|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | IS/ST |
| Nr laboratorium | 8 | Data wykonania | 08.05.2025 | Grupa | 1 |
| Ocena |  | Data oddania | 08.05.2025 | 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.

1. MainActivity.kt:

2. package com.example.pumlab8\_2

3. import android.content.Context

4. import android.graphics.Canvas

5. import android.graphics.Color

6. import android.graphics.Paint

7. import android.os.Bundle

8. import android.os.Handler

9. import android.util.AttributeSet

10. import android.view.MotionEvent

11. import android.view.SurfaceHolder

12. import android.view.SurfaceView

13. import android.widget.Button

14. import android.widget.FrameLayout

15. import androidx.appcompat.app.AppCompatActivity

16. import androidx.core.view.setPadding

17. import kotlin.math.hypot

18. import kotlin.random.Random

19. class MainActivity : AppCompatActivity() {

20. private lateinit var gameView: GameSurfaceView

21. private lateinit var restartButton: Button

22. override fun onCreate(savedInstanceState: Bundle?) {

23. super.onCreate(savedInstanceState)

24. setContentView(R.layout.activity\_main)

25. val container = findViewById<FrameLayout>(R.id.gameContainer)

26. val gameView = GameSurfaceView(this)

27. container.addView(gameView)

28. restartButton = findViewById(R.id.restartButton)

29. restartButton.setOnClickListener {

30. gameView.resetGame()

31. }

32. }

33. }

34. class GameSurfaceView(context: Context, attrs: AttributeSet? = null) :

35. SurfaceView(context, attrs), SurfaceHolder.Callback {

36. private val fps = 60

37. private val handler = Handler()

38. private val pilki = mutableListOf<Pilka>()

39. private val paint = Paint()

40. private var isInitialized = false

41. private var screenWidth = 0

42. private var screenHeight = 0

43. private var frameCount = 0

44. private var lastTime = System.currentTimeMillis()

45. private var currentFps = 0

46. init {

47. holder.addCallback(this)

48. isFocusable = true

49. }

50. fun resetGame() {

51. pilki.clear()

52. initializePilki()

53. }

54. private val gameLoop = object : Runnable {

55. override fun run() {

56. updateGameLogic()

57. renderGame()

58. updateFps()

59. handler.postDelayed(this, (1000L / fps))

60. }

61. }

62. override fun surfaceCreated(holder: SurfaceHolder) {

63. screenWidth = width

64. screenHeight = height

65. if (!isInitialized) {

66. initializePilki()

67. isInitialized = true

68. }

69. handler.post(gameLoop)

70. }

71. override fun surfaceChanged(holder: SurfaceHolder, format: Int, width: Int,

72. height: Int) {}

73. override fun surfaceDestroyed(holder: SurfaceHolder) {

74. handler.removeCallbacks(gameLoop)

75. }

76. private fun initializePilki() {

77. pilki.clear()

78. val liczbaPilek = 20

79. val maxSpeed = 10

80. val maxRadius = 60

81. val random = Random

82. repeat(liczbaPilek) {

83. val randomValueSpeedX = random.nextInt(-maxSpeed, maxSpeed +

84. 1).toFloat()val randomValueSpeedY = random.nextInt(-maxSpeed, maxSpeed +

85. 1).toFloat()

86. val color = Color.rgb(random.nextInt(256), random.nextInt(256),

87. random.nextInt(256))

88. val radius = random.nextInt(10, maxRadius + 1).toFloat()

89. pilki.add(Pilka(screenWidth / 2f, screenHeight / 2f, radius,

90. randomValueSpeedX, randomValueSpeedY, color))

91. }

92. }

93. private fun updateGameLogic() {

94. // aktualizacja pozycji piłek

95. pilki.forEach { it.update(screenWidth, screenHeight) }

96. // detekcja kolizji między piłkami

97. for (i in 0 until pilki.size) {

98. for (j in i + 1 until pilki.size) {

99. pilki[i].resolveCollision(pilki[j])

100. }

101. }

102. }

103. private fun renderGame() {

104. val canvas = holder.lockCanvas()

105. if (canvas != null) {

106. canvas.drawColor(Color.WHITE)

107. pilki.forEach { it.draw(canvas, paint) }

108. paint.color = Color.BLACK

109. paint.textSize = 50f

110. canvas.drawText("FPS: $currentFps", 50f, 80f, paint)

111. holder.unlockCanvasAndPost(canvas)

112. }

113. }

114. private fun updateFps() {

115. frameCount++

116. val currentTime = System.currentTimeMillis()

117. if (currentTime - lastTime >= 1000) {

118. currentFps = frameCount

119. frameCount = 0

120. lastTime = currentTime

121. }

122. }

123. override fun onTouchEvent(event: MotionEvent): Boolean {

124. if (event.action == MotionEvent.ACTION\_DOWN) {

125. val touchX = event.x

126. val touchY = event.y

127. // Sprawdzenie, czy kliknięto piłkę

128. pilki.forEach {

129. val dx = it.x - touchX

130. val dy = it.y - touchY

131. if (hypot(dx, dy) < it.radius) {

132. it.speedY = -20f // podbicie w górę

133. }

134. }

135. }

136. return true

137. }

138. }

139. class Pilka(

140. var x: Float,

141. var y: Float,

142. val radius: Float,

143. var speedX: Float,

144. var speedY: Float,

145. val color: Int

146. ) {

147. companion object {

148. private const val grawitacja = 0.05f

149. private const val opor = 0.99f

150. private const val wytracanie = 0.8f

151. }

152. fun update(width: Int, height: Int) {

153. speedY += grawitacja

154. x += speedX

155. y += speedY

156. speedX \*= opor

157. speedY \*= opor

158. if (y >= height - radius) y = height - radius

159. if (y - radius <= 0) y = radius

160. if (y - radius <= 0 || y + radius >= height) {

161. speedY = -speedY \* wytracanie

162. }

163. if (x - radius <= 0 || x + radius >= width) {

164. speedX = -speedX

165. }

166. }

167. fun draw(canvas: Canvas, paint: Paint) {

168. paint.color = color

169. canvas.drawCircle(x, y, radius, paint)

170. }

171. fun resolveCollision(other: Pilka) {

172. val dx = other.x - x

173. val dy = other.y - y

174. val distance = hypot(dx, dy)

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

176. if (distance < radius + other.radius) {

177. // Masa jako pole koła

178. val mass1 = radius \* radius

179. val mass2 = other.radius \* other.radius

180. // Normal i tangent wektor

181. val nx = dx / distance

182. val ny = dy / distance

183. val tx = -ny

184. val ty = nx

185. // Projekcje prędkości na wektory normalny i styczny

186. val v1n = speedX \* nx + speedY \* ny

187. val v1t = speedX \* tx + speedY \* ty

188. val v2n = other.speedX \* nx + other.speedY \* ny

189. val v2t = other.speedX \* tx + other.speedY \* ty

190. // Nowe prędkości normalne po kolizji (prawo zachowania pędu i

191. energii) val v1nAfter = (v1n \* (mass1 - mass2) + 2 \* mass2 \* v2n) / (mass1 +

192. mass2) val v2nAfter = (v2n \* (mass2 - mass1) + 2 \* mass1 \* v1n) / (mass1 +

193. mass2)

194. // Zamiana na prędkości wektorowe

195. speedX = v1nAfter \* nx + v1t \* tx

196. speedY = v1nAfter \* ny + v1t \* ty

197. other.speedX = v2nAfter \* nx + v2t \* tx

198. other.speedY = v2nAfter \* ny + v2t \* ty

199. // Korekta pozycji aby piłki się nie nakładały

200. val overlap = 0.5f \* (radius + other.radius - distance)

201. x -= overlap \* nx

202. y -= overlap \* ny

203. other.x += overlap \* nx

204. other.y += overlap \* ny

205. }

206. }

207. }

208.

activity\_main.xml:

1. <?xml version="1.0" encoding="utf-8"?>

2. <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"

3. android:id="@+id/rootLayout"

4. android:layout\_width="match\_parent"

5. android:layout\_height="match\_parent">

6. <FrameLayout

7. android:id="@+id/gameContainer"

8. android:layout\_width="match\_parent"

9. android:layout\_height="match\_parent" />

10. <Button

11. android:id="@+id/restartButton"

12. android:layout\_width="wrap\_content"

13. android:layout\_height="wrap\_content"

14. android:text="Restart"

15. android:layout\_gravity="bottom|center\_horizontal"

16. android:layout\_marginBottom="20dp" />

17. </FrameLayout>

18.

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.

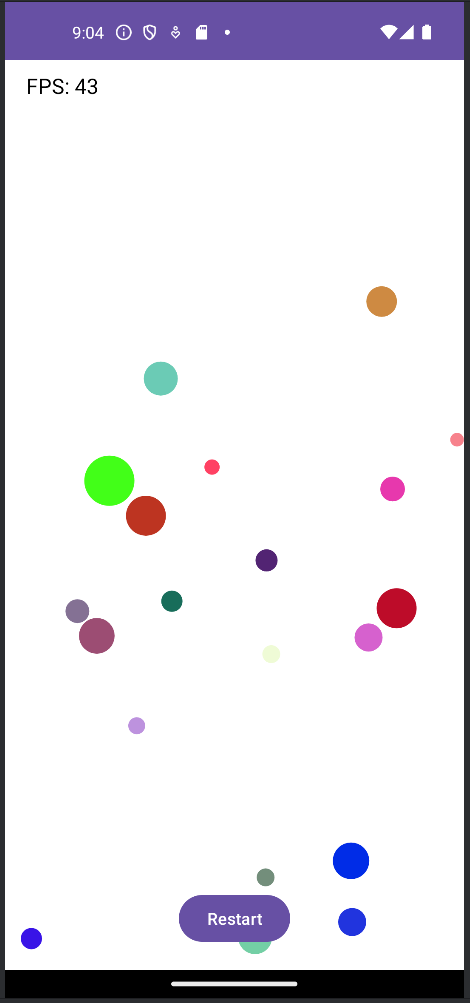
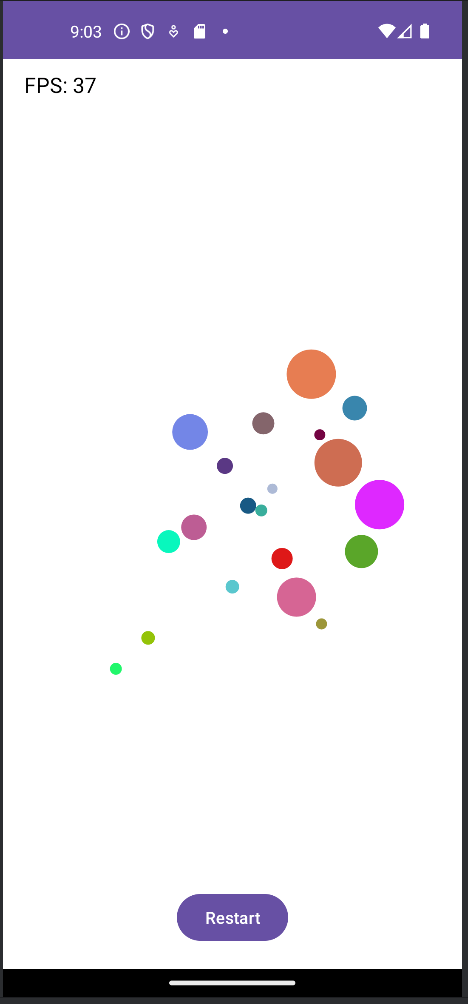
• Sprawdzenie, czy wysokość odbicia maleje.

• 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.

7. 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.