|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | 7 | Data wykonania |  | Grupa |  |
| Ocena |  | Data oddania |  | Imię Nazwisko |  |
| Nazwa ćwiczenia | Cykl życia aplikacji, dynamiczne tworzenie interfejsu, animacja komponentów, obsługa dotyku | | |

**Cel ćwiczenia laboratoryjnego**

Celem tego ćwiczenia laboratoryjnego jest nauczenie studentów podstawowych technik programowania aplikacji mobilnych na platformę Android przy użyciu języka Java. Studenci będą projektować i rozwijać interaktywną grę opartą na GridLayout, w której będą używać przycisków do interakcji z użytkownikiem. W projekcie zastosowano kilka ważnych elementów programowania. Po pierwsze, studenci zapoznają się z tworzeniem interfejsu użytkownika przy użyciu layoutów takich jak LinearLayout i GridLayout. Następnie nauczą się manipulować właściwościami widoków, takimi jak ustawianie tła, marginesów oraz kolorów, a także dodawanie tagów do widoków. Ważnym aspektem będzie obsługa zdarzeń dotykowych i przycisków, poprzez implementację setOnClickListener i dodawanie animacji do przycisków. Studenci dowiedzą się również, jak zarządzać cyklem życia aktywności, korzystając z metod onCreate oraz nowoczesnego podejścia do obsługi przycisku "Wstecz" przy pomocy OnBackPressedCallback. Kolejnym elementem jest losowanie i zarządzanie danymi gry, w tym tworzenie losowych kolorów, parowanie ich i aktualizacja stanu gry.

**Zadanie**

Studenci zostaną poproszeni o utworzenie gry, którą będzie rozszerzeniem przykładu zawartego w ćwiczeniu oraz zastosowaniem wiedzy zdobytej na poprzednich zajęciach.

**Opcjonalne rozszerzenia**

Studenci mogą dodać licznik czasu, który mierzy czas gry i wyświetla go na ekranie. Możliwe jest również wprowadzenie poziomów trudności poprzez zmienianie liczby wierszy, kolumn i kolorów. Kolejnym rozszerzeniem może być integracja z bazą danych, aby przechowywać wyniki graczy w zasobach aplikacji.

**Oczekiwane wyniki**

Oczekuje się, że studenci stworzą funkcjonalną grę z interaktywnymi przyciskami, w której użytkownik może naciskać przyciski, aby odkrywać kolory i tworzyć pary. Interfejs użytkownika powinien być płynny, a animacje przycisków wykonane starannie. Gra powinna również zarządzać cyklem życia aplikacji, umożliwiając jej zresetowanie np. przy użyciu przycisku "Wstecz". Aplikacja powinna być zaprojektowana w sposób umożliwiający dalszy rozwój i dodawanie nowych funkcji np. menu.

**Zadanie**

Na podstawie przedstawionego poniżej szkieletu aplikacji implementującego pętle gry należy uzupełnić rozgrywkę inspirując się klasyczną grą Memory. Jest to to gra, gdzie wymieszane kartoniki należy ułożyć obrazkami skierowanymi w dół. Następnie losujemy dwa z nich – jeśli są takie same, zabieramy je jako zdobytą przez nas parę i możemy losować ponownie. Jeśli są różne – odkładamy je na miejsce, a ruch należy do przeciwnika. Wygrywa osoba, która zdobędzie najwięcej par. W wersji na urządzenia mobilne zadaniem gracza może być jak najszybsze odkrycie wszystkich par.

**Zadania szczegółowe**

1. Dodaj dźwięki do gry (Laboratorium 4).
2. Dodaj pomiar i wyświetlanie czasu.
3. Dodaj warunek zwycięstwa lub porażki (kiedy np. upłynie ustalony czas).
4. Dodaj ekrany: ekran menu, ekran opcji, ekran informacji o autorach, ekran listy wyników (Laboratorium 5).
5. W opcjach gracz, może wybierać rozmiar siatki, poziom trudności - kolory widoczne lub niewidoczne. Można zastosować liczby zamiast kolorów lub kolory i liczby (tak jak w grze 15)
6. Zapamiętanie stanu gry i wyników (tak jak w Laboratorium 2)
7. Opcjonalnie zamiast kolorów/liczb można zastosować obrazki.

**Zasady gry**

Gra polega na znajdowaniu i parowaniu przycisków o tym samym kolorze. Użytkownik klika przyciski, aby odkryć ich kolory. Gdy odkryje dwa przyciski o tym samym kolorze, przyciski pozostają odsłonięte. Gdy odkryje dwa przyciski o różnych kolorach, przyciski są zakrywane po krótkiej animacji. Gra kończy się, gdy wszystkie pary przycisków zostaną poprawnie odkryte.

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

Ten program jest prostą aplikacją na Androida, która dynamicznie tworzy siatkę przycisków w układzie GridLayout. Każdy przycisk ma przypisany losowy kolor. Gdy użytkownik naciśnie przycisk, kolor przycisku zostaje animowany, a następnie porównywany z kolorem wcześniej naciśniętego przycisku. Jeśli kolory są takie same, oba przyciski zostają "ukryte" przez zmianę ich koloru na biały i dezaktywację.

**Tworzenie interfejsu użytkownika za pomocą XML**

Program używa LinearLayout do organizowania elementów interfejsu użytkownika w pionowej kolejności.

GridLayout jest używany do dynamicznego tworzenia siatki przycisków.

**Tworzenie elementów interfejsu użytkownika dynamicznie w Java**

Button jest tworzony dynamicznie w kodzie Java, a następnie dodawany do GridLayout.

**Praca z właściwościami wizualnymi**

Każdy przycisk ma przypisany kolor z tablicy zdefiniowanej w programie.

Po naciśnięciu przycisku, jego marginesy są zwiększane, a tło jest zmieniane, by pokazać wizualny feedback.

**Przetwarzanie zdarzeń (Event Handling)**

Program używa OnClickListener do obsługi zdarzeń kliknięcia przycisku.

Każdy przycisk ma tag zawierający jego pozycję (wiersz i kolumna), który jest używany do identyfikacji naciśniętego przycisku.

**Animacje**

Używana jest animacja powiększenia przycisku (scaleX i scaleY) po jego naciśnięciu, która daje wrażenie animacji przycisku.

**Dynamiczna zmiana właściwości interfejsu użytkownika**

Program dynamicznie zmienia marginesy i tło przycisków, gdy są klikane.

**Zarządzanie stanem aplikacji**

Używany jest OnBackPressedCallback, który pozwala na przechwycenie zdarzenia naciśnięcia przycisku "Wstecz" i restartuje aplikację.

**Utworzenie nowego projektu w Android Studio**

Uruchom Android Studio.

Wybierz opcję "New Project".

Wybierz szablon "Empty Views Activity" i kliknij "Next".

Skonfiguruj nazwę i lokalizację projektu oraz inne parametry według własnych potrzeb.

Kliknij "Finish", aby utworzyć projekt.

**Activity\_main.xml**

*<?*xml version="1.0" encoding="utf-8"*?>*<LinearLayout 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:orientation="vertical"  
 tools:context=".MainActivity">  
  
 <TextView  
 android:id="@+id/topTextView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:textSize="20sp" />  
  
 <GridLayout  
 android:id="@+id/gridLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" />  
</LinearLayout>

**MainActivity.java**

package com.example.simplegridlayoutapp;  
import android.graphics.Color;  
import android.os.Bundle;  
import androidx.activity.OnBackPressedCallback;  
import androidx.appcompat.app.AppCompatActivity;  
import android.view.View;  
import android.widget.Button;  
import android.widget.GridLayout;  
import android.widget.TextView;  
import java.util.ArrayList;  
import java.util.Collections;  
  
public class MainActivity extends AppCompatActivity {  
 int kolumny = 5;  
 int wiersze = 8;  
 int[] colors = {  
 Color.*rgb*(255, 0, 0), Color.*rgb*(0, 255, 0), Color.*rgb*(0, 0, 255),  
 Color.*rgb*(255, 255, 0), Color.*rgb*(0, 255, 255), Color.*rgb*(255, 0, 255),  
 Color.*rgb*(255, 165, 0), Color.*rgb*(128, 0, 128), Color.*rgb*(0, 128, 0),  
 Color.*rgb*(0, 0, 128), Color.*rgb*(75, 0, 130), Color.*rgb*(238, 130, 238),  
 Color.*rgb*(165, 42, 42), Color.*rgb*(255, 192, 203), Color.*rgb*(128, 128, 0),  
 Color.*rgb*(128, 0, 0), Color.*rgb*(255, 127, 80), Color.*rgb*(127, 255, 0),  
 Color.*rgb*(255, 20, 147), Color.*rgb*(46, 139, 87), Color.*rgb*(255, 255, 255),  
 Color.*rgb*(0, 0, 0)  
 };  
  
 int[] indexColors;  
 int clickCounter = 0;  
 int lastColor = -1;  
 int actualColor = 0;  
 int lastRow = -1;  
 int lastCol = -1;  
 int actualRow = 0;  
 int actualCol = 0;  
 public TextView topTextView;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
 topTextView = findViewById(R.id.*topTextView*);  
 topTextView.setText("Czas: ");  
 GridLayout gridLayout = findViewById(R.id.*gridLayout*);  
 gridLayout.setRowCount(wiersze);  
 gridLayout.setColumnCount(kolumny);  
 indexColors = *losujKolory*(kolumny \* wiersze);  
  
 for (int i = 0; i < wiersze; i++) {  
 for (int j = 0; j < kolumny; j++) {  
 Button button = new Button(this);  
 int[] position = {i, j};  
 button.setTag(position);  
 button.setBackgroundColor(colors[indexColors[i \* kolumny + j]]);  
 button.setOnClickListener(this::onButtonClick);  
 GridLayout.LayoutParams params = new GridLayout.LayoutParams(  
 GridLayout.*spec*(i, 1f),  
 GridLayout.*spec*(j, 1f)  
 );  
 params.width = 0;  
 params.height = 0;  
 int marginInPixels = 4;  
 params.setMargins(marginInPixels, marginInPixels, marginInPixels, marginInPixels);  
 button.setLayoutParams(params);  
 gridLayout.addView(button);  
 }  
 }  
 *// Obsługa przycisku "Wstecz"* OnBackPressedCallback callback = new OnBackPressedCallback(true) {  
 @Override  
 public void handleOnBackPressed() {  
 recreate(); *// Restartuj aktywność* }  
 };  
 getOnBackPressedDispatcher().addCallback(this, callback);  
 }  
  
 public void onButtonClick(View v) {  
 int[] position = (int[]) v.getTag();  
 int row = position[0];  
 int col = position[1];  
 actualColor = colors[indexColors[row \* kolumny + col]];  
 actualCol = col; actualRow = row;  
 if (clickCounter >= 2) {  
 clickCounter = 0;  
 }  
 if (lastColor == actualColor && !(lastRow == actualRow && lastCol == actualCol)) {  
 indexColors[actualRow \* kolumny + actualCol] = Color.*WHITE*;  
 indexColors[lastRow \* kolumny + lastCol] = Color.*WHITE*;  
 clickCounter = 0;  
 lastColor = -1;  
 lastRow = -1;  
 lastCol = -1;  
 } else {  
 clickCounter++;  
 lastColor = actualColor;  
 lastCol = actualCol;  
 lastRow = actualRow;  
 }  
 ukryjKolory();  
 *// Animacja do przycisków* v.animate().scaleX(.5f).scaleY(0.5f).setDuration(100).withEndAction(new Runnable() {  
 @Override  
 public void run() {  
 v.animate().scaleX(.8f).scaleY(0.8f).setDuration(100);  
 }  
 });  
 }  
  
 public void ukryjKolory() {  
 GridLayout gridLayout = findViewById(R.id.*gridLayout*);  
 int childCount = gridLayout.getChildCount();  
 int whiteColor = Color.*WHITE*;  
 for (int i = 0; i < childCount; i++) {  
 View child = gridLayout.getChildAt(i);  
 if (child instanceof Button) {  
 child.animate().scaleX(1.0f).scaleY(1.0f).setDuration(100);  
 if (indexColors[i] == whiteColor) {  
 child.setBackgroundColor(whiteColor);  
 child.setEnabled(false);  
 }  
 }  
 }  
 }  
  
 public static int[] losujKolory(int liczbaPol) {  
 if (liczbaPol % 2 != 0) {  
 throw new IllegalArgumentException("losujKolory działa tylko dla liczb parzystych.");  
 }  
 int[] wynik = new int[liczbaPol];  
 ArrayList<Integer> lista = new ArrayList<>();  
  
 for (int i = 0; i < liczbaPol / 2; i++) {  
 lista.add(i);  
 lista.add(i);  
 }  
 Collections.*shuffle*(lista);  
  
 for (int i = 0; i < liczbaPol; i++) {  
 wynik[i] = lista.get(i);  
 }  
 return wynik;  
 }  
}

Zaprezentowana aplikacja nie obsługuje warunku zwycięstwa i wymaga implementacji tej funkcjonalności. Należy połączyć szkielet obsługi gry, dodać dźwięki, menu, wyniki, wybór poziomu trudności i inne elementy, które opisano w części zadania szczegółowe.

**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.