|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | 4 | Data wykonania | 06.04.2023 | Grupa | 1 |
| Ocena |  | Data oddania | 06.04.2023 | Imię Nazwisko | Cezary Tytko |
| Nazwa ćwiczenia | Interfejs użytkownika, wyświetlanie obrazów, dźwięki | | |

**Cel ćwiczenia laboratoryjnego**

Celem ćwiczenia jest zapoznanie studentów z tworzeniem prostej gry mobilnej na platformie Android przy użyciu interfejsu użytkownika **EditText,** **Button,** **TextView, EditText, ImageView, MediaPlayer**. Studenci mają nauczyć się:

1. Projektować interfejs użytkownika, który obejmuje wprowadzanie danych, przycisk do sprawdzania odpowiedzi oraz wyświetlanie wyników i komunikatów.
2. Programować obsługę zdarzeń związaną z rozgrywką.
3. Wykonywać proste operacje na danych wejściowych.
4. Wyświetlać wynik, obrazy i komunikaty na ekranie w czytelny sposób.
5. Obsługiwać odtwarzanie dźwięków w aplikacji.

**3. Opis projektu:**

Gra "Wisielec" to aplikacja mobilna na system Android, w której użytkownik ma za zadanie odgadnąć ukryte słowo, zgadując pojedyncze litery. Aplikacja zapewnia interfejs użytkownika umożliwiający wprowadzanie liter, wyświetlanie aktualnego stanu słowa oraz zarządzanie liczbą pozostałych prób. Gra kończy się wygraną, jeśli użytkownik odgadnie wszystkie litery, lub przegraną, jeśli wyczerpie dostępne próby. Instrukcja z poprzednich zajęć rozszerzona została o obsługę obrazów i dźwięków na odpowiednich etapach gry.

Interfejs użytkownika zawiera następujące elementy:

* **TextView**: do wyświetlania aktualnego stanu zgadywanego słowa,
* **TextView**: do wyświetlania liczby pozostałych prób,
* **TextView**: do wyświetlania komunikatów o stanie gry,
* **EditText**: pole do wpisywania liter przez użytkownika,
* **Button**: przycisk do sprawdzania wprowadzonej litery,
* **Button**: przycisk do resetowania gry.
* **ImageView**: wyświetlanie obrazka wisielca, który zmienia się w zależności od liczby pozostałych prób. Obrazek zmienia się przy każdej błędnej próbie, pokazując postęp w rysunku wisielca.
* **MediaPlayer**: obsługuje odtwarzanie dźwięków w grze. W zależności od stanu gry, odtwarzane są różne dźwięki**:**

 **startSound**: dźwięk powitania na początku gry,

 **correctSound**: dźwięk przy poprawnej odpowiedzi,

 **wrongSound**: dźwięk przy błędnej odpowiedzi,

 **winSound**: dźwięk przy wygranej,

 **loseSound**: dźwięk przy przegranej

**4. Implementacja:**

**Kod został napisany w języku Kotlin w środowisku Android Studio.**

**MainActivity.kt:**

1. package com.example.pumlab3

2.

3.

4. import android.media.MediaPlayer

5. import android.os.Bundle

6. import android.util.Log

7. import android.view.View

8. import android.widget.\*

9. import androidx.appcompat.app.AppCompatActivity

10. import kotlin.random.Random

11.

12. class MainActivity : AppCompatActivity() {

13. private lateinit var wordTextView: TextView

14. private lateinit var triesTextView: TextView

15. private lateinit var statusTextView: TextView

16. private lateinit var letterInput: EditText

17. private lateinit var checkButton: Button

18. private lateinit var resetButton: Button

19. private lateinit var hangmanImage: ImageView

20.

21. private val TAG = "MainActivity"

22. private var wordToGuess = ""

23. private var guessedWord = charArrayOf()

24. private var triesLeft = 9

25. private var imageNumber = 1

26. private val guessedLetters = mutableSetOf<Char>()

27.

28.

29. private lateinit var startSound: MediaPlayer

30. private lateinit var correctSound: MediaPlayer

31. private lateinit var wrongSound: MediaPlayer

32. private lateinit var winSound: MediaPlayer

33. private lateinit var loseSound: MediaPlayer

34.

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

36. super.onCreate(savedInstanceState)

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

38.

39. wordTextView = findViewById(R.id.wordTextView)

40. triesTextView = findViewById(R.id.triesTextView)

41. statusTextView = findViewById(R.id.statusTextView)

42. letterInput = findViewById(R.id.letterInput)

43. checkButton = findViewById(R.id.checkButton)

44. resetButton = findViewById(R.id.resetButton)

45. hangmanImage = findViewById(R.id.hangmanImage)

46.

47. startSound = MediaPlayer.create(this, R.raw.start)

48. correctSound = MediaPlayer.create(this, R.raw.poprawna)

49. wrongSound = MediaPlayer.create(this, R.raw.niepoprawna)

50. winSound = MediaPlayer.create(this, R.raw.wygrana)

51. loseSound = MediaPlayer.create(this, R.raw.przegrana)

52.

53. resetGame()

54. //

55. val imageId = resources.getIdentifier("image${0}", "drawable", packageName)

56. hangmanImage.setImageResource(imageId)

57.

58. checkButton.setOnClickListener { checkLetter() }

59. resetButton.setOnClickListener { resetGame() }

60. }

61.

62. private fun resetGame() {

63. startSound.start()

64. wordToGuess = WORDS[Random.nextInt(WORDS.size)]

65. guessedWord = CharArray(wordToGuess.length) { '\_' }

66. guessedLetters.clear()

67. updateGuessedWord(' ')

68. triesLeft = 9

69. imageNumber = 1

70. checkButton.isEnabled = true

71. statusTextView.text = ""

72. Log.d(TAG, wordToGuess)

73. updateWordDisplay()

74. updateHangmanImage()

75. }

76.

77. private fun updateWordDisplay() {

78. val displayedWord = wordToGuess.mapIndexed { index, char ->

79. if (char.lowercaseChar() in guessedLetters) char else '\_'

80. }.joinToString(" ")

81.

82. wordTextView.text = displayedWord

83. triesTextView.text = "Pozostałe próby: $triesLeft"

84. }

85. private fun updateGuessedWord(letter: Char): Boolean {

86. if (!guessedLetters.contains(letter))

87. {

88. guessedLetters.add(letter)

89. }

90. if (wordToGuess.lowercase().contains(letter)) {

91. // correctSound.start()

92. for (i in wordToGuess.indices) {

93. if (wordToGuess[i] == letter) {

94. guessedWord[i] = letter

95. }

96. }

97. return true

98. } else {

99. // wrongSound.start()

100. triesLeft--

101. imageNumber++

102. return false

103. }

104. }

105.

106. private fun checkLetter() {

107. val input = letterInput.text.toString()

108. if (input.isEmpty() || input.length > 1) {

109. Toast.makeText(this, "Podaj jedną literę!", Toast.LENGTH\_SHORT).show()

110. return

111. }

112.

113. val letter = input[0].lowercaseChar()

114. letterInput.text.clear()

115.

116. if (letter in guessedLetters) {

117. Toast.makeText(this, "Ta litera była już zgadywana!", Toast.LENGTH\_SHORT).show()

118. return

119. }

120.

121. val isCorrect = updateGuessedWord(letter)

122. updateHangmanImage()

123. updateWordDisplay()

124. val gameResult = checkGameStatus()

125. if (gameResult == null) {

126. if (isCorrect){

127. correctSound.start()

128. }

129. else{

130. wrongSound.start()

131. }

132. }

133. else{

134. if (gameResult){

135. winSound.start()

136. }else{

137. loseSound.start()

138. }

139. }

140. }

141.

142. private fun checkGameStatus(): Boolean? {

143. if (String(guessedWord) == wordToGuess.lowercase()) {

144. statusTextView.text = "Gratulacje! Odgadłeś słowo!"

145. checkButton.isEnabled = false

146. //winSound.start()

147. //

148. val imageId = resources.getIdentifier("image${0}", "drawable", packageName)

149. hangmanImage.setImageResource(imageId)

150. return true

151. } else if (triesLeft == 0) {

152. statusTextView.text = "Przegrałeś! Słowo to: $wordToGuess"

153. checkButton.isEnabled = false

154. //loseSound.start()

155. return false

156. }

157. return null

158. }

159.

160. private fun updateHangmanImage() {

161. val imageId = resources.getIdentifier("image${imageNumber}", "drawable", packageName)

162. hangmanImage.setImageResource(imageId)

163. }

164.

165. private val WORDS = listOf(

166. "komputer", "programowanie", "system", "algorytm", "aplikacja", "internet", "dane", "kod", "sieć", "web",

167. "serwer", "framework", "grafika", "programista", "strona", "dysk", "RAM", "CPU", "GPU", "baza danych",

168. "linux", "Windows", "macOS", "sztuczna inteligencja", "machine learning", "cyberbezpieczeństwo", "komponent",

169. "API", "git", "repository", "debugowanie", "skrypt", "hasło", "router", "firewall", "monitor", "cyfrowy",

170. "cryptocurrency", "kodowanie", "architektura", "wersjonowanie", "debugowanie", "interfejs", "responsywność",

171. "frontend", "backend", "hosting", "cyberatak", "synchronizacja", "dokumentacja", "proxy", "wirus", "malware",

172. "VPN", "sterownik", "intranet", "siec neuronowa", "framework", "kryptografia", "algorithm", "program",

173. "framework", "obiekt", "serwer", "komponent", "aplikacja mobilna", "rozwiązanie", "komunikacja", "testowanie",

174. "konfiguracja", "edycja", "plugin", "archiwum", "algorytm", "adres IP", "usługa", "web design", "klient",

175. "terminal", "zaszyfrowany", "deklaracja", "monitoring", "renderowanie", "API key", "kompilator", "hardware",

176. "serwer plików", "debugowanie", "rozpoznawanie", "zarządzanie danymi", "zintegrowany", "data mining",

177. "sieć społecznościowa", "ciasteczko", "wywołanie", "zabezpieczenie", "operating system", "usługa chmurowa",

178. "framework", "transmisja", "responsive design", "open-source", "hosting", "platforma", "komunikat", "serializacja",

179. "aplikacja webowa", "cache", "testing", "storage", "asynchroniczny", "rozwiązanie", "algorytmiczny", "rozpoznawanie",

180. "klucz API", "synchronizacja", "platforma cyfrowa", "zaszyfrowany", "synchronizacja", "open-source", "profil",

181. "cache", "komunikacja", "algorithm", "kompilator", "zabezpieczenie"

182. )

183. }

184.

185.

186.

**activity\_main.xml:**

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

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

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

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

5. android:orientation="vertical"

6. android:padding="16dp"

7. android:gravity="center">

8.

9. <ImageView

10. android:id="@+id/hangmanImage"

11. android:layout\_width="200dp"

12. android:layout\_height="200dp"

13. android:src="@drawable/image0"/>

14.

15. <TextView

16. android:id="@+id/wordTextView"

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

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

19. android:textSize="24sp"

20. android:textStyle="bold"

21. android:text="\_ \_ \_ \_ \_"

22. android:padding="16dp"/>

23.

24. <TextView

25. android:id="@+id/triesTextView"

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

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

28. android:text="Pozostałe próby: 6"

29. android:textSize="18sp"

30. android:padding="8dp"/>

31.

32. <EditText

33. android:id="@+id/letterInput"

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

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

36. android:hint="Podaj literę"

37. android:maxLength="1"

38. android:inputType="text"

39. android:textSize="18sp"

40. android:gravity="center"/>

41.

42. <Button

43. android:id="@+id/checkButton"

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

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

46. android:text="Sprawdź literę"

47. android:padding="8dp"

48. android:layout\_marginTop="10dp"/>

49.

50. <Button

51. android:id="@+id/resetButton"

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

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

54. android:text="Nowa gra"

55. android:padding="8dp"

56. android:layout\_marginTop="10dp"/>

57.

58. <TextView

59. android:id="@+id/statusTextView"

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

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

62. android:text=""

63. android:textSize="20sp"

64. android:textStyle="bold"

65. android:padding="10dp"/>

66. </LinearLayout>

67.

68.

69.

70.

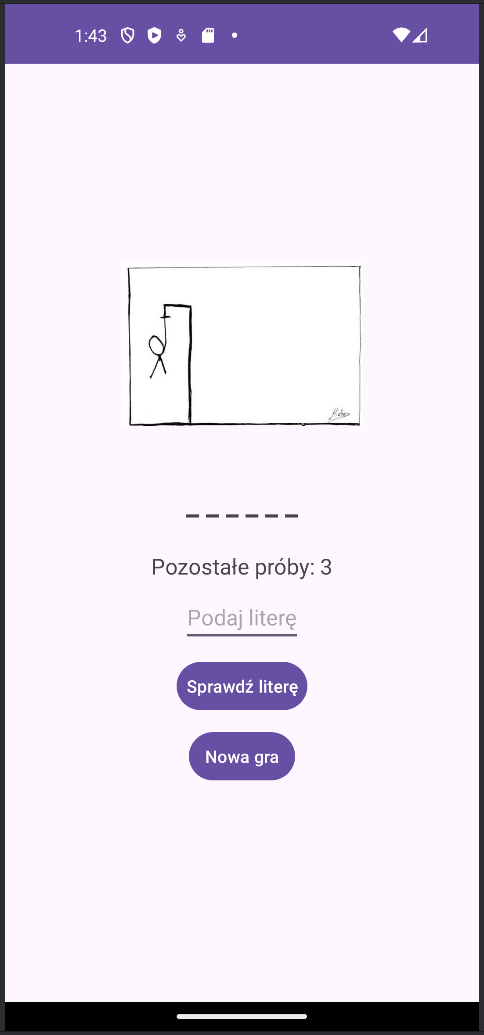
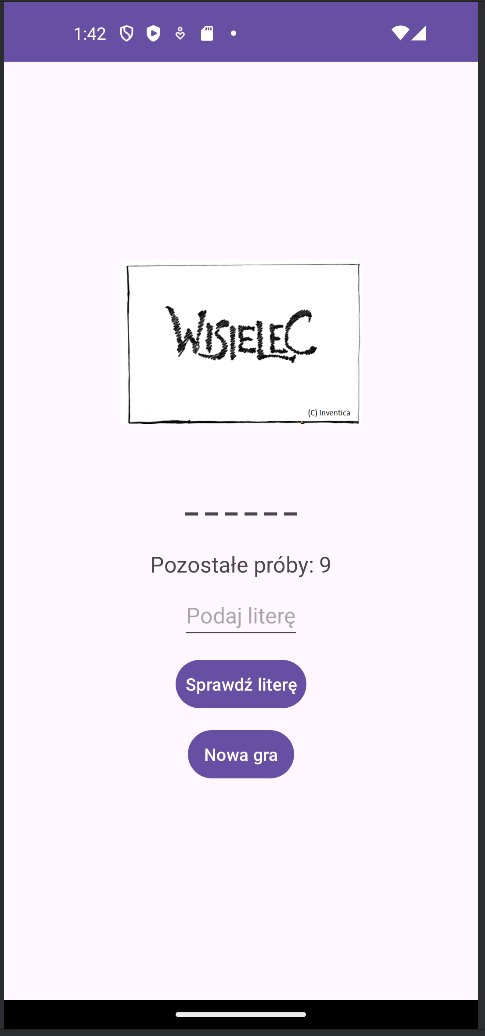
**5. Funkcje kluczowe:**

* **Obsługa zgadywania liter** – użytkownik może wpisywać pojedyncze litery i sprawdzać, czy występują w ukrytym słowie.
* **Zarządzanie stanem gry** – aplikacja aktualizuje wyświetlane słowo oraz liczbę pozostałych prób.
* **Wyświetlanie komunikatów** – informowanie użytkownika o poprawnym lub błędnym odgadnięciu litery, wygranej lub przegranej.
* **Resetowanie gry** – użytkownik może rozpocząć nową rozgrywkę.
* **Wywoływanie obrazów i dźwięku** – na odpowiednich etapach gry.

**6. Testowanie:**

* **Sprawdzenie poprawności wyświetlania ukrytego słowa i zgadywanych liter,**
* **Obsługę błędnych wejść (np. wpisanie więcej niż jednej litery),**
* **Weryfikację poprawnego zakończenia gry po odgadnięciu słowa lub przekroczeniu limitu prób.**
* **Weryfikacja wywoływania odpowiednich obrazów i dźwięków.**

**7. Wyniki:**





**8. Podsumowanie:**

Projekt realizuje klasyczną grę "Wisielec" w aplikacji mobilnej. Implementacja pozwoliła na zdobycie doświadczenia w tworzeniu aplikacji na Androida, obsłudze interakcji użytkownika oraz zarządzaniu stanem gry. Ćwiczenie umożliwiło lepsze zrozumienie obsługi zdarzeń i logiki gry. Instrukcja z poprzednich zajęć rozszerzona została o obsługę obrazów i dźwięków na odpowiednich etapach gry.

**9. Trudności i błędy:**

* Nie wystąpiły żadne trudności.

**10. Źródła i odniesienia:**

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

**11. Dodatkowe materiały:**

* Nie wykorzystano dodatkowych materiałów poza załączonymi do instrukcji plikami dźwiękowymi i graficznymi.