МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

кафедра програмних засобів

Звіт

з лабораторної роботи № 4

з дисципліни «Мобільні операційні системи»

на тему:

«Використання графіки в мобільних пристроях.  
Технологія створення ігор»

Виконав

ст. гр. КНТ-137 В.В. Козлов

Прийняв

старший викладач Є.М. Федорченко

1. Мета роботи

Ознайомитись із принципами використання графічних засобів в мобільних додатках. Створити гру для мобільного пристрою за допомогою середовища розробки Android Studio на основі використання засобів побудови комп’ютерної графіки.

1. Завдання на лабораторну роботу

Ознайомитись з основними теоретичними відомостями.

Створити гру для мобільного пристрою на основі використання засобів побудови комп’ютерної графіки.

Відповісти на контрольні запитання.

Оформити звіт.

1. Теоретичні відомості

Мобільна гра – це ігрова програма для мобільних пристроїв, наприклад стільникових телефонів, смартфонів тощо. Поширені мобільні ігри для платформи Java Micro Edition. Цю платформу підтримують більшість сучасних мобільних пристроїв, в тому числі більшість стільникових телефонів стандарту GSM.

Переважна більшість ігор написанf в жанрі «аркада» та «головоломка». Інколи зустрічаються представники й інших жанрів. Основним обмеженням для «розмаїття жанрів» є пристрій введення. Наприклад, для зручної гри в 3D-шутер бажана можливість одночасного наведення на ціль і стрільби – використання двох клавіш одночасно, що утруднено на маленьких клавіатурах мобільного телефону, а стратегії в реальному часі спочатку орієнтувалися на управління джойстиком або комп’ютерною мишею.

Наприкінці 2000-х років поширилися сенсорні телефони, на них основним жанром стала фізична головоломка (Angry Birds,Cut the Rope).

Щоб залучити потенційного покупця, часто випускаються мобільні ігри з використанням відомого бренду (наприклад, Doom RPG), або за мотивами фільму, що нещодавно з'явився.

Окремо виділяють аркадні ігрові автомати та мобільні пристрої, як смартфони чи планшетні комп'ютери, на яких також можна грати у відеоігри.

Деякі ігри можуть запускатися на різних платформах. Ця їх здатність називається кросплатформністю або багатоплатформністю. Для досягнення кросплатформності розробляється окрема версія гри для кожної платформи, або ця гра використовує технології, з яким працюють декілька платформ. Наприклад, браузерні ігри потребують лише наявності інтернет-браузера.

Стосовно розробки ігор під Android. Платформа від Google, існує у вигляді безкоштовної ОС, використовуваної виробниками телефонів. Мобільні ігри зазвичай пишуться на Java. Проте в Android SDK вже з'явилася можливість робити це і на C++.

1. Результати виконання роботи

Сучасній людині важко уявити собі життя без мобільних ігор. Чим ще зайнятися в маршрутці або черзі в лікарні, як не пограти в улюблену гру.

Хоча в пошуках витоків мобільних ігор можна дійти до найперших телефонів, але по-справжньому все почалося в 1997 році, коли фін Танелі Арманто створив для Nokia, напевно, все ще найвідомішу мобільну гру в історії – «Змійку» (Snake). Першим пристроєм, на якому можна було в неї пограти, був Nokia 6110, який розійшовся величезними тиражами. Гра стала справжнім символом всієї індустрії, і навіть сьогодні можна в неї зіграти на своєму телефоні.

Важливою частиною сучасних мобільних ігр є використання графічних засобів для побудови інтерфейсу та процесу гри. При створенні ігор для мобільного пристрою за допомогою середовища розробки Android Studio можна використовувати класи та методи, які вбудовано в стандартні бібліотеки, більшість яких додано в пакет android.graphics. Пакет android.graphics має всі необхідні бібліотеки для роботи з двомірної графікою.

* 1. Клас Color

Клас Color відповідає за кольори. Кольори можна описувати чотирма числами в форматі ARGB, по одному для кожного каналу (Alpha, Red, Green, Blue), або вибирати колір серед стандартного списку:

int myColor1 = Color.argb(0, 255, 0, 0);

int myColor2 = Color.RED;

* 1. Клас Paint

Клас Paint містить стилі, кольори й іншу графічну інформацію для малювання графічних об’єктів. Він дозволяє вибирати спосіб відображення графічних примітивів, які малюються на об’єкті Canvas за допомогою різних методів.

Створити об’єкт класу Paint можна за допомогою такого коду:

Paint myPaint = new Paint();

Змінюючи об’єкт класу Paint, можна контролювати колір, стиль, шрифт і спеціальні ефекти, використовувані при малюванні. Наприклад, щоб встановити суцільний колір для малювання лінії, потрібно викликати метод Paint.setColor ():

myPaint.setColor(Color.BLUE);

В цьому прикладі використана готова константа. Також можна вказати 32-бітове ціле число, закодоване в схемі ARGB8888. Так можна встановити колір за допомогою його складових:

myPaint.setARGB(alphaValue, redValue, greenValue, blueValue);

Стиль об’єкта класу Paint, що задається за допомогою методу setStyle (), дозволяє малювати або контур графічного примітиву (STROKE), або його заливку (FILL), або і те, і інше відразу (FILL\_AND\_STROKE):

myPaint.setStyle(Paint.Style.FILL\_AND\_STROKE);

Крім цих простих методів клас Paint підтримує прозорість і може бути змінений за допомогою різних шейдеров, фільтрів і ефектів, які надають багатий набір складних фарб і пензлів.

При створенні нового об’єкта Paint також можна передати в його конструктор кілька прапорів, які будуть впливати на спосіб відображення. Одним з найбільш цікавих з них вважається прапор ANTI\_ALIAS\_FLAG, що забезпечує згладжування діагональних ліній, змальованих об'єктом Paint (знижуючи при цьому продуктивність). Згладжування грає важливу роль в процесі відтворення тексту, значно спрощує його сприйняття. Також можна задати цей флаг програмно, використовуючи метод setAntiAlias (true). Створення об’єкта класу Paint, призначеного для використання при відтворенні тексту (класс Typeface використовується для визначення сімейства та типу шрифта):

textPaint = new Paint();

textPaint.setAntiAlias(true);

textPaint.setColor(Color.BLACK);

textPaint.setTextSize(48.0f);

textPaint.setTypeface(Typeface.create(Typeface.DEFAULT, Typeface.BOLD));

* 1. Клас Bitmap та BitmapFactory

Часто доводиться мати справу з зображеннями, які зберігаються в файлах JPG, PNG, GIF. По суті, будь-яке зображення, яке завантажується з графічного файлу, є набором кольорових крапок (пікселів). А інформацію про кожну точку можна зберегти в бітах. Звідси і назва – карта бітів або bitmap (іноді використовується термін растр або растрове зображення). В Android є спеціальний клас Bitmap для роботи з подібними картинками.

Існують готові растрові зображення в файлах, а щоб створити з нуля об’єкт класу Bitmap програмним способом, потрібно викликати метод createBitmap ():

Bitmap myBitmap =Bitmap.createBitmap(width, height, Bitmap.Config.ARGB\_8888);

В результаті вийде прямокутник із заданими розмірами в пікселях (перші два параметра – ширина та висота). Третій параметр відповідає за інформацію про прозорість та якість кольору.

Дуже часто потрібно знати розміри зображення. Щоб дізнатися його ширину і висоту в пікселях, необхідно використати відповідні методи:

int width = myBitmap.getWidth();

int height = myBitmap.getHeight();

Щоб зафарбувати окрему точку, можна використати метод setPixel () (парний йому метод getPixel дозволить дізнатися інформацію про точку):

myBitmap.setPixel(xValue, yValue, Color.RED);

Клас BitmapFactory дозволяє створити об’єкт класу Bitmap з файлу, потоку або байтового масиву. Даний клас часто використовується в роботі з зображеннями. Завантажити файл з ресурсів додатку можна за допомогою методу BitmapFactory.decodeResource () (Картинку можна додати в ресурси, завантаживши в папку додатку «res/drawable»):

myBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.imgName);

Для завантаження зображення з файлу призначений метод BitmapFactory.decodeFile ().

За допомогою методу Bitmap.createScaledBitmap () можна змінити розмір зображення:

myBitmap = Bitmap.createScaledBitmap(myBitmap, newWidth, newHight, false);

В останньому параметрі методу йде булева змінна, що відповідає за згладжування пікселів. Зазвичай його застосовують, коли маленьке зображення збільшують в розмірах, щоб поліпшити якість картинки. При зменшенні картинки, як правило, в цьому немає такої необхідності.

Створений малюнок не можна намалювати засобами класу Bitmap. Клас Bitmap не має своїх методів для малювання, для цього існує клас Canvas (Полотно), на якому можна розміщувати об’єкти Bitmap.

* 1. Клас Canvas, клас SurfaceView та SurfaceHolder

Клас Canvas (Полотно) надає методи для малювання, які відображають графічні примітиви на вихідному растровому зображенні. При цьому треба спочатку підготувати кисть (клас Paint).

Android підтримує напівпрозорість, градієнтні заливки, округлені прямокутники і згладжування. Через обмежених ресурсів векторна графіка поки що не підтримується, замість цього використовується традиційна растрова перерисовка.

Canvas працює з пікселями, тому слід дбати про конвертацію одиниць dp в px і навпаки при необхідності. Початок координат знаходиться в лівому верхньому кутку.

Отримати доступ до полотна можна за допомогою об’єкта Bitmap або компонента View. Дуже часто розробники створюють свій власний компонент, наслідуючи від View, і малюють на його полотні для реалізації своїх задумів.

За допомогою метода drawBitmap () можна намалювати Bitmap зображення на полотні Canvas:

canvas.drawBitmap(myBitmap, xCoordinate, yCoordinate, paint);

За допомогою метода drawText () можна відобразити текст на полотні Canvas:

canvas.drawText(textString, xCoordinate, yCoordinate, textPaint);

Клас SurfaceView надає об’єкт Surface, який підтримує малювання в фоновому потоці і дає можливість використовувати OpenGL для тривимірної графіки. Це відмінний варіант для насичених графікою елементів, які потребують частих оновленнях або повинні відображати складну графічну інформацію, як у випадку з іграми і тривимірною візуалізацією.

SurfaceView – обгортка навколо класу SurfaceHolder, який в свою чергу служить обгорткою класу Surface, використовуваного для відновлення зображення з фонових потоків.

SurfaceView використовується точно таким же чином, як будь-які похідні від View класи. Можна застосовувати анімацію і розміщувати їх всередині розмітки так само, як і інші компоненти.

Застосовуючи OpenGL, можна малювати на Surface будь-які підтримувані двовимірні або тривимірні об'єкти, отримуючи при цьому всі вигоди від апаратного прискорення (якщо таке є). Таким чином, значно підвищується продуктивність, якщо порівнювати з тими ж операціями, виконаними на двовимірному Canvas.

Припустимо, що необхідно намалювати Bitmap на якомусь елементі керування (саме намалювати, а не додати через Background).

Тоді необхідно створити клас, який успадковує клас SurfaceView (extends SurfaceView). Для елемента, на якому планується малювання додати екземпляр створено класу в якость View:

LinearLayout layout = (LinearLayout) findViewById(R.id.LayoutId);

Layout.addView(new YourViewClass());

В створеному класі необхідно створити функцію, в якій буде відбуватися малювання. В цій функції отримаємо полотно нашого layout за допомогою методів успадкованого класу SurfaceHolder – getSurface().lockCanvas() :

Canvas canvas = getHolder().lockCanvas();

При цьому автоматично передається полотно для малювання з елемента інтерфейсу, до якого було прикріплено за допомогою функції addView () екземпляр класу.

Теперь на цьому елементі можна малювати. Варто звернути увагу на те, що функція малювання повинна викликатися не один раз, адже після зміни активіті намальовані зображення зникнуть.

* 1. Відкриття іншого Activity. Клас Intent

Клас наміру (Intent) – це механізм для опису однієї операції, яку необхідно виконати додатку – вибрати фотографію, відправити лист, зробити дзвінок, запустити браузер або перейти за вказаною адресою. В Android-додатках багато операцій працюють через наміри.

Найбільш поширений сценарій використання Intent – це запуск іншої активності в своєму додатку.

Всі використовувані класи Activity повинні бути описані у файлі AndroidManifest.xml за допомогою елемента <activity>. Кожен подібний елемент містить як мінімум один атрибут «android: name», який встановлює ім'я класу Activity.

Однак activity – це стандартні класи java, які успадковуються від класу Activity або його спадкоємців. Тому замість вбудованих шаблонів в Android Studio (під час виконання лабораторних використовували empty activity) можемо додавати звичайні класи, і потім їх успадковувати від класу Activity. Однак в цьому випадку потрібно буде власноручі додавати в файл маніфесту дані про activity.

Для запуску нового екрану необхідно створити екземпляр класу Intent і вказати в першому параметрі контекст з поточним класом, а в другому – клас для переходу. Нехай поточний основний екран представлено класом CurrentActivity, а клас для переходу NewActivity:

Intent myIntent = new Intent(CurrentActivity.this, NewActivity.class);

Об’єкту класа Intent можна встановити різні флаги, які обумовлять його роботу. Наприклад, якщо не потрібна анімація переходу між активностями то можна використати флаг Intent.FLAG\_ACTIVITY\_NO\_ANIMATION:

myIntent.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);

Після встановлення необхідних налаштувань викликається метод startActivity (), який і запускає новий екран:

startActivity(myIntent);

* 1. Розробка гри для мобільного пристрою

Одним з найпопулярніших жанрів мобільних ігор є 2D платформери. Платформер (Platformer) – це жанр ігор, в яких основною рисою ігрового процесу є стрибання по платформах, лазання сходами, збирання предметів, зазвичай необхідних для завершення рівня в 2D просторі.

Ігри подібного жанру характеризуються нереалістичністю, мальованою мультяшною графікою. Героями таких ігор зазвичай бувають міфічні істоти або антропоморфні тварини.

За мету взято створення 2D гри без використання будь-яких ігрових движків, що зумовлює використання вбудованих засобів роботи з графікою.

* + 1. Вибір ідеї гри

В’єтнамським розробником Донг Нгуєном 24 травня 2013 року була випущена мобільна гра для платформ iOS та Android «Flappy Bird» (рис. 4.1), яка сподобалась мільйонам людей. Але нажаль вже 10 лютого 2014 року вона була видалена розробником.

Під час гри гравець керує польотом пташки, яка рухається між рядами перешкод у вигляді вертикальних зелених труб, що утворюють тунель складної форми. При зіткненні з ними гра завершується. Пташка летить вперед сама, управління здійснюється торканням до екрана – тоді пташка змахує крилами і робить невеликий ривок вгору. В іншому разі пташка не махає крилами і падає. За кожний успішний проліт між двома трубами зараховуються очки.

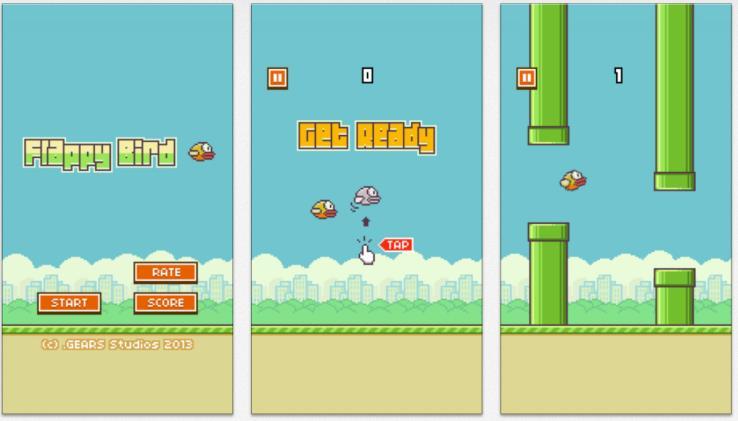


Рисунок 4.1 – Скриншоти оригінальної гри «Flappy Bird»

Тож було прийнято рішення розробити гру за мотивами «Flappy Bird».

* + 1. Розробка ресурсів для гри

В яксоті назви гри було прийнято рішення використати фразу «Game Bird», яка буде відсилкою до оригінальної гри.

З мережі інтернет було завантажено набір текстур з оригінальної гри (рис. 4.2).

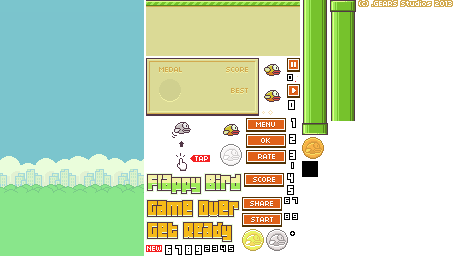


Рисунок 4.2 – Завантажені текстури

За допомогою графічного редактору фотошоп було вирізано і відредаговано необхідні текстури для гри, що розробляється (рис. 4.3).



Рисунок 4.3 – Текстури для гри, що розробляється

* + 1. Розробка інтерфейсу

В якості інтерфейсу було прийнято рішення використати два вікна (activity):

* меню гри (з кнопками «Розпочати гру» та «Закрити гру»);
* сама гра.

В результаті компонування елементів керування в контейнерах Layout було отримано ієрархію елементів вікна меню, яку зображено на рисунку 4.4.

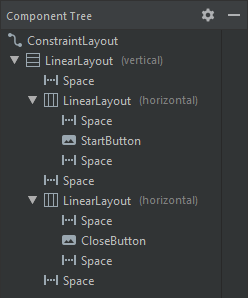


Рисунок 4.4 – Ієрархія View компонентів вікна меню

На рисунку 4.5 наведено зображення розробленого інтерфейсу вікна меню в середовищі Android Studio.

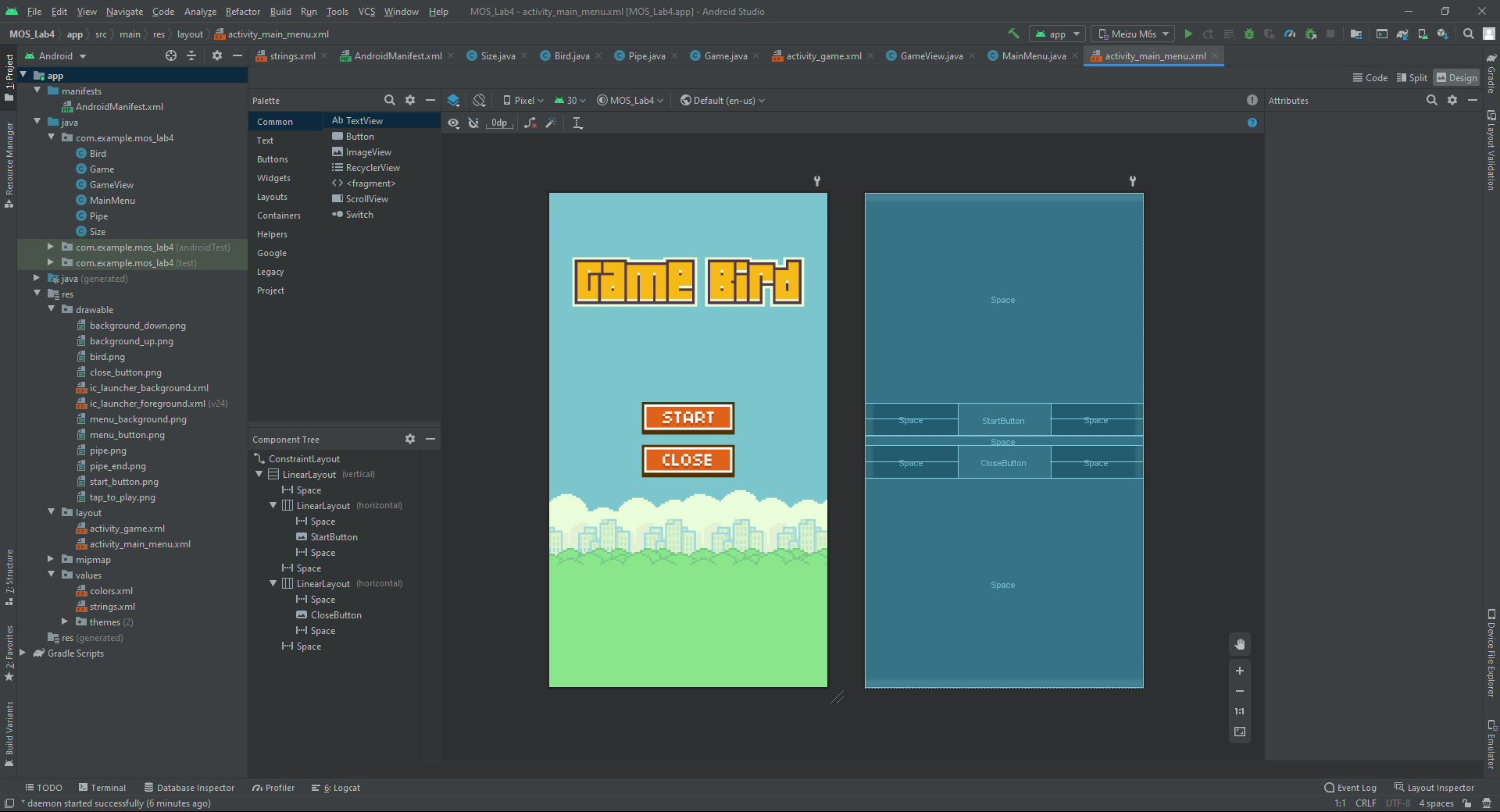


Рисунок 4.5 – Розроблений інтерфейс вікна меню в Android Studio

Вікно гри має такий дизайн, схожий на вікно меню. В результаті компонування елементів керування в контейнерах Layout було отримано ієрархію елементів вікна гри, яку зображено на рисунку 4.6.

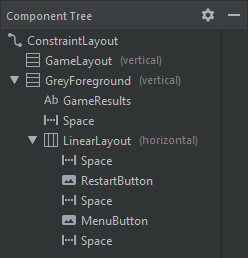


Рисунок 4.6 – Ієрархія View компонентів вікна гри

На рисунку 4.7 наведено зображення розробленого інтерфейсу вікна гри в середовищі Android Studio.

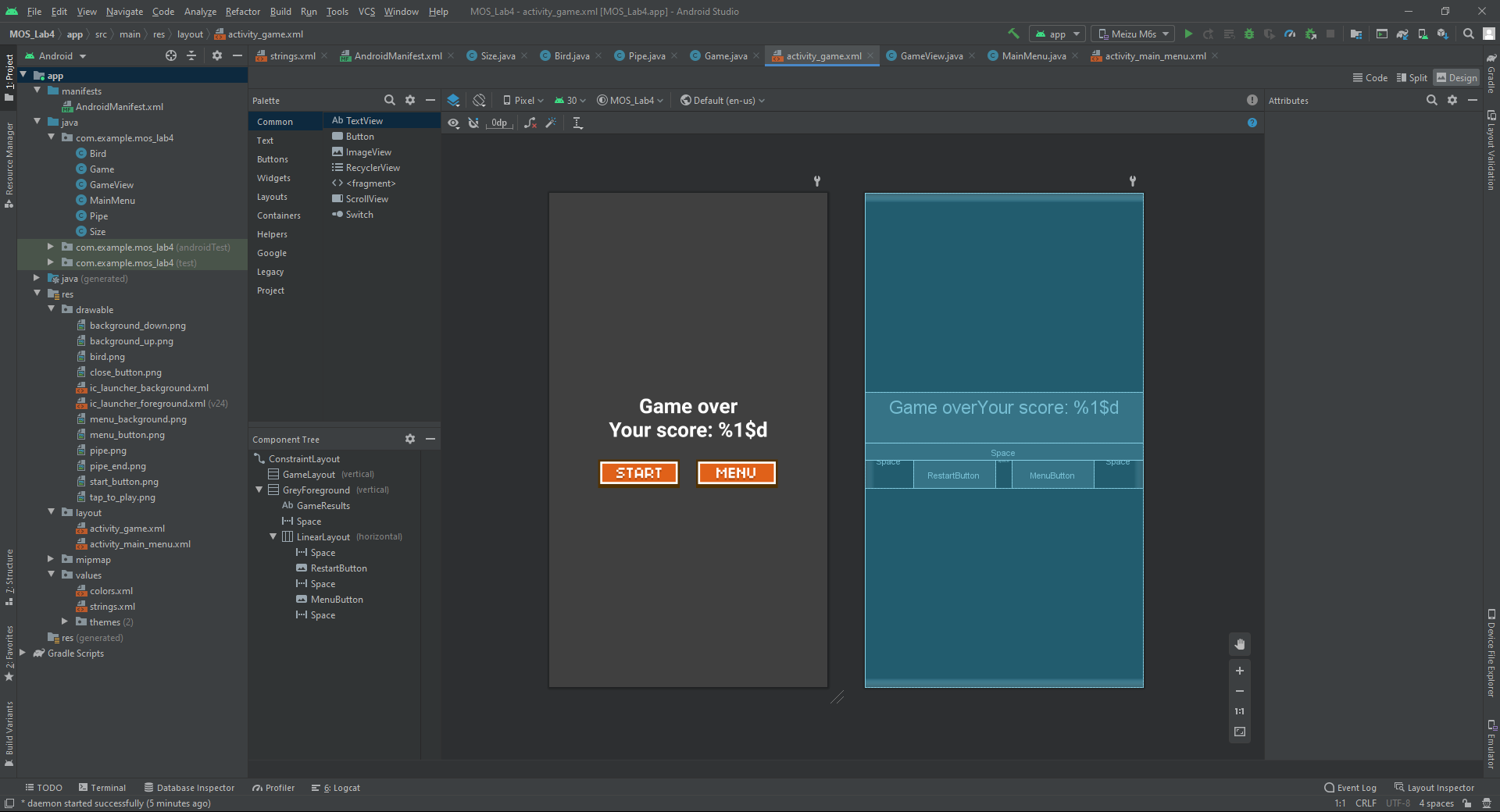


Рисунок 4.7 – Розроблений інтерфейс вікна гри в Android Studio

* + 1. Game Bird

При запуску розробленої гри відкривається головне меню з кнопками «Start» та «Close» (рис. 4.8).

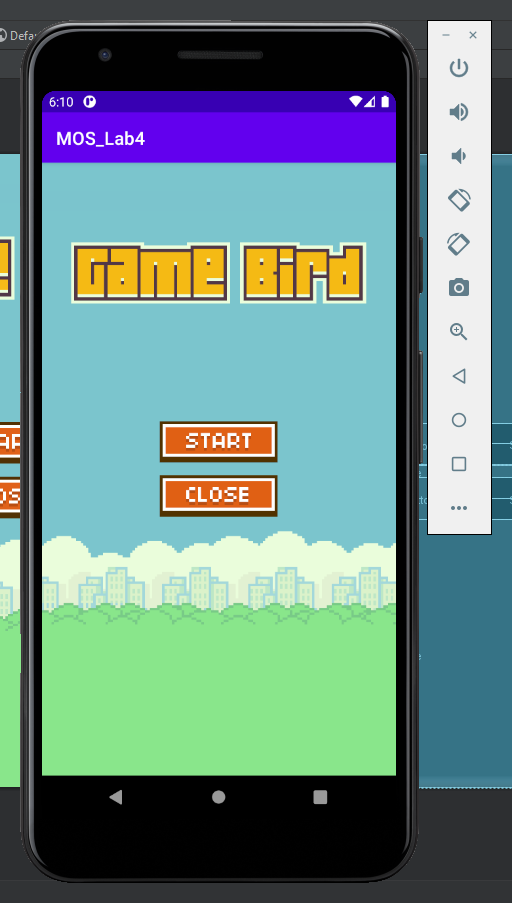


Рисунок 4.8 –Меню гри «Game Bird»

Після натискання на кнопку «Start» відкривається вікно з грою (рис. 4.9). На екрані відображено зображення з інструкцієї до гри. Щоб розпочати гру, необхідно натиснути на екран (тап по екрану в будь-якій точці).

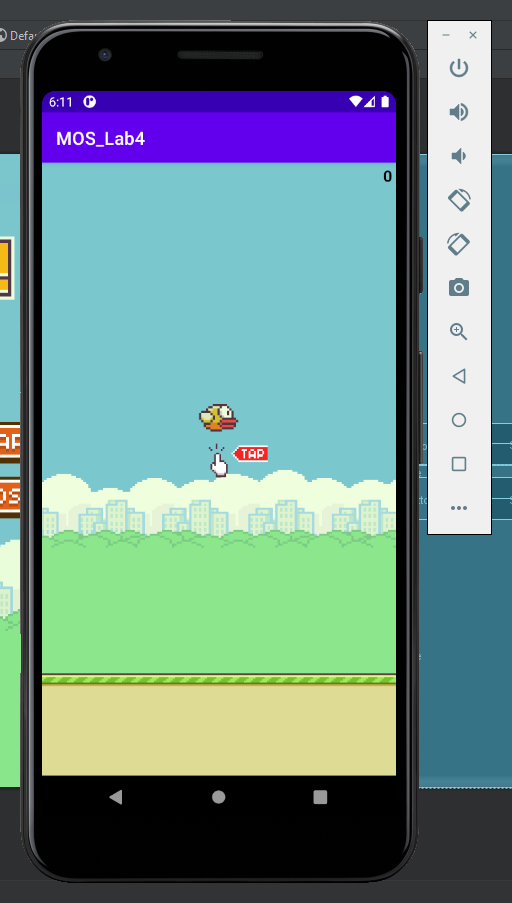


Рисунок 4.9 – Очікування початку гри «Game Bird»

Після натискання на екран ігра розпочата. При натисканні на екран птичка перестає падати та здіймається вище (рис. 4.10).

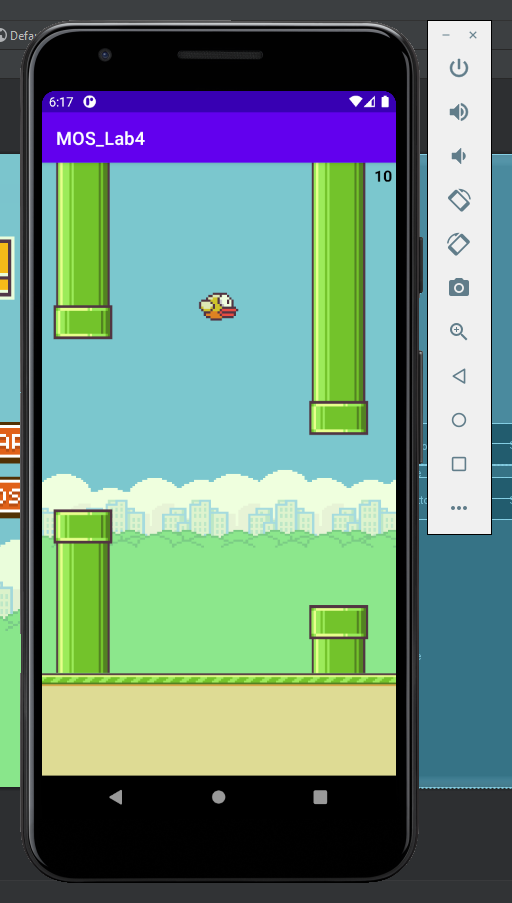


Рисунок 4.10 – Ігровий процес гри «Game Bird»

Якщо птичка доторкається до труби, верхньої або нижньої частини ігрового поля, то ігра завершується. На екрані з’являється повідомлення про закінчення гри з інформацією про кількість набраних очок (рис. 4.11).

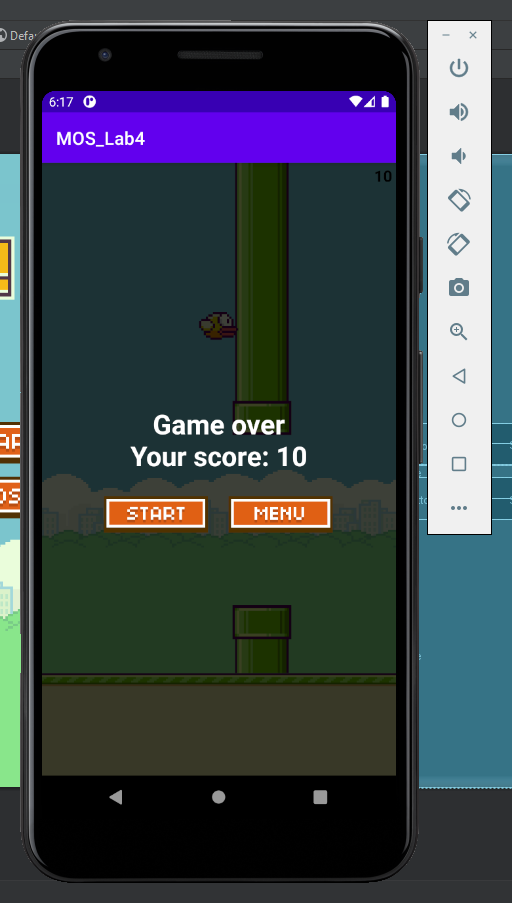


Рисунок 4.11 – Кінець гри «Game Bird»

При натисканні кнопки «Close» в меню з’являється Alert діалог закриття додатку з кнопками «Yes» та «No» (рис. 4.12).

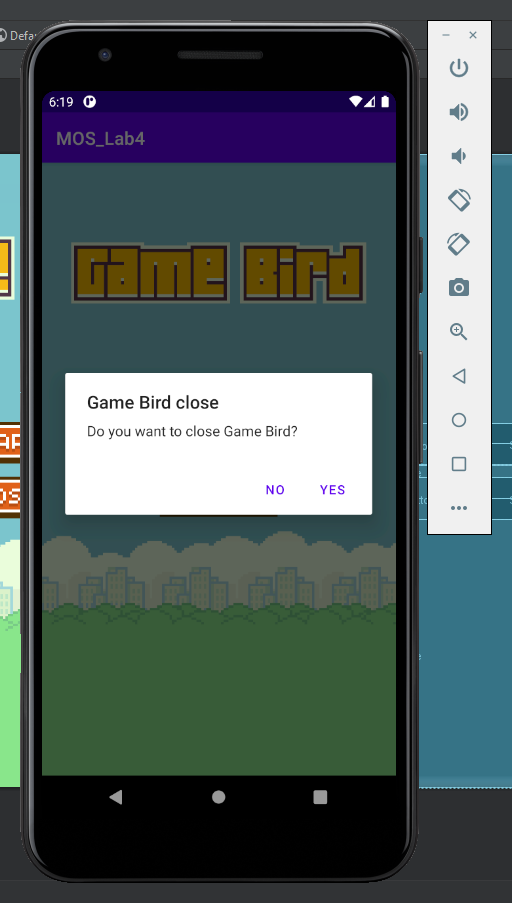


Рисунок 4.12 – Закриття мобільного додатку «Game Bird»

Ігра була розроблена так, щоб могла працювати та правильно відображатися на різних пристроях з різною роздільною здатністю екрана. На рисунку 4.13 наведено скріншот зроблений під час тестування гри не в емуляторі Android пристрою, а на смартфоні Meizu M6s.

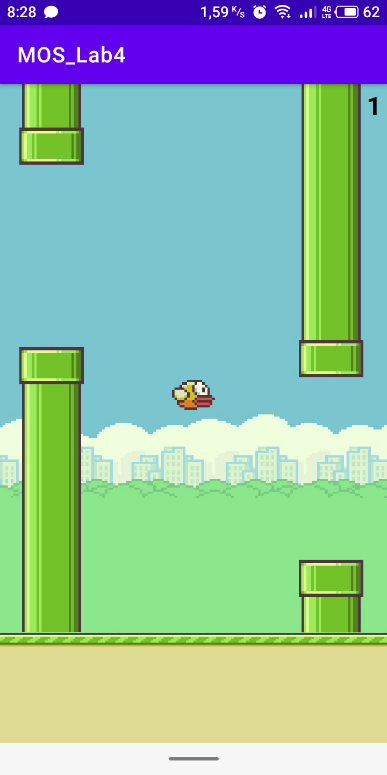


Рисунок 4.13 – Скріншот гри «Game Bird» запущеної на Meizu M6s

* + 1. Вихідний код

Проект гри розробленої в Android Studio містить такі основні класи:

* MainMenu – клас, що описує activity меню (вікно меню). При відкритті додатку створюється екземпляр класа MainMenu та відкривається інтерфейс, що описано в файлі «activity\_main\_menu.xml»;
* Game – клас, що описує activity гра (вікно гри). При відкритті вікна гри створюється екземпляр класа Game та відкривається інтерфейс, що описано в файлі «activity\_game.xml»;
* GameView – клас, що реалізує логіку гри й відповідає за малювання текстур та об’єктів у вікні гри.
* Bird – клас, що реалізує ігрового персонажа – птичку;
* Pipe – клас, що реалізує неживих ігрових персонажей – труби, які є перепонами для польоту птички;
* Size – mutable клас, який призначений для зберігання даних про двовимірний розмір об’єкта (ширина, висота). Створено для використання в якості альтернативи immutable класу android.util.Size.

Зміст основних файлів проекту наведено в пунктах 4.6.5.1 – 4.6.5.10.

* + - 1. AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.mos\_lab4">  
  
 <application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/Theme.MOS\_Lab4">  
 <activity android:name=".MainMenu">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 <activity android:name=".Game" />  
 </application>  
  
</manifest>

* + - 1. Bird.java

package com.example.mos\_lab4;  
  
import android.content.Context;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Canvas;  
import android.graphics.Paint;  
import android.graphics.PointF;  
  
public class Bird {  
 protected PointF coordinates;  
 protected Size sizeInUnits = new Size(2, 2);  
  
 protected float speed = 0;  
 protected float maxSpeed = (float) 1.25;  
 protected float minSpeed = (float) -1.5;  
 protected float velocity = (float) 0.2;  
  
 protected int imageId = R.drawable.bird;  
 protected Bitmap bitmap;  
  
 public Bird(Context context) {  
 coordinates = new PointF( (GameView.gameUnitCount.width - sizeInUnits.width) / (float) 2.0, (GameView.gameUnitCount.height - sizeInUnits.height) / (float) 2.0);  
 initImage(context);  
 }  
  
 private void initImage(Context context) {  
 Bitmap bitmapImage = BitmapFactory.decodeResource(context.getResources(), imageId);  
 bitmap = Bitmap.createScaledBitmap(bitmapImage, (int) (sizeInUnits.width \* GameView.unitSize.width), (int) (sizeInUnits.height \* GameView.unitSize.height), false);  
 bitmapImage.recycle();  
 }  
  
 public void touchBird() {  
 speed = maxSpeed;  
 }  
  
 public void draw(Paint paint, Canvas canvas) {  
 canvas.drawBitmap(bitmap, coordinates.x \* GameView.unitSize.width, coordinates.y \* GameView.unitSize.height, paint);  
 }  
  
 public void update() {  
 coordinates.y -= speed;  
 speed = Math.max(speed - velocity, minSpeed);  
 }  
  
 public boolean isHitFloorOrSky() {  
 return coordinates.y < 0 || coordinates.y + sizeInUnits.height > GameView.gameUnitCount.height;  
 }  
}

* + - 1. Game.java

package com.example.mos\_lab4;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.MotionEvent;  
import android.view.View;  
import android.widget.ImageButton;  
import android.widget.LinearLayout;  
import android.widget.TextView;  
  
public class Game extends AppCompatActivity implements View.OnTouchListener {  
 private LinearLayout GameLayout;  
 private GameView gameView;  
  
 private ImageButton RestartButton;  
 private ImageButton MenuButton;  
  
 private LinearLayout GreyLayout;  
 private TextView GameResults;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_game);  
  
 GreyLayout = (LinearLayout) findViewById(R.id.GreyForeground);  
 GameResults = (TextView) findViewById(R.id.GameResults);  
  
 RestartButton = (ImageButton) findViewById(R.id.RestartButton);  
 MenuButton = (ImageButton) findViewById(R.id.MenuButton);  
 RestartButton.setOnTouchListener(this);  
 MenuButton.setOnTouchListener(this);  
  
 GameLayout = (LinearLayout) findViewById(R.id.GameLayout);  
 GameLayout.setOnTouchListener(this);  
  
 startGame();  
 }  
  
 public void startGame() {  
 GreyLayout.setVisibility(View.GONE);  
 GameLayout.setEnabled(true);  
  
 if(gameView != null) {  
 GameLayout.removeView(gameView);  
 gameView.destroyDrawingCache();  
 }  
  
 gameView = new GameView(this);  
 GameLayout.addView(gameView);  
 }  
  
 @Override  
 public boolean onTouch(View v, MotionEvent event) {  
 if(v.getId() == R.id.RestartButton && event.getAction() == MotionEvent.ACTION\_DOWN) {  
 startGame();  
 }  
 else if(v.getId() == R.id.MenuButton && event.getAction() == MotionEvent.ACTION\_DOWN) {  
 Intent gameMenu = new Intent(this, MainMenu.class);  
 gameMenu.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);  
 startActivity(gameMenu);  
 finish();  
 overridePendingTransition(0, 0);  
 }  
 else if(v.getId() == R.id.GameLayout && event.getAction() == MotionEvent.ACTION\_DOWN)  
 gameView.performTouch();  
  
 return false;  
 }  
  
 public void showGameOver(int Score) {  
 runOnUiThread(new Runnable() {  
 @Override  
 public void run() {  
 GameLayout.setEnabled(false);  
 GreyLayout.setVisibility(View.VISIBLE);  
 GameResults.setText(getString(R.string.game\_results, Score));  
 }  
 });  
 }  
}

* + - 1. GameView.java

package com.example.mos\_lab4;  
  
import android.content.Context;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.graphics.Typeface;  
import android.view.SurfaceHolder;  
import android.view.SurfaceView;  
  
import java.util.ArrayList;  
  
public class GameView extends SurfaceView implements Runnable {  
 public static Size unitCount = new Size(18, 36);  
 public static Size gameUnitCount = new Size(18, 30);  
 public static Size unitSize = new Size(1, 1);  
  
 private int Score = 0;  
  
 private boolean firstIteration = true;  
 private boolean beforeGameStart = true;  
 private boolean gameRunning = true;  
  
 private Bird bird;  
 private ArrayList<Pipe> pipes = new ArrayList<Pipe>();  
  
 private Thread gameThread = null;  
 private Paint paint;  
 private Paint textPaint;  
 private Canvas canvas;  
 private SurfaceHolder surfaceHolder;  
  
 protected int backgroundUpId = R.drawable.background\_up;  
 protected int backgroundDownId = R.drawable.background\_down;  
 protected int tapToPlayId = R.drawable.tap\_to\_play;  
 protected Bitmap backgroundUp;  
 protected Bitmap backgroundDown;  
 protected Bitmap tapToPlay;  
 protected Size tapToPlaySize = new Size(3,2);  
  
 public GameView(Context context) {  
 super(context);  
 surfaceHolder = getHolder();  
  
 gameThread = new Thread(this);  
 gameThread.start();  
  
 paint = new Paint();  
 setupTextPaint();  
 }  
  
 public void setupTextPaint() {  
 textPaint = new Paint();  
 textPaint.setAntiAlias(true);  
 textPaint.setColor(Color.BLACK);  
 textPaint.setTextSize(48.0f);  
 textPaint.setTypeface(Typeface.create(Typeface.DEFAULT, Typeface.BOLD));  
 }  
  
 public void performTouch()  
 {  
 if(beforeGameStart)  
 beforeGameStart = false;  
  
 if(bird != null)  
 bird.touchBird();  
 }  
  
 @Override  
 public void run() {  
 while(beforeGameStart) {  
 draw();  
 pause();  
 }  
 while(gameRunning) {  
 update();  
 updateScore();  
 updatePipesList();  
 checkGameOverCollision();  
 draw();  
 pause();  
 }  
 ((Game) getContext()).showGameOver(Score);  
 }  
  
 private void update() {  
 if(!firstIteration) {  
 bird.update();  
 for(Pipe pipe : pipes)  
 pipe.update();  
 }  
 }  
  
 private void updateScore()  
 {  
 for(Pipe pipe : pipes)  
 if(pipe.isScoreUpdate(bird.coordinates, bird.sizeInUnits))  
 Score += 1;  
 }  
  
 private void updatePipesList() {  
 if(pipes.size() > 0 && pipes.get(0).isEnded()) {  
 pipes.remove(0);  
 pipes.add(new Pipe(getContext(), pipes.get(pipes.size() - 1).holeCoordinates));  
 }  
 }  
  
 private void checkGameOverCollision() {  
 if(bird != null && bird.isHitFloorOrSky())  
 gameRunning = false;  
  
 for(Pipe pipe : pipes) {  
 if(pipe.isCollision(bird.coordinates, bird.sizeInUnits))  
 gameRunning = false;  
 }  
 }  
  
 private void draw() {  
 if (surfaceHolder.getSurface().isValid()) {  
  
 if(firstIteration) {  
 firstIteration = false;  
  
 unitSize.width = surfaceHolder.getSurfaceFrame().width()/unitCount.width;  
 unitSize.height = surfaceHolder.getSurfaceFrame().height()/unitCount.height;  
  
 initBackgrounds();  
 initTapToPlayImage();  
  
 bird = new Bird(getContext());  
  
 pipes.add(new Pipe(getContext()));  
 pipes.add(new Pipe(getContext(), pipes.get(pipes.size() - 1).holeCoordinates));  
 pipes.add(new Pipe(getContext(), pipes.get(pipes.size() - 1).holeCoordinates));  
 }  
  
 canvas = surfaceHolder.lockCanvas();  
  
 canvas.drawBitmap(backgroundUp, 0, 0, paint);  
 canvas.drawBitmap(backgroundDown, 0, gameUnitCount.height \* unitSize.height, paint);  
  
 for(Pipe pipe : pipes)  
 pipe.draw(paint, canvas);  
  
 bird.draw(paint, canvas);  
  
 if(beforeGameStart) {  
 canvas.drawBitmap(tapToPlay, (bird.coordinates.x + (float) 0.5) \* unitSize.width, (bird.coordinates.y + bird.sizeInUnits.height + (float) 0.5) \* unitSize.height, paint);  
 }  
  
 float textWidth = textPaint.measureText(String.valueOf(Score));  
 canvas.drawText(String.valueOf(Score), gameUnitCount.width \* unitSize.width - textWidth - 10, textPaint.getTextSize() + 10, textPaint);  
  
 surfaceHolder.unlockCanvasAndPost(canvas);  
 }  
 }  
  
 private void initBackgrounds()  
 {  
 Bitmap bitmapImage = BitmapFactory.decodeResource(getContext().getResources(), backgroundUpId);  
 backgroundUp = Bitmap.createScaledBitmap(bitmapImage, (int)(gameUnitCount.width \* unitSize.width), (int)(gameUnitCount.height \* unitSize.height + 1), false);  
 bitmapImage.recycle();  
  
 bitmapImage = BitmapFactory.decodeResource(getContext().getResources(), backgroundDownId);  
 backgroundDown = Bitmap.createScaledBitmap(bitmapImage, (int)(unitCount.width \* unitSize.width), (int)((unitCount.height - gameUnitCount.height) \* unitSize.height), false);  
 bitmapImage.recycle();  
 }  
  
 private void initTapToPlayImage() {  
 Bitmap bitmapImage = BitmapFactory.decodeResource(getContext().getResources(), tapToPlayId);  
 tapToPlay = Bitmap.createScaledBitmap(bitmapImage, (int)(tapToPlaySize.width \* unitSize.width), (int)(tapToPlaySize.height \* unitSize.height), false);  
 bitmapImage.recycle();  
 }  
  
 private void pause() {  
 try {  
 gameThread.sleep(20);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

* + - 1. MainMenu.java

package com.example.mos\_lab4;  
  
import androidx.appcompat.app.AlertDialog;  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.content.DialogInterface;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.MotionEvent;  
import android.view.View;  
import android.widget.ImageButton;  
  
public class MainMenu extends AppCompatActivity implements View.OnTouchListener {  
  
 protected ImageButton StartButton;  
 protected ImageButton CloseButton;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main\_menu);  
  
 StartButton = (ImageButton) findViewById(R.id.StartButton);  
 CloseButton = (ImageButton) findViewById(R.id.CloseButton);  
 StartButton.setOnTouchListener(this);  
 CloseButton.setOnTouchListener(this);  
 }  
  
 @Override  
 public boolean onTouch(View v, MotionEvent event) {  
 if(v.getId() == R.id.StartButton && event.getAction() == MotionEvent.ACTION\_DOWN)  
 {  
 Intent game = new Intent(this, Game.class);  
 game.addFlags(Intent.FLAG\_ACTIVITY\_NO\_ANIMATION);  
 startActivity(game);  
 finish();  
 overridePendingTransition(0, 0);  
 }  
 if(v.getId() == R.id.CloseButton && event.getAction() == MotionEvent.ACTION\_DOWN){  
 closeDialog();  
 }  
  
 return false;  
 }  
  
 protected void closeDialog()  
 {  
 AlertDialog.Builder builder = new AlertDialog.Builder(this);  
 builder.setMessage("Do you want to close Game Bird?")  
 .setCancelable(false)  
 .setPositiveButton("Yes", new DialogInterface.OnClickListener() {  
 public void onClick(DialogInterface dialog, int id) {  
 finish();  
 }  
 })  
 .setNegativeButton("No", new DialogInterface.OnClickListener() {  
 public void onClick(DialogInterface dialog, int id) {  
 dialog.cancel();  
 }  
 });  
 AlertDialog alert = builder.create();  
 alert.setTitle("Game Bird close");  
 alert.show();  
 }  
}

* + - 1. Pipe.java

package com.example.mos\_lab4;  
  
import android.content.Context;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Canvas;  
import android.graphics.Paint;  
import android.graphics.PointF;  
  
import java.util.Random;  
public class Pipe {  
 protected PointF holeCoordinates;  
 protected Size holeSizeInUnits = new Size(3, 10);  
 protected Size pipeEndSizeInUnits = new Size(holeSizeInUnits.width, 2);  
  
 protected float holeDifferenceInUnits = 14;  
 protected float pipeDistanceInUnits = 10;  
 protected float pipeDistanceFromField = pipeEndSizeInUnits.height \* 2;  
 protected float speed = (float) 0.3;  
 protected boolean isScored = false;  
  
 protected int pipeImageId = R.drawable.pipe;  
 protected int pipeEndImageId = R.drawable.pipe\_end;  
 protected Bitmap bitmapUpperPipe;  
 protected Bitmap bitmapLowerPipe;  
 protected Bitmap bitmapPipeEnd;  
  
 public Pipe(Context context) {  
 PointF previousCenterCoordinates = new PointF(GameView.gameUnitCount.width - pipeDistanceInUnits, (GameView.gameUnitCount.height - holeSizeInUnits.height) / (float) 2.0);  
 init(context, previousCenterCoordinates);  
 }  
  
 public Pipe(Context context, PointF previousCenterCoordinates) {  
 init(context, previousCenterCoordinates);  
 }  
  
 protected void init(Context context, PointF previousCenterCoordinates) {  
 Random random = new Random();  
 float randomHeightDifference = (2 \* random.nextFloat() - 1) \* holeDifferenceInUnits;  
 float newHoleCoordinateY = previousCenterCoordinates.y + randomHeightDifference;  
  
 // Random hole coordinate, that does not depend on the previous  
 // float newHoleCoordinateY = pipeDistanceFromField + random.nextFloat() \* (GameView.gameUnitCount.height - 2 \* pipeDistanceFromField - holeSizeInUnits.height);  
  
 newHoleCoordinateY = Math.max(pipeDistanceFromField, newHoleCoordinateY);  
 newHoleCoordinateY = Math.min(GameView.gameUnitCount.height - pipeDistanceFromField - holeSizeInUnits.height, newHoleCoordinateY);  
  
 holeCoordinates = new PointF(previousCenterCoordinates.x + holeSizeInUnits.width + pipeDistanceInUnits, newHoleCoordinateY);  
  
 initImage(context);  
 }  
  
 void initImage(Context context) {  
  
 Bitmap bitmapImage = BitmapFactory.decodeResource(context.getResources(), pipeEndImageId);  
 bitmapPipeEnd = Bitmap.createScaledBitmap(bitmapImage, (int) (pipeEndSizeInUnits.width \* GameView.unitSize.width), (int) ((pipeEndSizeInUnits.height \* GameView.unitSize.height) + 1), false);  
 bitmapImage.recycle();  
  
 bitmapImage = BitmapFactory.decodeResource(context.getResources(), pipeImageId);  
 bitmapUpperPipe = Bitmap.createScaledBitmap(bitmapImage, (int) (pipeEndSizeInUnits.width \* GameView.unitSize.width), (int) (((holeCoordinates.y - pipeEndSizeInUnits.height) \* GameView.unitSize.height) + 1), false);  
 bitmapImage.recycle();  
  
 bitmapImage = BitmapFactory.decodeResource(context.getResources(), pipeImageId);  
 bitmapLowerPipe = Bitmap.createScaledBitmap(bitmapImage, (int) (pipeEndSizeInUnits.width \* GameView.unitSize.width), (int) ((GameView.gameUnitCount.height - holeCoordinates.y - pipeEndSizeInUnits.height - holeSizeInUnits.height) \* GameView.unitSize.height), false);  
 bitmapImage.recycle();  
 }  
  
 public void draw(Paint paint, Canvas canvas) {  
 canvas.drawBitmap(bitmapUpperPipe, holeCoordinates.x \* GameView.unitSize.width, 0, paint);  
 canvas.drawBitmap(bitmapLowerPipe, holeCoordinates.x \* GameView.unitSize.width, (holeCoordinates.y + holeSizeInUnits.height + pipeEndSizeInUnits.height) \* GameView.unitSize.height - 1, paint);  
 canvas.drawBitmap(bitmapPipeEnd, holeCoordinates.x \* GameView.unitSize.width, (holeCoordinates.y - pipeEndSizeInUnits.height) \* GameView.unitSize.height, paint);  
 canvas.drawBitmap(bitmapPipeEnd, holeCoordinates.x \* GameView.unitSize.width, (holeCoordinates.y + holeSizeInUnits.height) \* GameView.unitSize.height, paint);  
 }  
  
 public boolean isEnded() {  
 return holeCoordinates.x + holeSizeInUnits.width < -2;  
 }  
  
 public boolean isCollision(PointF birdLocation, Size birdSizeInUnits) {  
 if(birdLocation.x + birdSizeInUnits.width < holeCoordinates.x || birdLocation.x > holeCoordinates.x + holeSizeInUnits.width)  
 return false;  
  
 if(birdLocation.y > holeCoordinates.y && birdLocation.y + birdSizeInUnits.height < holeCoordinates.y + holeSizeInUnits.height)  
 return false;  
  
 return true;  
 }  
  
 public boolean isScoreUpdate(PointF birdLocation, Size birdSizeInUnits)  
 {  
 if(!isScored && birdLocation.x > holeCoordinates.x + holeSizeInUnits.width / (float) 2.0) {  
 isScored = true;  
 return true;  
 }  
  
 return false;  
 }  
  
 public void update() {  
 holeCoordinates.x -= speed;  
 }  
}

* + - 1. Size.java

package com.example.mos\_lab4;  
  
public class Size {  
 public float width;  
 public float height;  
  
 public Size(float width, float height) {  
 this.width = width;  
 this.height = height;  
 }  
}

* + - 1. activity\_game.xml

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="#03FFFFFF"  
 tools:context=".Game">  
  
 <LinearLayout  
 android:id="@+id/GameLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical">  
  
 </LinearLayout>  
  
 <LinearLayout  
 android:id="@+id/GreyForeground"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="#BF000000"  
 android:gravity="center"  
 android:orientation="vertical">  
  
 <TextView  
 android:id="@+id/GameResults"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:textAlignment="center"  
 android:textColor="#FFFFFF"  
 android:textSize="30sp"  
 android:textStyle="bold"  
 tools:text="@string/game\_results" />  
  
 <Space  
 android:layout\_width="match\_parent"  
 android:layout\_height="25dp" />  
 <LinearLayout  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal">  
  
 <Space  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="3" />  
  
 <ImageButton  
 android:id="@+id/RestartButton"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="5"  
 android:adjustViewBounds="true"  
 android:contentDescription="@string/restart\_button"  
 android:padding="0dp"  
 android:scaleType="centerCrop"  
 app:srcCompat="@drawable/start\_button" />  
  
 <Space  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="1" />  
  
 <ImageButton  
 android:id="@+id/MenuButton"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="5"  
 android:adjustViewBounds="true"  
 android:contentDescription="@string/restart\_button"  
 android:padding="0dp"  
 android:scaleType="centerCrop"  
 app:srcCompat="@drawable/menu\_button" />  
  
 <Space  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="3" />  
 </LinearLayout>  
 </LinearLayout>  
  
</androidx.constraintlayout.widget.ConstraintLayout>

* + - 1. activity\_main\_menu.xml

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainMenu">  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:gravity="center"  
 android:orientation="vertical"  
 android:background="@drawable/menu\_background"  
 android:scaleType="centerCrop">  
  
 <Space  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="20" />  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:gravity="center"  
 android:orientation="horizontal">  
  
 <Space  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="5" />  
  
 <ImageButton  
 android:id="@+id/StartButton"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="5"  
 android:adjustViewBounds="true"  
 android:contentDescription="@string/start\_button"  
 android:padding="0dp"  
 android:scaleType="centerCrop"  
 app:srcCompat="@drawable/start\_button" />  
  
 <Space  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="5" />  
 </LinearLayout>  
  
 <Space  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="1" />  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:gravity="center"  
 android:orientation="horizontal">  
  
 <Space  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="5" />  
  
 <ImageButton  
 android:id="@+id/CloseButton"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="5"  
 android:adjustViewBounds="true"  
 android:contentDescription="@string/close\_button"  
 android:padding="0dp"  
 android:scaleType="centerCrop"  
 app:srcCompat="@drawable/close\_button" />  
  
 <Space  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="5" />  
 </LinearLayout>  
  
 <Space  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="20" />  
 </LinearLayout>  
</androidx.constraintlayout.widget.ConstraintLayout>

* + - 1. strings.xml

<resources>  
 <string name="app\_name">MOS\_Lab4</string>  
 <string name="game\_results">Game over\nYour score: %1$d</string>  
 <string name="restart\_button">Restart button</string>  
 <string name="start\_button">Start button</string>  
 <string name="close\_button">Close button</string>  
</resources>

1. Відповіді на контрольні запитання
   1. Що таке графіка? Особливості графіки у мобільних пристроях.

Комп'ютерна графіка – це розділ інформатики, який вивчає методи цифрового синтезу і обробки зорового контенту. Робота з комп'ютерною графікою – один з найпоширеніших напрямків використання мобільних пристроїв та комп'ютера.

Мобільні додатки сильно обмежені в ресурсах процесора та відеокарти мобільного пристрою. Проте такі обмеження роблять процес створення мобільних ігор більш простим у порівнянні з розробкою для комп’ютерних або консольних ігор.

* 1. Що таке фонове зображення?

Фонове зображення – це одне або група зображень, що формує задній фон додатку та може використовуватися в якості заставки (найчастіше у прикладних застосунках) або в якості сцени (найчастіше за допомогою шарів в ігрових застосунках).

* 1. Які класи слід використовувати для виводу графіки?

При виводі графіки можна використовувати клас Canvas. Цей клас надає методи для малювання, які відображають графічні примітиви на вихідному растровому зображенні. Отримати доступ до полотна можна за допомогою об’єкта Bitmap або компонента View. За допомогою метода drawBitmap () можна намалювати Bitmap зображення на полотні, а за допомогою drawText () можна відобразити текст.

* 1. Для чого призначений класи SurfaceView та SurfaceHolder?

Клас SurfaceView надає об’єкт Surface, який підтримує малювання в фоновому потоці і дає можливість використовувати OpenGL для тривимірної графіки. Це відмінний варіант для насичених графікою елементів, які потребують частих оновленнях або повинні відображати складну графічну інформацію, як у випадку з іграми і тривимірною візуалізацією.

SurfaceView – обгортка навколо класу SurfaceHolder, який в свою чергу служить обгорткою класу Surface, використовуваного для відновлення зображення з фонових потоків. SurfaceView використовується точно таким же чином, як будь-які похідні від View класи. Можна застосовувати анімацію і розміщувати їх всередині розмітки так само, як і інші компоненти.

* 1. Як запустити нову Activity?

Для запуску нових активностей в мобільному додатку варто використовувати клас Intent.

Для запуску нової Activity необхідно створити екземпляр класу Intent і вказати в першому параметрі конструктора контекст з поточним класом, а в другому – клас для переходу. Об’єкту класа Intent можна встановити різні флаги, які обумовлять його роботу. Після встановлення необхідних налаштувань необхідно викликати метод startActivity (), який і запустить нову Activity.

Висновки

Протягом виконання цієї лабораторної роботи ознайомився із принципами використання графічних засобів в мобільних додатках.

Розглянув принципи використання стандартних класів для роботи з графікою (Color, Paint, Bitmap, BitFactory, Canvas, SurfaceView, SurfaceHolder) та роботи з активностями (Intent).

Розробив гру «Game Bird» для мобільного пристрою за допомогою середовища розробки Android Studio на основі використання вивчених засобів побудови комп’ютерної графіки.