


Politechnika Bydgoska im. Jana i Jędrzeja Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki al. prof. S. Kaliskiego 7, 85-796 Bydgoszcz				 POLITECHNIKA BYDGOSKA Wydział Telekomunikacji, Informatyki i Elektrotechniki	
Przedmiot	Programowanie urządzeń mobilnych			Kierunek/Tryb	IS/ST
Nr laboratorium	8	Data wykonania	18.05.2025	Grupa	1
Ocena		Data oddania	18.05.2025	Imię Nazwisko	Cezary Tytko
Nazwa ćwiczenia	Animacja, SurfaceView, fizyka obiektów				

2. Cel ćwiczenia

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.

3. Opis projektu

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.

4. Implementacja

Kod został napisany w języku Kotlin w środowisku Android Studio.

MainActivity.kt:

```
package com.example.pumlalab8_2

import android.content.Context
import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint
import android.os.Bundle
import android.os.Handler
import android.util.AttributeSet
import android.view.MotionEvent
import android.view.SurfaceHolder
import android.view.SurfaceView
import android.widget.Button
import android.widget.FrameLayout
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.setPadding
import kotlin.math.hypot
import kotlin.random.Random

class MainActivity : AppCompatActivity() {
    private lateinit var gameView: GameSurfaceView
    private lateinit var restartButton: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

```

        val container = findViewById<FrameLayout>(R.id.gameContainer)
        val gameView = GameSurfaceView(this)
        container.addView(gameView)

        restartButton = findViewById(R.id.restartButton)

        restartButton.setOnClickListener {
            gameView.resetGame()
        }
    }
}

class GameSurfaceView(context: Context, attrs: AttributeSet? = null) :
    SurfaceView(context, attrs), SurfaceHolder.Callback {
    private val fps = 60
    private val handler = Handler()
    private val pilki = mutableListOf<Pilka>()
    private val paint = Paint()
    private var isInitialized = false
    private var screenWidth = 0
    private var screenHeight = 0

    private var frameCount = 0
    private var lastTime = System.currentTimeMillis()
    private var currentFps = 0

    init {
        holder.addCallback(this)
        isFocusable = true
    }

    fun resetGame() {
        pilki.clear()
        initializePilki()
    }

    private val gameLoop = object : Runnable {
        override fun run() {
            updateGameLogic()
            renderGame()
            updateFps()
            handler.postDelayed(this, (1000L / fps))
        }
    }

    override fun surfaceCreated(holder: SurfaceHolder) {
        screenWidth = width
        screenHeight = height
        if (!isInitialized) {
            initializePilki()
            isInitialized = true
        }
        handler.post(gameLoop)
    }

    override fun surfaceChanged(holder: SurfaceHolder, format: Int, width: Int,
        height: Int) {}

    override fun surfaceDestroyed(holder: SurfaceHolder) {
        handler.removeCallbacks(gameLoop)
    }

    private fun initializePilki() {
        pilki.clear()
        val liczbaPilek = 20
        val maxSpeed = 10
        val maxRadius = 60
        val random = Random

        repeat(liczbaPilek) {

```

```

        val randomValueSpeedX = random.nextInt(-maxSpeed, maxSpeed + 1).toFloat()
        val randomValueSpeedY = random.nextInt(-maxSpeed, maxSpeed + 1).toFloat()
        val color = Color.rgb(random.nextInt(256), random.nextInt(256), random.nextInt(256))
        val radius = random.nextInt(10, maxRadius + 1).toFloat()
        pilki.add(Pilka(screenWidth / 2f, screenHeight / 2f, radius, randomValueSpeedX, randomValueSpeedY, color))
    }

    private fun updateGameLogic() {
        // aktualizacja pozycji piłek
        pilki.forEach { it.update(screenWidth, screenHeight) }

        // detekcja kolizji między piłkami
        for (i in 0 until pilki.size) {
            for (j in i + 1 until pilki.size) {
                pilki[i].resolveCollision(pilki[j])
            }
        }
    }

    private fun renderGame() {
        val canvas = holder.lockCanvas()
        if (canvas != null) {
            canvas.drawColor(Color.WHITE)
            pilki.forEach { it.draw(canvas, paint) }

            paint.color = Color.BLACK
            paint.textSize = 50f
            canvas.drawText("FPS: $currentFps", 50f, 80f, paint)

            holder.unlockCanvasAndPost(canvas)
        }
    }

    private fun updateFps() {
        frameCount++
        val currentTime = System.currentTimeMillis()
        if (currentTime - lastTime >= 1000) {
            currentFps = frameCount
            frameCount = 0
            lastTime = currentTime
        }
    }

    override fun onTouchEvent(event: MotionEvent): Boolean {
        if (event.action == MotionEvent.ACTION_DOWN) {
            val touchX = event.x
            val touchY = event.y

            // Sprawdzenie, czy kliknięto piłkę
            pilki.forEach {
                val dx = it.x - touchX
                val dy = it.y - touchY
                if (hypot(dx, dy) < it.radius) {
                    it.speedY = -20f // podbicie w górę
                }
            }
        }
        return true
    }
}

class Pilka(
    var x: Float,
    var y: Float,
    val radius: Float,

```

```

var speedX: Float,
var speedY: Float,
val color: Int
) {
    companion object {
        private const val grawitacja = 0.05f
        private const val opor = 0.99f
        private const val wytracanie = 0.8f
    }

    fun update(width: Int, height: Int) {
        speedY += grawitacja

        x += speedX
        y += speedY

        speedX *= opor
        speedY *= opor

        if (y >= height - radius) y = height - radius
        if (y - radius <= 0) y = radius
        if (y - radius <= 0 || y + radius >= height) {
            speedY = -speedY * wytracanie
        }
        if (x - radius <= 0 || x + radius >= width) {
            speedX = -speedX
        }
    }

    fun draw(canvas: Canvas, paint: Paint) {
        paint.color = color
        canvas.drawCircle(x, y, radius, paint)
    }

    fun resolveCollision(other: Pilka) {
        val dx = other.x - x
        val dy = other.y - y
        val distance = hypot(dx, dy)

        if (distance == 0f) return // unikamy dzielenia przez zero

        if (distance < radius + other.radius) {
            // Masa jako pole koła
            val mass1 = radius * radius
            val mass2 = other.radius * other.radius

            // Normal i tangent wektor
            val nx = dx / distance
            val ny = dy / distance
            val tx = -ny
            val ty = nx

            // Projekcje prędkości na wektory normalny i styczny
            val v1n = speedX * nx + speedY * ny
            val v1t = speedX * tx + speedY * ty
            val v2n = other.speedX * nx + other.speedY * ny
            val v2t = other.speedX * tx + other.speedY * ty

            // Nowe prędkości normalne po kolizji (prawo zachowania pędu i
            energii)
            val v1nAfter = (v1n * (mass1 - mass2) + 2 * mass2 * v2n) / (mass1 +
            mass2)
            val v2nAfter = (v2n * (mass2 - mass1) + 2 * mass1 * v1n) / (mass1 +
            mass2)

            // Zamiana na prędkości wektorowe
            speedX = v1nAfter * nx + v1t * tx
            speedY = v1nAfter * ny + v1t * ty

```

```

other.speedX = v2nAfter * nx + v2t * tx
other.speedY = v2nAfter * ny + v2t * ty

// korekta pozycji aby piłki się nie nakładały
val overlap = 0.5f * (radius + other.radius - distance)
x -= overlap * nx
y -= overlap * ny
other.x += overlap * nx
other.y += overlap * ny
}
}
}

```

activity_main.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/rootLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:id="@+id/gameContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <Button
        android:id="@+id/restartButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Restart"
        android:layout_gravity="bottom|center_horizontal"
        android:layout_marginBottom="20dp" />

</FrameLayout>

```

5. Funkcje kluczowe

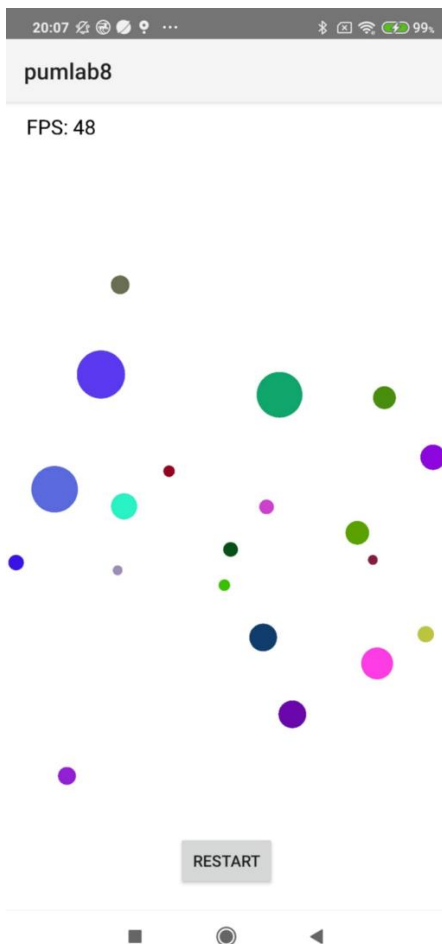
Demonstracja działania fizyki w grze – uwzględniona jest grawitacja, opór powietrza oraz prawo zachowania pędu przy kolizjach. Użytkownik może kliknąć na piłkę, aby ją podbić, a także zresetować grę przyciskiem. Aplikacja oblicza i wyświetla liczbę klatek na sekundę (FPS), a całość odbywa się w pętli animacyjnej na widoku SurfaceView.

6. Testowanie

Testowanie gry obejmowało:

- Sprawdzenie, czy piłka odbija się w górę po uderzeniu o dolną krawędź.
- Sprawdzenie, czy wysokość odbicia maleje.
- Sprawdzenie, czy piłka odbija się w dół, gdy uderzy w górną krawędź.
- Sprawdzenie, czy piłka odbija się w przeciwną stronę po uderzeniu w lewą lub prawą krawędź.
- Sprawdzenie, czy prędkość piłek stopniowo maleje.

1. Wyniki



8. Podsumowanie

Projekt realizuje symulację fizyki ruchu piłek w aplikacji mobilnej. Implementacja pozwoliła na zdobycie praktyki w programowaniu grafiki na Androida, obsłudze dotyku oraz realizacji animacji w czasie rzeczywistym. Ćwiczenie umożliwiło lepsze zrozumienie działania pętli gry, kolizji obiektów oraz zasad fizyki stosowanych w grach.

9. Trudności i błędy

- Nie wystąpiły żadne trudności ani błędy.

10. Źródła i odniesienia

- Nie korzystano ze źródeł i odniesień innych niż ta instrukcja.

11. Dodatkowe materiały

- Nie korzystano z dodatkowych materiałów.