζεται και το μήνυμα αναφοράς για το εάν έχουν ή δεν έχουν σταλεί και οι συντεταγμένες του σημείου.

```
@Override
public void onLocationChanged(Location location) {
    longitude = String.valueOf(location.getLongitude());
    latitude = String.valueOf(location.getLatitude());
}
```

Η παραπάνω μέθοδος είναι μία από τις τέσσερις μεθόδους (onLocation Changed, onStatusChanged, onProviderEnabled, onProviderDisabled), που διαχειρίζονται την υπηρεσία εντοπισμού θέσης της συσκευής Android.

Στην συγκεκριμένη *onLocationChanged*, αποθηκεύονται στις μεταβλητές longitude και latitude, οι συντεταγμένες κάθε φορά που εντοπίζει αλλαγή θέσης το GPS. Σε αυτήν την εφαρμογή δεν χρειάζεται να γραφτεί κώδικας για τις υπόλοιπες τρεις μεθόδους.

```
public void checkAbort(View view) {
    Intent intent = new Intent(this, Main2Activity.class);
    startActivity(intent);
}
```

Αυτή η μέθοδος συνδέεται με το button *Abort*, αντικείμενο του layout από το αρχείο activity_mail.xml. Όταν πατηθεί λοιπόν το κουμπί, δημιουργείται ένα Intent

με τις δύο βασικές κλάσεις: την MainActivity.class που εστιάζει η εφαρμογή εκείνη τη στιγμή και την Main2Activity.class. Στην συνέχεια, ενεργοποιείται το intent και πραγματοποιείται μεταφορά στο νέο Activity (Main2Activity.class), όπου θα δοθεί στο χρήστη η δυνατότητα να στείλει μήνυμα ακύρωσης του συναγερμού. Ο κώδικάς του νέου Activity θα αναλυθεί παρακάτω.

```
public void exitApp(View view) {
    if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED
                && ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION)
                != PackageManager.PERMISSION_GRANTED)
            locManager.removeUpdates(this);
    else locManager.removeUpdates(this);
    exit(0);
} }
```

Τέλος, η μέθοδος *exitApp*, συνδέεται με το button Exit, και μέσω αυτής υλοποιείται η έξοδος από την εφαρμογή, αφού πρώτα απενεργοποιηθεί η υπηρεσία εντοπισμού θέσης – GPS.

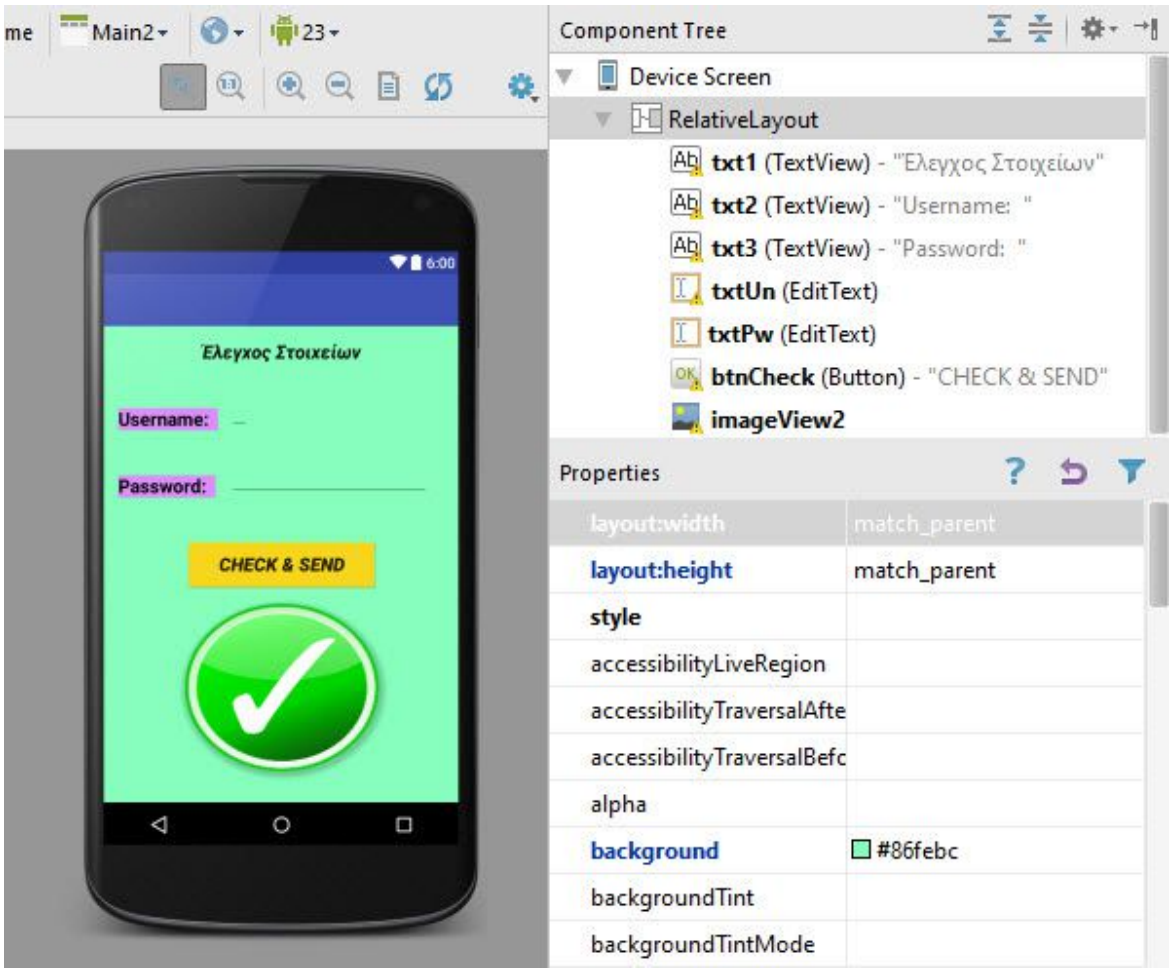
Εδώ, ολοκληρώνεται το πρώτο activity, όπου η εφαρμογή επικοινωνεί με το GPS για να λάβει τις συντεταγμένες του σημείου που βρίσκεται η Android συσκευή και έπειτα τις αποστέλλει μέσω γραπτού μηνύματος, σαν μήνυμα SOS.

Στην συνέχεια, θα παρουσιαστεί το δεύτερο activity, όπου εφόσον ο χρήστης κάνει login και δώσει τα σωστά στοιχεία, αποστέλλεται ένα μήνυμα ακύρωσης του μηνύματος SOS. Το username και το password είναι αναγκαία για να αποκλειστεί το ενδεχόμενο αποστολής του μηνύματος ακύρωσης από κάποιο τρίτο και κακόβουλο άτομο. Το δεύτερο αυτό activity θα εμφανιστεί όταν πατηθεί το κουμπί *Abort*, με την μέθοδο *checkAbort* που αναλύθηκε παραπάνω.

4.4 Layout της δεύτερης οθόνης (δεύτερο activity) – αρχείο xml της εφαρμογής MySendMessage

Αρχικά υλοποιείται πρώτα το αρχείο xml *(activity_main2.xml)*. Η οθόνη θα περιέχει τρία *Plain TextView*, ένα *Button* και ένα *ImageView* από την ομάδα

widgets, καθώς και ένα *Plain Text* και ένα *Password* από την ομάδα *text fields*, στην καρτέλα *Design*.



Εικόνα 4.5: Καρτέλα Design που δείχνει τη γραφική απεικόνιση του xml αρχείου

Από τα *Properties* (στην καρτέλα *Design*) ή / και μέσω κώδικα xml (στην καρτέλα *Text*) μεταβάλλονται οι ιδιότητες των αντικειμένων. Κομμάτια του κώδικα xml φαίνονται παρακάτω:

```
<TextView android:text="Ελεγχος Στοιχείων"
android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#000000"
    android:textSize="20dp"
    android:id="@+id/txt1"
    android:layout_centerHorizontal="true"
    android:textStyle="bold|italic" />
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Username:  "
    android:id="@+id/txt2"
    android:textSize="20dp"
    android:textColor="#000000"
    android:background="#dc8efa"
    android:textStyle="bold"
    android:layout_below="@+id/txt1"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="48dp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Password:  "
    android:id="@+id/txt3"
    android:textSize="20dp"
    android:layout_marginTop="48dp"
    android:textColor="#000000"
    android:background="#dc8efa"
    android:textStyle="bold"
    android:layout_below="@+id/txt2"
    android:layout_alignParentLeft="true"
    android:layout alignParentStart="true" />
```

Τα τρία *TextView* είναι για ενημέρωση του χρήστη ότι χρειάζεται να επαληθευτεί η ταυτότητά του με την εισαγωγή του ονόματος χρήστη και του κωδικού πρόσβασης. Ορίζονται λοιπόν τα id τους, τα ύψη τα πλάτη, η σχετική τοποθέτηση τους στην οθόνη της συσκευής, καθώς και η μορφή της γραμματοσειράς τους. Στο πρώτο *TextView* θέτεται η ιδιότητα text ως "Ελεγχος Στοιχείων" , στο δεύτερο ως "Username:" και στο τρίτο ως "Password:".

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/txtUn"
    android:layout_alignBottom="@+id/txt2"
    android:layout_marginLeft="120dp"
    android:layout_marginStart="120dp"/>
```

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="textPassword"
    android:ems="10"
    android:id="@+id/txtPw"
    android:layout_alignBottom="@+id/txt3"
    android:layout_alignLeft="@+id/txtUn"
    android:layout_alignStart="@+id/txtUn" />
```

Τα δύο επόμενα αντικείμενα είναι τύπου *EditText (*ένα *Plain* και ένα *Password)*, στα οποία ο χρήστης έχει την δυνατότητα να πληκτρολογήσει κείμενο μέσα σε αυτά. Το πρώτο (*Plain Text)*, με id "txtUn", είναι για την εισαγωγή του ονόματος χρήστη. Το δεύτερο, με id "txtPw", είναι για την εισαγωγή του κωδικού πρόσβασης και είναι τύπου *Password*, δηλαδή εμφανίζονται αστεράκια αντί των γραμμάτων που πληκτρολογεί ο χρήστης, ώστε να αποκρύβεται ο κωδικός.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="CHECK &amp; SEND"
    android:id="@+id/btnCheck"
    android:layout_below="@+id/txt3"
    android:layout_marginTop="48dp"
    android:textStyle="bold|italic"
    android:background="#f7d61d"
    android:textSize="20dp"
    android:layout_centerHorizontal="true"
    android:width="200dp" />
```

Οι ενέργειες που θα πραγματοποιήσει το παραπάνω *Button*, μέσω μιας μεθόδου στο αρχείο *Main2Activity.java* είναι να ελέγξει την ορθότητα των στοιχείων που πληκτρολόγησε ο χρήστης, και εάν είναι σωστά τότε να στείλει ένα μήνυμα. Ορίζονται το id, το κείμενο "CHECK & SEND" που θα αναγράφεται πάνω του, το ύψος και πλάτος του και η σχετική τοποθέτησή του.

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView2"
    android:layout_below="@+id/btnCheck"
    android:layout_centerHorizontal="true"
    android:background="@drawable/check" />
```

Τέλος το προαιρετικό *ImageView*, όπου στην ιδιότητα background έχει φορτωθεί η εικόνα S.O.S. (check.png) από τον φάκελο drawable στο res.

4.5 Κώδικας java του δεύτερου activity της εφαρμογής MySendMessage

Ολοκληρώνοντας τον σχεδιασμό του layout της δεύτερης οθόνης, ακολουθεί ο κώδικας του activity, στο αρχείο *Main2Activity.java*:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
```

Υλοποιείται η σύνδεση με τις απαραίτητες βιβλιοθήκες που θα χρειαστεί το activity.

```
public class Main2Activity extends AppCompatActivity {
    Button checkBtn;
    String phoneNo = "69......";
    String msgAbort = "Ακυρος ο συναγερμός. Είμαι εντάξει.";
```

Ορίζονται τα χαρακτηριστικά της κλάσης: ένα κουμπί *(Button)* και δύο *String*: ένα για τον αριθμό κινητού που θα σταλεί το μήνυμα (phoneNo) και ένα για το μήνυμα ακύρωσης της έκτακτης ανάγκης (msgAbort).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);

    checkBtn = (Button) findViewById(R.id.btnCheck);

    checkBtn.setOnClickListener(new View.OnClickListener() {
```

Στην αρχή της μεθόδου *onCreate* συνδέεται το button checkBtn με το αντίστοιχο button (Check & Send) του layout από το αρχείο activity_main2.xml. Στη συνέχεια, ορίζεται ένας ClickListener στο κουμπί checkBtn, έτσι ώστε να εντοπίσει το event του click και να τρέξει την μέθοδο *onClick*.

```
public void onClick(View view) {
    EditText username = (EditText) findViewById(R.id.txtUn);
    EditText password = (EditText) findViewById(R.id.txtPw);

    if (username.getText().toString().equals("m") &&
password.getText().toString().equals("k")) {
        sendMsgAbort(phoneNo, msgAbort);
    }
    else {
        Toast.makeText(getBaseContext(), "Δώσατε λάθος
username ή/και password.", Toast.LENGTH_LONG).show();
    }
} }); }
```

Στη μέθοδο *onClick* αρχικά ορίζονται και συνδέονται δύο αντικείμενα EditText *(username, password)* με τα αντίστοιχα στοιχεία *txtUn* και *txtPw* του layout από το αρχείο *activity_main2.xml*.

Στη συνέχεια με μια εντολή if, συγκρίνονται τα username και password που έχει πληκτρολογήσει ο χρήστης, με αυτά που έχουν οριστεί hardcoded στον κώδικα της εφαρμογής, δηλαδή το “m” σαν username και το “k” σαν password. Εφόσον είναι ίδια, τότε ταυτοποιείται ο χρήστης και εκτελείται η μέθοδος *sendMsgAbort* όπου στέλνει το μήνυμα ακύρωσης (ανάλυση παρακάτω). Εάν δεν έχει δοθεί ο σωστός συνδυασμός username / password, σημαίνει ότι πιθανόν η συσκευή Android έχει επέλθει στην κατοχή άλλου προσώπου, το οποίο προσπαθεί ψευδώς να στείλει το μήνυμα ότι δεν υπάρχει λόγος ανησυχίας. Έτσι δεν ταυτοποιείται ο χρήστης, το μήνυμα δεν αποστέλλεται και εμφανίζεται προειδοποίηση εισαγωγής λανθασμένου username ή/και password.

Ολοκληρώνεται η μέθοδος *onClick* καθώς και η *onCreate*.

```
private void sendMsgAbort(String phoneNo, String msgAbort) {
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(phoneNo, null, msgAbort, null, null);

    Toast.makeText(getBaseContext(), "Μήνυμα εστάλη!!",
Toast.LENGTH_LONG).show();
} }
```

Τελευταίο κομμάτι κώδικα της εφαρμογής η μέθοδος *sendMsgAbort*, η οποία λειτουργεί ακριβώς όπως και η μέθοδος sendMsgSOS που αναλύθηκε στο πρώτο activity. Έχει ως παραμέτρους το μήνυμα που θα στείλει (msgAbort), δηλαδή ότι όλα βαίνουν καλώς και ότι ο συναγερμός είναι άκυρος, καθώς και τον αριθμό τηλεφώνου στο οποίο θα το στείλει (phoneNo), ο οποίος λογικά θα είναι ο ίδιος. Επιπλέον, εμφανίζεται και η ειδοποίηση για την επιτυχημένη αποστολή.

Ολοκληρωμένος ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα, σελ. 93.

ΚΕΦΑΛΑΙΟ 5 – Το Video Game «Χαλασμένο UFO»

ΕΙΣΑΓΩΓΗ

Έχοντας κατά νου την ιδιαιτερότητα της πολυεπίπεδης πλατφόρμας Android, η ανάπτυξη μιας εφαρμογής παιχνιδιού αποτελεί μια πρόκληση για αρκετούς λόγους. Μερικοί από αυτούς είναι η ανάπτυξη σε υψηλότερο επίπεδο ανεξάρτητα από το υλικό και η έμφαση που δίνεται στις προγραμματιστικές τεχνικές υλοποίησης της εφαρμογής.

Το συγκεκριμένο παιχνίδι αποτελεί μια κλασσική ανάπτυξη των τυπικών όπως ονομάζονται “2d roll platform games”, στα οποία οι παίχτες καθοδηγούνται μέσα από μια κυλιόμενη δισδιάστατη απεικόνιση της οθόνης, προσπαθώντας να αποφύγουν όσο το δυνατό περισσότερους εχθρούς και εμπόδια, ώστε να επιτύχουν το μεγαλύτερο δυνατό σκορ. Η επιλογή αυτού του τύπου του παιχνιδιού έγινε επειδή επιτρέπει την εύκολη ανάπτυξη του κώδικα, τη δυνατότητα εκτέλεσης σε πολύ μεγάλη γκάμα συσκευών, την ενσωμάτωση και τη περιγραφή μιας σειράς από θεμελιώδεις τεχνικές στην ανάπτυξη εφαρμογών και τέλος την ευχάριστη και με δημιουργικό τρόπο εμβάθυνση στην πλατφόρμα Android.

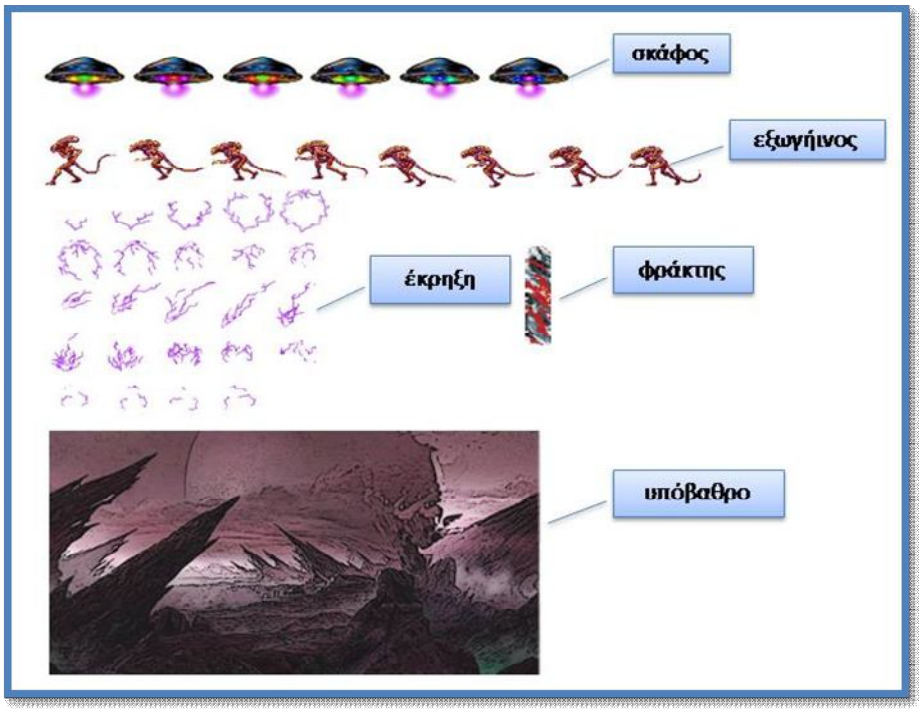
Παρόλα αυτά, στην περίπτωση που θεωρηθεί ιδιαίτερα απαιτητική και δυσνόητη για να την υλοποιήσουν οι μαθητές από την αρχή μόνοι τους, πιο χρήσιμο θα ήταν να διδαχθεί με εναλλακτικό τρόπο: π.χ. δίνοντας έτοιμα ορισμένα ελλιπή κομμάτια κώδικα για να τα συμπληρώσουν με εντολές που θα τους είναι πιο ευκολονόητες, ή ζητώντας να μεταβάλλουν κάποιες τιμές στις παραμέτρους ώστε να δουν τις διαφορές στο παιχνίδι. Επίσης, υπάρχουν ορισμένες παρόμοιες κλάσεις στην εφαρμογή, οπότε θα μπορούσαν οι μαθητές μελετώντας μία από αυτές, να προγραμματίσουν μόνοι τους τις άλλες.

Ακολουθεί λοιπόν η παρουσίαση της μεθοδολογίας ανάπτυξης της εφαρμογής, δίνοντας ιδιαίτερη έμφαση στα βασικά μέρη του κώδικα. Τα επιπλέον τμήματα του κώδικα που δεν θεωρηθήκαν αναγκαία, διότι ήταν παρόμοια με προηγούμενα, ενσωματώθηκαν στο Παράρτημα (σελ. 99) που επισυνάπτεται στο τέλος της εργασίας.

5.1 Η εφαρμογή GameApp – Οργανώνοντας τα αρχεία

Ξεκινώντας, δημιουργούμε το έργο project με ονομασία *"GameApp"*. Για τις ανάγκες της εφαρμογής μέσα στο υποφάκελο *res* δημιουργούμε ένα νέο υποφάκελο *drawable-nodpi*. Ο συγκεκριμένος υποφάκελος είναι αναγκαίος όταν χρειάζονται εικόνες για τα γραφικά του παιχνιδιού, ανεξάρτητα από το μέγεθος της ανάλυσης της οθόνης της συσκευής που θα χρησιμοποιηθεί. Αυτά τα αρχεία εικόνας είναι μια σειρά από *spriteSheets*, όπως ονομάζονται συνήθως, και περιέχουν διαφορετικά στιγμιότυπα της μορφής των γραφικών που θα χρησιμοποιηθούν για την οπτική απεικόνιση των αντικειμένων που εμφανίζονται στο παιχνίδι (σκάφος, εξωγήινος, έκρηξη).

Επιπλέον, εδώ περιέχονται και οι εικόνες για το υπόβαθρο (φόντο) της εφαρμογής, καθώς και για το φράκτη που θα σχεδιάζεται πάνω και κάτω στην οθόνη.



Εικόνα 5.1: Τα γραφικά του παιχνιδιού

Επίσης στο φάκελο *raw* πρέπει να αποθηκευτούν τα αρχεία που θα χρησιμοποιηθούν για τον ήχο υπόβαθρου και τον ήχο εφέ για την σύγκρουση του σκάφους (bg.mp3, crash.wav).

5.2 Η κλάση Game – Ξεκινώντας τον Κώδικα

Ξεκινώντας την περιγραφή της ανάπτυξης του κώδικα, είναι αναγκαίο να γίνει αναφορά στο κεντρικό *Activity* της εφαρμογής, δηλαδή το δομικό στοιχείο του Android από το οποίο ξεκινάνε όλες οι εφαρμογές της πλατφόρμας. Όμως, δεν θα ακολουθηθεί η ανάπτυξη μιας κλασσικής εφαρμογής που στηρίζεται στον σχεδιασμό ενός ή περισσοτέρων οθονών με την μεθοδολογία του οπτικού προγραμματισμού: εισάγοντας δηλαδή από μια εργαλειοθήκη αντικείμενα (πλαίσια κειμένου, ετικέτες, εικόνες, κ.λπ.), τα οποία αποθηκεύονται σε αρχεία xml και σε δεύτερο στάδιο γράφεται κώδικας πάνω σε αυτές τις σχεδιασμένες οθόνες (layouts). Σ' αυτή την εφαρμογή, η εισαγωγή και ο σχεδιασμός των αντικειμένων θα γίνει προγραμματιστικά στην επόμενη κλάση GamePanel, απλώς στέλνοντάς της ως περιεχόμενο αυτό το Activity.

```
public class Game extends Activity {
    MediaPlayer backgroundMusic;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       requestWindowFeature(Window.FEATURE_NO_TITLE);
       getWindow().setFlags(
                  WindowManager.LayoutParams.FLAG_FULLSCREEN,
                 WindowManager.LayoutParams.FLAG_FULLSCREEN);

       backgroundMusic = MediaPlayer.create(Game.this,
                                                R.raw.bg);
       backgroundMusic.setLooping(true);
       backgroundMusic.setVolume(0.1f, 0.1f);
       backgroundMusic.start();

       setContentView(new GamePanel(this));
    }
    @Override
    protected void onDestroy() {
       super.onDestroy();
       if(backgroundMusic!=null) {
            backgroundMusic.stop();
            backgroundMusic.release();  }
    }
}
```

Αρχικοποιείται λοιπόν, η οθόνη σε πλήρης και απενεργοποιείται η γραμμή τίτλου (μέθοδος *onCreate*). Επίσης, χρησιμοποιώντας ένα στιγμιότυπο της κλάσης MediaPlayer, δίνεται η δυνατότητα της εισαγωγής και εκτέλεσης στο υπόβαθρο του αρχείου, ενός ήχου bg.mp3, για όσο λειτουργεί η εφαρμογή. Τέλος,

στη μέθοδο *onDestroy* όπου παύει η λειτουργία της εφαρμογής, απελευθερώνεται η μνήμη από τον mediaplayer.

5.3 Η κλάση GamePanel – Η εξέλιξη του παιχνιδιού (GamePlay)

Η κλάση *GamePanel* περιλαμβάνει την επιφάνεια-χώρο αντικείμενο *SurfaceView*, όπου οργανώνονται όλα τα γραφικά αντικείμενα που συμμετέχουν στο σενάριο του παιχνιδιού. Επίσης, εδώ είναι το κεντρικό σημείο όπου ξεκινά η εκτέλεση του παιχνιδιού. Επομένως είναι αναγκαίο, για την κατανόηση της λειτουργίας του παιχνιδιού αλλά και για να παρουσιαστεί μια γεύση από την μεθοδολογία ανάπτυξης μιας τέτοιας εφαρμογής σε συσκευές Android, να γίνει μια πιο αναλυτική περιγραφή του κώδικα της κλάσης.

```
public class GamePanel extends SurfaceView implements
SurfaceHolder.Callback {…}
```

Αρχικά συμφώνα με το Android Developer Documentation, το *SurfaceView* αποτελεί μια ιδανική επιφάνεια για να φιλοξενήσει γραφικά και εικόνες επιτρέποντας την εύκολη διαχείριση τους. Επιπλέον, δίνει την δυνατότητα επέκτασης με την *SurfaceHolder.CallBack*, έτσι ώστε να μπορεί να συλλαμβάνει και να διαχειρίζεται γεγονότα που συμβαίνουν πάνω στην επιφάνεια της κινητής συσκευής, όπως το χτύπημα ή και το σύρσιμο (touch, drag) του δάκτυλου πάνω στην επιφάνεια της οθόνης.

Κατόπιν κατά την περιγραφή του κώδικα της κλάσης όπως παρουσιάζεται παρακάτω, πρέπει να καθοριστούν τα βασικά χαρακτηριστικά του GamePanel, όπως το πλάτος και ύψος της οθόνης, η ταχύτητα κύλισης του φόντου κ.α. που θα αναφερθούν και κατά την χρησιμοποίηση τους.

```
public static final int WIDTH = 840;
public static final int HEIGHT = 420;
public static final int ROLLSPEED = -5;
private int difficulty = 10;
private Random rnd = new Random();
private boolean newGameCreated;
private int best;
private long startReset;
private boolean reset;
private boolean started;
```

Έτσι αρχικά, ορίζεται το πλάτος (WIDTH) και το ύψος (HIGH) αντίστοιχα της εφαρμογής σε 840 x 420, και η σταθερά που ελέγχει την ταχύτητα κύλισης (ROLLSPEED) της οθόνης. Επίσης, καθορίζεται ο βαθμός δυσκολίας (difficulty) με ένα παράγοντα πολλαπλασιαστή που ελέγχει τη ταχύτητα του παιχνιδιού. Επιπλέον, ορίζεται η μεταβλητή που τροφοδοτείται από τη μηχανή παραγωγής τυχαίων αριθμών που χρησιμεύει σε διάφορα σημεία του κώδικα όπου απαιτείται τυχαία επιλογή, κίνηση, δημιουργία, κλπ. Σημαντικές επίσης, είναι η μεταβλητή newGameCreated που ελέγχει την δημιουργία ενός νέου παιχνιδιού, η best που αποθηκεύει το καλύτερο σκορ και τέλος οι μεταβλητές startReset, reset και started για τη μέτρηση του χρόνου σε διάφορα σημεία του παιχνιδιού, οι οποίες βασίζονται στην ώρα του συστήματος και η μέτρηση τους γίνεται σε ns.

Στη συνέχεια του κώδικα πρέπει να γίνει ο ορισμός των αντικειμένων που θα αποτελέσουν τα γραφικά που χρησιμοποιούνται στο παιχνίδι, όπως φαίνονται στην παρακάτω εικόνα, ενώ η λειτουργία τους θα περιγραφεί κατά την ανάλυση των αντίστοιχων κλάσεων.

Εικόνα 5.2: Τα γραφικά του παιχνιδιού

```
private Background bg;
private Ufo ufo;
private boolean dissapear;
private ArrayList<EngineRun> engineRun;
private long engineStartTime;
private ElectricCrash electricCrash;
```

Αρχικά ορίζεται το αντικείμενο (bg) που περιέχει την εικόνα του υπόβαθρου το οποίο βρίσκεται σε διαρκή κύλιση και το γραφικό με το σκάφος (ufo) που ελέγχει ο παίκτης. Επίσης, απαραίτητη είναι η βοηθητική μεταβλητή dissapear που ελέγχει την ορατότητα του σκάφους όταν συγκρουστεί με κάποιο άλλο αντικείμενο, καθώς και ένας πίνακας λίστας (engineRun) που θα περιέχει σχήματα κύκλου διαφορετικού μεγέθους και ελαφρώς διαφοροποιημένου χρώματος, τα οποία με την εναλλασσόμενη εμφάνιση και εξαφάνιση τους πίσω από το σκάφος, παίζουν το ρόλο του καπνού που εξάγεται από τη μηχανή του σκάφους που λειτουργεί. Μαζί, ορίζεται και η βοηθητική μεταβλητή engineStartTime που ελέγχει τη συχνότητα με την οποία θα δημιουργείται το αντικείμενο καπνού. Τέλος, ορίζεται το αντικείμενο electricCrash που σκοπό έχει να αντικαταστήσει το σκάφος με την εικόνα της έκρηξης του.

```
private ArrayList<WalkingAllien> walkingAlliens;
private long walkingAllienstartTime;
```

Οι εξωγήινοι (walkingAlliens) είναι τα γραφικά που ορίζονται ως πίνακας λίστας ArrayList, ο οποίος θα συμπληρώνεται από τα αντικείμενα της κλάσης *WalkingAllien*, όπως θα εμφανίζονται και θα εξαφανίζονται στο παιχνίδι.

Η βοηθητική μεταβλητή walkingAllienstartTime ελέγχει τη συχνότητα που θα δημιουργούνται τα αντικείμενα των εξωγήινων ανάλογα με το πέρασμα του χρόνου.

```
private ArrayList<TopFence> topFences;
private ArrayList<BottomFence> botFences;
```

Οι φράκτες είναι τα γραφικά που καθορίζουν τα πάνω και τα κάτω όρια της οθόνης, όπου μπορεί να κινηθεί το σκάφος για να μην καταστραφεί αν πέσει επάνω τους. Έτσι, τα αντικείμενα αυτά ορίζονται ως Πινάκα λίστας, που γεμίζουν συνεχώς την οθόνη με ψευδοτυχαίο μεταβλητό ύψος, καθώς εκείνη κυλά

συνέχεια. Το μεταβλητό ύψος του φράχτη χτίζεται εισάγοντας διαφορετικό σύνολο από γραφικά το ένα πάνω από το άλλο, κάθε φορά που κυλά η οθόνη ένα βήμα. Βεβαία, την πρώτη φορά όταν αρχικοποιείται η οθόνη, πρέπει να γεμίσει όλη με τον πάνω και κάτω φράκτη. Έτσι, εδώ ορίζονται οι πίνακες λίστας (topFences, botFences) που θα περιέχουν τα γραφικά του πάνω φράκτη και αντίστοιχα του κάτω φράκτη που θα καθορίζει συνεχώς το κάτω όριο της οθόνης.

```
private int maxFenceHeight;
private int minFenceHeigh
private boolean topDown = true;
private boolean botDown = true;
```

Επιπλέον, είναι αναγκαίο να οριστούν βοηθητικές μεταβλητές που ελέγχουν το μέγιστο και ελάχιστο ύψος του φράκτη, καθώς και πότε θα αυξάνει και πότε θα μειώνεται το ύψος του.

```
private SoundPool sounds;
private int sndmale;
```

Για τις ανάγκες του παιχνιδιού ορίζεται και το αντικείμενο για τα εφέ ήχου (sounds). Επειδή χρησιμοποιείται ο μηχανισμός (SoundPool) από το Android API, τα εφέ ήχου θα είναι διαθέσιμα από την έκδοση 5.0 Lollipop και νεώτερη. Έτσι, έχει ορισθεί η μεταβλητή sndmale που χρησιμοποιείται στο εφέ της σύγκρουσης του σκάφους.

```
private MainThread thread;
```

Το επόμενο βασικό αντικείμενο (thread) είναι υπεύθυνο για την διαχείριση της διεργασίας που αποτελεί το *Game Loop* του παιχνιδιού. Το Game Loop αναλαμβάνει την αποτελεσματική φόρτωση στη μνήμη και εκτέλεση της επαναληπτικής διαδικασίας ενημέρωσης των αντικειμένων και επανασχεδιασμού τους στην οθόνη (update-draw), έτσι ώστε από τη μια το παιχνίδι να λειτουργεί με την επιθυμητή δυνατή ταχύτητα και από την άλλη η συσκευή να συνεχίζει να αποκρίνεται σωστά.

```
public GamePanel(Context ontext) {
 super(context);
 getHolder().addCallback(this);
    setFocusable(true);     }
```

Ξεκινώντας το κύριο σώμα του κώδικα της κλάσης, ορίζεται ο κατασκευαστής που δημιουργεί όταν καλείται το SurfaceView. Ως περιεχόμενο (context) ο κατασκευαστής δέχεται το αρχικό “Activity Game“ που έκανε την κλήση για το GamePanel. Επίσης, μέσα στο κατασκευαστή ενεργοποιείται η δυνατότητα σύλληψης και διαχείρισης γεγονότων (events) καθώς και η δυνατότητα εστίασης πάνω στην οθόνη.

```
@TargetApi(Build.VERSION_CODES.LOLLIPOP)
@Override
public void surfaceCreated(SurfaceHolder holder) {
  bg = new
Background(BitmapFactory.decodeResource(getResources(),
  R.drawable.road));
  ufo = new Ufo(BitmapFactory.decodeResource(getResources(),
  R.drawable.ship), 70, 40, 6);
  engineRun = new ArrayList<EngineRun>();
  engineStartTime = System.nanoTime();
  walkingAlliens = new ArrayList<WalkingAllien>();
  walkingAllienstartTime = System.nanoTime();
  topFences = new ArrayList<TopFence>();
  botFences = new ArrayList<BottomFence>();
```

Τώρα κατά την κλήση της μεθόδου *surfaceCreated()*, που εκτελείται αφού έχει δημιουργηθεί η επιφάνεια, πρέπει να δημιουργηθούν ένα-ένα τα αντικείμενα που θα χρησιμοποιηθούν στο παιχνίδι.

```
AudioAttributes.Builder attributeBuilder = new
AudioAttributes.Builder();
attributeBuilder.setUsage(AudioAttributes.USAGE_GAME);
final AudioAttributes.Builder builder =
attributeBuilder.setContentType(AudioAttributes.CONTENT_TYPE_
SONIFICATION);
AudioAttributes attributes = attributeBuilder.build();
SoundPool.Builder soundBuilder = new SoundPool.Builder();
soundBuilder.setAudioAttributes(attributes);
sounds = soundBuilder.build();
sndmale = sounds.load(this.getContext(), R.raw.crash, 1);
```

Ιδιαίτερος είναι ο τρόπος που πρέπει να δημιουργηθεί ο μηχανισμός παραγωγής των εφέ ήχου. Έτσι, ακολουθήθηκε η διαδικασία όπως περιγράφεται στο δικτυακό τόπο Android Developers, στην αναφορά για την λειτουργία της κλάσης SoundPool.
(https://developer.android.com/reference/android/media/SoundPool.html).

```
thread = new MainThread(getHolder(), this);
thread.setRunning(true);
thread.start();
```

Τέλος, μέσα στην μέθοδο *surfaceCreated()* του GamePanel, πρέπει να δημιουργηθεί το αντικείμενο thread που ελέγχει την κύρια διεργασία του παιχνιδιού. Επίσης, εδώ ξεκινά η εκτέλεση του παιχνιδιού θέτοντας τιμή αληθές στη μέθοδο setRunning(), η οποία ελέγχει αν ήδη εκτελείται ή όχι το παιχνίδι, και κατόπιν γίνεται κλήση στη μέθοδο start().

```
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
  boolean retry = true;
  sounds.release();
  while (retry) {
            try {
                 thread.setRunning(false);
                 thread.join();
                 retry = false;
                 thread=null;}
           catch (InterruptedException e) {
                 e.printStackTrace(); }
        }
    }  }
```

Συνέχεια με τη μέθοδο *surfaceDestroyed()*, όπου καλείται όταν πρέπει να καταστρέφει η επιφάνεια, όπως π.χ. στην περίπτωση που πατηθεί το κουμπί back στη συσκευή android για να σταματήσει το παιχνίδι, οπότε πρέπει να τερματιστεί η διεργασία (thread), δηλαδή να σταματήσει το Game Loop του παιχνιδιού. Η μέθοδος thread.join() και κατόπιν η “thread=null” αποτελούν την διαδικασία τερματισμού. Επειδή όμως οι εντολές είναι πιθανόν να αποτύχουν να εκτελεστούν επιστρέφοντας κάποιο μήνυμα λάθους, γι’ αυτό ενσωματώθηκαν σε ένα ατέρμονο “while loop” μέχρι να επιτευχθεί ο τερματισμός τους. Η εντολή “try-catch”

χρησιμοποιείται σε όλες τις περιπτώσεις που υπάρχει πιθανότητα να προκληθούν exception errors που οδηγούν σε καταστροφικό κόλλημα ή τερματισμό (crash).

```
@Override
public void surfaceChanged(SurfaceHolder holder, int format,
int width, int height) { }
```

Μετά από τις δυο παραπάνω μεθόδους (surfaceCreated και surfaceDestroyed), λόγω του εκτεταμένου ορισμού της κλάσης GamePanel για να δέχεται callbacks, είναι υποχρεωτικός και ο ορισμός της μεθόδου *surfaceChanged()*, έστω και αν δεν περιέχει κώδικα.

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {

        if (!ufo.isStartAnimate()&& newGameCreated && reset)
{
            ufo.setStartAnimate(true);
            ufo.setUp(true);    }

        if(ufo.isStartAnimate()){
            if(!started)started=true;
                reset=false;
                ufo.setUp(true);     }
        return true;       }

    if (event.getAction() == MotionEvent.ACTION_UP) {
        ufo.setUp(false);
        return true;      }
    return super.onTouchEvent(event);
}
```

Στο επόμενο βήμα πρέπει να γίνει ο ορισμός της μεθόδου *onTouchEvent()*, που δέχεται όλα τα συμβάντα από την οθόνης αφής και τα διαχειρίζεται μέσω της κλάσης “event”, παρέχοντας όλους τους δυνατούς τύπους ελέγχου κίνησης όπως πάτημα, σύρσιμο κλπ. Στη συγκεκριμένη περίπτωση γίνεται έλεγχος της ενέργειας όταν πατηθεί η οθόνη (MotionEvent.ACTION_DOWN), καθώς και της ενέργειας που πρέπει να εκτελεστεί όταν αφεθεί η οθόνη (MotionEvent.ACTION_UP). Στην πρώτη περίπτωση, όταν πατηθεί η οθόνη καλείται η μέθοδος της κλάσης “*Ufo*” .setUp(true), η οποία μετακινεί το σκάφος προς τα επάνω όπως θα δούμε και αναλυτικότερα στην ανάπτυξη της κλάσης. Βεβαία, γίνονται οι κατάλληλοι έλεγχοι για το εάν είναι η πρώτη φορά που έχει πατηθεί η οθόνη όταν έχει ξεκινήσει νέο

παιχνίδι, ώστε να κληθεί η μέθοδος που εμφανίζει το σκάφος να λειτουργεί (animated) και να αρχικοποιήσει τις μεταβλητές started και reset. Τώρα στην περίπτωση που αφεθεί η οθόνη, το σκάφος μετακινείται προς τα κάτω (.setUp(false)).

Η Τεχνική «Ενημέρωση-Σχεδίαση» (Update-Draw)

Στη συνέχεια, περιγράφεται στον κώδικα της κλάσης η ουσιαστική εκτέλεση της λειτουργίας του παιχνιδιού. Αυτή η λειτουργία βασίζεται στη χρησιμοποίηση της κλασικής τεχνικής σε παιχνίδια δυο διαστάσεων στην οποία καλούνται συνεχώς δύο βασικές μέθοδοι, η *update()* και η *draw()*, μέσα σε ένα συνεχές κύκλο εκτέλεσης του παιχνιδιού (Game Loop). Στη μέθοδο *update()* γίνεται η ενημέρωση οποιαδήποτε νέας κατάστασης – θέσης των αντικειμένων που εμφανίζονται μέσα στην οθόνη και συμμετέχουν στο παιχνίδι, για κάθε πλαίσιο (frame) που θα προβάλλεται στην μονάδα του χρόνου. Στη μέθοδο *draw()* αντίστοιχα για κάθε πλαίσιο (frame), σχεδιάζονται όλα αυτά τα αντικείμενα στην οθόνη του παιχνιδιού.

```
public void update() {
    if (ufo.isStartAnimate()) {
        if(botFences.isEmpty() || topFences.isEmpty()){
                ufo.setStartAnimate(false);
                return;
            }
```

Ξεκινώντας με το περιεχόμενο της μεθόδου *update()*, πρέπει όλη η ενημέρωση να γίνεται μέσα σε ένα συνεχή έλεγχο για τη κίνηση στο σκάφος του χρήστη. Επίσης λογικό κριτήριο για τη συνέχεια στην εκτέλεση του παιχνιδιού αποτελεί και το εάν αρχικά έχουν σχεδιαστεί οι φράκτες στην οθόνη.

```
bg.update();
ufo.update();
```

Στη συνέχεια καλείται η μέθοδος ενημέρωσης αντίστοιχα σε κάθε αντικείμενο που συμμετέχει στο παιχνίδι (background, ufo, κ.α.). Πρέπει να περιέχει το κώδικα που καθορίζει τις μεταβολές στην κατάσταση (θέση, κίνηση, κ.λπ) για το καθένα από αυτά.

```
    maxFenceHeight = 30 + ufo.getScore() / difficulty;
    if (maxFenceHeight == HEIGHT / 4) maxFenceHeight =
HEIGHT / 4;
         minFenceHeigh = 5 + ufo.getScore() / difficulty

    for(int i=0;i<topFences.size();i++){
       if(collission(topFences.get(i),ufo)) {
              // σταματάει το animation
               ufo.setStartAnimate(false);     }
             }
    for(int i=0;i<botFences.size();i++){
       if(collission(botFences.get(i),ufo)) {
               ufo.setStartAnimate(false);     }
             }
    this.updateTopFence();
    this.updateBottomFence();
```

Ιδιαίτερα στην περίπτωση των φρακτών, ορίζεται αρχικά το μέγιστο και το ελάχιστο ύψος τους, και πριν την ενημέρωση τους γίνεται έλεγχος για την πρόοδο του παιχνιδιού, ώστε να αυξάνεται η δυσκολία αλλάζοντας αυτά τα ύψη. Επίσης, εδώ ελέγχεται όλη η λίστα των αντικειμένων στους φράκτες για την περίπτωση που κάποιο έχει συγκρουστεί με το σκάφος (ufo), να εμφανιστεί το εφέ της σύγκρουσης και να σταματήσει το παιχνίδι. Κατόπιν γίνεται η ενημέρωση στους φράκτες, καλώντας τις εσωτερικές μεθόδους της κλάσης *.updateTopFence()* και *.updateBottomFence()*.

```
long engineElapsed = (System.nanoTime() - engineStartTime) /
1000000;
  if (engineElapsed > 120) {
     .add(new EngineRun(ufo.getX(), ufo.getY() + 10));
     engineStartTime = System.nanoTime();             }
  for (int i = 0; i < engineRun.size(); i++) {
      engineRun.get(i).update();
      if (engineRun.get(i).getX() < -10) {
                    engineRun.remove(i); }
  }
```

Όσον αφορά τους κύκλους καπνού που εμφανίζονται πίσω από το σκάφος, πρέπει να δημιουργηθεί ένας χρονομέτρης που ελέγχει κάθε πότε θα εμφανίζεται ένας καινούργιος κύκλος (engineElapsed). Χρησιμοποιείται το ρολόι του συστήματος για την αρχικοποίηση του χρονομέτρη και κάθε 120 msec δημιουργείται ένας κύκλος καπνού που προστίθεται στον πινάκα λίστας

engineRun. Για κάθε αντικείμενο της λίστας καλείται η μέθοδος ενημέρωσης. Επιπλέον, όσοι κύκλοι έχουν φύγει έξω από την οθόνη και δεν είναι πλέον ορατοί, απομακρύνονται από την λίστα για λόγους οικονομίας της μνήμης.

```
long WalkingAllienElapsed = (System.nanoTime()-
       walkingAllienstartTime) / 1000000;
if (WalkingAllienElapsed > 2000 - ufo.getScore() / 4) {
    WalkingAllien(BitmapFactory.decodeResource(getResources(),
    R.drawable.alien),(int) (8* WIDTH / 9* rnd.nextDouble())+10,

    HEIGHT, 114, 84, ufo.getScore(), 8));
   walkingAllienstartTime = System.nanoTime();         }

for (int i = 0; i < walkingAlliens.size(); i++) {
   walkingAlliens.get(i).update();
   if (walkingAlliens.get(i).getY() < -10) {
       walkingAlliens.remove(i);
       break;            }
   if (collission(walkingAlliens.get(i), ufo)) {
       ufo.setStartAnimate(false);
       walkingAlliens.remove(i);
       break;      }
}
```

Κατά αντίστοιχο τρόπο λειτουργεί και ο πινάκας λίστας με τα αντικείμενα εξωγήινοι (*walkingAlliens*) που διασχίζουν το δρόμο, με τη διάφορα ότι το βήμα του χρονομέτρη, που προθέτει νέα αντικείμενα στη λίστα, ελέγχεται με βάση το σκορ που έχει επιτευχθεί μέχρι την τρέχουσα στιγμή (*ufo.getScore()*). Δηλαδή, στην αρχή του παιχνιδιού οι εξωγήινοι εμφανίζονται πιο αραιά και όσο περνάει ο χρόνος (μεγαλώνει το σκορ) τότε εμφανίζονται πιο συχνά.

Η θέση εμφάνισης, η κίνηση και το μέγεθος των αντικειμένων ελέγχεται με τυχαίο τρόπο χρησιμοποιώντας την γεννήτρια *"rnd"*, με μέγιστο ύψος και πλάτος 114 και 84. Τέλος, ελέγχεται εάν κάποιο εξωγήινος έχει βρεθεί εκτός οθόνης ώστε να αφαιρεθεί από την λίστα, καθώς επίσης γίνεται έλεγχος για το εάν έχει συγκρουσθεί με το σκάφος *(ufo)*.

```
else {
    ufo.resetThrotle();
    if(!reset){
        newGameCreated=false;
        startReset=System.nanoTime();
        reset=true;
        dissapear=true;
        electricCrash=new ElectricCrash(BitmapFactory.
          decodeResource(getResources(),R.drawable.lightning),
          ufo.getX(),ufo.getY()-30,100,100,25);
    }
    electricCrash.update();
    long resetElapsed=(System.nanoTime()-startReset)/1000000;
    if(resetElapsed>2500 && !newGameCreated){
        newGame();      }
    }  }
```

Κατόπιν, στη συνέχεια της μεθόδου *update()* πρέπει να ελεγχθεί η εναλλακτική περίπτωση του κριτηρίου *ufo.isStartAnimate()* – στην αρχή της μεθόδου update(). Ο κώδικας αυτός στην ουσία θα εκτελεστεί είτε όταν το παιχνίδι ξεκινά πρώτη φορά, είτε όταν έχει γίνει σύγκρουση του σκάφους και έχει σταματήσει η εκτέλεση.

Εάν ισχύει η δεύτερη περίπτωση, η οποία ελέγχεται από το λογικό κριτήριο “reset”, ξανά αρχικοποιούνται οι μεταβλητές που ελέγχουν το σκάφος (π.χ. ο χρόνος του παιχνιδιού, η ταχύτητα) και εκτελείται το εφέ καταστροφής του, δηλαδή εξαφανίζεται το σκάφος και εμφανίζεται το αντικείμενο *electricCrash* που το αντικαθιστά στην ίδια θέση (συντεταγμένες X, Y). Τέλος, γίνεται η κλήση της μεθόδου *newGame()* που ξεκινά το νέο παιχνίδι.

```
public void draw(Canvas canvas) {
    float scaleFactorX = getWidth() / (GamePanel.WIDTH*1.0f);
    float scaleFactorY = getHeight() / (GamePanel.HEIGHT*1.0f);
    if (canvas != null) {
        final int savedState = canvas.save();
        canvas.scale(scaleFactorX,scaleFactorY);
        …
        canvas.restoreToCount(savedState);
    }
}
```

Μετά την μέθοδο *update()* ακολουθεί η μέθοδος *draw()*, που είναι υπεύθυνη για το σχεδιασμό της οθόνης για κάθε πλαίσιο (frame). Αντίστοιχα με την update(), η τεχνική βασίζεται στη λογική ότι για κάθε κλάση αντικειμένου έχει δημιουργηθεί μια

μέθοδος draw(), όπου καλείται μέσα στην κεντρική μέθοδο draw της κλάσης GamePanel.

Πριν όμως από την κλήση των μεθόδων draw, χρησιμοποιείται άλλη μια τεχνική: για κάθε frame αποθηκεύεται αρχικά η τρέχουσα κατάσταση του “canvas” της οθόνης, στο οποίο δουλεύεται ο σχεδιασμός όλων των αντικειμένων (*canvas.save()*), και στη συνέχεια τον μεγεθύνει και τον προσαρμόζει λαμβάνοντας υπόψη τις πραγματικές διαστάσεις της κάθε συσκευής Android (*canvas.scale(scaleFactorX,scaleFactorY)*). Επειδή όμως η ενημέρωση (*update()*) της τρέχουσας κατάστασης του κάθε αντικειμένου γίνεται με βάση τις προκαθορισμένες αρχικές διαστάσεις, πρέπει να επανέρχεται η οθόνη στο αρχικό της μέγεθος (*canvas.restoreToCount(savedState)*).

```
bg.draw(canvas);
if(!dissapear){
    ufo.draw(canvas);
    for (EngineRun engR : engineRun) {
        engR.draw(canvas); }
}
for (WalkingAllien walkingA : walkingAlliens) {
    walkingA.draw(canvas); }
for (TopFence topF:topFences){topF.draw(canvas); }
for(BottomFence botF:botFences){ botF.draw(canvas);}
if(started){ electricCrash.draw(canvas);    }
drawText(canvas);
```

Έτσι μέσα σ’ αυτή την επαναληπτική τεχνική εναλλαγής του canvas της οθόνης σχεδιάζονται το υπόβαθρο (bg), το σκάφος (ufo), όλα τα αντικείμενα της λίστας που εμφανίζουν το καπνό (EngineRun), τους εξωγήινους (WalkingAllien), τον πάνω και κάτω φράκτη (Top & Bottom Fence), κι αν πρέπει το εφέ της σύγκρουσης (electricCrash). Τέλος καλείται η τοπική μέθοδος drawText(), η οποία θα περιγραφεί παρακάτω, που εμφανίζει τα αναγκαία μηνύματα στην αρχή του παιχνιδιού, καθώς και το σκορ και την καλύτερη επίδοση.

```
public boolean collission(Actor a1, Actor a2) {
    if (Rect.intersects(a1.colRect(), a2.colRect())) {
        sounds.play(sndmale, 1, 1, 0, 0, 1);
        return true;    }
    return false;
}
```

Με το πέρας της περιγραφής της τεχνικής *«Ενημέρωση-Σχεδίαση»*, περιγράφονται οι μέθοδοι που καλούνται και χρησιμοποιούνται μέσα στις μεθόδους update-draw. Αρχικά, υπάρχει η μέθοδος *collision()*, η οποία ελέγχει εάν ένα αντικείμενο έχει συγκρουστεί με κάποιο άλλο (το ufo με έναν εξωγήινο ή ένα φράκτη), χρησιμοποιώντας την μέθοδο *intersects()* του Android API. Η μέθοδος intersects() έχει τη δυνατότητα να ελέγξει εάν δυο κλειστές περιοχές (εδώ δύο παραλληλόγραμμα - *Rect*) έχουν κοινά σημεία, δηλαδή εάν το ένα παραλληλόγραμμο έχει μπει μέσα στα όρια του άλλου (σύγκρουση). Εάν υπάρχουν κοινά σημεία, εφαρμόζεται και η μέθοδος που αναπαράγει τους ήχους σύγκρουσης, χρησιμοποιώντας το αντικείμενο soundPool.

```
public void updateTopFence() {
  int xpos = topFences.get(topFences.size() - 1).getX()+20;
  int hpos = 0;
  if (ufo.getScore() % 50 == 0) {
    int rand = (int) (rnd.nextDouble() * maxFenceHeight)+1;
    topFences.add(new TopFence(BitmapFactory.decodeResource(
    getResources(), R.drawable.barrier), xpos, 0, rand));  }
  for (int i = 0; i < topFences.size(); i++) {
    topFences.get(i).update();
    if (topFences.get(i).getX() < -20) {
       topFences.remove(i);
       if (topFences.get(topFences.size() - 1).getHeight()>=
          maxFenceHeight) { topDown = false; }
       if (topFences.get(topFences.size() - 1).getHeight()<=
          minFenceHeigh) {  topDown = true;  }
       if (topDown) { hpos = topFences.get(topFences.size()
          -1).getHeight() + 1;
          topFences.add(new
          TopFence(BitmapFactory.decodeResource(
          getResources(), R.drawable.barrier ,  xpos, 0,
          hpos));

       }
```

```
        else { hpos = topFences.get(topFences.size() -
                1).getHeight() - 1;
                topFences.add(new TopFence(
                BitmapFactory.decodeResource(
                getResources(), R.drawable.barrier), xpos, 0,
                hpos)); }
    } }
}
```

```
public void updateBottomFence() {
    int xpos = botFences.get(botFences.size() -1).getX()+20;
    int ypos = botFences.get(botFences.size() -1).getY()+1;
    Bitmap cImage=BitmapFactory.decodeResource(
           getResources(), R.drawable.barrier);
    if (ufo.getScore() % 40 == 0) {
        int rand = (int) (rnd.nextDouble() * maxFenceHeight) +
        (HEIGHT - maxFenceHeight);
        botFences.add(new BottomFence(cImage, xpos, rand));  }
    for (int i = 0; i < botFences.size(); i++) {
        botFences.get(i).update();
        if (botFences.get(i).getX() < -20) {
            botFences.remove(i);
            if (botFences.get(botFences.size() - 1).getHeight()
                >= maxFenceHeight) { botDown = false;   }
            if (botFences.get(botFences.size() - 1).getHeight()
                <= minFenceHeigh) {  botDown = true;    }
            if (!botDown) {
                ypos = botFences.get(botFences.size() -
                1).getY() - 1;     }

            botFences.add(new BottomFence(cImage, xpos, ypos));
      } }
}
```

Επόμενες είναι οι μέθοδοι που ενημερώνουν τον πάνω και κάτω φράκτη (*updateTopFence()*, *updateBottomFence()*). Οι φράκτες ακολουθούν τη τεχνική της συμπλήρωσης κομματιών στο τέλος της οθόνης που κυλά. Η θέση για κάθε νέο αντικείμενο που συμπληρώνει τον κάθε φράκτη, ενημερώνεται με βάση τη θέση του προηγούμενου αντικείμενου (*xpos, hpos*), προσθέτοντας ή αφαιρώντας εικόνες προς τα πάνω ή κάτω, δίνοντας την αίσθηση της μεταβολής του μεγέθους του, αλλά και ανεβάζοντας την δυσκολία στο παιχνίδι. Βέβαια, κατά την πρόοδο του σκορ γίνονται απότομες και τυχαίες αλλαγές στο μέγεθος και τη θέση του φράκτη, ώστε να δημιουργηθεί κάποια ασυνέχεια, να μην επαναλαμβάνεται

δηλαδή το ίδιο μοτίβο στο σχεδιασμό των φρακτών και να ελεγχθεί η προβλεψιμότητα του παιχνιδιού (*if (ufo.getScore()%50==0) { int rand = …*). Επιπλέον υπάρχει διαφοροποίηση με τον τρόπο και την τυχαιότητα που ενημερώνονται οι δυο φράκτες (*if (ufo.getScore()%50==0,* όταν η ακέραια διαίρεση του σκορ με το 50 έχει υπόλοιπο μηδέν*)* στον πάνω φράκτη και *if (ufo.getScore()%40==0)* στον κάτω φράκτη). Τέλος, γίνονται οι απαραίτητοι έλεγχοι ώστε να απομακρύνονται τα αντικείμενα της λίστας του κάθε φράκτη που βρίσκονται εκτός οθόνης (*topFences.remove(i)*).

```
public  void newGame(){
    dissapear=false;
    topFences.clear();
    botFences.clear();
    walkingAlliens.clear();
    engineRun.clear();

    if(ufo.getScore()*3>best){ best=ufo.getScore()*3; }
    ufo.resetScore();
    ufo.setY(HEIGHT/2);
    minFenceHeigh=5;
    maxFenceHeight=30;

    for(int i=0;i*20<WIDTH+40;i++){
        if(i==0){
            topFences.add(new TopFence(
              BitmapFactory.decodeResource(getResources(),
              R.drawable.barrier),i*20,0,10));
           botFences.add(new BottomFence(
             BitmapFactory.decodeResource(getResources(),
            R.drawable.barrier),i*20,HEIGHT-minFenceHeigh)); }
    else{
        topFences.add(new TopFence(
          BitmapFactory.decodeResource(getResources(),
          R.drawable.barrier),i*20,0,topFences.get(i-
          1).getHeight()+1));
        botFences.add(new BottomFence(
          BitmapFactory.decodeResource(getResources(),
          R.drawable.barrier),i*20,botFences.get(i-1).getY()-
          1)); }
    }
    newGameCreated=true;
}
```

Κατόπιν ακολουθεί η μέθοδος *newGame()*, που καλείται όταν ξεκινά ένα νέο παιχνίδι, είτε για πρώτη φορά είτε μετά από σύγκρουση του σκάφους. Αυτή η

μέθοδος είναι υπεύθυνη για τον καθαρισμό και αρχικοποίηση των μεταβλητών και των στιγμιότυπων των αντικειμένων (*topFences, botFences, walkingAlliens, engineRun*). Επίσης κρατάει το καλύτερο σκορ (*ufo.getScore()*). Τέλος, αναλαμβάνει να γεμίσει την οθόνη για πρώτη φορά με τους φράκτες, ώστε να είναι έτοιμοι πριν ξεκινήσει το παιχνίδι και η οθόνη αρχίσει να κυλά.

```
public void drawText(Canvas canvas){
    Paint paint=new Paint();
    paint.setTypeface(Typeface.create(Typeface.DEFAULT,
                                      Typeface.BOLD_ITALIC));
    paint.setTextSize(35);
    paint.setColor(Color.WHITE);
    canvas.drawText("Αποσταση:
"+(ufo.getScore()*3),10,HEIGHT-5,paint);
    canvas.drawText("Καλυτερος:"+best,WIDTH-250,HEIGHT-
10,paint);
    if(!ufo.isStartAnimate() && newGameCreated && reset){
       Paint paint1=new Paint();
       paint1.setColor(Color.CYAN);
       paint1.setStyle(Paint.Style.FILL);
       paint1.setAlpha(80); // Διαφάνεια
       canvas.drawRect(10,10,WIDTH-10,HEIGHT-10,paint1);
       paint1.setColor(Color.YELLOW);
       paint1.setTextSize(30);
       paint1.setTypeface(Typeface.create(Typeface.DEFAULT,
                          Typeface.BOLD));
       canvas.drawText("ΣΠΑΣΜΕΝΟ UFO ", WIDTH/2-140,HEIGHT/2-
70,paint1);
       canvas.drawText("Πάτα για να ξεκινήσει το παιχνίδι
και", WIDTH/2-300,HEIGHT/2-20,paint1);
       paint1.setTextSize(20);
       canvas.drawText("Συνέχισε να πατάς για να μετακινηθεί
το UFO προς τα πάνω:",WIDTH/2-350,HEIGHT/2+70,paint1);
       canvas.drawText("Σταμάτα  να πατάς για να μετακινηθεί
το UFO προς τα κατω:",WIDTH/2-300,HEIGHT/2+140,paint1);
        }
  } }
```

Και τελευταία μέσα στην κλάση GamePanel είναι η μέθοδος *drawText()*, που αναλαμβάνει την απεικόνιση κειμένου όπως το σκορ, τη βέλτιστη επίδοση και το βοηθητικό πλαίσιο κειμένου με οδηγίες και πληροφορίες πριν την έναρξη κάθε παιχνιδιού. Η μέθοδος αυτή βασίζεται στην χρήση της κλάσης *"Paint"*, η οποία παρέχεται από το Android API και υποστηρίζει όλες τις δυνατότητες απεικόνισης κειμένου (χρώμα, στυλ γραμματοσειράς, διαφάνεια, κ.α.).

5.4 Η κλάση MainThread - Τεχνική βασισμένη σε πλαίσια (frame-based)

Σ' αυτή την κλάση υλοποιείται η διαδικασία του GameLoop όπως έχει αναφερθεί και παραπάνω, δηλαδή η διαδικασία της διαρκούς ενημέρωσης της κατάστασης όλων των αντικειμένων που συμμετέχουν στο παιχνίδι και ο επανασχεδιασμός τους στην οθόνη (update-draw). Αυτό εκτελείται επαναληπτικά, για κάθε frame του παιχνιδιού. Στην ουσία καλώντας την MainThread παράγεται το νήμα εκτέλεσης εντολών για την κύρια διεργασία (process) της εφαρμογής, που δημιουργείται στη μνήμη της συσκευής. Όμως αυτή η διαδικασία εκτέλεσης πρέπει να έχει σταθερό ρυθμό διάρκειας σχεδιασμού και αναπαράστασης των αντικειμένων, έτσι ώστε αφενός όταν η οθόνη περιέχει πολλά αντικείμενα και μεγάλες απαιτήσεις σε διαδικασίες να μην καθυστερεί (lagging), αφετέρου όταν περιέχει λίγα και απλά αντικείμενα, να μην τρέχει.

```
public class MainThread extends Thread {
    private int FPS=30; // Frames Per Second
    private double averageFPS;
    private SurfaceHolder surfaceHolder;
    private GamePanel gamePanel;
    private boolean running;
    public static Canvas canvas;

    public MainThread(SurfaceHolder surfaceHolder,GamePanel
gamePanel){
        super();
        this.surfaceHolder=surfaceHolder;
        this.gamePanel=gamePanel;
    }
```

Υπάρχουν δύο βασικές τεχνικές υλοποίησης: η τεχνική βασισμένη στο χρόνο και η τεχνική βασισμένη στα πλαίσια. Στη συγκεκριμένη περίπτωση, χρησιμοποιείται η τεχνική σταθερής διάρκειας του πλαισίου της οθόνης (frame), το οποίο πρέπει να σχεδιάζεται στο 1/30 του δευτερολέπτου (*FPS*). Αυτή είναι μια τυπική διάρκεια και βασίζεται στην αδυναμία του ανθρώπινου ματιού να διακρίνει διακοπές σε τέτοιες συχνότητες. Επίσης, το πλαίσιο ορίζεται ως μια πλήρη σχεδιασμένη οθόνη με τα αντικείμενα που συμμετέχουν στη εφαρμογή για οποιαδήποτε τυχαία στιγμή της εκτέλεσης της.

Τώρα μέσα στην κλάση, για τις ανάγκες δημιουργίας του Thread, είναι απαραίτητος ο ορισμός της δομής *"surfaceHolder"*, ένα αφηρημένο interface που συνδέεται με την επιφάνεια (surfaceView) της εφαρμογής gamePanel. Επιπλέον,

είναι απαραίτητος ο ορισμός ενός πεδίου τύπου Canvas που ορίζει ένα πλαίσιο πάνω στην επιφάνεια, όπου επιτρέπεται ο σχεδιασμός αντικειμένων σε επίπεδο pixel (sprites). Τέλος, ορίζονται το πεδίο running για το λογικό έλεγχο εκτέλεσης του Thread και ένα πεδίο για την οπτική ενημέρωση του μέσου όρου πλαισίων που μπορούν να εμφανιστούν στην μονάδα του χρόνου (averageFPS).

```
@Override
public void run(){
    long startTime;
    long timeMillis;
    long waitTime;
    long totalTime=0;
    int  frameCount=0;
    long targetTime=1000/FPS;

    while (running){
        startTime=System.nanoTime();
        canvas=null;
        try {
            canvas= this.surfaceHolder.lockCanvas();
            synchronized (surfaceHolder){
                this.gamePanel.update();
                this.gamePanel.draw(canvas);}
        }
        catch (Exception e){}
        finally {
            if(canvas!=null){
                try{
                  surfaceHolder.unlockCanvasAndPost(canvas);}
                catch(Exception e){
                    e.printStackTrace(); }
            }
        }
```

Στο επόμενο βήμα πρέπει να καθοριστεί η βασική μέθοδος *run()* που ξεκινά την εκτέλεση του MainThread(). Εδώ καλούνται οι μέθοδοι ενημέρωσης και σχεδιασμού της GamePanel μεταξύ των μεθόδων *lockCanvas()* και *unlockCanvasAndPost()*. Η πρώτη επιτρέπει την επεξεργασία του πλαισίου Canvas, ενώ η δεύτερη τερματίζει την επεξεργασία και απεικονίζει το περιεχόμενο του στην οθόνη. Επειδή υπάρχει μεγάλη πιθανότητα να προκληθούν “exception errors”, τόσο κατά την εκτέλεση των εντολών όσο και κατά την ανάπτυξη της εφαρμογής, είναι χρήσιμο να περικλείονται οι εντολές σε “try catch”. Ειδικά η

unlockCanvasAndPost() περικλείεται και σε μια “finally”, ώστε να εκτελεστεί ακόμα και όταν έχει συλληφθεί ένα exception error. Επιπλέον, καλό είναι να συμπεριλαμβάνεται η εντολή synchronized, η οποία βεβαία λειτουργεί όταν υπάρχουν πολλαπλά thread που έχουν πρόσβαση στο ίδιο surfaceView, κάτι που όμως δεν συμβαίνει στην συγκεκριμένη εφαρμογή.

```
    timeMillis=(System.nanoTime() - startTime) / 1000000;
    waitTime=targetTime-timeMillis;
    try{
        this.sleep(waitTime); }
    catch (Exception e){}
    totalTime+=System.nanoTime()-startTime;
    frameCount++;
    if (frameCount==FPS){
        averageFPS=1000/((totalTime/frameCount)/1000000);
        System.out.println( averageFPS);
        frameCount=0;
        totalTime=0;   }
} }
```

Όπως φαίνεται στο κώδικα, χρησιμοποιείται ένας χρονομέτρης που μετρά τη διάρκεια κάθε πλαισίου (*timeMillis*) και χρησιμεύει στον υπολογισμό του χρόνου καθυστέρησης (*waitTime*) που πρέπει η εφαρμογή να περιμένει στην περίπτωση που το πλαίσιο έχει ολοκληρωθεί γρηγορότερα (*sleep(waitTime)*). Όταν όμως αυτός ο χρόνος είναι μεγαλύτερος από τον αναμενόμενο, τότε έχουμε υπέρβαση χρονισμού και τελικά χαμένα πλαίσια (drop frames). Επιπρόσθετα, ο υπολογισμός του μέσου όρου απεικόνισης πλαισίων (*averageFPS*) γίνεται απλά για την παρατήρηση των χαμένων πλαισίων, έτσι ώστε κατά την ανάπτυξη του παιχνιδιού να αναγνωριστούν οι ελάχιστες αναγκαίες απαιτήσεις της συσκευής για την εκτέλεση του.

```
public void setRunning(boolean b) {
    this.running=b;
}   }
```

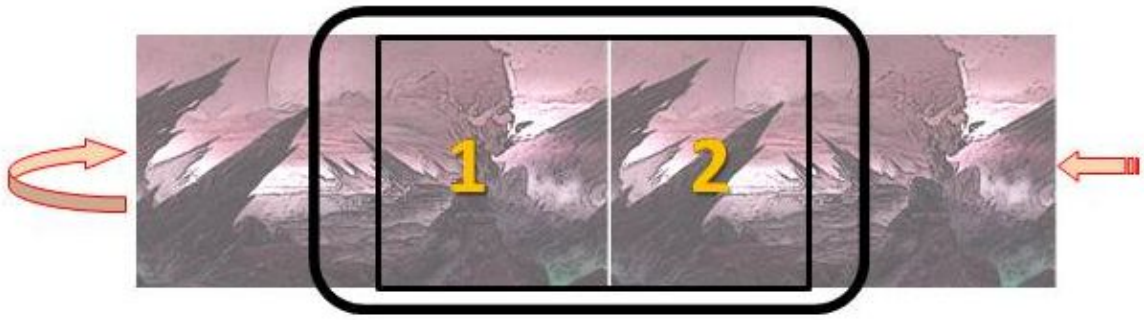
Τέλος, είναι αναγκαίο το λογικό πεδίο *running* για την εκτέλεση και τον τερματισμό του Thread.

Στη συνέχεια αυτής της εργασίας, θεωρείται απαραίτητο να περιγραφεί η ανάπτυξη του κώδικα συγκεκριμένων κλάσεων, όπως οι *Background* και *Ufo*,

επειδή ενσωματώνουν σημαντικές τεχνικές για την ανάπτυξη μιας τέτοιας εφαρμογής. Οι υπόλοιπες κλάσεις έχουν αναπτυχθεί με παρόμοιο τρόπο και έτσι δεν θα αναλυθούν, ο κώδικας τους όμως περικλείεται στο παράρτημα (σελ. 99) που βρίσκεται στο τέλος της εργασίας.

5.5 Η κλάση Background – Η τεχνική wraparound

Η βασική ιδέα για την ανάπτυξη της κλάσης του υπόβαθρου είναι η συνεχής περιστροφή δυο εικόνων με το ίδιο περιεχόμενο, όπου η μια συνδέεται στο τέλος της άλλης δίνοντας την αίσθηση συνεχής κίνησης, όπως φαίνεται στην παρακάτω εικόνα. Σημαντικό για την επιτυχία αυτής της τεχνικής είναι να ταιριάζει οπτικά το τέλος της εικόνας με την αρχή της, έτσι ώστε να μην φαίνεται κάποια έντονη ασυνέχεια.



Εικόνα 5.3: *Η τεχνική wraparound*

```
public class Background {
    private Bitmap image;
    private int x,dx;
    private int y;
    public Background(Bitmap resImage) {
        this.image=resImage;
        this.dx=GamePanel.ROLLSPEED;   }

    public void update(){
        x+=dx;
        if(x<=-GamePanel.WIDTH){
            x=0;        }
    }
    public void draw(Canvas canvas){
       canvas.drawBitmap(image,x,y,null);
       if(x<0){
          canvas.drawBitmap(image,x+GamePanel.WIDTH,y,null);}
}   }
```

Για την υλοποίηση της κλάσης είναι απαραίτητα η θέση *x, y* της εικόνας του background, το βήμα κύλισης σε pixels (*ROLLSPEED→dx*) σε -5 που έχει ορισθεί ήδη στην κλάση GamePanel και η φόρτωση της εικόνας που προσλαμβάνεται ως παράμετρος κατά την κλήση (*resImage→image*). Στη μέθοδο *update()*, όπου ενημερώνεται η θέση της εικόνας, ορίζεται με βάση το βήμα dx μια συνεχή μετακίνηση προς τα αριστερά μέχρι να βγει εξολοκλήρου έξω από την οθόνη, οπότε και επανέρχεται πάλι στην αρχική θέση για να επαναλάβει την ίδια διαδικασία. Λόγω της μετακίνησης αυτής προς τα αριστερά, δημιουργείται ένα αυξανόμενο κενό στα δεξιά της οθόνης, το οποίο και πρέπει να συμπληρωθεί με την ίδια εικόνα για να φανεί η συνεχόμενη κύλιση. Στη μέθοδο *draw()* λοιπόν, όπου γίνεται ο σχεδιασμός του αντικείμενου στην οθόνη, δεν σχεδιάζεται μια εικόνα αλλά δυο, εκτός βέβαια από την μόνη περίπτωση που η πρώτη εικόνα καταλαμβάνει πλήρως την οθόνη. Κατά την απεικόνιση στην οθόνη, η μια εικόνα σχεδιάζεται με βάση τις συνταγμένες που έχουν ορισθεί στην *update()*, ενώ η δεύτερη σχεδιάζεται στο τέλος της πρώτης εικόνας, σε σχέση πάντα με τις συντεταγμένες της πρώτης.

5.6 Οι κλάσεις Ufo και Actor – Εισαγωγή στην κινούμενη εικόνα (Animation)

Πριν επεκταθεί η περιγραφή στην ανάλυση του κώδικα της κλάσης *Ufo*, πρέπει να γίνει μια αναφορά στην αφηρημένη (Abstract) κλάση *Actor*. Η κλάση αυτή περιέχει μια σειρά από κοινά χαρακτηριστικά-πεδία, όπως οι συντεταγμένες θέσης, το ύψος και το πλάτος, τα βήματα μετακίνησης προς στις δυο κατευθύνσεις, καθώς και μια σειρά από κοινές μεθόδους (setters & getters), που μπορούν να μεταβάλουν τις τιμές των χαρακτηριστικών-πεδίων, προστατεύοντάς τα από την άμεση πρόσβαση από εξωτερικές κλάσεις-αντικείμενα. Όλα αυτά τα κοινά χαρακτηριστικά είναι αναγκαία και κληρονομούνται στις υπόλοιπες κλάσεις (Ufo, WalkingAllien, ElectricCrash, EngineRun κ.λπ.), επιτρέποντας έτσι την μη επανάληψη συγγραφής του κώδικα.

```
public abstract class Actor {
    protected int x;
    protected int y;
    protected int width;
    protected int height;
    protected int dirX;
    protected int dirY;

    public void setX(int x){ this.x=x; }
    public void setY(int y){ this.y=y; }
    public int getX() { return x; }
    public int getY() { return y; }
    public void setWidth(int width) { this.width = width; }
    public void setHeight(int height){ this.height = height;}
    public int getWidth() { return width;  }
    public int getHeight(){ return height;  }
    public void setDirX(int dirX) { this.dirX = dirX; }
    public void setDirY(int dirY) { this.dirY = dirY; }
    public int getDirX() { return dirX;  }
    public int getDirY() { return dirY;  }
    public Rect colRect(){ return new
Rect(x,y,x+width,y+height); }
}
```

Έτσι και στην συγκεκριμένη περίπτωση της κλάσης *Ufo*, γίνεται επέκταση (extend) σε κλάση *Actor*, ώστε να συμπεριλάβει τα παραπάνω χαρακτηριστικά. Επιπλέον, σ' αυτή την κλάση όπως και σε άλλες, κατά την δημιουργία ενός αντικειμένου εφαρμόζεται μια συνεχή εναλλαγή της εικόνας του γραφικού με παραπλήσια χαρακτηριστικά, όπως η μεταβολή του χρώματος, ή η μετακίνηση κάποιων τμημάτων, δίνοντας την αίσθηση μιας συνεχής κίνησης "animation" του αντικειμένου. Για την επίτευξη αυτής της δυνατότητας, χρησιμοποιείται ένα στιγμιότυπο της κλάσης *Animation*, που αναλαμβάνει την υλοποίηση αυτού του χαρακτηριστικού.

```
public class Ufo extends Actor{
    private Bitmap imgSprites;
    private int score;
    private boolean up;
    private boolean startAnimate;
    private Animation anim=new Animation();
    private Long startTime;

    public Ufo (Bitmap imageRes, int w, int h, int aniFrm){
        this.x=100;
        this.y=GamePanel.HEIGHT/2;
        this.width=w;
        this.height=h;
        this.score=0;
        this.dirY=0;
        Bitmap[] imageArray = new Bitmap[aniFrm];
        imgSprites=imageRes;
        for(int i=0;i<imageArray.length;i++){
           imageArray[i]=Bitmap.createBitmap(
                        imgSprites,i*width,0,width,height);
           anim.setFrames(imageArray);
           anim.setDelay(10);
           startTime=System.nanoTime();     }
     }
  …
}
```

Κατ' αρχήν σε κάθε κλάση που θα χρησιμοποιηθεί αυτή η δυνατότητα, πρέπει να υπάρχει μια εικόνα-σελίδα με όλα τα στιγμιότυπα του γραφικού που θα χρησιμοποιηθούν. Έτσι στην συγκεκριμένη περίπτωση του *Ufo*, χρησιμοποιείται η παρακάτω εικόνα, όπου τα στιγμιότυπα του γραφικού τοποθετούνται στον πίνακα *Bitmap[]* ως εικόνες ίδιου μεγέθους, και μεταφέρονται ως παράμετρος κατά την κλήση της μεθόδου *setFrames()* της κλάσης *Animation*.

Εικόνα 5.4: *Τα στιγμιότυπα του γραφικού Ufo*

Πριν συνεχίσουμε με τον κώδικα της κλάσης *Ufo*, πρέπει επίσης να γίνει αναφορά στον κώδικα της κλάσης *Animation*, η οποία είναι υπεύθυνη για την αναπαραγωγή των κινούμενων εικόνων (sprites).

```
public class Animation {
  private Bitmap[] frames;
  private int curFrame;
  private long delay;
  private long startTime;
  private boolean oneLoop;
  public void Animation(){    }
  public void setFrames(Bitmap[] frames){
      this.frames=frames;
      this.curFrame=0;
      startTime=System.nanoTime();     }
  public void setDelay(long delay) {
      this.delay = delay;     }
  public void setCurFrame(int curFrame) {
      this.curFrame = curFrame;     }

  public void update(){
      long elapsedTime=(System.nanoTime()-startTime)/1000000;
      if(elapsedTime>delay){
          curFrame++;
          startTime=System.nanoTime(); }
      if(curFrame==frames.length){
          curFrame=0;
          oneLoop=true;          }
  }
  public Bitmap getImage(){ return frames[curFrame]; }
  public int getFrame(){ return curFrame; }
  public boolean isOneLoop() { return oneLoop; }
}
```

Όπως φαίνεται παραπάνω, η μορφή του είναι πολύ απλή και σκοπό έχει να αλλάζει συνεχώς το τρέχον στιγμιότυπο του γραφικού, επιστρέφοντας το τρέχον πλαίσιο (*curFrame*) σε καθορισμένο χρονικό διάστημα. Αυτό το χρονικό διάστημα καθορίζεται κατά την κλήση της μεθόδου *setDelay()*. Επιπλέον, αυτό το εφέ κίνησης πάνω στα αντικείμενα μπορεί να είναι επαναλαμβανόμενο, όπως στην περίπτωση του ufo, ή να τερματίζεται αφού ολοκληρωθεί η εμφάνιση όλων των πλαισίων μία φορά, όπως στην περίπτωση που χρησιμοποιείται το αντικείμενο *ElelecticCrash* όταν καταστρέφεται το σκάφος. Για τον έλεγχο αυτής της επανάληψης του εφέ, χρησιμοποιείται η λογική τιμή που επιστρέφεται από την κλήση της μεθόδου *isOneLoop()*.

Φεύγοντας από την κλάση *Animation* και επιστρέφοντας στο κώδικα της κλάσης *Ufo*, θα ολοκληρωθεί με την περιγραφή μιας σειράς λειτουργιών που επιτελούνται.

```
public void setUp(boolean pressed){ this.up=pressed;}

public void update(){
     long elapsed=(System.nanoTime()-startTime)/1000000;
     if(elapsed>100){
          score+=1;
          startTime=System.nanoTime();
     }
     anim.update();
     if(up) {  dirY-=1;    }
     else   {  dirY+=1;    }
     if (dirY>14){dirY=14; }
     if (dirY<-14 ){dirY=-14; }
     y+=dirY*2;
}
```

Ξεκινώντας από την μέθοδο *update()*, εδώ γίνονται ο έλεγχος της εξέλιξης της βαθμολογίας με την πάροδο του χρόνου (*score*), η ενημέρωση του animation (*anim.update()*), και η αλληλεπίδραση με το χρήστη ορίζοντας τη θέση του σκάφους ανάλογα με την επαφή πάνω στην οθόνη της συσκευής από το χρήστη.

```
public void draw(Canvas canvas){
    canvas.drawBitmap(anim.getImage(),x,y,null);   }

public int getScore(){  return score;   }
public boolean isStartAnimate() {  return startAnimate;   }
public void setStartAnimate(boolean startAnimate) {
    this.startAnimate = startAnimate;   }
public void resetThrotle(){  dirY=0;   }
public void resetScore(){  score=0;   }
```

Τέλος, αναγκαίες είναι η μέθοδος του σχεδιασμού της οθόνης (*draw()*) και οι μέθοδοι που αρχικοποιούν και επαναφέρουν τις τιμές των πεδίων στις αρχικές τους καταστάσεις (*resetThrotle()*, κ.α.)

5.7 Η κλάση EngineRun – Εναλλακτική τεχνική παραγωγής κινούμενης εικόνας (Animation)

Σε αυτή τη κλάση δημιουργείται και σχεδιάζεται το εφέ του καπνού που βρίσκεται πίσω από το εν κινήσει ufo. Κατά την περιγραφή της κλάσης *Ufo* στην προηγούμενη ενότητα, η δυναμική κίνηση, το animation της μορφής του σκάφους, βασίστηκε στην συνεχή εναλλαγή ψηφιογραφικών εικόνων (bitmaps) που είναι

αποθηκευμένα σε μια δομή πίνακα. Εδώ στην κλάση *EngineRun*, όπως φαίνεται στο κώδικα, δεν χρησιμοποιείται αυτή η τεχνική, αλλά έγινε μια προσπάθεια εφαρμογής του σχεδιασμού της διαφορετικής μορφής του αντικειμένου σε πραγματικό χρόνο, χρησιμοποιώντας απλά διανυσματικά γραφικά, δηλαδή κύκλους. Στην πραγματικότητα, είναι μια στοιχειώδης υλοποίηση μιας διαδεδομένης τεχνικής, που υλοποιείται συνήθως ενσωματώνοντας εξωτερικά API τρίτων δημιουργών όπως το OpenGL, που προσφέρει πολυσύνθετες δυνατότητες.

```
public class EngineRun extends Actor{
    private int r;
    private Random rnd= new Random();

    public EngineRun(int x, int y){
        r=5;
        super.x=x;
        super.y=y;    }

    public void update(){
        x-=6;     }

    public void draw(Canvas canvas){
        Paint paint=new Paint();
        int rand=(int)(rnd.nextDouble()*10);
        paint.setColor(Color.YELLOW+(int)(rnd.nextDouble()*
300));
        paint.setStyle(Paint.Style.FILL);
        canvas.drawCircle(x - r, y - r, r+rand, paint);
        canvas.drawCircle(x-r+2,y-r-4,r+rand,paint);
        canvas.drawCircle(x-r+4,y-r,r+rand,paint);     }
}
```

Στην προκείμενη περίπτωση, χρησιμοποιείται η κλάση *Paint*, που προσφέρει τη δυνατότητα σχεδιασμού δισδιάστατων σχημάτων. Έτσι για να παραχθεί η ψευδαίσθηση του καπνού, σχηματίζονται τρεις αλληλεπικαλυπτόμενοι, διαφορετικού μεγέθους και χρώματος κύκλοι κάθε φορά που καλείται η μέθοδος *draw()* της κλάσης (*r*, rand, paint.setColor()). Βέβαια, πέρα από την εναλλαγή της μορφής, υπάρχει και σταδιακή μετακίνηση κάθε στιγμιότυπου προς τα αριστερά (x), μέχρι την έξοδο τους από την οθόνη όπου και καταστρέφονται τα αντικείμενα όπως είδαμε στην κλάση *GamePanel*.

5.8 Οι κλάσεις BottomFence, TopFence, WalkingAllien και ElectricCrash

Οι υπόλοιπες κλάσεις της εφαρμογής, όπως αναφέρθηκε και παραπάνω, έχουν αναπτυχθεί με παρόμοιο τρόπο με τις κλάσεις *Background* και *Ufo*, οπότε δεν χρειάζεται να παρουσιαστεί η ανάλυση του κώδικά τους. Συγκεκριμένα, οι κλάσεις BottomFence και TopFence έχουν πολλά κοινά σημεία με την κλάση Background, ενώ οι κλάσεις WalkingAllien και ElectricCrash έχουν κοινά σημεία με την κλάση Ufo. Παρόλα αυτά, ο κώδικάς τους βρίσκεται στο Παράρτημα (σελ. 99) της εργασίας.

Στις κλάσεις BottomFence και TopFence δημιουργούνται και σχεδιάζονται οι πάνω και κάτω φράκτες, που αποτελούν τα “όρια” για την κίνηση του σκάφους και εάν χτυπήσει επάνω τους καταστρέφεται. Ο κώδικας μέσω του οποίου μεγαλώνουν ή μικραίνουν οι φράκτες βρίσκεται και περιγράφτηκε στην κλάση GamePanel. Η κλάση WalkingAllien δημιουργεί και σχεδιάζει τους εξωγήινους – “εχθρούς” του ufo που πρέπει να αποφύγει. Η κλάση ElectricCrash είναι υπεύθυνη για το εφέ της σύγκρουσης, όταν το ufo χτυπήσει πάνω σε κάποιον φράκτη ή σε έναν εξωγήινο. Επιπρόσθετα, στην κλάση ElectricCrash το animation δεν είναι επαναλαμβανόμενο, όπως στις Ufo και WalkingAllien, αλλά ολοκληρώνεται όταν εμφανιστούν από μια φορά όλα τα διαφορετικά πλαίσια που απαρτίζουν το εφέ του.

Οι ομοιότητες που έχουν αυτές οι κλάσεις μπορούν να φανούν χρήσιμες στη διδασκαλία της συγκεκριμένης εφαρμογής, επιλέγοντας έναν πιο εναλλακτικό τρόπο διδασκαλίας όπως σημειώθηκε στην εισαγωγή. Να δοθούν δηλαδή έτοιμοι οι κώδικες των κλάσεων Background και Ufo, να πραγματοποιηθεί ανάλυσή τους και εφόσον έχουν γίνει κατανοητοί από τους μαθητές, τότε να προγραμματίσουν τις υπόλοιπες τέσσερις κλάσεις.

ΕΠΙΛΟΓΟΣ

Σε αυτήν την εργασία παρουσιάστηκαν παραδείγματα εφαρμογών, ανεπτυγμένα στην πλατφόρμα Android, με το σκεπτικό να διευκολύνουν κυρίως τους καθηγητές στη διδασκαλία της Ενότητας 2α: «Ανάπτυξη Εφαρμογών (APPS) για Android», του βιβλίου του μαθήματος «Ειδικά Θέματα στον Προγραμματισμό Υπολογιστών», το οποίο διδάσκεται στη Γ’ τάξη του τομέα Πληροφορικής του Επαγγελματικού Λυκείου.

Δεν πραγματοποιήθηκε εντρύφηση σε θέματα θεωρίας τα οποία καλύπτονται από το βιβλίο, απεναντίας αναλύθηκε μόνο προγραμματιστικός κώδικας. Οι εφαρμογές παρατέθηκαν ολοκληρωμένες με μια σύντομη εξήγηση στη καθεμία, δίνοντας την ευχέρεια στους καθηγητές να τις μελετήσουν και να τις χρησιμοποιήσουν με όποιον διδακτικό τρόπο θεωρήσουν και επιλέξουν ως εκπαιδευτικά αξιοποιήσιμο.

Στις επόμενες σελίδες ακολουθεί το παράρτημα, όπου αναγράφεται ο κώδικας των εφαρμογών, συνοδευόμενος από σχολιασμό στα κύρια σημεία του.

ΠΑΡΑΡΤΗΜΑ

Στο παράρτημα αυτό παρατίθεται ο κώδικας ανάπτυξης των εφαρμογών.

~ MyFirstApp ~

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Στοίχιση σχετική με τα άλλα element του layout -->
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:background="#00ffea">

    <!-- Η εικόνα για τη Μέρα και τη Νύχτα -->
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="300dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:background="@drawable/day" />

    <!-- Το label για το "Καλημέρα" και το "Καληνύχτα" -->
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="70dp"
        android:layout_below="@+id/imageView"
        android:layout_centerHorizontal="true"
        android:text=" Καλημέρα! "
        android:background="#ffea00"
        android:textSize="50dp"
        android:textStyle="bold|italic"
        android:textColor="#313030" />

    <!-- Το κουμπί για να εμφανιστεί η Μέρα -->
    <Button
        android:id="@+id/buttonDay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:text="  MEPA  "
        android:textStyle="bold"
        android:textSize="35dp"
        android:background="#000000"
        android:textColor="#ffffff"
        android:onClick="setDay" />
```

```
    <!-- Το κουμπί για να εμφανιστεί η Νύχτα -->
    <Button
        android:id="@+id/buttonNight"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:text="  ΝΥΧΤΑ  "
        android:textStyle="bold"
        android:textSize="35dp"
        android:background="#000000"
        android:textColor="#ffffff"
        android:onClick="setNight" />
</RelativeLayout>
```

MainActivity.java

```
// Το κύριο Activity
public class MainActivity extends AppCompatActivity {
    // Ορισμός των χαρακτηριστικών της κλάσης
    private ImageView imgView;
    private TextView txtView;
    private Button btnDay, btnNight;

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

        // Ορισμός αντικειμένων, που είναι συνδεδεμένα με τα στοιχεία
του layout από το αρχείο activity_main.xml
        imgView=(ImageView) findViewById(R.id.imageView);
        txtView=(TextView) findViewById(R.id.textView);
        btnDay=(Button) findViewById(R.id.buttonDay);
        btnNight=(Button) findViewById(R.id.buttonNight);
    }

    // Μέθοδος όπου αλλάζει το σκηνικό σε Μέρα (event onClick του
κουμπιού buttonDay στο activity_main.xml)
    public void setDay(View view) {
        imgView.setBackgroundResource(R.drawable.day);
        txtView.setText(" Καλημέρα! ");
        txtView.setBackgroundColor(Color.YELLOW);
        txtView.setTextColor(Color.DKGRAY);
    }

    // Μέθοδος όπου αλλάζει το σκηνικό σε Νύχτα (event onClick του
κουμπιού buttonNight στο activity_main.xml)
    public void setNight(View view) {
        imgView.setBackgroundResource(R.drawable.night);
        txtView.setText(" Καληνύχτα! ");
        txtView.setBackgroundColor(Color.DKGRAY);
        txtView.setTextColor(Color.RED);
    }

    @Override
    public void onResume() { super.onResume(); }

    @Override
    public void onPause() { super.onPause(); }

    @Override
```

```
    public void onDestroy() { super.onDestroy(); }
}
```

~~~~~~

~ MyLightSensor ~

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Στοίχιση σχετική με τα άλλα element του layout -->
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:background="#00ffea">

    <!-- Η εικόνα για τη Μέρα και τη Νύχτα -->
    <ImageView
       android:id="@+id/imageView"
       android:layout_width="wrap_content"
       android:layout_height="300dp"
       android:layout_alignParentTop="true"
       android:layout_centerHorizontal="true"
       android:background="@drawable/day" />

    <!-- Το label για το "Καλημέρα" και το "Καληνύχτα" -->
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="70dp"
        android:layout_below="@+id/imageView"
        android:layout_centerHorizontal="true"
        android:text=" Καλημέρα! "
        android:background="#ffea00"
        android:textSize="50dp"
        android:textStyle="bold|italic"
        android:textColor="#313030" />

    <!-- Το κουμπί για την ενεργοποίηση/απενεργοποίηση της επικοινωνίας
με τον light sensor -->
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true"
        android:text=" Light Sensor Off "
        android:background="#000000"
        android:textColor="#ffffff"
        android:textStyle="bold"
        android:onClick="sensorOnOff" />
```
~~~~~~

```
    <!-- Ένα label για το caption "Light Value: " -->
    <TextView
        android:id="@+id/textLight"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:text="Light Value:  " />

    <!-- Το label που εμφανίζει την τιμή του light sensor -->
    <TextView
        android:id="@+id/textValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button"
        android:layout_toRightOf="@+id/textLight"
        android:layout_toEndOf="@+id/textLight"
        android:text=" ... " />
</RelativeLayout>
```

MainActivity.java

```
// Το κύριο Activity, που περιέχει και έναν Listener αισθητήρων
public class MainActivity extends AppCompatActivity implements
SensorEventListener {
    // Ορίσμός των χαρακτηριστικών της κλάσης
    private ImageView imgView;
    private TextView txtView, txtValue;
    private Button btn;
    private SensorManager sensorManager;
    private Sensor lightSensor;
    private boolean isStarted=false;

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

        // Ορισμός αντικειμένων, που είναι συνδεδεμένα με τα στοιχεία
του layout από το αρχείο activity_main.xml
        imgView=(ImageView) findViewById(R.id.imageView);
        txtView=(TextView) findViewById(R.id.textView);
        btn=(Button)findViewById(R.id.button);
        txtValue=(TextView)findViewById(R.id.textValue);
        // Ορισμός του αισθητήρα φωτός (lightSensor)
        sensorManager=(SensorManager)getSystemService(Context.SENSOR_SERV
ICE);
        lightSensor=sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

    // Μέθοδος ενεργοποίησης / απενεργοποίησης της επικοινωνίας με τον
light sensor (event onClick του κουμπιού button στο activity_main.xml)
    public void sensorOnOff(View view) {
        // Εάν είναι αληθής (επικοινωνία ενεργοποιημένη), τότε να
απενεργοποιηθεί, ...
        if (isStarted) {
            isStarted=false;
            sensorManager.unregisterListener(this);
            btn.setText(" Light Sensor Off ");
        }
        // αλλιώς να ενεργοποιηθεί
```

```
        else {
            isStarted=true;
            /* Η τρίτη παράμετρος μας λέει κάθε πότε (πόσο γρήγορα) θα
καταγράφει τιμές ο sensor. Εδώ με βάση μια νορμάλ ταχύτητα */
            sensorManager.registerListener(this, lightSensor,
SensorManager.SENSOR_DELAY_NORMAL);
            btn.setText(" Light Sensor On ");
        }
    }

    // Μέθοδος όπου καλείται όταν συμβαίνουν αλλαγές στις τιμές που
διαβάζει ο light sensor
    @Override
    public void onSensorChanged(SensorEvent event) {
        // Απεικόνιση της τιμής που διάβασε ο sensor σε ένα text
        txtValue.setText(Float.toString(event.values[0]));
        // Εάν η τιμή είναι μεγαλύτερη του 10, τότε αλλαγή του σκηνικού
σε Μέρα, ...
        if (event.values[0]>10) {
            imgView.setBackgroundResource(R.drawable.day);
            txtView.setText(" Καλημέρα! ");
            txtView.setBackgroundColor(Color.YELLOW);
            txtView.setTextColor(Color.DKGRAY);
        }
        // αλλιώς αλλαγή του σκηνικού σε Νύχτα
        else {
            imgView.setBackgroundResource(R.drawable.night);
            txtView.setText(" Καληνύχτα! ");
            txtView.setBackgroundColor(Color.DKGRAY);
            txtView.setTextColor(Color.RED);
        }
    }

    // Μέθοδος όπου καλείται όταν συμβαίνουν αλλαγές στην ακρίβεια του
light sensor
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) { }
}
```

~~~~~~

~ MyProximitySensor ~

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Στοίχιση σχετική με τα άλλα element του layout -->
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:context=".MainActivity" >
    <!-- Η εικόνα για τη Μέρα και τη Νύχτα -->
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
```
~~~~~~

```
        android:layout_centerHorizontal="true"
        android:src="@drawable/happykitty" />
</RelativeLayout>
```

MainActivity.java

```
// Το κύριο Activity, που περιέχει και έναν Listener αισθητήρων
public class MainActivity extends Activity implements
SensorEventListener {
    // Ορίσμός των χαρακτηριστικών της κλάσης
    private SensorManager mSensorManager;
    private Sensor proxSensor;
    private ImageView imgView;
    private MediaPlayer meowSound, roarSound;

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

        // Ορισμός αντικειμένων, που είναι συνδεδεμένα με τα στοιχεία
του layout από το αρχείο activity_main.xml
        imgView = (ImageView) findViewById(R.id.imageView1);
        // Ορισμός του αισθητήρα απόστασης (proxSensor)
        mSensorManager = (SensorManager)
getSystemService(SENSOR_SERVICE);
        proxSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
    }

    // Μέθοδος onResume από τον κύκλο του activity
    @Override
    protected void onResume() {
        super.onResume();
        // Ενεργοποίηση της επικοινωνίας με τον proximity sensor
        mSensorManager.registerListener(this, proxSensor,
SensorManager.SENSOR_DELAY_NORMAL);
    }

    // Μέθοδος onPause από τον κύκλο του activity
    @Override
    protected void onPause() {
        super.onPause();
        // Απενεργοποίηση της επικοινωνίας με τον proximity sensor
        mSensorManager.unregisterListener(this);
    }

    // Μέθοδος όπου καλείται όταν συμβαίνουν αλλαγές στις τιμές που
διαβάζει ο proximity sensor
    @Override
    public void onSensorChanged(SensorEvent event) {
        /* Εάν η τιμή είναι 0, δηλαδή εάν έχει πλησιάσει πολύ ένα
αντικείμενο, τότε φόρτωσε την εικόνα της τίγρης και παίξε τον ήχο της
τίγρης... */
        if (event.values[0] == 0) {
            imgView.setImageResource(R.drawable.angrytiger);
            // Εδω ορίζεται και ξεκινά η μουσική .
            roarSound = MediaPlayer.create(this, R.raw.tigerroar);
            roarSound.start();
        }
        // αλλιώς, φόρτωσε την εικόνα της γάτας και παίξε τον ήχο της
γάτας
```

```
        else {
            imgView.setImageResource(R.drawable.happykitty);
            meowSound = MediaPlayer.create(this, R.raw.catmeow);
            meowSound.start();
        }
    }

    // Μέθοδος όπου καλείται όταν συμβαίνουν αλλαγές στην ακρίβεια του
proximity sensor
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }

    // Μέθοδος onDestroy από τον κύκλο του activity
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Απελευθέρωση της μνήμης από τον mediaplayer
        roarSound.stop();
        roarSound.release();
        meowSound.stop();
        meowSound.release();
    }
}
```

~~~~~~

~ MySendMessage ~

AndroidManifest.xml

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

    <!-- Ορισμός των απαραίτητων δικαιωμάτων πρόσβασης σε υπηρεσίες του
τηλεφώνου -->
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.SEND_SMS" />

    <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" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Main2Activity"
            android:label="@string/title_activity_main2"
```
~~~~~~

```
            android:theme="@style/AppTheme.NoActionBar" >
        </activity>
    </application>
</manifest>
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Στοίχιση σχετική με τα άλλα element του layout -->
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:background="#870f0f">

    <!-- Η εικόνα SOS -->
    <ImageView
        android:layout_width="350dp"
        android:layout_height="150dp"
        android:id="@+id/imageView"
        android:background="@drawable/sos"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <!-- Το label "Don't panic! Stay calm!" -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Don&apos;t panic! Stay calm!"
        android:id="@+id/textView1"
        android:textSize="25dp"
        android:textStyle="bold|italic"
        android:textColor="#ffffff"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="160dp" />

    <!-- Το κουμπί για την αποστολή του μηνύματος SOS -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="S.O.S."
        android:id="@+id/SOSbtn"
        android:textSize="30dp"
        android:textStyle="bold"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="220dp" />

<!-- Το κουμπί για την μεταφορά σε νέο activity για έλεγχο στοιχείων -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ABORT"
        android:id="@+id/abortBtn"
        android:textSize="30dp"
        android:textStyle="bold"
```

```
        android:onClick="checkAbort"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="300dp" />
    <!-- Το κουμπί για την έξοδο από την εφαρμογή -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="EXIT"
        android:id="@+id/exitBtn"
        android:textSize="25dp"
        android:textStyle="bold|italic"
        android:onClick="exitApp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />
</RelativeLayout>
```

MainActivity.java

```
// Το πρώτο κύριο Activity
public class MainActivity extends AppCompatActivity implements
LocationListener {
    //Ορίσμός των χαρακτηριστικών της κλάσης
    Button sendBtn, abortBtn, exitBnt;
    String phoneNo = "69......";  //Εδώ μπαίνει ο αριθμός κινητού που θα
σταλούν τα μηνύματα
    String msgSOS;
    String toastMsg;
    //Ορισμός των στοιχείων για την υπηρεσία GPS
    static String longitude;
    static String latitude;
    LocationManager locManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Ορισμός αντικειμένων, που είναι συνδεδεμένα με τα στοιχεία του
layout από το αρχείο activity_main.xml
        sendBtn = (Button) findViewById(R.id.SOSbtn);
        //abortBtn = (Button) findViewById(R.id.abortBtn);  έχει δηλωθεί
στο onClick property στο activity_main.xml
        //exitBnt = (Button) findViewById(R.id.exitBtn);  έχει δηλωθεί
στο onClick property στο activity_main.xml
        locManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
        //Έλεγχος των δικαιωμάτων πρόσβασης στην υπηρεσία GPS και
μεταφορά δεδομένων στον Location Manager
        if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED
                && ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {
locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
this);
            return;
        }
        else
locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
```

```
this);

        //Ορισμός ενός ClickListener στο κουμπί SOS ώστε να εντοπίσει το
event του click και να τρέξει την onClick
        sendBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                if (longitude != null) {
                    // Δημιουργία του μηνύματος SOS με τις συντεταγμένες
που ελήφθησαν από το GPS
                    msgSOS = "Χρειάζομαι βοήθεια! Βρίσκομαι στην
τοποθεσία με γεωγραφικό μήκος: " + longitude + "και γεωγραφικό πλάτος: "
+ latitude;
                    toastMsg = "Μήνυμα εστάλη!!";
                }
                else {
                    /*Εάν το GPS δεν δίνει στίγμα, δηλαδή εάν οι
μεταβλητές longitude και latitude δεν έχουν πάρει τιμή τότε θα σταλεί
ένα απλό μήνυμα βοήθειας*/
                    msgSOS = "Χρειάζομαι βοήθεια!";
                    toastMsg = "Μήνυμα εστάλη χωρίς την τοποθεσία.";
                }
                //Εκτέλεση της μεθόδου για αποστολή μηνύματος
                sendMsgSOS(phoneNo, msgSOS);
            }
        });
    } //Τέλος της onCreate

    //Μέθοδος για την αποστολή μηνύματος
    private void sendMsgSOS(String phoneNo, String msgSOS) {
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNo, null, msgSOS, null, null);
        // Εμφάνιση μηνύματος επαλήθευσης
        Toast.makeText(getBaseContext(), toastMsg,
Toast.LENGTH_LONG).show();
    }

    //Αποθήκευση σε μεταβλητές, των συντεταγμένων κάθε φορά που
εντοπίζει αλλαγή θέσης το GPS
    @Override
    public void onLocationChanged(Location location) {
        longitude = String.valueOf(location.getLongitude());
        latitude = String.valueOf(location.getLatitude());
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle
extras) { }

    @Override
    public void onProviderEnabled(String provider) { }

    @Override
    public void onProviderDisabled(String provider) { }

    //Μέθοδος που ξεκινά ένα Intent και μεταφέρεται σε νέο Activity,
όταν πατηθεί το κουμπί Abort
    public void checkAbort(View view) {
        // Δημιουργία Intent με τις δύο βασικές κλάσεις
        Intent intent = new Intent(this, Main2Activity.class);
        // Εκκίνηση της νέας διαδικασίας
        startActivity(intent);
    }

    //Μέθοδος για την έξοδο από την εφαρμογή (κουμπί Exit)
```

```
    public void exitApp(View view) {
        //Απενεργοποίηση του GPS
        if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED
                && ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION)
                != PackageManager.PERMISSION_GRANTED)
            locManager.removeUpdates(this);
        else locManager.removeUpdates(this);
        //Έξοδος από την εφαρμογή
        exit(0);
    }
}
```

activity_main2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Στοίχιση σχετική με τα άλλα element του layout -->
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context="com.example.kardara_alert.Main2Activity"
    android:background="#86febc">

    <!-- Το label "Ελεγχος Στοιχείων" -->
    <TextView android:text="Ελεγχος Στοιχείων"
android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:textSize="20dp"
        android:id="@+id/txt1"
        android:layout_centerHorizontal="true"
        android:textStyle="bold|italic" />

    <!-- Το label "Username" -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Username:  "
        android:id="@+id/txt2"
        android:textSize="20dp"
        android:textColor="#000000"
        android:background="#dc8efa"
        android:textStyle="bold"
        android:layout_below="@+id/txt1"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="48dp" />

    <!-- Το label "Password" -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Password:  "
```

```
        android:id="@+id/txt3"
        android:textSize="20dp"
        android:layout_marginTop="48dp"
        android:textColor="#000000"
        android:background="#dc8efa"
        android:textStyle="bold"
        android:layout_below="@+id/txt2"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <!-- Το Input TextBox για εισαγωγή του Username -->
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/txtUn"
        android:layout_alignBottom="@+id/txt2"
        android:layout_marginLeft="120dp"
        android:layout_marginStart="120dp"/>

    <!-- Το Input TextBox για εισαγωγή του Password -->
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:ems="10"
        android:id="@+id/txtPw"
        android:layout_alignBottom="@+id/txt3"
        android:layout_alignLeft="@+id/txtUn"
        android:layout_alignStart="@+id/txtUn" />

    <!-- Το κουμπί για την επιβεβαίωση των στοιχείων και την αποστολή
μηνύματος False alarm -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CHECK &amp;amp; SEND"
        android:id="@+id/btnCheck"
        android:layout_below="@+id/txt3"
        android:layout_marginTop="48dp"
        android:textStyle="bold|italic"
        android:background="#f7d61d"
        android:textSize="20dp"
        android:layout_centerHorizontal="true"
        android:clickable="false"
        android:width="200dp" />

    <!-- Η εικόνα Check -->
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView2"
        android:layout_below="@+id/btnCheck"
        android:layout_centerHorizontal="true"
        android:background="@drawable/check" />
</RelativeLayout>
```

Main2Activity.java

```
// Το δεύτερο Activity
public class Main2Activity extends AppCompatActivity {
    //Ορίσμός των χαρακτηριστικών της κλάσης
    Button checkBtn;
```

```
    String phoneNo = "69......";
    String msgAbort = "Άκυρος ο συναγερμός. Είμαι εντάξει.";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
        //Ορισμός αντικείμενου τύπου button, που είναι συνδεδεμένο με το
κουμπί btnCheck του layout από το αρχείο activity_main2.xml
        checkBtn = (Button) findViewById(R.id.btnCheck);
        //Ορισμός ενός ClickListener στο κουμπί btnCheck ώστε να
εντοπίσει το event του click και να τρέξει την onClick
        checkBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                /* Ορισμός δύο αντικειμένων τύπου EditText, που είναι
συνδεδεμένα με τα στοιχεία του layout txtUn και txtPw από το αρχείο
activity_main2.xml*/
                EditText username = (EditText) findViewById(R.id.txtUn);
                EditText password = (EditText) findViewById(R.id.txtPw);
                //Εδώ γίνεται ο έλεγχος για τη σωστή εισαγωγή στοιχείων
                if (username.getText().toString().equals("m") &&
password.getText().toString().equals("k")) {
                    //Εκτέλεση της μεθόδου για αποστολή μηνύματος
                    sendMsgAbort(phoneNo, msgAbort);
                }
                else {
                    //Εμφάνιση μηνύματος στην περίπτωση εισαγωγής λάθος
στοιχείων
                    Toast.makeText(getBaseContext(), "Δώσατε λάθος
username ή/και password.", Toast.LENGTH_LONG).show();
                }
            }
        });
    } // Τέλος της onCreate

    //Μέθοδος για την αποστολή μηνύματος
    private void sendMsgAbort(String phoneNo, String msgAbort) {
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNo, null, msgAbort, null, null);
        // Εμφάνιση μηνύματος επαλήθευσης
        Toast.makeText(getBaseContext(), "Μήνυμα εστάλη!!",
Toast.LENGTH_LONG).show();
    }
}
```

~~~~~~

~ MyGame ~

Game.java

```
// ************************** Η κλάση Game ****************************
// Η κλάση Game είναι το κύριο Activity στο οποίο γίνεται ο καθορισμός
των αρχικών ρυθμίσεων στην οθόνη, ενεργοποιείται το μουσικό υπόβαθρο και
αποστέλλεται ως περιεχόμενο (context) αυτό το κύριο Activity στο νέο
στιγμιότυπο της Gamepanel που δημιουργείται και δεν είναιτίποτα άλλο
παρά ένα νέο SurfaceView.
// ********************************************************************
```
~~~~~~

```
public class Game extends Activity {
    // Ορισμός αντικειμένου mediaPlayer για μουσική υπόβαθρου
    MediaPlayer backgroundMusic;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Απενεργοποίηση της γραμμής του τίτλου
        requestWindowFeature(Window.FEATURE_NO_TITLE);

        // Ορισμός σε πλήρη μέγεθος η οθόνη
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);

        // Εδώ ορίζεται και ξεκινά η μουσική στο υπόβαθρο.
        backgroundMusic = MediaPlayer.create(Game.this, R.raw.bg);
        backgroundMusic.setLooping(true);  // συνεχής επανάληψη
        backgroundMusic.setVolume(0.1f, 0.1f);  // ορισμός έντασης ήχου
        backgroundMusic.start();

        // Καθορισμός του περιεχόμενου της οθόνης δημιουργώντας ένα
στιγμιότυπο της κλάσης
        setContentView(new GamePanel(this));
    }

    @Override
    protected void onDestroy() {
        // Απελευθέρωση της μνήμης από τον mediaplayer
        super.onDestroy();
        if(backgroundMusic!=null) {
            backgroundMusic.stop();
            backgroundMusic.release();
        }
    }
}
```

GamePanel.java

```
// ************************ Η κλάση GamePanel ************************
// Η Κλάση GamePanel αποτελεί μια υλοποίηση της κλάσης SurfaceView με
προσαύξηση των δυνατοτήτων της ώστε να δέχεται συμβάντα, όπως η επαφή με
την επιφάνεια της οθόνης(touch). Ουσιαστικά στην κλάση αυτή
δημιουργούνται στιγμιότυπα από όλες τις υπόλοιπες κλάσεις που
συμμετέχουν στο παιχνίδι και καθορίζεται η συμπεριφορά τους. Επίσης εδώ
υπάρχει ο κώδικας που ξεκινά ένα νέο παιχνίδι καθώς και ο κώδικας με τις
διαδικασίες που καθορίζουν την εξέλιξη και τον τερματισμό του
παιχνιδιού.
// ******************************************************************

public class GamePanel extends SurfaceView implements
SurfaceHolder.Callback {
    // Τα χαρακτηριστικά της οθόνης και η ταχύτητα κυλισης
    public static final int WIDTH = 840;
    public static final int HEIGHT = 420;
    public static final int ROLLSPEED = -5;

    //  Το βασικό thread εκτέλεσης του game στη μνήμη
    private MainThread thread;

    // Το αντικείμενο που περιέχει το υπόβαθρο που συνεχώς κινείται
    private Background bg;

    // Ο παίκτης με το sprite ufo
    private Ufo ufo;
```

```
    // Ο πίνακας Λίστα για το εφέ του καπνού της μηχανής (περιέχονται όλες οι εικόνες του sprite που θα λειτουργήσουν ως animation)
    private ArrayList<EngineRun> engineRun;
    private long engineStartTime;  // μεταβλητή χρονου έναρξης του timer για την δημιουργία αντικειμένων κύκλων του καπνού της μηχανής

    // Ο πίνακας με τους εξωγήινους (περιέχονται όλες οι εικόνες του sprite που θα λειτουργήσουν ως animation)
    private ArrayList<WalkingAllien> walkingAlliens;
    private long walkingAllienstartTime;  // μεταβλητή για την έναρξη του χρονομέτρου των εξωγήινων

    // Πίνακες λίστες για τους φράκτες πάνω και κάτω (περιέχονται όλες οι εικόνες που θα λειτουργήσουν ως animation)
    private ArrayList<TopFence> topFences;
    private ArrayList<BottomFence> botFences;
    private int maxFenceHeight;  // μεγιστο υψος φράκτη
    private int minFenceHeigh;  // ελάχιστο ύψος φράκτη
    private boolean topDown = true;  // λογικές μεταβλητές εναλλαγής από αύξηση σε μείωση του φράκτη
    private boolean botDown = true;

    private int difficulty = 10;  // βαθμός δυσκολίας (π.χ. για την ταχύτητα του παιχνιδιού)
    private Random rnd = new Random();  // μηχανή παραγωγής τυχαίων αριθμών (για το ύψος των φραχτών, για το που θα εμφανίζονται τα allien)
    private boolean newGameCreated;  // η λογική μεταβλητή για την δημιουργία ή όχι νέου παιχνιδιού
    private int best;  // καλύτερο σκορ

    // ElectricCrash – το αντικείμενο για την εξαφάνιση και έκρηξη του σκάφους
    private ElectricCrash electricCrash;

    private long startReset;  // μεταβλητές για τη μέτρηση χρόνου σε διάφορα σημεία του παιχνιδιού
    private boolean reset;
    private boolean dissapear;  // μεταβλητή για την εξαφάνιση του ufo όταν χτυπήσει κάτι
    private boolean started;

    // Αντικείμενα που χρησιμεύουν για την εκτέλεση των εφέ ήχου (παίζει από lollipop και πάνω, σε ποιο παλιές εκδόσεις ήταν πιο πολύπλοκες εντολές)
    private SoundPool sounds;
    private int sndmale;

    // Ο Κατασκευαστής της κλάσης GamePanel
    public GamePanel(Context context) {
        // προωθεί στη μητρική κλάση surfaceView το περιεχόμενο της GamePanel
        super(context);

        // Προσθήκη των callbacks επάνω στο SurfaceHolder ώστε να διαχειρίζεται events πάνω στο touch screen
        getHolder().addCallback(this);
        // Εστίαση πάνω στο συγκεκριμένο surfaceView (εδώ μόνο 1, αλλά θα μπορούσαν να υπάρχουν κι άλλα)
        setFocusable(true);
    }

    @Override
    // Υποχρεωτικά πρέπει να υπάρχει έστω και χωρίς περιεχόμενο
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {  }
```

```
    @Override
    // Κώδικας όταν καταστρέφεται η Επιφάνεια
    public void surfaceDestroyed(SurfaceHolder holder) {
        // Εδώ σταματά το Game Loop
        boolean retry = true;
        sounds.release();
        // Επαναληπτικά γίνεται προσπάθεια μέχρι να επιτευχθεί ο
τερματισμός του thread του παιχνιδιού
        while (retry) {
            try {
                thread.setRunning(false);
                thread.join();
                retry = false;
                thread=null;
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    // Ενσωμάτωση δυνατοτήτων για τα εφέ ήχου που ισχύουν μόνο στην
έκδοση του Android 5.0 Lollipop
    @TargetApi(Build.VERSION_CODES.LOLLIPOP)

    @Override
    // Κατά την δημιουργία της επιφάνειας της κλάσης ορίζονται και
δημιουργούνται τα νέα στιγμιότυπα από κάθε κλάση που θα συμμετέχουν στο
παιχνίδι
    public void surfaceCreated(SurfaceHolder holder) {
        bg = new Background(BitmapFactory.decodeResource(getResources(),
R.drawable.road));  // Υπόβαθρο
        ufo = new Ufo(BitmapFactory.decodeResource(getResources(),
R.drawable.ship), 70, 40, 6);  // Το σκάφος
        engineRun = new ArrayList<EngineRun>();  // Οι καπνοί που
δηλώνουν την λειτουργία του σκάφους
        engineStartTime = System.nanoTime();  // Μεταβλητή για τη
χρονομέτρηση της αναγέννησης αντικειμένων της κλάσης EngineRun
        walkingAlliens = new ArrayList<WalkingAllien>();  // Οι
εξωγήινοι
        walkingAllienstartTime = System.nanoTime(); // Μεταβλητή για τη
χρονομέτρηση της αναγέννησης αντικειμένων της  κλάσης WalkingAlliens
        topFences = new ArrayList<TopFence>();  // Στιγμιότυπα της
κλάσης για τον επάνω φράκτη
        botFences = new ArrayList<BottomFence>();  // Στιγμιότυπα της
κλάσης για τον κάτω φράκτη

        // Δημιουργία αντικειμένων για την διαχείριση των εφέ ήχου
βασισμένη στην κλάση SoundPool, όπως περιγράφεται στο Android Developer,
και μόνο για Android Lollipop και μετά
        AudioAttributes.Builder attributeBuilder = new
AudioAttributes.Builder();
        attributeBuilder.setUsage(AudioAttributes.USAGE_GAME);
        final AudioAttributes.Builder builder =
attributeBuilder.setContentType(AudioAttributes.CONTENT_TYPE_SONIFICATIO
N);
        AudioAttributes attributes = attributeBuilder.build();
        SoundPool.Builder soundBuilder = new SoundPool.Builder();
        soundBuilder.setAudioAttributes(attributes);
        sounds = soundBuilder.build();
        sndmale = sounds.load(this.getContext(), R.raw.crash, 1);  //
Εδώ φορτώνεται από τα resources ο ήχος σύγκρουσης
```

```
        thread = new MainThread(getHolder(), this);  // Δημιουργείται το
thread στην μνήμη
        thread.setRunning(true);  // Αρχικοποιείται η μεταβλητή
εκκίνησης του thread
        thread.start();  // // Καλείται η διαδικασία που ξεκινά την
εκτέλεση του περιεχόμενου του thread (**GameLoop**)
    }

    @Override
    // Διαχείριση των συμβάντων πάνω στην οθόνη της συσκευής
    public boolean onTouchEvent(MotionEvent event) {
        // Εδώ συλλαμβάνεται το γεγονός όταν κάποιος ακουμπά την οθόνη
της συσκευής
        if (event.getAction() == MotionEvent.ACTION_DOWN ) {
            // Ξεκινά η κίνηση του σκάφους
            if (!ufo.isStartAnimate()&& newGameCreated && reset) {
                ufo.setStartAnimate(true);
                ufo.setUp(true);  // Πάει το ufo προς τα πάνω
            }
            if(ufo.isStartAnimate()){  // Αν έχει ήδη ξεκινήσει τότε
επανέφερε τις μεταβλητές στην κανονική λειτουργία
                if(!started)started=true;
                reset=false;
                ufo.setUp(true);
            }
            return true;
        }
        // Εδώ συλλαμβάνεται το γεγονός όταν κάποιος σταματά να ακουμπά
την οθόνη της συσκευής
        if (event.getAction() == MotionEvent.ACTION_UP) {
            ufo.setUp(false);
            return true;
        }
        return super.onTouchEvent(event);  // Επιστρέφει το τύπο του
γεγονότος στην μητρική κλάση
    }

    // Η διαδικασία update που καλείται σε κάθε κύκλο του thread (Game
Loop). Σκοπός της είναι να ενημερώνει σε κάθε κύκλο την συμπεριφορά των
αντικειμένων που βρίσκονται μέσα στο παιχνίδι.
    public void update() {
        if (ufo.isStartAnimate()) {
            // Έλεγχος αν έχουν σχεδιαστεί οι φράκτες
            if(botFences.isEmpty() || topFences.isEmpty()){
                ufo.setStartAnimate(false);
                return;
            }

            // Καλούνται οι διαδικασίες ενημέρωσης του Background και
του Ufo
            bg.update();
            ufo.update();

            // Καθορισμός του μέγιστου και ελάχιστου ύψους του φράκτη
καθώς εξελίσσεται το παιχνίδι και το σκορ
            maxFenceHeight = 30 + ufo.getScore() / difficulty;
            if (maxFenceHeight == HEIGHT / 4) maxFenceHeight = HEIGHT /
4;
            minFenceHeigh = 5 + ufo.getScore() / difficulty;

            // Εδώ γίνεται έλεγχος για την περίπτωση που το σκάφος
ακουμπήσει στον πάνω και κάτω φράκτη (είναι array list, οπότε ελέγχει με
όλα τα αντικείμενα της λίστας)
```

```
            for(int i=0;i<topFences.size();i++){
                if(collission(topFences.get(i),ufo)) {
                    ufo.setStartAnimate(false);  // Σταματάει το
animation
                }
            }
            for(int i=0;i<botFences.size();i++){
                if(collission(botFences.get(i),ufo)) {
                    ufo.setStartAnimate(false);
                }
            }

            // Αντίστοιχα καλείται η διαδικασία ενημέρωσης του πάνω και
κάτω φράκτη
            this.updateTopFence();
            this.updateBottomFence();

            // Με βάση το χρόνο δημιουργούνται και τοποθετούνται τα
αντικείμενα καπνού της κλάσης EngineRun
            long engineElapsed = (System.nanoTime() - engineStartTime) /
1000000;  // millisecond
            if (engineElapsed > 120) {  // Εάν έχουν περάσει 120
millisecond δείξε το επόμενο κυκλάκι του καπνου
                engineRun.add(new EngineRun(ufo.getX(), ufo.getY() +
10));
                engineStartTime = System.nanoTime();
            }

            // Εδώ γίνεται ο καθαρισμός της λίστας από τα αντικείμενα
καπνού που έχουν βγει εκτός οθόνης
            for (int i = 0; i < engineRun.size(); i++) {
                engineRun.get(i).update();
                if (engineRun.get(i).getX() < -10) {
                    engineRun.remove(i);
                }
            }

            // Εδώ δημιουργούνται με βάση το χρόνο τα αντικείμενα
εξωγήινοι. Η θέση των αντικειμένων εξαρτάται από την εξέλιξη και το σκορ
του παιχνιδιού. Στην αρχή εμφανίζονται πιο αραιά, κι όσο περνάει ο
χρόνος πιο συχνά)
            long WalkingAllienElapsed = (System.nanoTime() -
walkingAllienstartTime) / 1000000;
            if (WalkingAllienElapsed > 2000 - ufo.getScore() / 4) {
                walkingAlliens.add(new
WalkingAllien(BitmapFactory.decodeResource(getResources(),
R.drawable.alien),
                        (int) (8* WIDTH / 9* rnd.nextDouble())+10,
HEIGHT, 114, 84, ufo.getScore(), 8));
                walkingAllienstartTime = System.nanoTime();
            }
            // Πρέπει όταν τα αντικείμενα εξωγήινοι έχουν βγει εκτός
οθόνης διαγράφονται από την λίστα, έτσι ώστε να αποδεσμεύεται η μνήμη.
            for (int i = 0; i < walkingAlliens.size(); i++) {
                walkingAlliens.get(i).update();
                if (walkingAlliens.get(i).getY() < -10) {
                    walkingAlliens.remove(i);
                    break;
                }
                // Επίσης πρέπει να ελεγχθεί και η περίπτωση που ακουμπά
με το σκάφος
                if (collission(walkingAlliens.get(i), ufo)) {
                    ufo.setStartAnimate(false);
```

```
                    walkingAlliens.remove(i);
                    break;
                }
            }
        }
        // Σε περίπτωση που δεν κινείται το ufo, τότε:
        else {
            ufo.resetThrotle();
            // 1. Αν έχει συγκρουσθεί με κάποιο άλλο αντικείμενο τότε
πρέπει να εμφανιστεί το εφέ της σύγκρουσης και να εξαφανιστεί το σκάφος
            if(!reset){
                newGameCreated=false;
                startReset=System.nanoTime();  // Αρχικοποιείται ο
χρόνος του παιχνιδιού για να μετράει ο χρόνος από την αρχή
                reset=true;
                dissapear=true;  // Εξαφανίζεται το ufo
                electricCrash=new
ElectricCrash(BitmapFactory.decodeResource(getResources(),R.drawable.lig
htning),ufo.getX(),ufo.getY()-30,100,100,25);
            }
            electricCrash.update();  // Εμφανίζεται το εφέ της
σύγκρουσης
            long resetElapsed=(System.nanoTime()-startReset)/1000000;
            // 2. και επίσης να επαναρχικοποιηθεί το παιχνίδι ώστε να
ξεκινήσει ξανά
            if(resetElapsed>2500 && !newGameCreated){
                newGame(); }
        }
    }

    @Override
    // Η διαδικασία που σχεδιάζει όλες τις αλλαγές που έχουν συμβεί λόγω
των αλλαγών που προκαλούνται από την συμπεριφορά του κάθε αντικειμένου
στην οθόνη
    public void draw(Canvas canvas) {
        // Υπολογίζονται οι παράγοντες αύξησης ή μείωσης των διαστάσεων
της οθόνης του παιχνιδιού με βάση τις διαστάσεις της οθόνης της
συσκευής. Αλλάζεται έτσι το μέγεθος της οθόνης ώστε να καλύπτεται πλήρως
και να είναι ακριβής ο σχεδιασμός.
        float scaleFactorX = getWidth() / (GamePanel.WIDTH*1.0f);
        float scaleFactorY = getHeight() / (GamePanel.HEIGHT*1.0f);
        // Χρησιμέυει για τον αρχικό έλεγχο κατά την δοκιμή του κώδικα
System.out.print(" SFX:" +scaleFactorX + " ORWIDTH:"+getWidth()+" SFY:"
+scaleFactorY+" ORHEIGHT" +getHeight()+"\n");

        if (canvas != null) {
            // Εδώ αποθηκεύεται η κατάσταση του καμβά και γίνεται η
αλλαγή βάση των παραγόντων που έχουν υπολογισθεί
            final int savedState = canvas.save();
            canvas.scale(scaleFactorX, scaleFactorY);
            //Κατόπιν καλούνται οι διαδικασίες της κάθε κλάσης που
σχεδιάζουν τον καμβά με την ενημερωμένη συμπεριφορά του κάθε
αντικειμένου
            bg.draw(canvas);
            if(!dissapear){
                ufo.draw(canvas);
                for (EngineRun engR : engineRun) {
                    engR.draw(canvas); }
            }
            for (WalkingAllien walkingA : walkingAlliens) {
                walkingA.draw(canvas); }
            for (TopFence topF:topFences){
```

```
                topF.draw(canvas); }
            for(BottomFence botF:botFences){
                botF.draw(canvas); }
            if(started){
                electricCrash.draw(canvas); }
            // Εδώ καλείται η διαδικασία drawtext που σχεδιάζει τα
αρχικά μηνύματα καθώς και το κείμενο με το σκορ και την εξέλιξη του
παιχνιδιού
            drawText(canvas);
            // και επανέρχεται η οθόνη στην αποθηκευμένη κατάσταση, με
το αρχικό ύψος και πλάτος
            canvas.restoreToCount(savedState);
        }
    }

    // Η διαδικασία που ελέγχει αν έχει συμβεί σύγκρουση με κάποιο άλλο
αντικείμενο
    public boolean collission(Actor a1, Actor a2) {
        if (Rect.intersects(a1.colRect(), a2.colRect())) {
            sounds.play(sndmale, 1, 1, 0, 0, 1);
            return true;
        }
        return false;
    }

    // Διαδικασίες που ενημερώνουν την θέση και το μέγεθος και
κατασκευάζουν το πάνω και κάτω φράκτη κατά την εξέλιξη του παιχνιδιού.
    public void updateTopFence() {
        // Το xpos, hpos θα χρησιμοποιηθούν για να χτιστεί το επόμενο
κομματάκι του φράκτη το topFences είναι array, οπότε η get επιστρέφει το
τελευταίο κομμάτι ((το τελευταίο δεξιά ώστε να ξέρει που έχει τελειώσει
ο φράχτης για να χτίσει το επόμενο κομμάτι getx + 20 (20 pixel το
μέγεθος του κομματιού))
        int xpos = topFences.get(topFences.size() - 1).getX() + 20;
        int hpos = 0;

        // Για να μην είναι προβλέψιμη η αλλαγή του φράκτη ανά τακτά
διαστήματα, σημειώνεται μια μεγάλη αλλαγή (όταν η ακέραια διαίρεση του
σκορ με το 50 έχει υπόλοιπο 0)
        if (ufo.getScore() % 50 == 0) {
            int rand = (int) (rnd.nextDouble() * maxFenceHeight) + 1;
            topFences.add(new
TopFence(BitmapFactory.decodeResource(getResources(),
R.drawable.barrier), xpos, 0, rand));
        }

        // Εδώ σχεδιάζεται ο φράκτης και γίνεται έλεγχος ώστε σταδιακά
να αυξομειώνεται το ύψος του φράκτη και επίσης σε περίπτωση που είναι
έξω από την οθόνη να διαγράφονται τα στιγμιότυπα που δεν χρειάζονται
        for (int i = 0; i < topFences.size(); i++) {
            topFences.get(i).update();
            if (topFences.get(i).getX() < -20) {
                topFences.remove(i);
                if (topFences.get(topFences.size() - 1).getHeight() >=
maxFenceHeight) {
                topDown = false;  }
                if (topFences.get(topFences.size() - 1).getHeight() <=
minFenceHeigh) {
                topDown = true;  }
                if (topDown) {
                hpos = topFences.get(topFences.size() - 1).getHeight() +
1;
                topFences.add(new
```

```
TopFence(BitmapFactory.decodeResource(getResources(),
R.drawable.barrier), xpos, 0, hpos));
                }
                else {
                hpos = topFences.get(topFences.size() - 1).getHeight() -
1;
                topFences.add(new
TopFence(BitmapFactory.decodeResource(getResources(),
R.drawable.barrier), xpos, 0, hpos));
                 }
            }
        }
    }

    // Αντίστοιχα η διαδικασία για τον κάτω φράκτη
    public void updateBottomFence() {
        int xpos = botFences.get(botFences.size() - 1).getX() + 20;
        int ypos = botFences.get(botFences.size() - 1).getY() + 1;
        Bitmap cImage=BitmapFactory.decodeResource(getResources(),
R.drawable.barrier);

        // Εδώ η σημαντική αλλαγή που συμβαίνει ανά τακτά χρονικά
διαστήματα διαφοροποιείται από τον πάνω  φράκτη (όταν η ακέραια διαίρεση
με το 40 έχει υπόλοιπο 0)
        if (ufo.getScore() % 40 == 0) {
            int rand = (int) (rnd.nextDouble() * maxFenceHeight) +
(HEIGHT - maxFenceHeight);
            botFences.add(new BottomFence(cImage, xpos, rand));
        }

        for (int i = 0; i < botFences.size(); i++) {
            botFences.get(i).update();
            if (botFences.get(i).getX() < -20) {
                botFences.remove(i);
                if (botFences.get(botFences.size() - 1).getHeight() >=
maxFenceHeight) {
                    botDown = false;   }
                if (botFences.get(botFences.size() - 1).getHeight() <=
minFenceHeigh) {
                    botDown = true;   }

                if (!botDown) {
                    ypos = botFences.get(botFences.size() - 1).getY() -
1;
                }
                botFences.add(new BottomFence(cImage, xpos, ypos));
            }
        }
    }

    // Η διαδικασία που αρχικοποιεί την εφαρμογή για την έναρξη ενός
νέου παιχνιδιού
    public  void newGame(){
        // Αρχικοποίηση των μεταβλητών
        dissapear=false;
        topFences.clear();
        botFences.clear();
        walkingAlliens.clear();
        engineRun.clear();
        // Ο υπολογισμός του καλύτερου σκορ
        if(ufo.getScore()*3>best){
            best=ufo.getScore()*3;   }
        ufo.resetScore();
```

```
        ufo.setY(HEIGHT/2);
        minFenceHeigh=5;
        maxFenceHeight=30;

        // Κατά την έναρξη πρέπει να σχεδιασθεί όλη η οθόνη με τον πάνω
και κάτω φράκτη
        for(int i=0;i*20<WIDTH+40;i++){
            if(i==0){
                topFences.add(new TopFence(
BitmapFactory.decodeResource(getResources(),
R.drawable.barrier),i*20,0,10));
                botFences.add(new BottomFence(
BitmapFactory.decodeResource(getResources()
                        , R.drawable.barrier),i*20,HEIGHT-
minFenceHeigh));
            }
            else{
                topFences.add(new TopFence(
BitmapFactory.decodeResource(getResources()
                        , R.drawable.barrier),i*20,0,topFences.get(i-
1).getHeight()+1));
                botFences.add(new BottomFence(
BitmapFactory.decodeResource(getResources()
                        , R.drawable.barrier),i*20,botFences.get(i-
1).getY()-1));
            }
        }
        newGameCreated=true;
    }

    // Η διαδικασία που χρησιμοποιείται για να εμφανίσει το κείμενο με
τις πληροφορίες για την έναρξη ενός νέου παιχνιδιού, καθώς και τις
πληροφορίες με το σκορ και της καλύτερης επίδοσης
    public void drawText(Canvas canvas){
        Paint paint=new Paint();  // Χρησιμοποιείται η κλάση Paint
paint.setTypeface(Typeface.create(Typeface.DEFAULT,Typeface.BOLD_ITALIC)
);
        paint.setTextSize(35);
        paint.setColor(Color.WHITE);
        canvas.drawText("Αποσταση: "+(ufo.getScore()*3),10,HEIGHT-
5,paint);
        canvas.drawText("Καλυτερος:"+best,WIDTH-250,HEIGHT-10,paint);

        // Εδώ  εμφανίζονται οι πληροφορίες για ένα νέο παιχνίδι
        if(!ufo.isStartAnimate() && newGameCreated && reset){
            Paint paint1=new Paint();
            paint1.setColor(Color.CYAN);  // Ορισμός χρώματος
            paint1.setStyle(Paint.Style.FILL);  // Το χρώμα εσωτερικά
του πλαισίου κειμένου
            paint1.setAlpha(80);  // Διαφάνεια 0-255
            canvas.drawRect(10,10,WIDTH-10,HEIGHT-10,paint1);
            paint1.setColor(Color.YELLOW);
            paint1.setTextSize(30);  // Μέγεθος γραμματοσειράς
paint1.setTypeface(Typeface.create(Typeface.DEFAULT,Typeface.BOLD));  //
Στυλ έντονο
            canvas.drawText("ΣΠΑΣΜΕΝΟ UFO ", WIDTH/2-140,HEIGHT/2-
70,paint1);
            canvas.drawText("Πάτα για να ξεκινήσει το παιχνίδι και",
WIDTH/2-300,HEIGHT/2-20,paint1);
            paint1.setTextSize(20);
            canvas.drawText("Συνέχισε να πατάς για να μετακινηθεί το UFO
```

```
προς τα πάνω:",WIDTH/2-350,HEIGHT/2+70,paint1);
            canvas.drawText("Σταμάτα  να πατάς για να μετακινηθεί το UFO
προς τα κατω:",WIDTH/2-300,HEIGHT/2+140,paint1);
        }
    }
}
```

MainThread.java

```
// ********************* Η κλάση MainThread ***************************
// Η Κλάση MainThread αποτελεί το κύριο Loop του παιχνιδιού. Δηλαδή μέσα
σε μια While επανάληψη καλούνται σε κάθε κύκλο οι διαδικασίες ενημέρωσης
και σχεδίασης (update(),draw()) της κλάσης GamePanel. Ο κάθε κύκλος
πρέπει κανονικά να γίνεται σε σταθερό χρόνο 1/30 του δευτερολέπτου (1
frame). Όμως, επειδή σε κάποιες περιπτώσεις μπορεί ο κύκλος να
ολοκληρώνεται σε μικρότερο χρόνο, πρέπει να ορισθεί κάποια καθυστέρηση.
Επιπλέον σε κάποιες περιπτώσεις ο κύκλος μπορεί να ολοκληρώνεται σε
μεγαλύτερο χρόνο και κάποια πλαίσια να χαθούν.
// ********************************************************************
public class MainThread extends Thread {
    // Αρχικοποιούνται τα πεδία-fields της κλάσης
    private int FPS=30;  // frames per second, πόσες φορές σχεδιάζεται η
εικόνα το δευτερόλεπτο για να είναι η κίνηση ομαλή, και όχι
διακοπτόμμενη
    private double averageFPS;  // Μέσος όρος frames
    private SurfaceHolder surfaceHolder;  // Η επιφάνεια όπου εκτελείται
ο κύκλος του παιχνιδιού
    private GamePanel gamePanel;  // Η κλάση GamePanel
    private boolean running;  // Η λογική μεταβλητή εκτέλεσης
    public static Canvas canvas;  // Ο Καμβάς

    // Ο κατασκευαστής της κλάσης που συνδέεται με την επιφάνεια και τον
καμβά
    public MainThread(SurfaceHolder surfaceHolder,GamePanel gamePanel){
        super();
        this.surfaceHolder=surfaceHolder;
        this.gamePanel=gamePanel;
    }

    @Override
    // Η διαδικασία που εκκινεί την εκτέλεση του thread
    public void run(){
        long startTime;
        long timeMillis;
        long waitTime;
        long totalTime=0;
        int frameCount=0;
        long targetTime=1000/FPS;

        // Η επανάληψη
        while (running){
            startTime=System.nanoTime();
            canvas=null;
            // επιδιώκεται η δέσμευση του καμβά για την επεξεργασία των
εικονοστοιχείων
            try {
                canvas= this.surfaceHolder.lockCanvas();
                // Εδώ γίνεται η κλήση των διαδικασιών του Loop
                synchronized (surfaceHolder){
                    this.gamePanel.update();
                    this.gamePanel.draw(canvas);
```

```
                }
            }
            catch (Exception e){}
            finally {
                if(canvas!=null){
                    try{
                        // Απελευθέρωση του καμβά γι' αυτό το κύκλο
                        surfaceHolder.unlockCanvasAndPost(canvas);  }
                    catch(Exception e){
                        e.printStackTrace();  }
                }
            }
            // Υπολογισμός του χρόνου που έχει περάσει και πόσος χρόνος
απομένει
            timeMillis=(System.nanoTime() - startTime) / 1000000;
            waitTime=targetTime-timeMillis;
            try{
                // Αναμονή γι' αυτό το χρόνο που υπολογίστηκε
                this.sleep(waitTime);  }
            catch (Exception e){}
            // Υπολογισμοί χρόνων για δοκιμές και έλεγχο απόδοσης
            totalTime+=System.nanoTime()-startTime;
            frameCount++;
            if (frameCount==FPS){
                averageFPS=1000/((totalTime /  frameCount)/1000000);
                System.out.println( averageFPS);
                frameCount=0;
                totalTime=0;
            }
        }
    }
    //  Η διαδικασία που ορίζει εάν έχει ξεκινήσει ή όχι η εκτέλεση του
thread
    public void setRunning(boolean b) {
        this.running=b;  }
}
```

Background.java

```
// ********************* Η κλάση Background ****************************
// Η κλάση Background σκοπό έχει να εμφανίσει μια εικόνα ως υπόβαθρο της
οθόνης και επιπλέον πρέπει να την μετακινεί, "κυλλάει" στην αντίθετη
κατεύθυνση από το σκάφος ώστε να φαίνεται ότι το σκάφος κινείται. Όμως
επειδή το μέγεθος της εικόνας δεν είναι τεράστιο πρέπει να διπλωθεί,
όπως διατυπώνεται συνήθως, το τέλος με την αρχή της εικόνας. Για να
επιτευχθεί αυτό χρησιμοποιείται ένα δεύτερο αντίγραφο της εικόνας, το
οποίο το κολλάμε στο τέλος της πρώτης εικόνας και μετακινείται μαζί της.
Όταν τώρα βγει όλη η πρώτη εικόνα έξω από την οθόνη επανέρχεται στην
αρχική της θέση.
// ***********************************************************************
public class Background {
    // x,y η θέση που θα μπει αρχικά, dx το βήμα κύλισης
    private Bitmap image;
    private int x,dx;
    private int y;  // Δεν υπάρχει λόγος, παρά μόνο για επέκταση των
δυνατοτήτων του παιχνιδιού

    // Ο κατασκευαστής φορτώνει την εικόνα και ορίζει το αρχικό βήμα
κύλισης
```

```
    public Background(Bitmap resImage) {
        this.image=resImage;
        this.dx=GamePanel.ROLLSPEED;
    }

    // Η διαδικασία που ενημερώνει την τρέχουσα θέση της εικόνας κάθε
φορά που καλείται
    public void update(){
        x+=dx;
        // Κυλάει προς τα αριστερά, μειώνεται κάτε φορά μέχρι να βγει
εκτός οθόνης, οπότε την επαναφέρει στη θέση χ=0
        if(x<=-GamePanel.WIDTH){
            x=0;  }
    }

    // Η διαδικασία που σχεδιάζει τη εικόνα πάνω στο καμβά
    public void draw(Canvas canvas){
        // Η πρώτη εικόνα που έχει μετακινηθεί σύμφωνα με το βήμα
ROLLSPEED
        canvas.drawBitmap(image,x,y,null);
        // Η δεύτερη εικόνα που συμπληρώνει το κενό στο τέλος της πρώτης
εικόνας (στα δεξιά)
        if(x<0){
            canvas.drawBitmap(image,x+GamePanel.WIDTH,y,null);
        }
    }
}
```

Actor.java

```
// ********************** Η κλάση Actor ****************************
// Η κλάση Actor αποτελεί μια αφηρημένη κλάση που τα χαρακτηριστικά της
μπορούν να κληρονομήσουν οι υπόλοιπες βασικές κλάσεις του παιχνιδιού.
Έτσι εδώ καθορίζονται οι μέθοδοι και οι ιδιότητες που είναι κοινές για
τις βασικές κλάσεις που συμμετέχουν στο παιχνίδι.
// *****************************************************************

public abstract class Actor {
    // Βασικές ιδιότητες όπως θέση (x,y), ύψος , πλάτος, κατεύθυνση
(dirX,dirY)
    protected int x;
    protected int y;
    protected  int width;
    protected int height;
    protected  int dirX;
    protected int dirY;

    /* Βασικές μέθοδοι όπως ορισμός νέας θέσης (x,y), επιστροφή της
θέσης (x,y), ενημέρωσης του ύψους και του πλάτους, της κατεύθυνσης
(dirX,dirY) και τέλος του πλαισίου με τα τέσσερα σημεία των κορυφών του,
που χρησιμεύει στην ανίχνευση σύγκρουσης των αντικειμένων. */
    public void setX(int x){ this.x=x;}
    public void setY(int y){this.y=y;}
    public int getX() {
        return x; }
    public int getY() {
        return y; }

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

```
    public int getWidth() {
        return width; }
    public int getHeight() {
        return height; }

    public void setDirX(int dirX) {
        this.dirX = dirX; }
    public void setDirY(int dirY) {
        this.dirY = dirY; }
    public int getDirX() {
        return dirX; }
    public int getDirY() {
        return dirY; }

    public Rect colRect(){
        return new Rect(x,y,x+width,y+height); }
}
```

Ufo.java

```
// ********************** Η κλάση Ufo ********************************
// Η κλάση Ufo αποτελεί το αντικείμενο που χειρίζεται ο χρήστης του
παιχνιδιού και εκτελεί τις παρακάτω λειτουργίες: πρέπει να ενημερώνεται
η θέση του σκάφους καθώς ο χρήστης ακουμπά την οθόνη, πρέπει να
εναλλάσσονται τα πλαίσια του σκάφους ώστε να δίνει την αίσθηση της
κίνησης – περιστροφής, και τέλος πρέπει να ενημερώνει το σκορ καθώς
εξελίσσεται το παιχνίδι.
// ********************************************************************
public class Ufo extends Actor{
    private Bitmap imgSprites;
    private int score;
    private boolean up;
    private boolean startAnimate;
    private Animation anim=new Animation();
    private Long startTime;

    // Ο κατασκευαστής της κλάσης
    public Ufo (Bitmap imageRes, int w, int h, int aniFrm){
        // Οι αρχικές τιμές για την θέση, μέγεθος και κατεύθυνση του
σκάφους
        this.x=100;
        this.y=GamePanel.HEIGHT/2;
        this.width=w;
        this.height=h;
        this.score=0;
        this.dirY=0;

        /* Δημιουργείται ένας πίνακας από εικόνες ο οποίος περιέχει τα
διαφορετικά πλαίσια του σκάφους τα οποία θα σταλούν ως παράμετροι στην
κλάση Animation, ώστε να ξεκινήσει η εναλλαγή των εικόνων */
        Bitmap[] imageArray = new Bitmap[aniFrm];
        imgSprites=imageRes;
        for(int i=0;i<imageArray.length;i++){
imageArray[i]=Bitmap.createBitmap(imgSprites,i*width,0,width,height);
            anim.setFrames(imageArray);
            anim.setDelay(10);
            startTime=System.nanoTime();
        }
    }

    // Η διαδικασία setUp που όταν καλείται δηλώνει ότι ο χρήστης πάτησε
στην οθόνη
```

```
    public void setUp(boolean pressed){ this.up=pressed;}

    // Η διαδικασία ενημέρωσης της συμπεριφοράς του σκάφους
    public void update(){
        // Υπολογίζεται ο χρόνος που πέρασε από τον χρόνο εκκίνησης
        long elapsed=(System.nanoTime()-startTime)/1000000;
        // Ανάλογα με τον υπολογισμό του χρόνου που πέρασε ενημερώνεται
το σκορ και επαναρχικοποιείται ο χρόνος
        if(elapsed>100){
            score+=1;
            startTime=System.nanoTime();
        }
        anim.update();  // Καλείται η κλάση Animation

        // Ελέγχεται η κίνηση ανάλογα με την τιμή της μεταβλητής up, που
καθορίζει αν πατήθηκε ή όχι
        // η οθόνη. Όσο αυξάνονται τα πατήματα αυξάνεται και το βήμα,
αλλιώς μειώνεται αντίστοιχα.
        if(up) {
            dirY-=1; }
        else {
            dirY+=1; }
        if (dirY>14){ dirY=14; }
        if (dirY<-14 ){ dirY=-14; }
        y+=dirY*2;
    }

    // Η διαδικασία draw της κλάσης που σχεδιάζει το ενημερωμένο πλαίσιο
στην κατάλληλη θέση στον καμβά
    public void draw(Canvas canvas){
        canvas.drawBitmap(anim.getImage(),x,y,null); }

    // Η διαδικασία που όταν καλείται, επιστρέφει τη τρέχουσα τιμή του
σκορ
    public int getScore(){
        return score; }

    // Η διαδικασία που επιστρέφει αληθές ή ψευδές ανάλογα με το εάν
έχει ξεκινήσει το animation του σκάφους
    public boolean isStartAnimate() {
        return startAnimate; }

    // Η διαδικασία που όταν κληθεί ορίζει το ξεκίνημα του animation του
σκάφους ή  όχι
    public void setStartAnimate(boolean startAnimate) {
        this.startAnimate = startAnimate; }

    // Η διαδικασία που επαναφέρει την παράμετρο ενίσχυσης της κίνησης
προς τα πάνω, όταν πατά ο χρήστης την οθόνη της συσκευής
    public void resetThrotle(){ dirY=0; }

    // Η διαδικασία που μηδενίζει το σκορ όταν καλείται
    public void resetScore(){ score=0; }
}
```

Animation.java

```
// ********************** Η κλάση Animation ***************************
// Η κλάση Animation αποτελεί την αναπαραγωγή κινούμενων εικόνων –
sprites, ώστε να δίνεται η εντύπωση της πραγματικής κίνησης, όπως
περπάτημα, περιστροφή, και γενικά αλλαγή στιγμιότυπων εικόνων. Στην
ουσία η λειτουργία της βασίζεται στην εναλλαγή εικόνων κάθε φορά που
```

```
καλείται.
// ***********************************************************************
public class Animation {
    // Οι βασικές μεταβλητές
    private Bitmap[] frames;  // Ο πίνακας που περιέχει τις εικόνες-
πλαίσια που δημιουργούν το κινούμενο γραφικό (sprite)
    private int curFrame;  // Το τρέχον πλαίσιο εικόνας
    private long delay;  // Η καθυστέρηση εναλλαγής εικόνων
    private long startTime; // Η μεταβλητή εκκίνησης
    private boolean oneLoop;  // η μεταβλητή που καθορίζει αν έχει
ολοκληρωθεί μια πλήρης εναλλαγή εικόνων (η περίπτωση του εφέ καταστροφής
του σκάφους πρέπει να εκτελείται μόνο μια φορά)

    public void Animation(){ }  // Ο κατασκευαστής δεν περιέχει τίποτα

    // Η διαδικασία όπου ενημερώνονται τα πλαίσια του Animation με τις
εικόνες που θα εναλλάσσονται. Αρχικοποιείται το πρώτο πλαίσιο και ο
χρόνος.
    public void setFrames(Bitmap[] frames){
        this.frames=frames;
        this.curFrame=0;
        startTime=System.nanoTime();
    }

    // Η διαδικασία που ορίζει την καθυστέρηση
    public void setDelay(long delay) {
        this.delay = delay; }

    // Η διαδικασία που ορίζει το τρέχων πλαίσιο
    public void setCurFrame(int curFrame) {
        this.curFrame = curFrame; }

    // Η διαδικασία ενημέρωσης με το επόμενο πλαίσιο και η αλλαγή των
μεταβλητών χρόνου, τρέχοντα πλαισίου, και ολοκλήρωσης της πλήρης
εναλλαγής
    public void update(){
        long elapsedTime=(System.nanoTime()-startTime)/1000000;
        if(elapsedTime>delay){
            curFrame++;
            startTime=System.nanoTime();
        }
        if(curFrame==frames.length){
            curFrame=0;
            oneLoop=true;
        }
    }

    // Η διαδικασία που επιστρέφει τη τρέχουσα εικόνα
    public Bitmap getImage(){
        return frames[curFrame]; }

    // Η διαδικασία που επιστρέφει το τρέχον πλαίσιο
    public int getFrame(){
        return curFrame; }

    // Η διαδικασία που επιστρέφει την λογική εά ολοκληρώθηκε ή όχι μια
πλήρης εναλλαγή πλαισίων
    public boolean isOneLoop() {
        return oneLoop; }
}
```

EngineRun.java

```
// ********************** Η κλάση EngineRun ***************************
// Η κλάση EngineRun υλοποιεί έναν εναλλακτικό τρόπο παραγωγής
animation, με την σχεδίαση σχημάτων. Πιο συγκεκριμένα, δημιουργεί τρείς
κύκλους με ένα τυχαίο μέγεθος και σε τυχαία θέση δίνοντας την εντύπωση
καπνού.
// ********************************************************************

public class EngineRun extends Actor{
    // Πεδία που χρησιμοποιούνται είναι η ακτίνα και ένας τυχαίος
αριθμός
    private int r;
    private Random rnd= new Random();

    // Ο κατασκευαστής αρχικοποιεί την ακτίνα και την αρχική θέση
    public EngineRun(int x, int y){
        r=5;
        super.x=x;
        super.y=y;
    }

    // Η διαδικασία ενημέρωσης μειώνει τη θέση στον οριζόντιο άξονα κατά
-6 βήματα
    public void update(){
        x-=6; }

    // Η διαδικασία σχεδίασης με την βοήθεια της κλάσης Paint σχεδιάζει
τρεις κύκλους  με τυχαίο μέγεθος και θέση μέσα σε συγκεκριμένα όρια
    public void draw(Canvas canvas){
        Paint paint=new Paint();
        int rand=(int)(rnd.nextDouble()*10);
        paint.setColor(Color.YELLOW+(int)(rnd.nextDouble()*300));
        paint.setStyle(Paint.Style.FILL);
        canvas.drawCircle(x - r, y - r, r+rand, paint);
        canvas.drawCircle(x-r+2,y-r-4,r+rand,paint);
        canvas.drawCircle(x-r+4,y-r,r+rand,paint);
    }
}
```

BottomFence.java

```
// ********************** Η κλάση BottomFence ************************
// Η κλάση BottomFence αποτελεί τα στιγμιότυπα που σχεδιάζουν τον φράκτη
στο κάτω μέρος της οθόνης. Κατά την έναρξη του παιχνιδιού καλούνται οι
update και draw διαδικασίες ώστε να γεμίσουν όλη την οθόνη, ενώ κατόπιν
με βάση το βήμα σχεδιάζουν το καμβά ώστε να συμπληρώσουν το κενό του
φράκτη που δημιουργείται.
// ********************************************************************

public class BottomFence extends Actor {
    private Bitmap image;
    private int dx;

    // Ο κατασκευαστής της κλάσης που κατασκευάζει ένα στιγμιότυπο του
φράκτη στη θέση που ορίζεται
    public BottomFence(Bitmap resImageSprite, int x, int y){
        this.x=x;
        this.y=y;
        this.width=20;
        this.height=200;
        dx=GamePanel.ROLLSPEED;  // το βήμα κύλισης
        image=Bitmap.createBitmap(resImageSprite,0,0,width,height);
```

```
    }

    // Η διαδικασία που ενημερώνει την νέα θέση του στιγμιότυπου με βάση
το βήμα ROLLSPEED
    public void update(){
        x+=dx; }

    // Η διαδικασία που σχεδιάζει κάθε φορά τα αντικείμενα του φράκτη
    public void draw(Canvas canvas){
        canvas.drawBitmap(image,x,y,null);
    }
}
```

TopFence.java

```
// ********************** Η κλάση TopFence ***************************
// Η κλάση TopFence αποτελεί τα στιγμιότυπα που σχεδιάζουν τον φράκτη
στο πάνω μέρος της οθόνης. Κατά την έναρξη του παιχνιδιού καλούνται οι
update και draw διαδικασίες ώστε να γεμίσουν όλη την οθόνη, ενώ κατόπιν
με βάση το βήμα σχεδιάζουν το καμβά ώστε να συμπληρώσουν το κενό του
φράκτη που δημιουργείται.
// ***********************************************************************
public class TopFence extends Actor{
    private Bitmap image;
    private int dx;

    // Ο κατασκευαστής της κλάσης που κατασκευάζει ένα στιγμιότυπο του
φράκτη στη θέση που ορίζεται
    public TopFence(Bitmap resImageSprite,int x,int y,int h) {
        super();
        this.x=x;
        this.y=y;
        this.width=20;
        this.height=h;
        dx=GamePanel.ROLLSPEED;  // το βήμα κύλισης
        image=Bitmap.createBitmap(resImageSprite,0,0,width,height);
    }

    // Η διαδικασία που ενημερώνει την νέα θέση του στιγμιότυπου με βάση
το βήμα ROLLSPEED
    public void update(){
        x+=dx; }

    // Η διαδικασία που σχεδιάζει κάθε φορά τα αντικείμενα του φράκτη
    public void draw(Canvas canvas){
        canvas.drawBitmap(image,x,y,null);
    }
}
```

WalkingAllien.java

```
// ********************* Η κλάση WalkingAllien ************************
// Η κλάση WalkingAllien χρησιμοποιείται για την δημιουργία των
στιγμιότυπων των εξωγήινων, τα οποία κινούνται μέσα στην σκηνή και
πρέπει το σκάφος να αποφύγει. Για να επιτευχθεί αυτό πρέπει να
επιτελούνται οι εξής λειτουργίες: Το κάθε στιγμιότυπο πρέπει να κινείται
με τυχαία κατεύθυνση και ταχύτητα, (μέσα σε καθορισμένα όρια), και
πρέπει να εναλλάσσονται πλαίσια εικόνων που να δίνουν την αίσθηση ότι ο
εξωγήινος περπατάει.
// ***********************************************************************
```

```
public class WalkingAllien extends Actor {
    // Τα πεδία που θα χρησιμοποιηθούν στην κλάση
    private int score;
    private int speed;
    private int direction;
    private Random rnd= new Random();
    private Animation anim= new Animation();
    private Bitmap imgSprites;

    // Ο κατασκευαστής της κλάσης που αρχικοποιεί την θέση, κατεύθυνση
και ταχύτητα, καθώς και τα
    // πλαίσια εικόνας που θα σταλούν ως παράμετροι στην κλάση Animation
    public WalkingAllien( Bitmap imageRes,int x,int y, int w, int h, int
s, int aniFrm){
        super.x=x;
        super.y=y;
        width=w;
        height=h;
        score=s;
        // Η κατεύθυνση και η ταχύτητα ορίζονται με τυχαίο τρόπο, αλλά
μέσα σε όρια
        direction=(int)(rnd.nextDouble()*6)+7;
        // Αύξηση της ταχύτητας
        speed=5+ (int) rnd.nextDouble()*score/30;
        // Μεγίστη ταχύτητα
        if(speed>40){ speed=40; }

        // Εδώ ορίζονται τα πλαίσια εικόνας μέσα σε ένα πίνακα και
στέλνονται ως παράμετροι στην κλάση Animation
        Bitmap[] image=new Bitmap[aniFrm];
        imgSprites=imageRes;
        for(int i=0;i<image.length;i++){
image[i]=Bitmap.createBitmap(imgSprites,i*width,0,width,height);
        }
        anim.setFrames(image);
        anim.setDelay(200-speed);
    }

    // Η διαδικασία ενημέρωσης της νέας θέσης x,y με βάση την ταχύτητα
και την κατεύθυνση, και το νέο πλαίσιο εικόνας που καθορίζεται από την
κλάση Animation
    public void  update(){
        y-=speed;
        x-=direction;
        anim.update();
    }

    // Η διαδικασία draw που σχεδιάζει το ενημερωμένο πλαίσιο στον καμβά
    public void draw(Canvas canvas){
        try{
            canvas.drawBitmap(anim.getImage(),x,y,null);    }
        catch (Exception e){System.out.println("ERRoorrr");  }
    }
}
```

ElectricCrash.java

```
// ********************** Η κλάση ElectricCrash **********************
// Η κλάση ElectricCrash εμφανίζει το εφέ καταστροφής του σκάφους στη
θέση x,y, που καθορίζεται κατά τη δημιουργία του στιγμιότυπου. Στην
ουσία εκτελείται το animation που αποτελείται από την εναλλαγή μιας
σειράς εικόνων, για μια φορά.
```

```
// **********************************************************************
public class ElectricCrash {
    // Αρχικοποιούνται οι μεταβλητές θέσης και εικόνων του στιγμιότυπου
    private int x;
    private int y;
    private int width;
    private int height;
    private int row;
    private Animation anim=new Animation();
    private Bitmap imgSprite;

    // Καλείται ο κατασκευαστής που χρησιμοποιεί την κλάση Animation για
να υλοποιήσει την εναλλαγή των εικόνων
    public ElectricCrash(Bitmap resImage,int x,int y, int w,int h,int
aniFrames) {
        // Αρχικοποίηση της θέσης που θα εκτελεστεί το εφέ
        this.x=x;
        this.y=y;
        this.width=w;
        this.height=h;
        Bitmap[] images=new Bitmap [aniFrames];
        this.imgSprite=resImage;

        // Δημιουργείται ο πίνακας με τις εικόνες που θα χρησιμοποιηθούν
        for(int i=0;i<images.length;i++){
            if(i%5==0 && i>0) row++;
            images[i]=Bitmap.createBitmap(imgSprite,(i-(5*row))*width,
row*height, width,height);
        }
        // Χρησιμοποιείται η κλάση Animation όπου ενημερώνεται για τα
πλαίσια, στέλνοντας τις εικόνες και το χρόνο καθυστέρησης
        anim.setFrames(images);
        anim.setDelay(10);
    }

    // Η διαδικασία που ενημερώνει το επόμενο πλαίσιο του animation,
εφόσον δεν έχει ολοκληρωθεί μια πλήρη εναλλαγή
    public void update(){
        if(!anim.isOneLoop()){
            anim.update();
        }
    }

    // Η διαδικασία που σχεδιάζει το καμβά με το επόμενο πλαίσιο, εφόσον
δεν έχει ολοκληρωθεί μια πλήρη εναλλαγή
    public void draw(Canvas canvas) {
        if (!anim.isOneLoop()) {
            canvas.drawBitmap(anim.getImage(), x, y, null);
        }
    }

    // Η διαδικασία που επιστρέφει το ύψος του πλαισίου
    public int getHeight(){
        return height;  }
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] DiMarzio, J. F., 2015. Android Studio Game Development: Concepts and Design. Apress.

[2] Lauren Darcey, L., Conder, S., 2010. Sams Teach Yourself Android Application Development in 24 Hours. Sams, United States of America (First Printing June 2010).

[3] Smyth, N., 2015. Android Studio Development Essentials – Android 6 Edition. CreateSpace Independent Publishing Platform (December 9, 2015).

[4] van Drongelen, M., 2015. Android Studio Cookbook. Design, debug, and test your apps using Android Studio. Packt Publishing.

[5] Αλέπης, Ε., Δρακούλης, Σ., 2015. "Android Geolocation", *Σημειώσεις του ΠΜΣ «Εφαρμοσμένα Πληροφοριακά Συστήματα»*, Τμήμα Ηλεκτρονικών Υπολογιστικών Συστημάτων, ΑΕΙ Πειραιά Τ.Τ. (Οκτώβριος 2015).

[6] Αλέπης, Ε., Δρακούλης, Σ., 2015. "Android Sensors", *Σημειώσεις του ΠΜΣ «Εφαρμοσμένα Πληροφοριακά Συστήματα»*, Τμήμα Ηλεκτρονικών Υπολογιστικών Συστημάτων, ΑΕΙ Πειραιά Τ.Τ. (Οκτώβριος 2015).

ΙΣΤΟΣΕΛΙΔΕΣ

[1] Intro To Game Development (Part 1): The Game Loop link: http://blog.heeresonline.com/2015/01/intro-to-game-development-part-1-the-game-loop/

[2] JavaCodeGeeks ,2011. Android Game Development – The Game Loop. link: https://www.javacodegeeks.com/2011/07/android-game-development-game-loop.html

[3] Karthik M., How to import existing Android Studio Project In to Android Studio with New Package Name. https://www.youtube.com/watch?v=isr3zXK42po

[4] paymon wang-lotfi, How to make a 2D game for Android (Episodes 1-8). https://www.youtube.com/watch?v=rJcm5Oyi3YA

[5] Music: extremeaction.mp3 - https://www.bensound.com/royalty-free-music