|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Politechnika Bydgoska im. Jana i Jędrzeja Śniadeckich  Wydział Telekomunikacji, Informatyki i Elektrotechniki  al. prof. S. Kaliskiego 7, 85-796 Bydgoszcz | | | |  | |
| Przedmiot | **Programowanie urządzeń mobilnych** | | | Kierunek/Tryb |  |
| Nr laboratorium | 8 | Data wykonania |  | Grupa |  |
| Ocena |  | Data oddania |  | Imię Nazwisko |  |
| Nazwa ćwiczenia | Animacja, SurfaceView, fizyka obiektów | | |

**Cel ćwiczenia laboratoryjnego**

Celem ćwiczenia laboratoryjnego jest zapoznanie studentów z technikami programowania aplikacji mobilnych na platformę Android, ze szczególnym uwzględnieniem implementacji animacji oraz interakcji użytkownika z aplikacją. Studenci mają za zadanie stworzyć aplikację, która wykorzystuje mechanizmy animacji obiektów graficznych, takie jak piłki, oraz implementuje dodatkowe funkcjonalności, takie jak grawitacja, opór powietrza, wytracanie prędkości, a także interfejs użytkownika z menu i opcjami. Ćwiczenie ma na celu rozwinięcie umiejętności programistycznych, zrozumienie zasad działania pętli gry oraz zastosowanie różnych technik programistycznych w praktyce.

**Zakres Tematyki**

W ramach ćwiczenia poruszane są następujące zagadnienia:

* Implementacja animacji w aplikacjach mobilnych na platformę Android.
* Tworzenie i zarządzanie obiektami graficznymi w aplikacji.
* Zastosowanie fizyki w animacjach, w tym grawitacji, oporu powietrza i wytracania prędkości.
* Projektowanie interfejsu użytkownika z wykorzystaniem widoków i elementów interfejsu, takich jak menu, opcje i ekrany informacyjne.
* Implementacja pętli gry, która zarządza aktualizacją logiki gry oraz renderowaniem grafiki.
* Zastosowanie dźwięków w aplikacji mobilnej.
* Zapamiętywanie stanu i pozycji obiektów w aplikacji.

**Zadanie**

Zadanie stojące przed studentami polega na stworzeniu aplikacji mobilnej na platformę Android, która wykorzystuje animacje obiektów graficznych z uwzględnieniem fizyki (grawitacja, opór powietrza, wytracanie prędkości), oraz implementuje interfejs użytkownika z menu, opcjami i dodatkowymi ekranami, a także funkcjonalności dźwiękowe i zapamiętywanie stanu aplikacji.

**Opcjonalne rozszerzenia**

* Zastosowanie obrazków zamiast prostych obiektów geometrycznych: Zamiast rysowania prostych kształtów, takich jak piłki, kwadraty czy trójkąty, można użyć obrazków, aby nadać aplikacji bardziej atrakcyjny wizualnie wygląd.
* Dodanie efektów specjalnych: Implementacja dodatkowych efektów graficznych, takich jak cienie, odbicia, czy efekty świetlne, aby zwiększyć atrakcyjność wizualną aplikacji.
* Rozbudowa interakcji użytkownika: Dodanie bardziej zaawansowanych interakcji użytkownika, takich jak gesty dotykowe (przesuwanie, powiększanie, obracanie) oraz reakcje na różne rodzaje dotyku (krótkie i długie naciśnięcie).

**Oczekiwane wyniki**

Oczekiwane wyniki ćwiczenia obejmują:

* Stworzenie działającej aplikacji mobilnej na platformę Android, która realizuje animację obiektów graficznych.
* Implementacja mechanizmów fizycznych, takich jak grawitacja, opór powietrza i wytracanie prędkości, w animacji obiektów.
* Zaprojektowanie i zaimplementowanie interfejsu użytkownika z menu, opcjami i ekranami informacyjnymi.
* Dodanie funkcjonalności dźwiękowych do aplikacji.
* Zapamiętywanie stanu i pozycji obiektów w aplikacji, umożliwiające kontynuowanie po jej wznowieniu.
* Zrozumienie i zastosowanie pętli gry do zarządzania aktualizacją logiki gry oraz renderowaniem grafiki.

**Zadania**

1. Do zaprezentowanej na końcu przykładowej pętli gry SimpleGameLoopApp dodaj grawitację, opór powietrza i wytracanie prędkości piłki, tak jak w przykładzie MyAnimationViewApp.
2. Dodaj inne klasy reprezentujące figury geometryczne np. kwadrat, trójkąt.
3. Dodaj warunek zakończenia (wstrzymania obliczeń nowych pozycji obiektów) i wyświetleniem informacji o możliwości przejścia do menu opcje.
4. Dodaj ekrany: ekran menu, ekran opcji, ekran informacji o autorach (Laboratorium 5).
5. W opcjach pozwól na zmianę szybkości, liczby i rodzajów obiektów.
6. Dodaj dźwięki np. do ekranu opcji, menu (Laboratorium 4).
7. Dodaj zapamiętanie stanu i pozycji obiektów (tak jak w Laboratorium 2)
8. Opcjonalnie zamiast obiektów można zastosować obrazki.

**Przykłady do wykorzystania w ćwiczeniu i opis propozycji rozwiązania zadania**

**Przykład 1. MyAnimationViewApp**

Poniższy przykład prezentuje aplikację na system Android, w której animacja poruszających się po ekranie piłek została zrealizowana przy użyciu klasy MyAnimationView, będącej rozszerzeniem klasy View. Program wykorzystuje mechanizmy losowania oraz aktualizacji pozycji piłek, aby uzyskać efekt animacji.

1. Otwórz Android Studio i utwórz nowy projekt.
2. Wybierz “Empty Views Activity” jako szablon projektu.
3. Nazwij projekt np. MyAnimationViewApp
4. Wprowadź zmiany w pliku MainActivity.java

package com.example.myanimationviewapp;  
import android.os.Bundle;  
import androidx.appcompat.app.AppCompatActivity;  
public class MainActivity extends AppCompatActivity {  
 private MyAnimationView animationView;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 *// Ustawiamy nasz widok jako główny content* animationView = new MyAnimationView(this);  
 setContentView(animationView);  
 }  
}

1. Dodaj Klasę Pilka i uzupełnij kod

package com.example.myanimationviewapp;  
import android.graphics.Canvas;  
import android.graphics.Paint;  
  
public class Pilka {  
 float x, y, radius, speedX, speedY;  
 int color;  
  
 public Pilka(float x, float y, float radius, float speedX, float speedY, int color) {  
 this.x = x;  
 this.y = y;  
 this.radius = radius;  
 this.speedX = speedX;  
 this.speedY = speedY;  
 this.color = color;  
 }  
  
 *// Metoda do aktualizacji pozycji piłki* public void update(float width, float height) {  
 *// Sprawdzenie kolizji z krawędziami ekranu* if (x + radius > width || x - radius <= 0) {  
 speedX = -speedX;  
 }  
 if (y + radius > height || y - radius <= 0) {  
 speedY = -speedY;  
 }  
  
 x += speedX;  
 y += speedY;  
 }  
  
 *// Metoda do rysowania piłki* public void draw(Canvas canvas, Paint paint) {  
 paint.setColor(color);  
 canvas.drawCircle(x, y, radius, paint);  
 }  
}

1. Dodaj klasę MyAnimationView i uzupełnij kod

package com.example.myanimationviewapp;  
  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.view.View;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
  
public class MyAnimationView extends View {  
 private List<Pilka> pilki = new ArrayList<>();  
 private Paint paint = new Paint();  
 private boolean isInitialized = false; *// Flaga, aby upewnić się, że piłki są tworzone tylko raz* public MyAnimationView(Context context) {  
 super(context); }  
  
 *// Ta metoda jest wywoływana, gdy zmienia się rozmiar widoku (np. po załadowaniu layoutu)* @Override  
 protected void onSizeChanged(int width, int height, int oldWidth, int oldHeight) {  
 super.onSizeChanged(width, height, oldWidth, oldHeight);  
  
 *// Sprawdzenie, czy widok już został zainicjalizowany, aby nie dodawać piłek ponownie* if (!isInitialized) {  
 initializePilki(width, height);  
 isInitialized = true; *// Ustawienie flagi, że piłki zostały utworzone* }  
 }  
  
 *// Metoda do inicjalizacji piłek* private void initializePilki(int width, int height) {  
 int liczbaPilek = 20;  
 int maxSpeed = 10;  
 int maxRadius = 60;  
 Random random = new Random();  
 for (int i = 0; i < liczbaPilek; i++) {  
 *// Losowanie współrzędnych X i Y w zakresie od -width do width* float randomValueWidth = random.nextInt( width + 1);  
 float randomValueHeight = random.nextInt(height + 1);  
 *// Losowanie prędkości, kolorów, promienia itp.* float randomValueSpeedX = random.nextInt(2 \* maxSpeed + 1) - maxSpeed;  
 float randomValueSpeedY = random.nextInt(2 \* maxSpeed + 1) - maxSpeed;  
 int randomValueColorR = random.nextInt(256); *// Zakres 0-255* int randomValueColorG = random.nextInt(256);  
 int randomValueColorB = random.nextInt(256);  
 float randomValueRadius = random.nextInt(maxRadius + 1);  
 *// Utworzenie piłek z losowymi wartościami* pilki.add(new Pilka(width/2, height/2, randomValueRadius,  
 randomValueSpeedX, randomValueSpeedY, Color.*rgb*(randomValueColorR, randomValueColorG, randomValueColorB)));  
 }  
 }  
 @Override  
 protected void onDraw(Canvas canvas) {  
 super.onDraw(canvas);  
  
 *// Aktualizacja pozycji każdej piłki* for (Pilka pilka : pilki) {  
 pilka.update(getWidth(), getHeight());  
 }  
  
 *// Rysowanie piłek* for (Pilka pilka : pilki) {  
 pilka.draw(canvas, paint);  
 }  
  
 *// Inwalidacja widoku, aby wywołać ponownie onDraw()* invalidate();  
 }  
}

1. Dodaj grawitację piłek

package com.example.myanimationviewapp;  
  
import android.graphics.Canvas;  
import android.graphics.Paint;  
  
public class Pilka {  
 private float x, y;  
 private float radius;  
 private float speedX, speedY;  
 private int color;  
 private static final float *grawitacja* = 0.05f; *// Jeszcze mniejsza wartość grawitacji* private boolean isFalling = true;  
  
 public Pilka(float x, float y, float radius, float speedX, float speedY, int color) {  
 this.x = x;  
 this.y = y;  
 this.radius = radius;  
 this.speedX = speedX;  
 this.speedY = speedY;  
 this.color = color;  
 }  
  
 public void update(int width, int height) {  
 if (isFalling) {  
 speedY += *grawitacja*; *// Dodaj grawitację do prędkości pionowej* }  
  
 x += speedX;  
 y += speedY;  
  
 *// Sprawdź, czy piłka dotknęła dołu ekranu* if (y + radius >= height+2\*radius) {  
 *//y = height - radius;* speedY = 0; *// Zatrzymaj piłkę po dotarciu do dołu* speedX = 0;  
 isFalling = false; *// Piłka już nie spada* }  
  
 *// Sprawdzenie kolizji z krawędziami ekranu* if (x + radius > width || x - radius <= 0) {  
 speedX = -speedX;  
 }  
 *//if (y + radius > height ) {  
 // speedY = -speedY;  
 //}* if ( y - radius <= 0) {  
 speedY = -speedY;  
 }  
  
 x += speedX;  
 y += speedY;  
 }  
  
  
 public void draw(Canvas canvas, Paint paint) {  
 paint.setColor(color);  
 canvas.drawCircle(x, y, radius, paint);  
 }  
}

1. Dodaj współczynnik oporu powietrza i wytracanie prędkości przy odbiciu piłki.

package com.example.myanimationviewapp;  
import android.graphics.Canvas;  
import android.graphics.Paint;  
  
public class Pilka {  
 private float x, y;  
 private float radius;  
 private float speedX, speedY;  
 private int color;  
 private static final float *grawitacja* = 0.05f; *// Wartość grawitacji* private static final float *opor* = 0.99f; *// Współczynnik oporu powietrza* private static final float *wytracanie* = 0.8f; *// Współczynnik wytracania prędkości przy odbiciu* public Pilka(float x, float y, float radius, float speedX, float speedY, int color) {  
 this.x = x;  
 this.y = y;  
 this.radius = radius;  
 this.speedX = speedX;  
 this.speedY = speedY;  
 this.color = color;  
 }  
  
 public void update(int width, int height) {  
 speedY += *grawitacja*; *// Dodaj grawitację do prędkości pionowej* x += speedX;  
 y += speedY;  
  
 *// Zmniejsz prędkości o opór powietrza* speedX \*= *opor*;  
 speedY \*= *opor*;  
  
  
  
 *// Odbij piłkę od dolnej i górnej krawędzi ekranu* if (y - radius <= 0 || y + radius >= height) {  
 speedY = -speedY\**wytracanie*;  
 }  
  
 *// Odbij piłkę od bocznych ścian* if (x - radius <= 0 || x + radius >= width) {  
 speedX = -speedX;  
 }  
 if (y>=height-radius) y=height-radius;  
 *//System.out.println("height "+height+" radius "+radius+" y "+y);* }  
  
 public void draw(Canvas canvas, Paint paint) {  
 paint.setColor(color);  
 canvas.drawCircle(x, y, radius, paint);  
 }  
}

**Przykład 2. SimleGameLoopApp**

Kolejny przykład przedstawia prosty szkielet pętli gry w Android Studio. Zaczniemy od stworzenia nowej aplikacji, a następnie dodamy pętlę gry z wyświetlaniem FPS i TPS (ticks per second).

1. Otwórz Android Studio i utwórz nowy projekt.
2. Wybierz “Empty Views Activity” jako szablon projektu.
3. Nazwij projekt np. SimpleGameLoopApp
4. Wprowadź zmiany w pliku MainActivity.java

package com.example.simplegameloopapp;  
import androidx.appcompat.app.AppCompatActivity;  
import android.os.Bundle;  
import android.os.Handler;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
 private final int FRAMES\_PER\_SECOND = 60;  
 private final int TICKS\_PER\_SECOND = 30;  
 private boolean running = true;  
 private TextView fpsTextView;  
 private TextView ticksTextView;  
 private Handler handler = new Handler();  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 fpsTextView = findViewById(R.id.*fpsTextView*);  
 ticksTextView = findViewById(R.id.*ticksTextView*);  
  
 startGameLoop();  
 }  
  
 private void startGameLoop() {  
 Runnable gameLoop = new Runnable() {  
 @Override  
 public void run() {  
 if (!running) return;  
  
 updateGameLogic();  
 renderGame();  
  
 handler.postDelayed(this, 1000 / FRAMES\_PER\_SECOND);  
 }  
 };  
  
 handler.post(gameLoop);  
 }  
  
 private void updateGameLogic() {  
 *// Aktualizacja logiki gry* ticksTextView.setText("TPS: " + TICKS\_PER\_SECOND);  
 }  
  
 private void renderGame() {  
 *// Renderowanie gry* fpsTextView.setText("FPS: " + FRAMES\_PER\_SECOND);  
 }  
}

1. Wprowadź zmiany w pliku activity\_main.xml

*<?*xml version="1.0" encoding="utf-8"*?>*<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"  
 tools:context=".MainActivity">  
  
 <TextView  
 android:id="@+id/fpsTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentTop="true"  
 android:text="FPS: 0"  
 android:textColor="#00FF00"  
 android:textSize="24sp" />  
  
 <TextView  
 android:id="@+id/ticksTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_below="@id/fpsTextView"  
 android:layout\_marginTop="5dp"  
 android:text="TPS: 0"  
 android:textColor="#00FF00"  
 android:textSize="24sp" />  
</RelativeLayout>

1. Zamiast view zastosuj canvas I surfaceview. Wprowadź zmiany w pliku MainActivity.java

*<?*xml version="1.0" encoding="utf-8"*?>*<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"  
 tools:context=".MainActivity">  
  
 <SurfaceView  
 android:id="@+id/surfaceView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"/>  
  
 <TextView  
 android:id="@+id/fpsTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentTop="true"  
 android:text="FPS: 0"  
 android:textColor="#00FF00"  
 android:textSize="24sp" />  
  
 <TextView  
 android:id="@+id/ticksTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_below="@id/fpsTextView"  
 android:layout\_marginTop="5dp"  
 android:text="TPS: 0"  
 android:textColor="#00FF00"  
 android:textSize="24sp" />  
</RelativeLayout>

1. Wprowadź zmiany w pliku MainActivity.java

package com.example.simplegameloopapp;  
  
import androidx.appcompat.app.AppCompatActivity;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.graphics.Point;  
import android.os.Bundle;  
import android.os.Handler;  
import android.view.Display;  
import android.view.SurfaceHolder;  
import android.view.SurfaceView;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
 private final int FRAMES\_PER\_SECOND = 60;  
 private final int TICKS\_PER\_SECOND = 30;  
 private boolean running = true;  
 private TextView fpsTextView;  
 private TextView ticksTextView;  
 private Handler handler = new Handler();  
 private Pilka pilka;  
 private Paint paint;  
 private int screenWidth;  
 private int screenHeight;  
 private SurfaceView surfaceView;  
 private SurfaceHolder surfaceHolder;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 fpsTextView = findViewById(R.id.*fpsTextView*);  
 ticksTextView = findViewById(R.id.*ticksTextView*);  
 surfaceView = findViewById(R.id.*surfaceView*);  
 surfaceHolder = surfaceView.getHolder();  
  
 *// Pobieranie rozmiarów ekranu* Display display = getWindowManager().getDefaultDisplay();  
 Point size = new Point();  
 display.getSize(size);  
 screenWidth = size.x;  
 screenHeight = size.y;  
  
 *// Inicjalizacja piłki* pilka = new Pilka(100, 100, 15, 17, Color.*RED*, 20);  
 paint = new Paint();  
  
 startGameLoop();  
 }  
  
 private void startGameLoop() {  
 Runnable gameLoop = new Runnable() {  
 @Override  
 public void run() {  
 if (!running) return;  
  
 updateGameLogic();  
 renderGame();  
  
 handler.postDelayed(this, 1000 / FRAMES\_PER\_SECOND);  
 }  
 };  
  
 handler.post(gameLoop);  
 }  
  
 private void updateGameLogic() {  
 *// Aktualizacja logiki gry* pilka.update(screenWidth, screenHeight);  
 ticksTextView.setText("TPS: " + TICKS\_PER\_SECOND);  
 }  
  
 private void renderGame() {  
 if (!surfaceHolder.getSurface().isValid()) {  
 return;  
 }  
 Canvas canvas = surfaceHolder.lockCanvas();  
 canvas.drawColor(Color.*WHITE*); *// Clear the canvas* pilka.draw(canvas, paint);  
 surfaceHolder.unlockCanvasAndPost(canvas);  
 fpsTextView.setText("FPS: " + FRAMES\_PER\_SECOND);  
 }  
}

1. Dodaj klasę Pilka

package com.example.simplegameloopapp;  
  
import android.graphics.Canvas;  
import android.graphics.Paint;  
  
public class Pilka {  
 private float x, y;  
 private float speedX, speedY;  
 private int color;  
 private float radius;  
  
 public Pilka(float x, float y, float speedX, float speedY, int color, float radius) {  
 this.x = x;  
 this.y = y;  
 this.speedX = speedX;  
 this.speedY = speedY;  
 this.color = color;  
 this.radius = radius;  
 }  
  
 public void update(float screenWidth, float screenHeight) {  
 x += speedX;  
 y += speedY;  
  
 *// Odbijanie od ścian* if (x - radius <= 0 || x + radius >= screenWidth) {  
 speedX = -speedX;  
 }  
 if (y - radius <= 0 || y + radius\*4 >= screenHeight) {  
 speedY = -speedY;  
 }  
 }  
  
 public void draw(Canvas canvas, Paint paint) {  
 paint.setColor(color);  
 canvas.drawCircle(x, y, radius, paint);  
 }  
  
 *// Getter dla x, y - dla przyszłego użycia* public float getX() { return x; }  
 public float getY() { return y; }  
}

Ten przykład to prosty silnik gry w Androidzie z klasą Pilka, która odbija się od ścian. Opis poszczególnych części kodu:

MainActivity - Główna aktywność aplikacji, która inicjalizuje piłkę, ustawia pętlę gry i aktualizuje ekran.

Layout - activity\_main.xml zawiera SurfaceView do renderowania oraz TextView do wyświetlania FPS i TPS.

Pilka - Klasa reprezentująca piłkę. Ma atrybuty takie jak x, y, speedX, speedY, color i radius. Metoda update aktualizuje pozycję piłki, a metoda draw rysuje piłkę na ekranie.

Pętla gry - W MainActivity pętla gry jest realizowana za pomocą Handler. Pętla regularnie wywołuje metody updateGameLogic i renderGame:

updateGameLogic() - Aktualizuje logikę gry, w tym pozycję piłki.

renderGame() - Renderuje grę, czyli rysuje piłkę na SurfaceView.

SurfaceView - Używany do renderowania grafiki. SurfaceHolder pozwala na rysowanie na SurfaceView poprzez blokowanie i odblokowywanie powierzchni rysowania.

Aktualizacja i rysowanie - Pozycja piłki jest aktualizowana w zależności od jej prędkości i sprawdzane są kolizje z krawędziami ekranu. Piłka odbija się od ścian poprzez odwracanie kierunku prędkości.

Wyświetlanie FPS i TPS - TextView w activity\_main.xml wyświetlają aktualne wartości FPS i ticków. Te wartości są aktualizowane w pętli gry.

Teraz uporządkujemy naszą aplikację. Podzielimy program na klasy, aby ułatwić przyszłe rozbudowywanie. Proponuję podzielić go na klasy: MainActivity, Pilka, oraz dodanie klasy GameSurfaceView. Klasa Pilka pozostaje bez zmian. Dodamy klasę GameLoop.

package com.example.simplegameloopapp;  
import android.graphics.Canvas;  
import android.view.SurfaceHolder;  
  
public class GameLoop extends Thread {  
 private final int FRAMES\_PER\_SECOND = 60;  
 private final GameSurfaceView gameSurfaceView;  
 private final SurfaceHolder surfaceHolder;  
 private boolean running;  
  
 public GameLoop(GameSurfaceView gameSurfaceView, SurfaceHolder surfaceHolder) {  
 this.gameSurfaceView = gameSurfaceView;  
 this.surfaceHolder = surfaceHolder;  
 }  
  
 public void setRunning(boolean running) {  
 this.running = running;  
 }  
  
 @Override  
 public void run() {  
 long startTime, timeMillis, waitTime;  
 long targetTime = 1000 / FRAMES\_PER\_SECOND;  
  
 while (running) {  
 startTime = System.*nanoTime*();  
 Canvas canvas = null;  
  
 try {  
 canvas = surfaceHolder.lockCanvas();  
 synchronized (surfaceHolder) {  
 gameSurfaceView.updateGameLogic();  
 gameSurfaceView.renderGame();  
 }  
 } finally {  
 if (canvas != null) {  
 surfaceHolder.unlockCanvasAndPost(canvas);  
 }  
 }  
  
 timeMillis = (System.*nanoTime*() - startTime) / 1000000;  
 waitTime = targetTime - timeMillis;  
  
 try {  
 if (waitTime > 0) {  
 *sleep*(waitTime);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}  
Dodaj klasę GameSurfaceView

package com.example.simplegameloopapp;  
  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.util.AttributeSet;  
import android.view.Choreographer;  
import android.view.SurfaceHolder;  
import android.view.SurfaceView;  
import android.graphics.Point;  
import android.view.Display;  
import android.view.WindowManager;  
  
public class GameSurfaceView extends SurfaceView implements SurfaceHolder.Callback, Choreographer.FrameCallback {  
  
 public interface Callbacks {  
 void onFrameRendered(int fps);  
 void onGameUpdated(int ticks);  
 }  
  
 private SurfaceHolder surfaceHolder;  
 private Paint paint;  
 private Pilka pilka;  
 private int screenWidth, screenHeight;  
 private final int TICKS\_PER\_SECOND = 30;  
 private int frameCount = 0;  
 private int tickCount = 0;  
 private long lastTime = System.*currentTimeMillis*();  
 private long tickTime;  
 private Callbacks callbacks;  
  
 public GameSurfaceView(Context context, AttributeSet attrs) {  
 super(context, attrs);  
 init(context);  
 }  
  
 private void init(Context context) {  
 surfaceHolder = getHolder();  
 surfaceHolder.addCallback(this);  
 paint = new Paint();  
  
 WindowManager wm = (WindowManager) context.getSystemService(Context.*WINDOW\_SERVICE*);  
 Display display = wm.getDefaultDisplay();  
 Point size = new Point();  
 display.getSize(size);  
 screenWidth = size.x;  
 screenHeight = size.y;  
  
 pilka = new Pilka(100, 100, 15, 17, Color.*RED*, 20);  
 }  
  
 public void setCallbacks(Callbacks callbacks) {  
 this.callbacks = callbacks;  
 }  
  
 @Override  
 public void doFrame(long frameTimeNanos) {  
 updateGameLogic();  
 renderGame();  
  
 frameCount++;  
 tickCount++;  
  
 long currentTime = System.*currentTimeMillis*();  
 if (currentTime - lastTime >= 1000) {  
 if (callbacks != null) {  
 callbacks.onFrameRendered(frameCount);  
 callbacks.onGameUpdated(tickCount);  
 }  
 frameCount = 0;  
 tickCount = 0;  
 lastTime = currentTime;  
 }  
  
 Choreographer.*getInstance*().postFrameCallback(this);  
 }  
  
 public void startGameLoop() {  
 tickTime = System.*currentTimeMillis*();  
 Choreographer.*getInstance*().postFrameCallback(this);  
 }  
  
 public void updateGameLogic() {  
 long currentTime = System.*currentTimeMillis*();  
 if (currentTime - tickTime >= 1000 / TICKS\_PER\_SECOND) {  
 pilka.update(screenWidth, screenHeight);  
 tickTime = currentTime;  
 }  
 }  
  
 public void renderGame() {  
 if (!surfaceHolder.getSurface().isValid()) {  
 return;  
 }  
 Canvas canvas = surfaceHolder.lockCanvas();  
 canvas.drawColor(Color.*WHITE*);  
 pilka.draw(canvas, paint);  
 surfaceHolder.unlockCanvasAndPost(canvas);  
 }  
  
 @Override  
 public void surfaceCreated(SurfaceHolder holder) {  
 startGameLoop();  
 }  
  
 @Override  
 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {  
 *// Ignored* }  
  
 @Override  
 public void surfaceDestroyed(SurfaceHolder holder) {  
 Choreographer.*getInstance*().removeFrameCallback(this);  
 }  
}

Zmodyfikuj klasę MainActivity

package com.example.simplegameloopapp;  
  
import androidx.appcompat.app.AppCompatActivity;  
import android.os.Bundle;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
 private TextView fpsTextView;  
 private TextView ticksTextView;  
 private GameSurfaceView gameSurfaceView;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 fpsTextView = findViewById(R.id.*fpsTextView*);  
 ticksTextView = findViewById(R.id.*ticksTextView*);  
 gameSurfaceView = findViewById(R.id.*gameSurfaceView*);  
  
 gameSurfaceView.setCallbacks(new GameSurfaceView.Callbacks() {  
 @Override  
 public void onFrameRendered(int fps) {  
 fpsTextView.setText("FPS: " + fps);  
 }  
  
 @Override  
 public void onGameUpdated(int ticks) {  
 ticksTextView.setText("TPS: " + ticks);  
 }  
 });  
 }  
}

Na koniec zmodyfikuj, (jeżeli trzeba) plik activity\_main.xml

*<?*xml version="1.0" encoding="utf-8"*?>*<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"  
 tools:context=".MainActivity">  
  
 <com.example.simplegameloopapp.GameSurfaceView  
 android:id="@+id/gameSurfaceView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"/>  
  
 <TextView  
 android:id="@+id/fpsTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentTop="true"  
 android:text="FPS: 0"  
 android:textColor="#00FF00"  
 android:textSize="24sp" />  
  
 <TextView  
 android:id="@+id/ticksTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_below="@id/fpsTextView"  
 android:layout\_marginTop="0dp"  
 android:text="TPS: 0"  
 android:textColor="#00FF00"  
 android:textSize="24sp" />  
</RelativeLayout>

Struktura aplikacji

MainActivity - Główna aktywność, która uruchamia widok gry i wyświetla FPS i TPS.

GameSurfaceView - Rozszerza SurfaceView i zarządza renderowaniem oraz interakcjami z powierzchnią rysowania.

Pilka - Klasa reprezentująca piłkę, która zawiera jej właściwości i logikę poruszania się.

GameLoop - Klasa zarządzająca pętlą gry, która kontroluje aktualizację logiki (TPS) i renderowanie (FPS).

Klasa MainActivity

Inicjalizuje GameSurfaceView i ustawia TextView do wyświetlania FPS i TPS.

Ustawia callbacki, aby GameSurfaceView mogło aktualizować te wartości.

Klasa GameSurfaceView

Rozszerza SurfaceView i implementuje SurfaceHolder.Callback.

Inicjalizuje SurfaceHolder, obiekt Pilka i rysowanie (Paint).

Przechowuje i zarządza obiektem GameLoop.

Obsługuje metody surfaceCreated(), surfaceChanged() i surfaceDestroyed().

Klasa Pilka

Reprezentuje piłkę z parametrami takimi jak pozycja (x, y), prędkość (speedX, speedY), kolor (color) i promień (radius).

Implementuje logikę poruszania się i kolizji.

Zawiera metodę draw() do rysowania piłki na płótnie (Canvas).

Klasa GameLoop

Zarządza pętlą gry, aktualizując logikę i renderując obraz w określonych odstępach czasu.

Używa Handler do realizacji pętli gry.

Wywołuje metody updateGameLogic() i renderGame() w GameSurfaceView.

Interakcje między klasami

MainActivity ustawia GameSurfaceView jako główny widok i inicjalizuje go.

GameSurfaceView tworzy i zarządza instancją Pilka.

GameLoop jest odpowiedzialny za wywoływanie updateGameLogic() i renderGame() w odpowiednich odstępach czasu.

Pilka aktualizuje swoje położenie i kolizje oraz rysuje siebie na płótnie.

GameSurfaceView odświeża widok za pomocą metod surfaceCreated(), surfaceChanged() i surfaceDestroyed().

Biorąc pod uwagę powyższy podział aplikacji na klasy, ostateczna wersja szkieletu aplikacji może wyglądać następująco:

Plik activity\_main.xml

*<?*xml version="1.0" encoding="utf-8"*?>*<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"  
 tools:context=".MainActivity">  
  
 <com.example.simplegameloopapp.GameSurfaceView  
 android:id="@+id/gameSurfaceView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"/>  
  
 <TextView  
 android:id="@+id/fpsTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentTop="true"  
 android:text="FPS: 0"  
 android:textColor="#00FF00"  
 android:textSize="24sp" />  
  
 <TextView  
 android:id="@+id/ticksTextView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_below="@id/fpsTextView"  
 android:layout\_marginTop="0dp"  
 android:text="TPS: 0"  
 android:textColor="#00FF00"  
 android:textSize="24sp" />  
</RelativeLayout>

MainActivity.java

package com.example.simplegameloopapp;  
  
import androidx.appcompat.app.AppCompatActivity;  
import android.os.Bundle;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
 private TextView fpsTextView;  
 private TextView ticksTextView;  
 private GameSurfaceView gameSurfaceView;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 fpsTextView = findViewById(R.id.*fpsTextView*);  
 ticksTextView = findViewById(R.id.*ticksTextView*);  
 gameSurfaceView = findViewById(R.id.*gameSurfaceView*);  
  
 gameSurfaceView.setCallbacks(new GameSurfaceView.Callbacks() {  
 @Override  
 public void onFrameRendered(int fps) {  
 fpsTextView.setText("Frames Per Second: " + fps);  
 }  
  
 @Override  
 public void onGameUpdated(int ticks) {  
 ticksTextView.setText("Ticks: " + ticks);  
 }  
 });  
 }  
}

Pilka.java

package com.example.simplegameloopapp;  
  
import android.graphics.Canvas;  
import android.graphics.Paint;  
  
public class Pilka {  
 private float x, y;  
 private float speedX, speedY;  
 private int color;  
 private float radius;  
  
 public Pilka(float x, float y, float speedX, float speedY, int color, float radius) {  
 this.x = x;  
 this.y = y;  
 this.speedX = speedX;  
 this.speedY = speedY;  
 this.color = color;  
 this.radius = radius;  
 }  
  
 public void update(float screenWidth, float screenHeight) {  
 x += speedX;  
 y += speedY;  
  
 if (x - radius < 0 || x + radius > screenWidth) {  
 speedX = -speedX;  
 }  
 if (y - radius < 0 || y + radius > screenHeight) {  
 speedY = -speedY;  
 }  
 }  
  
 public void draw(Canvas canvas, Paint paint) {  
 paint.setColor(color);  
 canvas.drawCircle(x, y, radius, paint);  
 }  
}

GameLoop.java

package com.example.simplegameloopapp;  
  
import android.os.Handler;  
import android.os.Looper;  
  
public class GameLoop {  
 private final GameSurfaceView gameSurfaceView;  
 private final Handler handler = new Handler(Looper.*getMainLooper*());  
 private boolean running = false;  
 private int frameCount = 0;  
 private int tickCount = 0;  
 private long lastTime = System.*currentTimeMillis*();  
 private final int FRAMES\_PER\_SECOND = 60;  
 private final int TICKS\_PER\_SECOND = 30;  
 private long tickTime;  
  
 public GameLoop(GameSurfaceView gameSurfaceView) {  
 this.gameSurfaceView = gameSurfaceView;  
 }  
  
 public void start() {  
 running = true;  
 tickTime = System.*currentTimeMillis*();  
 handler.post(gameLoop);  
 }  
  
 public void stop() {  
 running = false;  
 handler.removeCallbacks(gameLoop);  
 }  
  
 private final Runnable gameLoop = new Runnable() {  
 @Override  
 public void run() {  
 if (!running) return;  
  
 long currentTime = System.*currentTimeMillis*();  
 gameSurfaceView.updateGameLogic();  
 frameCount++;  
  
 if (currentTime - tickTime >= 1000 / TICKS\_PER\_SECOND) {  
 tickCount++;  
 tickTime = currentTime;  
 }  
  
 if (currentTime - lastTime >= 1000) {  
 if (gameSurfaceView.callbacks != null) {  
 gameSurfaceView.callbacks.onFrameRendered(frameCount);  
 gameSurfaceView.callbacks.onGameUpdated(tickCount);  
 }  
 frameCount = 0;  
 tickCount = 0;  
 lastTime = currentTime;  
 }  
  
 handler.postDelayed(gameLoop, 1000 / FRAMES\_PER\_SECOND);  
 }  
 };  
}

GameSurface.java

package com.example.simplegameloopapp;  
  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.util.AttributeSet;  
import android.view.Choreographer;  
import android.view.SurfaceHolder;  
import android.view.SurfaceView;  
import android.graphics.Point;  
import android.view.Display;  
import android.view.WindowManager;  
  
public class GameSurfaceView extends SurfaceView implements SurfaceHolder.Callback, Choreographer.FrameCallback {  
  
 public interface Callbacks {  
 void onFrameRendered(int fps);  
 void onGameUpdated(int ticks);  
 }  
  
 private SurfaceHolder surfaceHolder;  
 private Paint paint;  
 private Pilka pilka;  
 private int screenWidth, screenHeight;  
 private final int TICKS\_PER\_SECOND = 30;  
 private int frameCount = 0;  
 private int tickCount = 0;  
 private long lastTime = System.*currentTimeMillis*();  
 private long tickTime;  
 public Callbacks callbacks;  
  
 public GameSurfaceView(Context context, AttributeSet attrs) {  
 super(context, attrs);  
 init(context);  
 }  
  
 private void init(Context context) {  
 surfaceHolder = getHolder();  
 surfaceHolder.addCallback(this);  
 paint = new Paint();  
  
 WindowManager wm = (WindowManager) context.getSystemService(Context.*WINDOW\_SERVICE*);  
 Display display = wm.getDefaultDisplay();  
 Point size = new Point();  
 display.getSize(size);  
 screenWidth = size.x;  
 screenHeight = size.y;  
  
 pilka = new Pilka(100, 100, 15, 17, Color.*RED*, 20);  
 }  
  
 public void setCallbacks(Callbacks callbacks) {  
 this.callbacks = callbacks;  
 }  
  
 public Callbacks getCallbacks() {  
 return callbacks;  
 }  
  
 @Override  
 public void doFrame(long frameTimeNanos) {  
 updateGameLogic();  
 renderGame();  
  
 frameCount++;  
 tickCount++;  
  
 long currentTime = System.*currentTimeMillis*();  
 if (currentTime - lastTime >= 1000) {  
 if (getCallbacks() != null) {  
 getCallbacks().onFrameRendered(frameCount);  
 getCallbacks().onGameUpdated(tickCount);  
 }  
 frameCount = 0;  
 tickCount = 0;  
 lastTime = currentTime;  
 }  
  
 Choreographer.*getInstance*().postFrameCallback(this);  
 }  
  
 public void startGameLoop() {  
 tickTime = System.*currentTimeMillis*();  
 Choreographer.*getInstance*().postFrameCallback(this);  
 }  
  
 public void updateGameLogic() {  
 long currentTime = System.*currentTimeMillis*();  
 if (currentTime - tickTime >= 1000 / TICKS\_PER\_SECOND) {  
 pilka.update(screenWidth, screenHeight);  
 tickTime = currentTime;  
 }  
 }  
  
 public void renderGame() {  
 if (!surfaceHolder.getSurface().isValid()) {  
 return;  
 }  
 Canvas canvas = surfaceHolder.lockCanvas();  
 canvas.drawColor(Color.*WHITE*); *// Clear the canvas* pilka.draw(canvas, paint);  
 surfaceHolder.unlockCanvasAndPost(canvas);  
 }  
  
 @Override  
 public void surfaceCreated(SurfaceHolder holder) {  
 startGameLoop();  
 }  
  
 @Override  
 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {  
 *// Ignored* }  
  
 @Override  
 public void surfaceDestroyed(SurfaceHolder holder) {  
 Choreographer.*getInstance*().removeFrameCallback(this);  
 }  
}

**Sprawozdanie**

Sprawozdanie z ćwiczenia w ramach nauki programowania aplikacji mobilnych powinno zawierać istotne informacje i dokumentację dotyczącą zadania oraz jego realizacji. Oto kilka elementów, które powinny być uwzględnione w sprawozdaniu:

**1. Tytuł i informacje ogólne**

* Tytuł ćwiczenia.
* Imię i nazwisko studenta.
* Data realizacji ćwiczenia.

**2. Cel ćwiczenia**

* Krótka informacja o celu i znaczeniu ćwiczenia, jak również o tym, czego studenci mieli się nauczyć.

**3. Opis projektu**

* Opis funkcjonalności gry w ramach projektu.
* Charakteryzacja interfejsu użytkownika, zawierająca informacje o wykorzystanych widokach i elementach interfejsu.

**4. Implementacja**

* Opis procesu tworzenia projektu, wraz z krokami realizacji gry.
* Omówienie wykorzystanych narzędzi i technologii, w tym Android Studio, język Java/Kotlin.
* Przedstawienie kodu źródłowego aplikacji, zarówno XML (layout) jak i kodu Java/Kotlin.

**5. Funkcje kluczowe**

* Omówienie kluczowych funkcji aplikacji, takich jak obsługa puzzli, zarządzanie stanem gry oraz sposób prezentacji wyników.

**6. Testowanie**

* Opis testowania gry, włączając w to przykłady testów przeprowadzonych w trakcie implementacji.
* Przykładowe przypadki testowe i raport z wynikami testów.

**7. Wyniki**

* Przedstawienie wyników działania aplikacji, w tym zrzuty ekranu demonstrujące działanie gry.

**8. Podsumowanie**

* Krótka ocena projektu i osiągnięć w kontekście zrealizowanego zadania.
* Wnioski wynikające z ćwiczenia, jakie umiejętności i doświadczenie zdobyli studenci.

**9. Trudności i błędy**

* Informacje na temat ewentualnych problemów napotkanych podczas implementacji gry i jak zostały one rozwiązane.

**10. Źródła i odniesienia**

* Jeśli korzystano z materiałów lub źródeł zewnętrznych, uwzględnij je w tekście.

**11. Dodatkowe materiały**

* Ewentualne dodatkowe materiały, takie jak kody źródłowe, zrzuty ekranu lub inne dokumentacje, które uzupełniają sprawozdanie.

**Zalecenia ogólne**

* Sprawozdanie powinno być czytelne i przejrzyste, z odpowiednimi nagłówkami i numeracją stron.
* Projektowanie i implementacja powinny być opisane w sposób logiczny i zrozumiały.
* Sprawozdanie powinno być dostatecznie szczegółowe, aby inny programista mógł zrozumieć projekt i ewentualnie go udoskonalić lub wykorzystać w przyszłości.